Prescriptive Semantics for Big-Step Modelling Languages

by

Shahram Esmaeilsabzali

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2011

Abstract

With the popularity of model-driven methodologies and the abundance of modelling languages, a major question for a modeller is: Which language is suitable for modelling a system under study? To answer this question, one not only needs to know the range of relevant languages for modelling the system under study, but also needs to be able to compare these languages. In this dissertation, I consider these challenges from a semantic point of view for a diverse range of behavioural modelling languages that I refer to as the family of Big-Step Modelling Languages (BSMLs). There is a plethora of BSMLs, including statecharts, its variants, SCR, un-clocked variants of synchronous languages (e.g., Esterel and Argos), and reactive modules. BSMLs are often used to model systems that continuously interact with their environments. In a BSML model, the reaction of the model to an environmental input is a big step, which consists of a sequence of small steps, each of which can be the concurrent execution of a set of transitions. To provide a systematic method to understand and compare the semantics of BSMLs, this dissertation introduces the big-step semantic deconstruction framework that deconstructs the semantic design space of BSMLs into eight high-level, independent semantic aspects together with the enumeration of the common semantic options of each semantic aspect. The dissertation also presents a comparative analysis of the semantic options of each semantic aspect to assist one to choose one semantic option over another. A key idea in the big-step semantic deconstruction is that the high-level semantic aspects in the deconstruction recognize a big step as a whole, rather than only considering its constituent transitions operationally.

A novelty of the big-step semantic deconstruction is that it lends itself to a systematic semantic formalization of most of the languages in the deconstruction. The dissertation presents a parametric, formal semantic definition method whose parameters correspond to the semantic aspects of the deconstruction, and thus it produces prescriptive semantics: The manifestation of a semantic option in the semantics of a BSML can be clearly identified.

The way transitions are ordered to form a big step in a BSML is a source of semantic complexity: A modeller needs to be aware of the possible orders of the execution of transitions when constructing and analyzing a model. The dissertation introduces three semantic quality attributes that each exempts a modeller from considering an aspect of ordering in big steps. The ranges of BSMLs that support each of these semantic quality attributes are formally specified. These specifications indicate that achieving a semantic quality attribute in a BSML is a cross-cutting concern over the choices of its different semantic options. The semantic quality attributes together with the semantic analysis of individual semantic options can be used in tandem to assist a modeller or a semanticist to compare two BSMLs or to create a new, desired BSML from scratch.

Through the big-step semantic deconstruction, I have discovered that some of the semantic aspects of BSMLs can be uniformly described as forms of synchronization. The dissertation presents a general synchronization framework for behavioural modelling languages. This framework is based on a notion of synchronization between transitions of complementary roles. It is parameterized by the number of interactions a transition can take part in, i.e., one vs. many, and the arity of the interaction mechanisms, i.e., exclusive vs. shared, which are considered for the complementary roles to result in 16 synchronization types. To enhance BSMLs with the capability to use the synchronization types, a synchronizer syntax is introduced for BSMLs, resulting in the family of Synchronizing Big-Step Modelling Languages (SBSMLs). Using the expressiveness of SBSMLs, the dissertation describes how underlying the semantics of many modelling constructs, such as multi-source, multi-destination transitions, various composition operators, and workflow patterns, there is a notion of synchronization that can be systematically modelled in SBSMLs.

Acknowledgements

For the past six years, Professor Nancy A. Day has been all I could have asked for as a Ph.D. supervisor. She taught me research, writing, self criticism, and many non-research life lessons. She gave me the freedom to pursue my research ideas, while advising and helping me when I was in trouble. This work heavily relies on her insights and advice.

I am grateful to have had Professor Joanne M. Atlee as a member of my Ph.D. committee and as a collaborator. Her original, insightful suggestions have been crucial in improving my work.

Professor Daniel M. Berry has been influential in my research and personal development during my graduate studies. I am fortunate that I have had a chance to learn so much from him.

I am appreciative that Professor Krzysztof Czarnecki has been a member of my Ph.D. committee. His encompassing comments have enabled me to broaden my research perspective.

I am thankful to Professor Juergen Dingel for being on my Ph.D. committee. His clear, insightful comments have provided me with a fresh perspective about my work.

I would like to thank the members of the Watform research lab, past and present, for their support and friendship over the last six years. I thank the people in my neighbouring office, Shoham Ben-David, Zarrin Langari, and Ali Taleghani, for being there to chat and laugh. Vlad Ciubotariu, Alma Juarez Dominguez, Hazem Shehata, and other Watform students, have been all great friends over these years. Pourya Shaker has been a great friend and officemate. Professor Jianwei Niu has been a great friend and collaborator. The staffs in Cheriton School of Computer Science have been so helpful and friendly. Wendy Rush has been a great supportive friend.

The Frizzell and Brummelhuis families have been like a second family for me in Canada. Evelyn and Don Frizzell provided the perfect environment in their house for me to focus on my work. Teresa and Gerry Brummelhuis have been caring, loyal friends for all these years, including me frequently in their family functions.

I thank my parents, Esmat and Karim, for their love, support, and constant encouragement to learn. I thank my brothers, Bahram, Hadi, and Kourosh, for their love, friendship, and constant support. I thank Harry Roettele for being a kind, always-stimulating friend, as well as a great sports buddy. The Roettele-Bonn family has been a great source of support and fun.

Christa gave me the love and energy to finish this work. Without her love and patience this work would not have been possible.

To Christa

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

“A general language-independent framework of semantical concepts would help to standardize terminology, clarify similarities and differences between languages, and allow rigorous formulation and proof of semantic properties of languages. A language designer could analyze proposed constructs to help find undesirable restrictions, incompatibilities, ambiguities, and so on.” [94, p.437]

Robert Tennent

With the increasing presence of software systems in our environment, there is a need for systematic, reliable ways to specify, create, verify, and maintain these systems. Many believe that it is through the use of models that the complexity of ever-growing software systems can be conquered. A model is an abstraction of a phenomenon, which is represented in a modelling language. Often when modelling a software system, there are many alternative languages that can be used. To narrow the range of alternatives, a modeller needs to answer the question of why language *A*, and not language *B*, is a more appropriate choice in a certain context. This dissertation considers this question from a semantic point of view: What are the semantic criteria to compare *A* and *B* to choose one over another?

In this dissertation, I undertake the above research challenge for the class of *big-step modelling languages*. I introduce the term Big-Step Modelling Languages (BSMLs)[1] to describe

[1]In this dissertation, all of the abbreviations are intended to be pronounced as their constituent letters, and not as the phrase they represent. As such, based on this pronunciation convention, I use the appropriate form of the indefinite article for an abbreviation.

a family of behavioural modelling languages that are often used for the requirements specification of interactive and reactive systems, which communicate with their environments continuously. There is a plethora of BSMLs, many with graphical syntax (e.g., some statecharts variants [41, 99] and Argos [68]), some with textual syntax (e.g., reactive modules [3] and Esterel [14]), and some with tabular format (e.g., SCR [47, 46]). These languages have in common that the reaction of a system to an environmental input is a *big step*, which consists of a sequence of *small steps*, each of which is the execution of a set of transitions. Commonly, the syntax of a BSML includes a combination of hierarchical control states, events, and variables syntax that are used in a transition syntax that often has guard and action parts. BSMLs provide two major advantages to a modeller. First, the reaction of a model to an environmental input can be conveniently modelled as multiple small steps, without worrying about a new environmental input being missed during the reaction of the model to the current environmental input. And second, since the reaction of a model to an environmental input can consist of more than one transition, a model can be decomposed into orthogonal parts, each of which can take part separately in the reaction. As such, a modeller can decompose a model into parts, each of which either corresponds to a physical component of a system under study or is used to facilitate the separation of concerns in modelling.

The semantics of many BSMLs have been a contentious area of research. For example, searching on the internet for the articles whose titles include both the terms “statecharts” (or “statechart”) and “semantics” returns 139 articles. (This search was carried out using Google Scholar on December 20, 2010.) Among these articles, there are ones that introduce a new semantics for statecharts and articles that use different semantic definition methods for defining the semantics of statecharts. While the above situation demonstrates the difficulties of categorizing and comparing two BSMLs even when they are labelled with the same name, a more complicated situation arises when the semantics of BSMLs with different names are considered, which usually have less in common with each other than the ones with the same name.

To compare the semantics of two BSMLs, *A* and *B*, their semantics must be known sufficiently clearly and there must be a semantic criterion. But semantics of modelling languages are defined in different ways, and therefore, either they need to be transformed to a single format or the semantic criteria by which they are compared must be relevant for different kinds of semantic definitions. Furthermore, one might be interested in knowing whether there is yet another modelling language *C* that is even better than *A* and *B* according to certain semantic criteria; or whether there is a way to define such a superior language. Thus, instead of considering only individual pairs of languages and identifying their comparison semantic criteria, it is desirable to create a common semantic framework for BSMLs in which distinct languages can be distinguished and can be compared according to some common semantic criteria.

Creating such a semantic framework for a large group of languages with different syntactic and semantic characteristics, however, is a major challenge. In principle, such a semantic framework for a set of languages should consist of a set of *semantic decision points* that categorize the semantics of the languages based on the *semantic decisions* that each language adopts at each semantic decision point. Ideally, (i) these semantic decision points correspond to a set of high-level, already-established semantic concepts that are understandable to the users of these languages; and (ii) the semantic decisions are formalizable in a manner such that a resulting semantics for a language clearly embodies its semantic decisions. Thus, if one understands such high-level semantic decision points and semantic decisions, she/he is likely to understand their formal semantics. The resulting formal semantics are *prescriptive semantics* [4, 5] because the formalism is used in an "active" [4, 5] role to design, and prescribe, a semantics, based on its semantic decisions. In contrast, in a *descriptive semantics* [4, 5], a semanticist seems to have been employed "*as a describer*" only "for recording design decisions already made", as opposed to, "playing a part in the language design process" [5].

Another major challenge is to identify the semantic criteria that can be used to compare two languages in a manner that helps a language designer or a modeller to choose one over another. Some of these criteria could be *local* in that each compares the semantic decisions of two languages at a certain semantic decision point. Other criteria could be *global* in that each compares the set of semantic decisions of two languages at multiple semantic decision points collectively. Both kinds of semantic criteria, however, are useful: The former kind of criteria helps one to make individual semantic decisions when choosing or creating a language, while the latter kind of criteria helps one to compare two semantics as a whole.

1.1 Approaches to Semantic Categorization and Comparison

This section briefly overviews the different approaches that have been used in the literature to categorize modelling languages. In general, these approaches can be categorized into three groups: (i) informal, imprecise approaches, (ii) formal, implementation-biased approaches, and (iii) formal, deconstructional approaches.

1.1.1 Informal, imprecise approaches

A framework in this category is often a useful, survey-like categorization of a set of related modelling languages. An example of this approach is the informal comparison of statecharts variants by von der Beeck [99]. This seminal work compares 20 statecharts variants based on a list of 19 “problems” [99], which includes a combination of syntactic, semantic, and semantic-formalization issues. In similar frameworks, Fidge compares process algebras CCS [72], CSP [48], and LOTOS [52], and, Crane and Dingel compare different variations of UML StateMachines and their supporting tools [20]. These frameworks, by definition, are usually insightful summarizations of different, often ad hoc, features of a set of languages. They are presented at different levels of preciseness and systematicness. In general, however, the summarization that each offers can neither be easily extended with new analytical insights nor can be used as the basis for a unified formalization of the semantics of the languages that are considered. Much effort is needed to interpret the imprecisely stated features of these languages. Therefore, even if a uniform semantic formalization of these languages would be possible, it would lead to descriptive semantics: For most semantic features of the languages, it is not clear how to formalize them in a way that they are manifested clearly in a semantic definition, prescriptively.

1.1.2 Formal, implementation-biased approaches

A framework in this category offers a set of semantic decision points that are derived from the tool suite that it represents. I call such a tool suite a *tool-support generator framework* (TGF), which takes the definition of a language, including its semantics, as input, and generates tool support, such as model checking and simulation capability for that language, as output [81, 25, 28, 75, 65, 6, 38, 87]. TGFs differ in the *semantic input formats* (SIF) they use, and the procedure by which they obtain tool support for a language. An SIF can be an existing formalism, such as higher-order logic [25], structural operational semantic format [28], or a new formalism, such as template semantics [75, 74]. A TGF often strives for open-ended flexibility and extensibility, to accommodate new notations, and thus its SIF is a general, expressive format for semantic definition. An SIF, by its mission, does not represent a high-level semantic framework with intuitively understandable semantic decision points, but rather it is an expressive semantic definition language that is designed to be flexible, extensible, and implementable. Technically, the semantics of a language specified in an SIF can be considered as prescriptive: All its semantic decisions are trivially embodied in its semantic definition. However, it would be misleading to consider these semantic definitions as prescriptive, because they are based on a semantic framework whose semantic decision points are often general semantic concepts that are applicable to a wide, open-ended group of languages. As such, an SIF often produces descriptive semantics because it aims "for generality at the expense of simplicity and elegance." [5, p.284]

1.1.3 Formal, deconstructional approaches

A framework in this category is organized around a set of semantic decision points that are intuitively understandable for stakeholders of the semantics of languages. These frameworks are also accompanied by semantic formalizations that produce prescriptive semantics. A notable example of these frameworks is the semantic framework of Huizing and Gerth [50] for a class of BSML semantics that supports only internal events. The only semantic decision point in their framework is for the semantics of internal events. The semantic formalization method that they choose is specialized not only to embody each of the five possible semantic decisions, but also to highlight the differences between these decisions when formalized. Furthermore, they identify three semantic criteria that allows one to choose one event semantics over another.

In the *Unifying Theories of Programming* [49], Hoare and Jifeng advocate a set of principles for unification and categorization of languages. They consider these principles in the context of semantic decision points and semantic decisions that are mainly relevant for process-algebraic languages and programming languages.

Other frameworks can be considered in more than one of the above three categories. For example, Maggiolo-Schettini, Peron, and Tini compare three semantic variations of statecharts in the context of a Structured Operational Semantic (SOS) semantic definition framework [67]. I categorize their work into the third category of formal, deconstructional approaches because they identify semantic decision points that correspond to understandable semantic concepts for the family of statecharts. However, their work can also be categorized under the second category because they intentionally adopt a limited compositional syntax for statecharts, similar to process-algebras, in order to be able to use SOS.

This dissertation introduces a semantic framework for BSMLs in the formal, deconstructional approach to semantic categorization and comparison.

1.2 Thesis Overview

This dissertation introduces the *big-step semantic deconstruction* framework for BSMLs, which unifies the semantics of a large group of seemingly different modelling languages into the family of BSMLs. The big-step semantic deconstruction *deconstructs* the semantics of various BSMLs into eight *semantic aspects* and enumerates the common *semantic options* found in existing BSMLs for each semantic aspect. In a few cases, I have added semantic options that complement the ones found in the existing BSMLs; these semantic options are included to make the range of possible semantic options for a semantic aspect more systematic. The semantic aspects and the semantic options are the semantic decision points and the semantic decisions of the family of BSMLs, respectively. The dissertation presents a parametric semantic definition method that uniformly formalizes the semantics of most of the languages in the deconstruction, producing prescriptive semantic definitions. To compare the languages in the deconstruction, the dissertation presents semantic criteria that differentiate two BSMLs based on their differences at the scope of a single semantic aspect. Furthermore, three *semantic quality attributes* are introduced that compare two BSMLs based on their semantic options for corresponding semantic aspects.

Like any other deconstructional analyses of a set of languages, the big-step semantic deconstruction provides insights about the range of possible BSML semantics, their interrelationships, as well as, clues about ways to further the unification of the BSML semantics. This dissertation describes how some of the semantic aspects in the big-step semantic deconstruction can be unified as different forms of *synchronization* that distinguish different BSMLs based on whether they support certain kinds of synchronizing transitions or not. Hence, BSMLs can be extended with a synchronization capability, to result in the family of *synchronizing big-step modelling languages* (SBSMLs). With the expressive power of explicit synchronization, SBSMLs can be used to model the semantics of various existing modelling constructs, revealing that these modelling constructs all use different forms of synchronization in their semantics, and that they can be adopted also by the languages in the family of SBSMLs.

> **Thesis Statement.** The big-step semantic deconstruction is a novel, high-level semantic framework for the family of BSMLs, with a formal semantic definition method that produces prescriptive semantics for most of the languages in the family. Using this framework, BSMLs can be compared at individual semantic decision points. Some BSMLs offer novel semantic quality attributes that each relieves a

modeller from dealing with some of the complexity of ordering of transitions in a big step of model. The set of all BSMLs that subscribe to a semantic quality attribute can be formally specified by enumerating all combinations of the semantic decisions that each yields a BSML semantics that subscribes to the semantic quality attribute. BSMLs can be compared based on the semantic quality attributes that each BSML has. The family of SBSMLs introduces synchronization capability to BSMLs. The semantics of SBSMLs can be formally described in a prescriptive manner, similar to the way the semantics of BSMLs are described. The introduction of synchronization for BSMLs deems some of the semantic decision points of the big-step semantic deconstruction as unnecessary, because these semantic decision points and their corresponding semantic decisions can be uniformly described as forms of synchronization. Lastly, SBSMLs are expressive enough to model the semantics of many existing modelling constructs, such as the semantics of multi-source, multi-destination transitions, some of the composition operators of template semantics, and some workflow patterns. These modelling constructs can be seamlessly adopted by SBSMLs.

Big-Step Semantic Deconstruction. The various ways that the semantics of events, variables, concurrency, and priority can be defined in BSMLs create a large design space for the semantics of BSMLs.

This dissertation introduces the big-step semantic deconstruction that is a novel method to decompose and organize the semantics of various BSMLs into eight semantic aspects and the common semantic options found in existing BSMLs for each semantic aspect. The semantic aspects are identified mainly based on conceptual sequentiality in the process of creating a big step in a BSML. The choice of a semantic option for a semantic aspect is independent of the choice of a semantic option for another semantic aspect, except for a few cases where certain combinations of semantic options lead to inconsistent BSML semantics. These cases are excluded by the big-step semantic deconstruction; cf., Figure 3.3 on page 34. To achieve understandability in the big-step semantic deconstruction and prescriptiveness in its formalization, whenever applicable, I have considered a big step as a whole, rather than considering only its constituent transitions operationally. For the same reasons, I have used a common normal-form syntax that is expressive enough to model the syntax of many BSMLs. As a result, I have been able to create a framework that focuses on semantics, without being sidetracked unnecessarily by the syntactic variations of BSMLs. An existing BSML can be identified in this framework by, first, determining a mapping from its syntax to the normal-form syntax, and second, by determining the set of semantic options that represent its semantics. A new BSML can be defined in this framework by choosing a set of syntactic features and semantic options that have not been considered together in a language previously.

In the big-step semantic deconstruction, I have considered only those languages that each (i) has an explicit stage in its semantics for sensing the environmental inputs, and (ii) its operational semantics specifies the reaction of a model to an environmental input as a sequence of small steps, instead of one single step. For example, process algebras [9] are not considered in the big-step semantic deconstruction because they support neither of the above two criteria. Typically, a language that supports these criteria supports also a combination of events and variables in its syntax, in order to provide a mechanism to relate the small steps in the sequence. This dissertation does not consider languages such as UML StateMachines [78]that can *buffer* the received environmental inputs or the generated events of a model. However, many of the semantic aspects in the big-step semantic deconstruction are relevant also for these languages. For example, the notion of run to completion in UML StateMachines [78] is similar to the notion of maximality of a big step in the big-step semantic deconstruction.

To assist a modeller to choose one semantic option over another, for each semantic option in the big-step semantic deconstruction, the dissertation presents a set of semantic properties, each of which is labeled as an *advantage* or a *disadvantage* of the semantic option. These labels are determined based on agreed-upon, common wisdom in the literature or a straightforward rationalization presented in the dissertation that is supported by examples.

Prescriptive Semantics. The formalization of the semantics of different subsets of BSMLs has been a contentious area of research, as evident by the large number of publications devoted to the formalization of the semantics of these subsets of BSMLs. The stakeholders of these formalizations, like all other formal semantics, vary from tool developers, to modellers, to semanticists. These stakeholders, however, have competing interests. For example, a tool developer is usually interested in a precise, operational formalization of a semantics; a modeller might compromise between understandability and preciseness; and a semanticist might be more interested in a formalization that reveals the semantic decisions and semantic properties of a semantics clearly. The big-step semantic deconstruction provides an opportunity to decouple such concerns.

This dissertation presents a semantic definition framework that produces prescriptive, formal semantic definitions for a large subset of BSMLs: The high-level semantic options of a BSML, chosen by the various stakeholders of the BSML, can be traced clearly as separate parts of its semantic definition. The semantic definition framework is a parametric *semantic definition schema* to formalize the semantics of most of BSMLs in the big-step semantic deconstruction. By instantiating the parameters of the semantic definition schema, an operational BSML semantics is derived. The semantic aspects of BSMLs correspond to disjoint parameters of the semantic definition schema, and the semantic options of each semantic aspect correspond to the possible values for the parameter that represents the semantic aspect. The semantic definition schema, its parameters, and the values of the parameters are specified in standard logic and set theory. Except for a couple of cases, the specification of a value of a parameter of the semantic definition schema is independent of the specification of a value of another parameter. The exceptions deal with semantics that support a notion of *combo step*, which partitions a big step to consecutive segments of small steps.

The big-step semantic deconstruction together with this semantic definition framework allow the underlying semantic options of a BSML to be chosen before being formalized. Therefore, the semantic formalization process is not used as a way to *discover* the range of possible semantic design decisions at the time of formalization but as a medium to *specify* the already-made semantic design decisions of a BSML. By analogy, BNF is a prescriptive method for defining syntax, as opposed to pre-BNF methods, which were descriptive [4]. “In general, the descriptive approach aims for generality even at the expense of simplicity and elegance, while the prescriptive approach aims for simplicity and elegance even at the expense of generality.” [5, p.284] A corollary of a prescriptive semantic definition method is that it specifies a clear scope for a class of semantics.

To validate the correctness of my semantic definition framework formally, a set of formal, reference semantic definitions for existing languages are needed to check my formalization against them. However, for each BSML, or a subclass of BSMLs, there are usually many semantic definitions available in the literature, specified using a range of different semantic definition methods. Thus, instead of proving the correctness of my formalization selectively with respect to one or more semantic definition, I have used inspection as a method to gain confidence in my formalization: While mapping the semantics of an existing BSML into my semantic aspects and their options, I have used many example models as witnesses for the correctness of my mapping.

Semantic Quality Attributes. The complexity of dealing with the semantic intricacies related to the ordering of the executions of the small steps of a big step can be a source of complexity and distraction for a modeller. For example, a modeller, or a model reviewer, might need to ensure that a certain enabled transition does not mistakenly become disabled in certain execution scenarios. A semantic quality attribute of a modelling language is a desired semantic property that is common to all models specified in that language.

This dissertation introduces three semantic quality attributes for BSMLs, each of which aims to alleviate a kind of semantic intricacy related to the ordering of the small steps of big steps. The dissertation presents the outlines of the proofs that demonstrate that each of the three semantic quality attributes is realized by any BSML whose constituent semantic options satisfy a set of identified necessary and sufficient constraints over the choices of the semantic options. These constraints reveal positive and negative interrelationships among seemingly independent semantic options by identifying their collective effect. Also, the dissertation shows formally how it is possible to achieve a semantic quality attribute for a BSML by constraining its syntax via syntactic well-formedness conditions. This latter approach to achieve a semantic quality attribute advocates an approach in design of modelling languages in which the syntax and the semantics of a languages are considered together, as opposed to an approach in which semantics is merely a function that maps syntax to its meaning.

Using the semantic deconstruction, the relative advantages and disadvantages of the semantic options, and the characterization of the semantic quality attributes in terms of semantic options, a modeller or a language designer can either (i) use the semantic quality attributes to narrow the range of semantic options for a language, or (ii) gain insights about a language's attributes after choosing its semantic options. The above two means for language comparison can be used: (a) as a semantic catalog, to compare the semantics of existing BSMLs and choose an appropriate BSML; (b) as a semantic scale, to assess the semantic properties of a BSML; or (c) as a semantic menu, to help design a BSML from scratch.

Synchronization for BSMLs. The prescriptive semantic definition framework in this dissertation formalizes the *enabledness* semantic aspects and the *structural* semantic aspects differently. The enabledness semantic aspects deal mainly with how the state of a model changes from one small step to the next. The structural semantic aspects deal with the meaning of the hierarchial structure of a model. While the enabledness semantic aspects are uniformly formalized using a snapshot-element-based approach, inspired by and adapted from template semantics [75, 74], the structural semantic aspects are formalized via logical predicates that determine how a set of enabled transitions can form a small step. Compared to the formalization of the enabledness semantic aspect, the formalization of the structural semantic aspects are less systematic in that they do not use a uniform specification method, similar to the snapshot elements used for the formalization of the enabledness semantic aspects. In searching for a more systematic method, I discovered how underlying different structural semantic aspects, there is a unifying theme: Each represents a form of synchronization.

This dissertation introduces a notion of *synchronization type* and a *synchronizer* syntax that not only preclude the necessity of having most of the structural semantic aspects in the big-step semantic deconstruction, but also provide the means to explain and recognize that the semantics of various modelling constructs are forms of synchronization. It shows how each of the semantics of multi-source, multi-destination transitions [41, 86], the composition operators of template semantics [75, 74], and the essence of many workflow patterns [96] uses its own different form of synchronization. Introducing synchronization to BSMLs results in the class of *synchronizing big-step modelling languages* (SBSMLs). A synchronizer in an SBSML model has one of the 16 synchronization types. A synchronizer is associated with a set of transitions whose executions are governed by the synchronization constraints that the synchronizer enforces. A transition might be controlled by more than one synchronizer.

The formalization of the semantics of synchronization types is done via a novel, declarative approach that uses relation types to characterize the set of all synchronizing transitions of a model according to a certain synchronization type.

This dissertation presents also transformation schemes, in the form of algorithms, (i) to model the semantic options of the structural semantic aspects of BSMLs, and (ii) to model the semantics of the modelling constructs whose semantics can be described by synchronizers. The dissertation presents the outlines of the proofs that demonstrate that each transformation scheme is correct with respect to its formal description, in case (i), and with respect to its natural-language description, in case (ii).

1.3 Validation

The big-step semantic deconstruction is a **novel** semantic framework in that it covers a range of seemingly unrelated modelling languages that have not been considered together previously in a unifying semantic framework. Chapter 3 presents this framework, its high-level semantic aspects, the semantic options of each semantic aspect, and the example BSMLs that subscribe to each semantic option. The enabledness semantic aspects and their corresponding semantic options are **high level** in that each considers a big step as a whole. The structural semantic aspects are **high level** in that each corresponds to an already-established semantic concept such as concurrency, preemption, and priority. The dissertation validates that the semantics of a **wide range of BSMLs** can be expressed in my semantic framework by enumerating the constituent semantic options of each, as summarized in Table 3.12, on page 83.

The semantic definition schema for formalizing BSML semantics, presented in Chapter 4, produces **prescriptive semantics**. For each semantic aspect, the semantic definition schema has a parameter. For those semantic options of the semantic aspect that are supported by the semantic definition schema, I provide a parameter value for its corresponding parameter. Thus, a semantic definition of a BSML can be partitioned into parts, each of which corresponds to a constituent semantic option of the BSML. The prescriptiveness of these semantics can be inspected, and validated, immediately: The formalization of the values of the parameters of the semantic definition schema are mainly independent.

A semantic option of a semantic aspect can be compared with another on the basis of their relative advantages and disadvantages. To facilitate such **comparisons**, for the semantic options of each semantic aspect, the list of their corresponding advantages and disadvantages are presented in a tabular format, in Chapter 3. These tables include also a list of example BSMLs that subscribe to each semantic option.

The formal semantics of two BSMLs can be compared on the basis of the **novel** semantic quality **attributes** that each supports. Chapter 5 introduces three novel semantic quality attributes together with the enumeration of the BSMLs that support each of the semantic quality attributes. A BSML *A* can then be compared with BSML *B* on the basis of these three semantic quality attributes. Such a **comparison** considers the collective effect of the constituent semantic options of *A* and *B*: For each BSML *A* and *B*, it is determined whether its set of constituent semantic options satisfies each of the three semantic quality attributes or not, using the formal specification of the semantic quality attributes in Section 5.3. The dissertation validates the **soundness** of the comparison of BSMLs based on these semantic quality attributes by proving that the specification of the classes of BSML semantics that support each of the semantic quality attribute is correct.

The family of SBSMLs provides **synchronization** capability for the languages in the family of BSMLs. The formal semantics of SBSMLs is presented in Chapter 7, in a similar **prescriptive** way as the formal semantics of BSMLs are described. Compared to the semantics of BSMLs, with synchronization capability available for SBSMLs, some of the structural semantic aspects of BSMLs become **unnecessary** in SBSMLs because their semantics can be restated using a notion of synchronization, as described in Section 6.4.2. Section 7.4.2 presents algorithms that specify how the semantic options of a structural semantic aspect can be modelled by synchronization. It validates the **correctness** of each of these algorithms by proving its correctness with respect to the semantics of its corresponding semantic option.

Finally, SBSMLs are expressive enough to model the semantics of a range of modelling constructs, including the semantics of multi-source, multi-destination transitions [41, 86], composition operators of template semantics [75, 74], and many workflow patterns [96], as described in Section 6.4. Section 7.4 validates the **expressiveness** of SBSMLs by presenting algorithms that specify how the semantics of these modelling constructs can be described using synchronization. The **correctness** of each of these algorithms is validated by proving its correctness with respect to the natural-language description of its corresponding modelling construct.

1.4 Contributions of the Thesis

The following list summarizes the contributions of this dissertation:

- The dissertation introduces a high-level, deconstructional semantic framework for the family of BSMLs in the form of semantic aspects and their corresponding semantic options. This framework is called the big-step semantic deconstruction. The semantic aspects and the semantic options of the big-step semantic deconstruction relate a large number of modelling languages through their underlying unifying semantic concepts.
- The dissertation introduces a prescriptive method to define the semantics of most of the BSMLs in the big-step semantic deconstruction in a manner that distinguishable parts of a semantic definition can be traced back to the high-level semantic concepts of the deconstruction.
- The dissertation introduces a set of semantic criteria for comparing two BSMLs. These semantic criteria enable the comparison of two BSMLs, (i) by enumerating the relative advantages and disadvantages of each of the constituent semantic options of the two BSMLs,

and (ii) by identifying the overall semantic quality attributes of the constituent semantic options of each of the two BSMLs.

- The dissertation introduces an explicit synchronization capability to the family of big-step modelling languages, resulting in the new class of synchronizing big-step modelling languages.
- The dissertation introduces transformation schemes that use the synchronization capability of synchronizing big-step modelling languages to model some of the semantic aspects in the big-step semantic deconstruction, as well as, to model the semantics of various modeling constructs. These transformation schemes reveal that underlying the semantics of these semantic aspects and modelling constructs there is a notion of synchronization.

1.5 Outline of the Thesis

Chapter 2 introduces the common syntactic constructs and semantic concepts that are used throughout the thesis. It also briefly describes how the syntax of various big-step modelling language (BSMLs) can be represented using the common syntax that is introduced in this chapter.

Chapter 3 presents the deconstruction of BSML semantics into eight semantic aspects and their corresponding semantic options. Each semantic aspect is presented in a separate section, accompanied by example BSMLs that use each of its semantic option and by example models that demonstrate the role of each semantic option. The research results reported in this chapter have been published [35, 36], with a more detailed version of the results disseminated in a technical report [34].

Chapter 4 presents the formalization of the semantics of BSMLs. First, a parametric semantic definition schema is introduced whose parameters correspond to semantic aspects. Then, the possible values of each parameter are presented. A version of this semantic definition framework, which does not cover all of the semantic aspects and options, has been published [31].

Chapter 5 presents the three semantic quality attributes for BSMLs. It also formally specifies the subset of BSMLs that satisfy each of the semantic quality attributes. For each of these specifications, the outline of the proof of its correctness is presented. A summary of the research results in this chapter has been accepted to be published [32].

Chapter 6 introduces a synchronization capability for BSMLs, resulting in the class of synchronizing big-step modelling languages (SBSMLs). Also, it informally describes how various modelling constructs and semantic options can be modelled using the synchronization capability of SBSMLs. A summary of the research results in this chapter has been published [33].

Chapter 7 presents a semantic definition framework for SBSMLs. It also presents transformation schemes, in the form of algorithms, that describe how SBSMLs can be used to specify the semantics of the modelling constructs and the semantic options of the structural semantic aspects. For each transformation scheme, the outline of the proof of its correctness is presented.

Lastly, Chapter 8 presents the conclusions of the thesis and discusses new directions for future work.

Each of the Chapter 3, Chapter 4, Chapter 5, Chapter 6 has its own separate related work section, at the end of the chapter.

Chapters 5 and 6 can be read independently of Chapter 4, except for some of the proofs in Section 5.3.

In the list of references at the end of the dissertation, a bibliographic entry is followed by the list of the pages in the dissertation that reference that entry.

Chapter 2

Common Syntax and Semantics

"Computer scientists collectively suffer from what I call the Whorfian syndrome[1] – the confusion of language with reality. Since these devices are described in different languages, they must all be different. In fact, they are all naturally described as state machines.
[1] See `http://en.wikipedia.org/wiki/Sapir-Whorfhypothesis`" [60, p.60]

Leslie Lamport

This chapter introduces the syntactic constructs and the semantic concepts that are used, in Chapter 3, to describe the semantic deconstruction of BSMLs. Section 2.1 presents a normal-form syntax for BSMLs. Section 2.2 presents the common basic semantics for big-step modelling languages. Section 2.3 discusses how the syntax of many BSMLs can be translated into the normal-form syntax of BSMLs described in Section 2.1.

2.1 Normal-Form Syntax

There is a plethora of BSMLs, including those with graphical syntax (e.g., statecharts variants [99], Argos [68]), those with textual syntax (e.g., reactive modules [3], Esterel [14]), and those with tabular/equational syntax (e.g., SCR [46, 47]). As is usual when studying a class of related notations, a *normal-form syntax* [49] is used that is sufficiently expressive to represent the

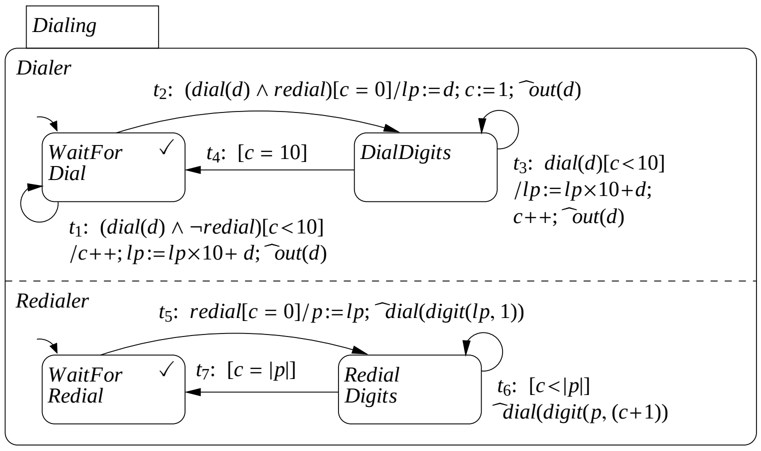

Dialing
Dialer
t_2: $(dial(d) \wedge redial)[c = 0]/lp := d; c := 1; {}^\frown out(d)$
WaitFor Dial ✓
t_4: $[c = 10]$
DialDigits
t_3: $dial(d)[c < 10]$ $/lp := lp \times 10 + d;$ $c{+}{+}; {}^\frown out(d)$
t_1: $(dial(d) \wedge \neg redial)[c < 10]$ $/c{+}{+}; lp := lp \times 10 + d; {}^\frown out(d)$
Redialer
t_5: $redial[c = 0]/p := lp; {}^\frown dial(digit(lp, 1))$
WaitFor Redial ✓
t_7: $[c = |p|]$
Redial Digits
t_6: $[c < |p|]$ ${}^\frown dial(digit(p, (c+1)))$

Figure 2.1: A model for dialing and redialing.

syntax of other notations. This section presents a normal-form syntax for BSMLs. A BNF representation of this syntax is presented at the end of Section 2.1.2, after describing the graphical representation of the syntax.

In the normal-form syntax of BSMLs, a model is defined through two main components: (i) a set of *control states* that are organized as a *hierarchy tree*; and (ii) a set of *transitions* between the control states. Figure 2.1 shows a BSML model in this syntax for a dialing system that has two functionalities. It can either collect a 10-digit phone number or redial a previously dialed number. The syntactic elements of this model are described next. But for now, it is helpful to note that the rounded boxes create a hierarchy tree of control states and an arrow between two control states is a transition.

2.1.1 Control States

A control state (e.g., *DialDigits* in Figure 2.1) is a named artifact that a modeller uses to represent a noteworthy moment in the execution of a model. Such a moment is an abstraction that groups together the past behaviours (consisting of inputs received by the model and the model's past reactions to these inputs) that have a common set of future behaviours. By using a control state, a modeller can describe future behaviour in terms of the current control state and the current environmental inputs.

A control state has a name and a *type*, which is either *Basic*, *Or*, or *And*. Graphically, a control state is shown by a rounded rectangular with a label on it that is its name. The set of control states of a model form a *hierarchy tree*. A leaf node of a hierarchy tree is a *Basic* control state. An *And* or an *Or* control state, which is a non-leaf node of a hierarchy tree, is called a *compound* control state. Relations *child*, *descendant*, *parent*, and *ancestor* are defined with their usual meanings, as follows. The child relation relates a control state with the immediate control state below it in the hierarchy tree. The root of the hierarchy tree is not a child of any control state; this control state is referred to as the *root*. A control state is descendent of another if it is its child through transitivity. The parent relation is the inverse of the child relation: A control state, s, is the parent of a control state, s', if s' is its child. Each control state, except for the root, has a unique parent by the definition of the hierarchy of control states, which is a tree. A *Basic* control state is not a parent of any control state. A control state is ancestor of another if it is its parent through transitivity. In the model in Figure 2.1, control state *Dialing* is an *And* control state and has two *Or* child control states, *Dialer* and *Redialer*; control state *DialDigits* is a child of *Dialer* and a descendant of *Dialing*. The name of an *And* control state is specified by a separate solid box attached to the top left of the rounded box that represents it. The children of an *And* control state are separated by dashed lines. An *And* control state is required to have more than one child, while an *Or* control state need not. When a child of an *And* control state is an *Or* control state, the rounded box that represents it is not drawn because the children of the *Or* control state can be surrounded by the border lines of its surrounding *And* control state and its dashed lines. An *Or* control state has a *default* control state, which is its child and is identified by an incoming arrow that has no source control state. In the model in Figure 2.1, *WaitForDial* is the default control state of *Dialer*. The control states with "✓" on them, e.g., *WaitForDial*, are *stable* control states, and have a semantic role in determining the length of a big step, or subsegments of a big step, as will be described in the next chapter.

A model may have no *And* control states. The root control state must be an *Or* control state so that the arena of every transition, as described in Section 2.1.2, is guaranteed to exist. An *And* control state may have a *Basic* control state as its child, although usually a *Basic* control state is a child of an *Or* control state. In some of the examples in this dissertation, the root control state of the model is not shown.

Two control states *overlap* if they are the same or one is an ancestor of the other. For example, in the model in Figure 2.1, control states *Dialing* and *Dialler* are overlapping, but not control states *Dialler* and *Redialler*. The *least common ancestor* of two control states is the lowest control state (closest to the leaves of the hierarchy tree) that is an ancestor of both. In the model in Figure 2.1, the least common ancestor of *DialDigits* and *RedialDigits* is *Dialing*. Two control states are *orthogonal* if neither is an ancestor of the other and their least common ancestor is an *And* control state. In Figure 2.1, *DialDigits* and *RedialDigits* are orthogonal. The *scope* of a transition is the least common ancestor of its source and destination control states. The *arena* of a transition is the lowest *Or* control state in the hierarchy tree that is the ancestor of both the source and destination control states of the transition. In the model in Figure 2.1, both the scope and the arena of transition t_1 is the *Or* control state *Dialer*. In general, however, the scope and arena of a transition need not be the same.

2.1.2 Transitions

Each transition (e.g., t_1 in Figure 2.1) has a name, both a *source* and a *destination* control state, and four optional parts: (i) an *event trigger*, which is a conjunction of event literals, some of which may be negated (a negated event being prefixed by a “¬”); (ii) a *guard condition* (GC) (enclosed by “[ ]”), which is a boolean expression over the set of variables of the model; (iii) a set of *assignments* (prefixed by a “/”); and (iv) a set of *generated events* (prefixed by a “⁀”). A generated event may have a parameter that can be modelled by associating a variable with it.

An assignment consists of a *left-hand side variable* (LHS), and a *right-hand side expression* (RHS). All variable expressions and assignments of models are assumed to be well-typed. Variables and events are global; local variables and scoped events can be modelled by a renaming that makes them globally unique.

Two transitions are *orthogonal* if their source control states are orthogonal, as well as their destination control states. A transition t is an *interrupt for* transition t' when the sources of the transitions are orthogonal and one of the following conditions holds: (i) the destination of t' is orthogonal with the source of t, and the destination of t is not orthogonal with the sources of either transitions (Figure 2.2(a)); or (ii) the destination of neither transition is orthogonal with the sources of the two transitions, but the destination of t is a descendant of the destination of t' (Figure 2.2(b)).

The normal-form syntax is a collection of various syntactic constructs adopted from different BSMLs. Some of these constructs have been adapted to fit the overall design of the normal-form syntax. For example, the notions of *And* and *Or* control states are adopted from Harel’s statecharts [41]; a few of the syntactic definitions for control states are adopted from Pnueli

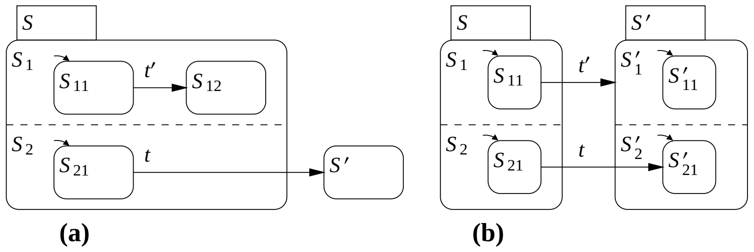


Figure 2.2: Interrupting transitions.

and Shalev's work on the semantics of statecharts [86]; and the notion of *stable* control state is adopted from the `pause` command in Berry and Gonthier's work that introduces Esterel [14].

2.1.3 BSML Syntax in BNF

Figure 2.3 is the BNF representation of the normal-form syntax of BSMLs, as described in Section 2.1.1 and Section 2.1.2. For the sake of brevity, the BNF in Figure 2.3 does not include the declaration of events, variables, etc. The normal-form syntax uses both boolean expressions, represented via the "b-expression" symbol in the BNF, and numeric expressions, represented via the "num-expression" symbol. It is assumed that all expressions and assignments are well-typed.

The default and stable control states, which were graphically represented by an arrow without a source and a "✓" label, respectively, are represented textually in the BNF, via the "Default"and "Stable" symbols, respectively. The symbol "identifier" is a unique name to identify a syntactic element such as a control state or a transition; an identifier starts with a character but can also include numbers.

Throughout this dissertation, I refer to the normal-form syntax of a BSML through a combination of elements in the BNF grammar in Figure 2.3 and a set of helper definitions, including the ones described in Section 2.1.1 and Section 2.1.2. The BNF grammar allows me to specify inductive definitions over the hierarchy tree of models, while the helper definitions allows me to specify operational definitions that deal with the sequences of small steps of big steps.

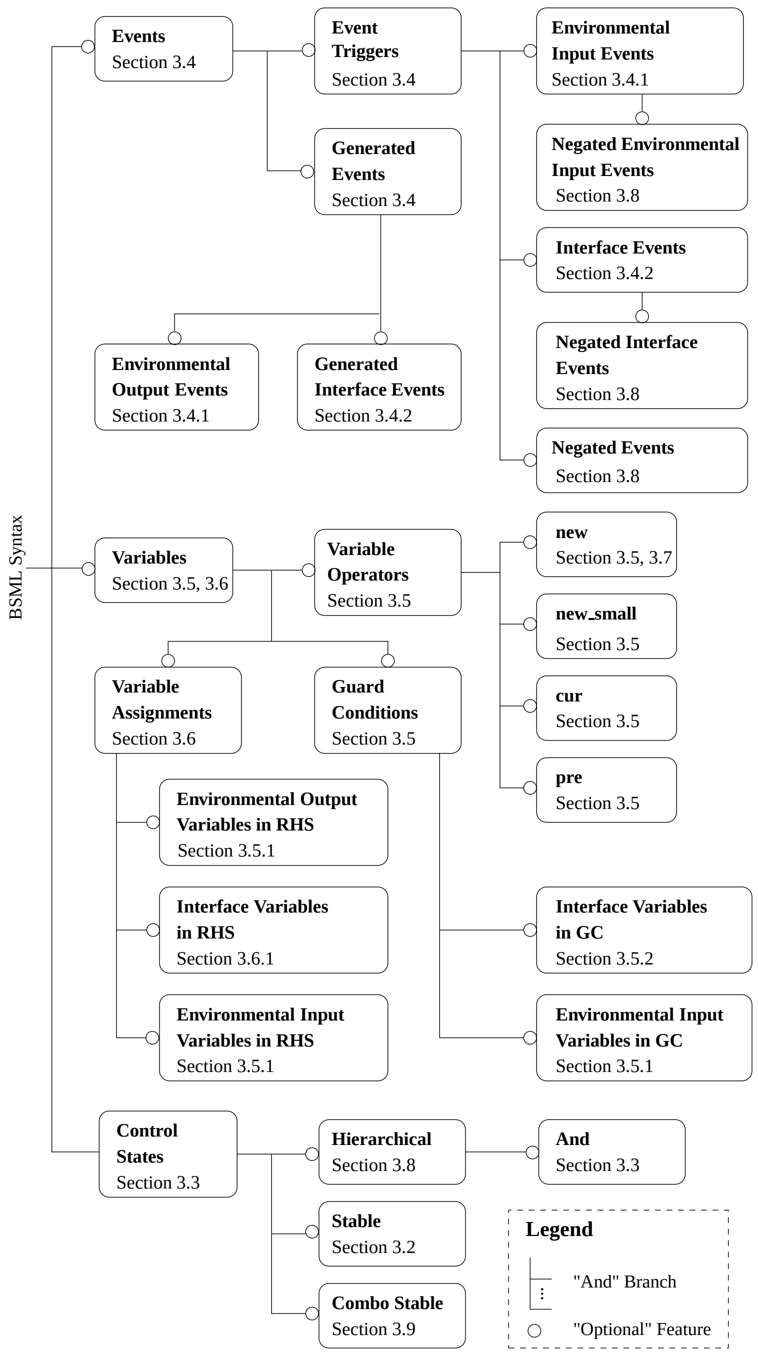
⟨root⟩	::=	“Or” ⟨state⟩+
⟨state⟩	::=	⟨Orstate⟩ | ⟨Andstate⟩ | ⟨Basicstate⟩
⟨Orstate⟩	::=	“Or” ⟨identifier⟩ ⟨state⟩+
⟨Andstate⟩	::=	“And” ⟨identifier⟩ (⟨state⟩ ⟨state⟩+)
⟨Basicstate⟩	::=	(“Basic” | “Default” | “Stable” | “Combo-Stable”) ⟨identifier⟩
⟨transitions⟩	::=	⟨transition⟩+
⟨transition⟩	::=	⟨identifier⟩ ⟨source⟩ ⟨destination⟩
		⟨trigger⟩ (“[”⟨guard⟩“]”)
		(“/”⟨assignment⟩∗) (“⁀” “{”⟨genevent⟩∗“}”)
⟨source⟩	::=	⟨state⟩
⟨destination⟩	::=	⟨state⟩
⟨trigger⟩	::=	(⟨postevent⟩ | ⟨negevent⟩)∗
⟨guard⟩	::=	⟨b-expression⟩
⟨assignment⟩	::=	⟨variable⟩“:=”⟨expression⟩“;”
⟨expression⟩	::=	⟨b-expression⟩ | ⟨num-expression⟩
⟨genevent⟩	::=	⟨posevent⟩
⟨posevent⟩	::=	⟨identifier⟩
⟨negevent⟩	::=	“¬”⟨posevent⟩
⟨variable⟩	::=	⟨identifier⟩

Figure 2.3: The BNF for the BSML normal-form syntax.

2.1.4 BSML Syntactic Features

Figure 2.4 is a feature diagram [56] that represents the combination of syntactic constructs of BSMLs that are of interest for the semantic decision points (semantic aspects) of BSMLs. Each feature in the diagram is labelled with the sections in Chapter 3 that describe its role and detailed semantics. The syntax of a BSML must have a notion of transition to specify the behaviour of a system, thus control states are necessary to define transitions. However, all other syntactic features in the feature diagram of Figure 2.4 are optional. In practice, the syntax of most useful BSMLs support at least events or variables.

A leaf node of the feature diagram represents a primitive syntactic feature of BSMLs. For example, the **Negated Events** node is the syntactic feature that allows the negation of an internal event to be used in the event trigger of a transition. A non-leaf node represents a syntactic feature that has additional syntactic sub-features in its children nodes. For example, the **Event Triggers** node is the syntactic feature that has syntactic sub-features, **Environmental Input**



BSML Syntax
Events Section 3.4
Event Triggers Section 3.4
Environmental Input Events Section 3.4.1
Negated Environmental Input Events Section 3.8
Generated Events Section 3.4
Interface Events Section 3.4.2
Negated Interface Events Section 3.8
Environmental Output Events Section 3.4.1
Generated Interface Events Section 3.4.2
Negated Events Section 3.8
Variables Section 3.5, 3.6
Variable Operators Section 3.5
new Section 3.5, 3.7
new_small Section 3.5
cur Section 3.5
pre Section 3.5
Variable Assignments Section 3.6
Guard Conditions Section 3.5
Environmental Output Variables in RHS Section 3.5.1
Interface Variables in RHS Section 3.6.1
Environmental Input Variables in RHS Section 3.5.1
Interface Variables in GC Section 3.5.2
Environmental Input Variables in GC Section 3.5.1
Control States Section 3.3
Hierarchical Section 3.8
And Section 3.3
Stable Section 3.2
Combo Stable Section 3.9
Legend
"And" Branch
"Optional" Feature

Figure 2.4: Feature diagram for the syntactic variation points of interest to BSML semantic aspects.

Events, **Interface Events**, and **Negated Events**. In the feature diagram in Figure 2.4, only "and" branches are used for sub-features of a feature: In these branches, if a feature is chosen, then all of its children sub-features are also chosen, except for the sub-features that are connected to a small circle, which are "optional" sub-features. An optional feature, as opposed to a "mandatory" feature, need not be chosen if its parent feature is chosen. All of the features in the diagram in Figure 2.4, except the **Control States** feature, are optional features.

The **Events** and **Variables** nodes both have child nodes that each represents the necessary syntactic feature for a specific kind of communication semantics. For example, the **Environmental Input Events** and **Environmental Input Variables** features are used for the environmental communication through events and variables, respectively. Similar syntactic features as for the environmental communication exist for communication through interface events and interface variables.

The **Stable** and **Combo Stable** child nodes of the **Control States** feature represent special kinds of control states that are used to determine when a big step ends or when a segment of a big step ends, respectively.

2.2 Common Basic Semantics

Initially, a model resides in the default control states of its root control state, no event is present, and its variables have their initial values. If a model resides in an *And* control state, it resides in all of its children. If a model resides in an *Or* control state, it resides in one of its children, which is by default its "default" child. The operational semantics of a BSML describes how a model reacts to an *environmental input* via a *big step*. An environmental input is a set of events and variable assignments that are received from the environment. Figure 2.5 depicts a big step T, which is a reaction of a model to environmental input I. A big step is an alternating sequence of *small steps* and *snapshots*, where a small step is the execution of a set of transitions (t_i's), and a snapshot is a tuple that stores information.[1] The T_i's ($1 \leq i \leq n$) are small steps of T, and sp, sp', and sp_i's ($1 \leq i < n$) are its snapshots. In the examples throughout this dissertation, a big step is represented as the sequence of its small steps; e.g., T is represented as $\langle T_1, T_2, \cdots, T_n \rangle$.

[1]Big steps and small steps are often called macro steps and micro steps, respectively. I adopt new terms to avoid association with the fixed semantics of the languages that use those terms. The big-step/small-step terminology has been used in the study of the operational semantics of programming languages in a similar spirit as used here [83].

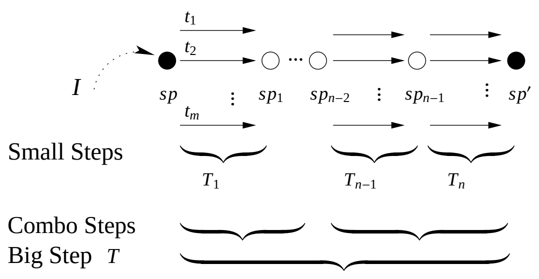


Figure 2.5: Steps.

Some BSMLs, such as RSML [63] and Statemate [43], introduce an intermediate grouping of a sequence of small steps into a *combo step*. The small steps of a combo step hide some of their effects, e.g., the effect of their assignments, from one another. Sections 3.4, 3.5, 3.6, and 3.9 describe when combo steps are useful. In representing a big step, the scope of each of its combo steps is identified by a surrounding "$\llparenthesis\ \rrparenthesis$". For example, $\langle \llparenthesis T_1, T_2 \rrparenthesis, \llparenthesis T_3, T_4 \rrparenthesis \rangle$ is a big step that consists of two combo steps and four small steps.

2.2.1 Snapshots

A snapshot of a model is a tuple that consists of sets of information that each captures a facet of the computation of a model in a particular moment of execution. As the execution of the model proceeds, its current snapshot gets updated. A snapshot often consists of: (i) a *configuration*, which is a set of control states; (ii) a *variable evaluation*, which is a set of ⟨variable name, value⟩ pairs; and (iii) a set of *events*. Each of a big step, a small step, or a combo step has a *source* and *destination* snapshot (e.g., sp and sp' are the source and destination snapshots of T).

2.2.2 Enabledness

In each small step of a BSML model, a set of *enabled*, *high-priority* transitions is chosen to be executed. In general, a transition is enabled if its event trigger and guard condition are satisfied, and its source control state is in the source configuration of the small step. Different semantic options use different snapshots of a big step to define enabledness. A transition is high priority if it cannot be replaced with another transition of higher priority, according to the semantics of priority in the BSML. At each snapshot, there could exist multiple sets of enabled transition, each of which is a *potential small step* that can be taken.

2.2.3 Execution

The effects of the execution of the transitions of a small step create its destination snapshot. When a transition is executed, it leaves its source control state (and its descendants), and enters its destination control state (and its descendants). When entering an *Or* control state, a transition enters its default control state, and when entering an *And* control state, it enters all of its children. Thus, if the source (destination) control state of a transition is an *And* control state, the execution of the transition includes exiting (entering) the children of the source (destination) control state. The semantics of event generation and variable assignment differ between BSMLs.

In a few, non-common cases, transition execution can be more involved; e.g., when the least common ancestor of the source and destination control states of a transition is an *And* control state. A discussion of these cases is included in Section 4.3.

The execution of a small step is *atomic*: the variable assignments and event generation of one transition cannot be seen by another transition, except for one of the semantic options for events, described in Section 3.4. Because of atomicity, a sequence of assignments on a transition can be converted to a set of assignments [61, 64].

2.2.4 Environmental inputs

When choosing a BSML for modelling a system under study, the domain of the system must satisfy the assumptions of the BSML regarding the model's ability to take multiple transitions in response to an environmental input and not miss other inputs. There are three types of assumptions:

- *Fast computation*: This assumption, which is usually referred to as the "synchrony hypothesis" or the "zero-time assumption" [14, 40], postulates that the system is fast enough, and thus never misses an input. The domain of systems that are modelled using this paradigm is called "reactive systems" [14, 40, 44]. A reactive system is usually a mission-critical system that is meant to react to environmental inputs in a timely manner, at the rate produced by the environment; e.g., the controller system of a nuclear reactor. No environmental inputs are missed. Therefore, the implementation of a reactive system should guarantee that the system is fast enough that all environmental inputs are processed.

  A reactive system might be either implemented as embedded software on a piece of hardware, or directly as a piece of hardware [10, 12, 29]. As such, many of the BSMLs that support the synchrony hypothesis adopt their underlying principles from the principles of hardware. For example, a BSML might equate a big-step as a reaction of the model during a "tick" of the global clock of the system; e.g., Esterel [14] and Argos [68]. In this dissertation, the synchronous languages whose syntax and semantics are closer to the hardware and directly support a notion of clock, such as Lustre [39] and Signal [7], are not considered.

- *Helpful environment*:[2] This assumption postulates that the environment is helpful by issuing an input only when the system is ready [40]. The domain of systems that are modelled using this paradigm is called "interactive systems" [40]. An interactive system is different from a reactive system in that the rate of environmental inputs is dictated by the system, rather than by the environment. An example of an interactive system is an automated banking machine, which interacts with its environment (i.e., a customer) at its own rate when it is ready, rather than at the rate the customer might like to provide inputs for it. An environmental input might be missed by the system when the system is busy processing a previous environmental input. Therefore, a modeller needs to ensure that the requirements of a system are consistent with the assumption that an environmental input might be missed.

- *Asynchronous communication*: This assumption postulates that the system has a buffering mechanism to store the environmental inputs, and thus never misses an environmental input. As such, no constraints are imposed on the computation speed of the system, or on the frequency of the arrival of environmental inputs.

In this dissertation, only the BSMLs with the first two assumptions are considered. The third assumption is mutually exclusive with these two assumptions. The BSMLs that adhere to the first two assumptions share many semantic options. As such, sometimes it is difficult and unnecessary to label a BSML conclusively as following one or the other assumption.

[2]The term "helpful environment" is adopted from a similar notion in *Interface Automata* [26].

2.3 Representing BSMLs in the Normal-Form Syntax

It is straightforward to represent the syntax of many BSMLs in our normal-form syntax. Template semantics [75, 74], which has a "composed hierarchical syntax" comparable to our normal-form syntax, describes the mapping of the syntax of many BSMLs to its syntax. In this section, the syntactic representations of a few less obvious constructs are considered. Additionally, a few syntactic representations are discussed in Chapter 3, when their corresponding semantic decision points are presented. Also, Chapter 6 presents the syntactic representations of modelling constructs whose semantics relies on synchronization.

2.3.1 Control States

A BSML may not include the notion of control states. If a model's reaction to an environmental input is always independent of its past behaviours, then the notion of control state is not useful for the model. In our normal-form syntax, one way to represent the syntax of a BSML that does not have control states is to create a single control state that serves as the source and destination control states of all transitions. The notion of the hierarchy of control states might still be useful for specifying priority between transitions in such a BSML (cf., Section 3.8 for priority semantics).

A BSML with a textual syntax without explicit control states, such as Esterel [14], realizes a line of a program as a control state. For example, in Esterel [14], an `exit` statement within a parallel command of a model moves the flow of control from within the parallel command to the next command outside the scope of the parallel command. The parallel command and the command after it can be conceptually considered as control states with the parallel command being an *And* control state. The `exit` statement can be considered as a transition that connects the two control states.

SCR [46, 47] is a BSML that uses a tabular format. The notions of "modes" and "transitions between modes" in its syntax can be represented by the notions of control states and transitions between control states, respectively.

2.3.2 Transitions

In the normal-form syntax for transitions, event triggers with disjunctions are not allowed, because an event trigger that has disjuncts can be split into multiple transitions, each with only one of the disjuncts of the original event trigger and exactly the same other elements as the original; such a transformation yields a model that is semantically the same as the original model [86].

To model multi-source, multi-destination transitions using single-source, single-destination transitions, they can be split into multiple transitions that are either taken together, or are not taken at all. Such an execution scheme requires synchronization between these split transitions. In Chapter 6, where a notion of synchronization for BSMLs is introduced, such a translation is described.

2.4 Summary

This chapter presented a normal-form syntax for BSMLs that is designed to model the syntax of many BSMLs. It first described this syntax informally, followed by a presentation of its BNF. It presented a feature diagram that shows the variation points in the BSML syntax that are relevant for the semantic decision points. It presented also a common semantics for the normal formal syntax of BSMLs. The variations of this semantics are described in Chapter 3.

Chapter 3

Semantic Deconstruction

"No one gets angry at a mathematician or a physicist whom he or she doesn't understand, or at someone who speaks a foreign language, but rather at someone who tampers with your own language, with this "relation," which is yours . . . " [27, p.115]

Jacques Derrida

This chapter introduces a deconstruction of BSML semantics into eight *semantic aspects* and their corresponding *semantic options*. Section 3.1 is an overview of the deconstruction, followed by sections that describe each semantic aspect. Section 3.10 describes the few identified *side effects* between the semantic options of different semantic aspects. Section 3.11 provides a summary of the semantic options via a table that specifies the semantic options of some common BSMLs. The formalization of the semantics described in this chapter are presented in Chapter 4.

3.1 Overview of Semantic Aspects

The operation of a big step can be deconstructed into the stages described in Figure 3.1. This systematic deconstruction is based on: (i) conceptual sequentiality in the process of creating a small step (partly based on the syntactic elements of the model), (ii) orthogonal concerns in the operation of a big step, and (iii) semantic variation points in existing BSMLs.[1] Each

[1] In this dissertation, I use the terms "semantic variation points" and "semantic variations" interchangeably with the terms "semantic decision points" and "semantic decisions", respectively. I use the former pair of terms to refer

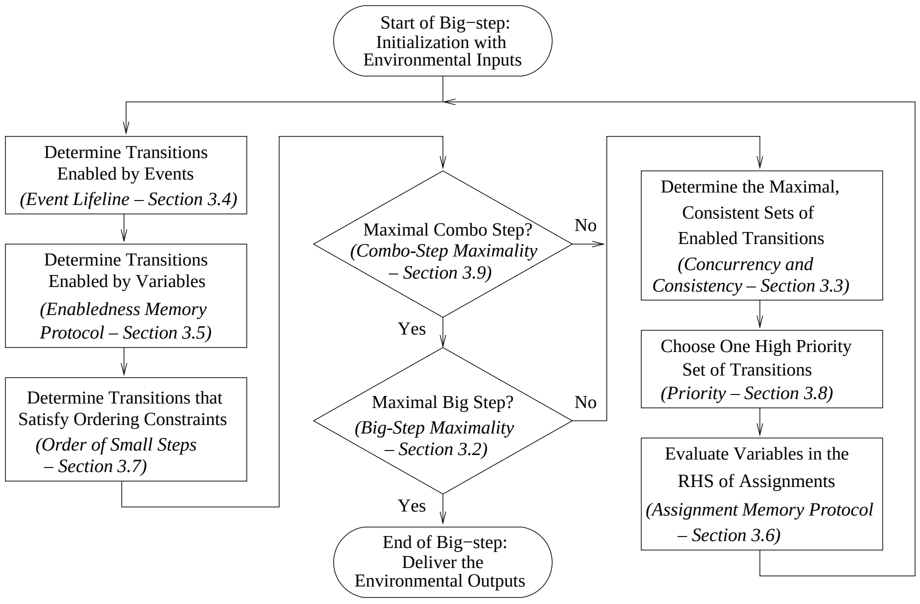

Start of Big−step: Initialization with Environmental Inputs
Determine Transitions Enabled by Events
(Event Lifeline – Section 3.4)
Determine Transitions Enabled by Variables
(Enabledness Memory Protocol – Section 3.5)
Determine Transitions that Satisfy Ordering Constraints
(Order of Small Steps – Section 3.7)
Maximal Combo Step?
(Combo-Step Maximality – Section 3.9)
No
Yes
Maximal Big Step?
(Big-Step Maximality – Section 3.2)
No
Yes
End of Big−step: Deliver the Environmental Outputs
Determine the Maximal, Consistent Sets of Enabled Transitions
(Concurrency and Consistency – Section 3.3)
Choose One High Priority Set of Transitions
(Priority – Section 3.8)
Evaluate Variables in the RHS of Assignments
(Assignment Memory Protocol – Section 3.6)

Figure 3.1: Operation of a big step.

stage of the diagram is associated with one of the *semantic aspects* and is labelled with the corresponding section of the chapter that describes it. A semantic aspect may be decomposed into some semantic sub-aspects. A semantic aspect or sub-aspect has a number of *semantic options*, each of which is a semantic variation for carrying out a stage. In Section 3.11, it is shown how the semantics of different BSMLs can indeed be specified through the stages of this diagram, or more particularly, using the semantic aspects and their options. Therefore, it is shown that the semantic aspects cover all semantic variation points of the languages that are considered in the scope of this dissertation. In a few cases, I have added semantic options for a semantic aspect that complement the ones found in the existing BSMLs; these semantic options are included to make the range of possible semantic options for a semantic aspect more systematic.

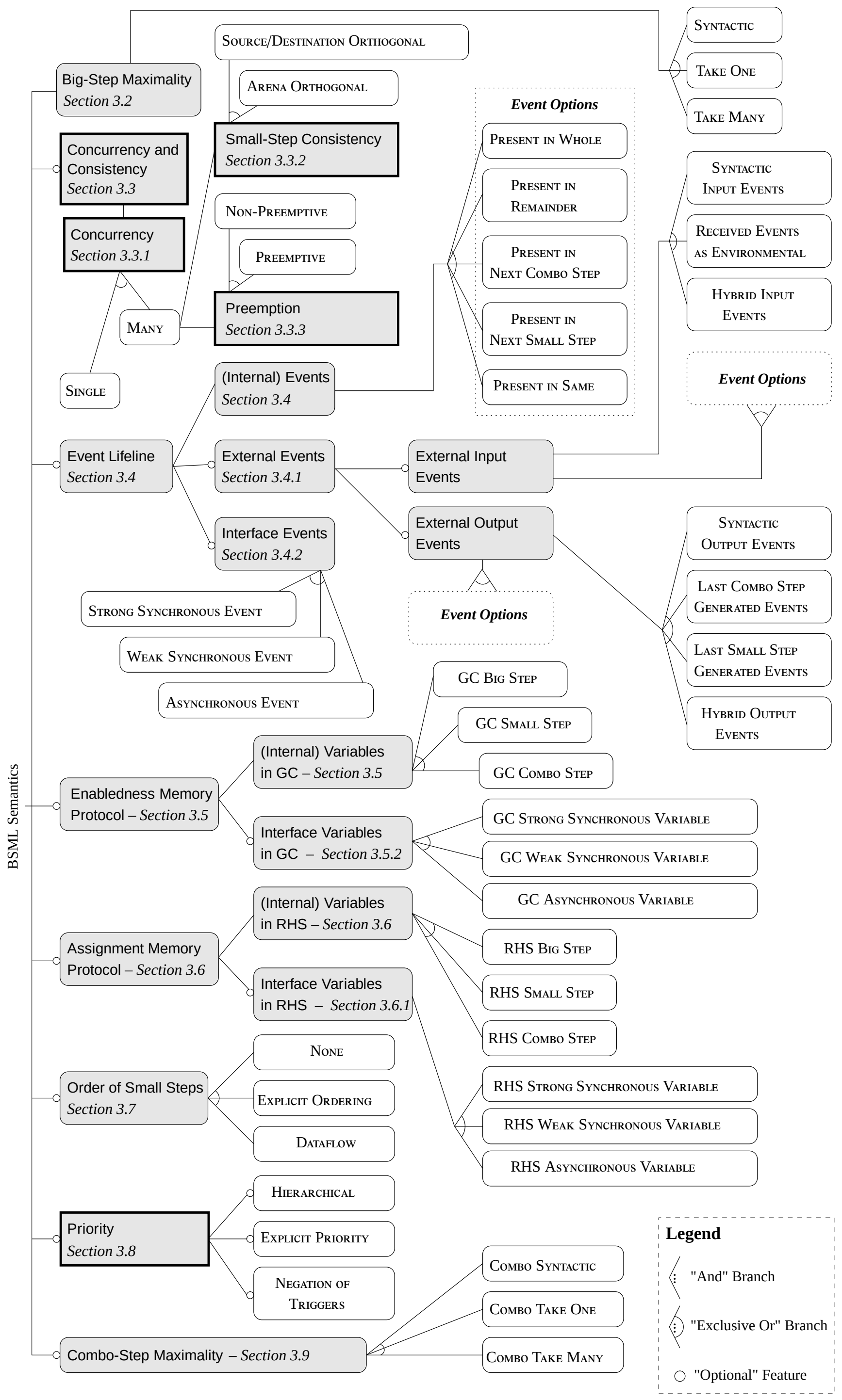
Next, the role of each stage in Figure 3.1, i.e., each semantic aspect, is described briefly. I use the Sans Serif font to distinguish the name of semantic aspects from normal text. The Big-Step Maximality semantic aspect specifies when a big step ends, at which point a new big step starts by sensing new environmental inputs. The Combo-Step Maximality semantic aspect specifies when

to the semantic differences between two existing BSMLs and use the latter pair of terms to refer to the semantic possibilities when designing a BSML. The distinction between semantic variation points and semantic decision points is similar to the distinction between the notion of “variation points” and the notion of “variable features” in generative programming [21].

a combo step ends, at which point a new combo step starts by adjusting the values of variables and/or the statuses of events, based on the details of a combo-step semantics, to reflect the effect of the execution of the small steps of the combo step. The Event Lifeline semantic aspect specifies how far within a big step a generated event can be sensed as present to trigger a transition. Separate sub-aspects are considered for the semantics of *internal events*, which are not meant to be observed by the environment of a model, for *external events*, which are used to communicate with the environment, and for *interface events*, which are used to specify communications among the different disjoint components of a model. The Enabledness Memory Protocol semantic aspect specifies the snapshot from which the values of variables are read to enable the guard condition of a transition. Similar to events, *internal variables* and *interface variables*, and their semantics, are distinguished. The Order of Small Steps semantic aspect describes options for the order of transitions that execute within a big step. From the set of transitions enabled by events, variables, and ordering constraints, the Concurrency and Consistency semantic aspect determines the set of potential small steps: first, it specifies whether more than one transition can be taken in a small step; and second, if more than one transition can be taken, it specifies the consistency criteria for including multiple transitions in a small step. The Priority semantic aspect chooses a small step from the set of potential small steps. The Assignment Memory Protocol semantic aspect specifies the snapshot from which the value of a variable in the right-hand side of an assignment is read.

The feature diagram in Figure 3.2 shows the eight semantic aspects for BSMLs together with their corresponding semantic options. Semantic aspects are represented by shaded boxes and the Sans Serif font, and semantic options are represented by clear boxes and the Small Cap font. Each semantic aspect is labelled with the section in this chapter that describes it. An arced branch in the diagram represents an "exclusive or": If a feature is chosen, then exactly one of its sub-features is chosen. For example, if the Big-Step Maximality semantic aspect is chosen, then exactly one of its options, Syntactic, Take One, or Take Many should be chosen. To achieve a concise diagram, a set of recurring semantic options for event-related semantic sub-aspects are grouped together as "Event Options", which is referenced via this label in the diagram.

I partition the BSML semantic aspects into two categories: The *enabledness* semantic aspects and the *structural* semantic aspects. The enabledness semantic aspects deal with the semantics of how a single transition can be included in a big step and what is the effect of its execution. The structural semantic aspects deal with how a set of enabled transitions can be taken together in a small step. In the feature diagram in Figure 3.2, enabledness semantic aspects and structural



BSML Semantics
Big-Step Maximality Section 3.2
Syntactic
Take One
Take Many
Source/Destination Orthogonal
Arena Orthogonal
Concurrency and Consistency Section 3.3
Small-Step Consistency Section 3.3.2
Concurrency Section 3.3.1
Non-Preemptive
Preemptive
Preemption Section 3.3.3
Many
Single
Event Options
Present in Whole
Present in Remainder
Present in Next Combo Step
Present in Next Small Step
Present in Same
Syntactic Input Events
Received Events as Environmental
Hybrid Input Events
Event Options
(Internal) Events Section 3.4
Event Lifeline Section 3.4
External Events Section 3.4.1
External Input Events
External Output Events
Interface Events Section 3.4.2
Event Options
Syntactic Output Events
Last Combo Step Generated Events
Last Small Step Generated Events
Hybrid Output Events
Strong Synchronous Event
Weak Synchronous Event
Asynchronous Event
GC Big Step
GC Small Step
GC Combo Step
(Internal) Variables in GC – Section 3.5
Enabledness Memory Protocol – Section 3.5
Interface Variables in GC – Section 3.5.2
GC Strong Synchronous Variable
GC Weak Synchronous Variable
GC Asynchronous Variable
(Internal) Variables in RHS – Section 3.6
Assignment Memory Protocol – Section 3.6
Interface Variables in RHS – Section 3.6.1
RHS Big Step
RHS Small Step
RHS Combo Step
RHS Strong Synchronous Variable
RHS Weak Synchronous Variable
RHS Asynchronous Variable
None
Order of Small Steps Section 3.7
Explicit Ordering
Dataflow
Hierarchical
Priority Section 3.8
Explicit Priority
Negation of Triggers
Combo Syntactic
Combo Take One
Combo Take Many
Combo-Step Maximality – Section 3.9
Legend
"And" Branch
"Exclusive Or" Branch
"Optional" Feature

Figure 3.2: The feature diagram representing the BSML semantic deconstruction. The shaded and clear boxes are semantic aspects and semantic options, respectively. The rounded, shaded boxes and the solid, shaded boxes are the enabledness and the structural semantic aspects, respectively.

semantic aspects are distinguished by rounded and solid boxes, respectively. As Chapter 4 will describe, different specification methods are used for formalizing the semantics of enabledness and structural semantic aspects.

A BSML semantics must subscribe to a Big-step Maximality semantics, as shown by the corresponding mandatory feature in the diagram in Figure 3.2. The other aspects are optional and depend on the syntactic features included in the BSML. A BSML semantics might have more than one priority semantic option, which together constitute its priority semantics (cf., Section 3.8).

A semantic aspect or a semantic option might be relevant for the semantics of a BSML only if a certain syntactic construct is allowed in the BSML. Figure 3.3 enumerates the dependencies between the syntactic and semantic features. To describe these dependencies, the names of syntactic features in Figure 2.4 and the names of semantic aspects and semantic options in Figure 3.2 are used as propositions, which indicate the choice of the feature in the corresponding feature diagram. The standard logical operators describe these dependencies. The "$p \Rightarrow q$" operator is logical implication: if p is true then q must be true. The "$p \Leftrightarrow q$" operator is logical equivalence: either p and q are both true, or both are false. The "$p \vee q$" operator is logical or: either p, q, or both are true. The "$p \wedge q$" operator is logical and: both p and q are true. For example, the first dependency asserts that if the syntax for events is used in a BSML, i.e., the "**Events**" is true, there must exist an event lifeline semantic option for it in the BSML, i.e., the "Event Lifeline" is true, and vice versa. As such, some of the semantic aspects are relevant only for the BSMLs whose syntax support certain syntax.

The last three dependencies in Figure 3.3 are between semantic features, as opposed to between syntactic and semantic features. These dependencies will be explained in the sections on the semantic aspects.

In the feature diagram in Figure 3.2, a semantic (sub-)aspect, or its parent, is labelled with the section in which it is described. The order of these sections is intended to minimize the required forward referencing to other semantics aspects (although some forward referencing cannot be avoided). In the following sections, for each semantic aspect, its semantic options are summarized in a table that includes a brief description of each semantic option, a list of its characteristics, and a list of representative BSMLs for each option. Each characteristic of a semantic option is identified as a relative advantage or disadvantage, signified by a "+" or "-", respectively, which is determined based on the conventional wisdom on this characteristic. Such wisdom may not

1. **Events** ⇔ Event Lifeline
2. (**Interface Events** ⇔ **Generated Interface Events**) ∧
   ((**Interface Events** ∧ **Generated Interface Events**) ⇔ Interface Events)
3. **Environmental Input Events** ⇔ Syntactic Input Events
4. **Environmental Output Events** ⇔ Syntactic Output Events
5. (**Negated Events** ∨ **Negated Interface Events** ∨ **Negated External Events**) ⇔
   Negation of Triggers
6. **Variable Conditions** ⇔ Enabledness Memory Protocol
7. **Variable Assignments** ⇔ Assignment Memory Protocol
8. **Interface Variables in GC** ⇔ Interface Variables in GC
9. **new** ⇔ Dataflow
10. **new** ⇒ (GC Big Step ∨ GC Small Step ∨ RHS Big Step ∨ RHS Small Step)
11. **new_small** ⇒ (GC Small Step ∨ RHS Small Step)
12. **cur** ⇒ (GC Big Step ∨ RHS Big Step)
13. **pre** ⇒ (GC Small Step ∨ RHS Small Step)
14. **Interface Variables in RHS** ⇔ Interface Variables in RHS
15. Hierarchical ⇒ **Hierarchical**
16. **And** ⇔ Concurrency and Consistency
17. **Stable** ⇔ Syntactic
18. **Combo Stable** ⇔ Combo Syntactic
19. Combo-Step Maximality ⇔
    (Present in Next Combo Step ∨ GC Combo Step ∨ RHS Combo Step)
20. (Combo Syntactic ∨ Combo Take Many)⇒ (Syntactic ∨ Take Many)
21. Present in Same ⇒ Many

Figure 3.3: Dependencies between syntactic features (in Figure 2.4) and semantic features (in Figure 3.2). (**Bold:** syntactic features, Sans Serif: semantic aspects, and Small Cap: semantic options.)

always be appropriate for a model depending on the domain of the system under study, the preference of the modeller, etc. These options cover the variations found in most existing BSMLs. I have also introduced a few semantic options that do not have a witness in existing languages; these semantic options are introduced to provide a systematic coverage of the range of possible semantic options for a particular semantic aspect. As in Figure 3.2, the SMALL CAP font is used to express the names of semantic options. Throughout the chapter, many examples are presented that are meant to demonstrate the differences between semantic options (but not to endorse one over another). The model snippets in the examples are not complete. Finally in Section 3.11, a table is presented that summarizes the semantic options chosen by a number of BSMLs.

3.2 Big-Step Maximality

The big-step maximality semantics of a BSML specifies when the sequence of small steps of a big step concludes. Table 3.1 lists the three possible semantic options. In the SYNTACTIC option, a BSML allows a modeller to designate syntactically a basic control state of a model as a *stable* control state. During a big step, once a transition *t* that enters a stable control state is executed, no other transition whose arena overlaps with the arena of *t* can be executed. In the TAKE ONE option, once a transition *t* is executed during a big step, no other transition whose arena overlaps with the arena of *t* can be executed. As such, each *Or* control state can contribute a maximum of one transition to a big step. Lastly, the TAKE MANY option allows a sequence of small steps to continue until there are no more enabled transitions to be executed.

Scope of a big step: In the TAKE ONE and the TAKE MANY options, the destination snapshot of a big step is not obvious, which can be complicated for a modeller. In the SYNTACTIC option, the end of a big step can be traced syntactically, which can be helpful for constructing and understanding a model.

Sequential transitions vs. non-terminating big steps: In the SYNTACTIC and TAKE MANY options, it is possible to specify a computation as a big step that consists of multiple sequential transitions within an *Or* control state. But, in these two semantics, it is also possible for a big step to never terminate because the execution of the big step never reaches a snapshot in which there are no more transitions to be executed. In the SYNTACTIC maximality semantics, additionally,

Table 3.1: Big-step maximality semantic options.

Options	Definition	Characteristics	Examples
SYNTACTIC	No two transitions with overlapping arenas that enter designated "stable" control states can be taken in the same big step.	(+) Syntactic scope for big steps (+) Sequential *Or* transitions (-) Non-terminating big steps	`pause` command in Esterel [14], "run to completion" in Rhapsody [42] and UML StateMachines [78]
TAKE ONE	No two transitions with overlapping arenas can be taken in the same big step.	(+) Terminating big steps (-) Unclear, non-syntactic scope for big steps	statecharts [41, 45, 86], reactive modules [3], and Argos [68]
TAKE MANY	Small steps continue until there are no more enabled transitions.	(+) Sequential *Or* transitions (-) Unclear, non-syntactic scope for big steps (-) Non-terminating big steps	Statemate [43] and RSML [63]

a big step may never terminate because the model never reaches a syntactically designated stable control state. Some BSMLs with the SYNTACTIC semantics require the non-stable control states of a model to have "else" transitions so that a big step can always reach a stable configuration (e.g., [42, 78]). Otherwise, a big step may halt because no transition is enabled to be executed although the big step is not maximal yet. In the TAKE ONE semantics, a sequence of transitions in an *Or* control state cannot be included in a big step, but a big step always terminates.

Stable control states can be used to model the semantics of the `pause` command in Esterel [14, 93]. During a big step, once all non-overlapping control states of the model's configuration have executed the `pause` command, the big step ends. As such, if the `pause` command is executed outside of a parallel command, then the big step terminates. But if the `pause`

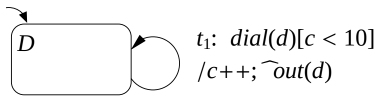


Figure 3.4: Dialer system.

command is executed inside a branch of a parallel command, then the big step terminates when every branch of the parallel command has executed the `pause` command. Stable control states can also be used to model the semantics of "compound transitions" in Rhapsody [42] and UML StateMachines [78]: The "pseudo states" of a model are modelled as non-stable control states, and "states" are modelled as stable control states. Some of the BSMLs that support the TAKE ONE semantics, such as reactive modules [3] and Argos [68], are influenced by the principles of synchronous hardware, which assumes that, during a big step, a non-concurrent part of a model can take only one transition (equivalently, each hardware component reacts once during a clock tick). The TAKE MANY semantic option is usually used by the BSMLs that support the notion of combo step (e.g., Statemate [43] and RSML [63]). The Statemate tool suite can be configured to use either the TAKE ONE semantic option, whose big steps are referred to as "steps", or the TAKE MANY semantic option together with combo steps, whose big steps are referred to as "super steps" [43].

Example 1 *The model in Figure 3.4 collects a dialed digit of a phone device (environmental input event* dial(d)*) and transmits the dialed digit* d *to the IP network via generated event* out(d).[2] *Variable* c *allows a maximum of 10 digits to be collected, at which point the central IP system would connect the caller to the dialed callee (the connection functionality of the system is not described). The "++" operator denotes increment by one.*

Let us consider a BSML semantics in which if an environmental input event is received at the beginning of a big step, it persists until the end of the big step. Also, let us consider the source snapshot where event dial(d) *is received from the environment and* c *is zero. If the* TAKE MANY *big-step semantics is chosen, then transition* t_1 *is executed 10 times in succession, sending the same digit,* d*, 10 times. If the* TAKE ONE *big-step maximality semantics is chosen, or the* SYNTACTIC *semantics is chosen and control state* D *is designated as stable, then the model behaves correctly.*

[2]Throughout the dissertation, when mentioning a syntactic element of a model in an example, whose body is in the Italic font, I use the normal font to highlight the syntactic element from the rest of the text.

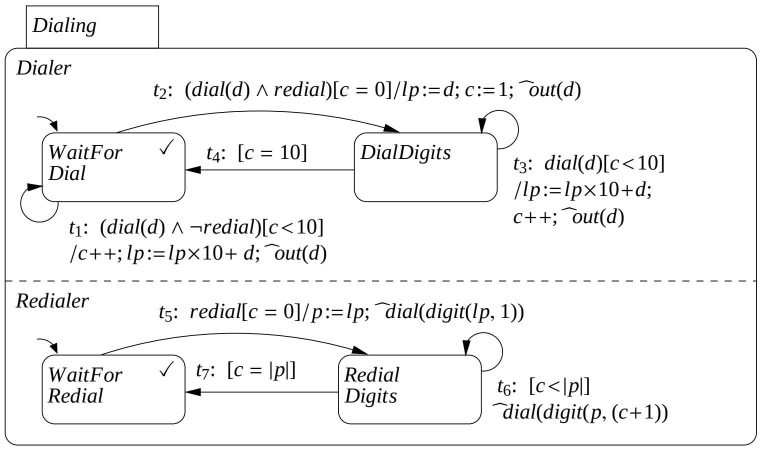

Dialing
Dialer
t_2: $(dial(d) \wedge redial)[c = 0]/lp := d; c := 1; \frown out(d)$
WaitFor Dial
✓
t_4: $[c = 10]$
DialDigits
t_3: $dial(d)[c < 10]$ $/lp := lp \times 10 + d;$ $c{+}{+}; \frown out(d)$
t_1: $(dial(d) \wedge \neg redial)[c < 10]$ $/c{+}{+}; lp := lp \times 10 + d; \frown out(d)$
Redialer
t_5: $redial[c = 0]/p := lp; \frown dial(digit(lp, 1))$
WaitFor Redial
✓
t_7: $[c = |p|]$
Redial Digits
t_6: $[c < |p|]$ $\frown dial(digit(p, (c+1))$

Figure 3.5: A model for dialing and redialing, copied from page 17.

Example 2 *The model in Figure 3.5, which is the same as the model in Figure 2.1, on page 17, copied here for convenience, is an extension of the simple dialer in Figure 3.4 to support redial functionality. The model uses the* SYNTACTIC *semantic option for big-step maximality. Control states* WaitForDial *and* WaitForRedial*, each signified by a "✓", are stable control states. For example,* WaitForRedial *is used to terminate a big step after the model receives a* redial *input in control state* WaitForRedial *and dials all the digits of the last dialed number. Once all the digits of the last-dialed phone number are redialled, control state* WaitForRedial *is entered again via transition* t_7.

Example 3 *The model in Figure 3.6 is for a two-bit counter.*[3] *Control states* Bit_1 *and* Bit_2 *model the least and most significant bits of the counter, respectively. Each time the environmental input event* tk_0*, which represents a clock tick, is received, the counter increments by one. Let us consider a semantics where a received environmental input event persists throughout the big step. After an even number of ticks,* Bit_1 *sends event* tk_1*, thereby instructs* Bit_2 *to toggle its status. After counting four clock ticks, the* Counter *generates the* done *event. Consider the snapshot where the model resides in control states* Bit_{11} *and* Bit_{21} *and a semantics where each small step comprises the execution of exactly one transition. If the* TAKE ONE *big-step semantics is chosen, then the model behaves correctly. The first* tk_0 *input event produces the big step* $\langle\{t_1\}\rangle$*, the second* tk_0 *input event produces the big step* $\langle\{t_2\}, \{t_3\}\rangle$*, the third* tk_0 *input event again produces the big step*

[3]This example is adopted from [68], where a more elaborate version of it is used as the running example of the paper.

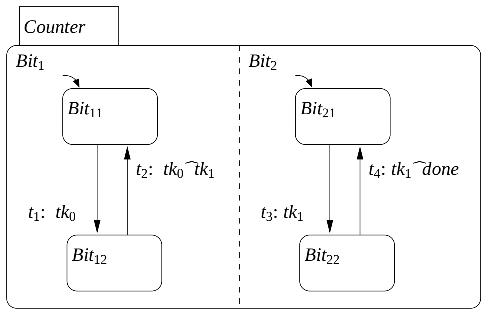


Figure 3.6: A two-bit counter.

⟨{t$_1$}⟩, *and lastly, the fourth* tk$_0$ *input event produces the big step* ⟨{t$_2$}, {t$_4$}⟩, *which generates event* done. *If the* Take Many *big-step semantics is chosen, then the model behaves incorrectly by creating non-terminating big steps; for example, upon receiving the first* tk$_0$ *input event, the model can engage in the following non-terminating big step:* ⟨{t$_1$}, {t$_2$}, {t$_1$}, {t$_2$}, $\cdots$⟩.

3.3 Concurrency and Consistency

BSMLs vary in how the enabled transitions of a model execute together in a small step. In the examples in the previous section, each small step has exactly one transition, but there are other options. Table 3.2 lists the three concurrency and consistency semantic sub-aspects that specify: (i) concurrency: whether more than one transition can be taken in a small step, and if so, (ii) small-step consistency: which transitions can be taken together, considering the composition tree of a model, and (iii) preemption: whether the execution of one transition in a small step can *preempt* the execution of another transition or not.

3.3.1 Concurrency

There is a dichotomy in hardware and software about how to model the execution of a system: *single-transition* vs. *many-transition* [71, 88, 90, 97]. Similarly, in BSMLs, there are two options: (i) a small step can execute only one transition in a small step (the Single option), and (ii) all enabled transitions that can be taken together are taken in a small step (the Many option).

Table 3.2: Concurrency and consistency semantic options.

Options	Definition	Characteristics	Examples
Concurrency			
SINGLE	A small step consists of the execution of exactly one transition.	(+) Simplicity (-) Non-determinism	statecharts [41, 45, 86], Stateflow [22], and reactive modules [3]
MANY	A small step may consist of the execution of more than one transition.	(+) Low chance for non-determinism (-) Race conditions	Argos [68] and Esterel [14]
Small-Step Consistency			
ARENA ORTHOGONAL	The arenas of two distinct transitions of a small step are orthogonal.	(+) Simplicity (-) High chance for non-determinism	Argos [68] and Esterel [14]
SOURCE/ DESTINATION ORTHOGONAL	The source control states and destination control states of two distinct transitions of a small step are pairwise orthogonal.	(+) Low chance for non-determinism (-) Complex	N/A
Preemption			
NON-PREEMPTIVE	Two transitions that one is an "interrupt for" another can be taken in a small step.	(+) Support for "last wishes" (-) Counterintuitive flow of control	Argos [68], and semantics of `exit` and `trap` statements in Esterel [14]
PREEMPTIVE	Two transitions that one is an "interrupt for" another cannot be taken in a small step.	(+) Simple flow of control (-) No support for "last wishes"	N/A

The SINGLE option is simple because it does not have to deal with the complexities of executing multiple transitions (e.g., race conditions), but it can cause undesired non-determinism because two enabled transitions can execute in different orders.

Race conditions: A model has a *race condition* when more than one transition in a small step assign values to a variable. Typically, one of the assignments is chosen non-deterministically [75], but there are other options [34].

Example 4 *Figure 3.7 shows the model for describing the behaviour of a simple traffic light system at an intersection.*[4] *The model consists of And control state* TrafficLight, *which itself consists of two Or control states: the* NS *control state controls the traffic in the north-south direction and the* EW *control state controls the traffic in the east-west direction. It is assumed that the environment provides the sequence of environmental input events:* end, change, end, change, $\cdots$, *in a timely manner according to the schedule of the traffic light. Environmental input event* end *designates the end of green light for a direction by changing its green lights to yellow. Environmental input event* change *changes the direction of traffic by switching the red lights to green lights, and the yellow lights to red lights. The system is initialized so that the lights for north-south direction are green, and the lights for east-west direction are red. Consider the snapshot where the model resides in control states* EW_Red *and* NS_Yellow, *and environmental input event* change *is received. If the* TAKE ONE *big-step maximality semantics together with the* SINGLE *concurrency semantics are chosen, then the model can choose to execute the big step consisting of the sequence of transitions* $\langle\{t_2\}, \{t_4\}\rangle$, *or the sequence of transitions* $\langle\{t_4\}, \{t_2\}\rangle$, *non-deterministically. However, executing the latter sequence of transitions allows the model to arrive at snapshot* EW_Green *and* NS_Yellow, *which is not a desirable behaviour. If the* MANY *concurrency semantics is chosen, then the model executes big step* $\langle\{t_2, t_4\}\rangle$, *arriving at control states* EW_Green *and* NS_Red.

Next, two semantic sub-aspects are considered that specify how the set of transitions can be combined to be taken together in a small step, when the MANY semantics is chosen. The *small-step consistency* sub-aspect deals with transitions that do not preempt each other. The *preemption* sub-aspect deals with transitions that do preempt each other. The two sub-aspects deal with disjoint sets of transitions of a model.

[4]This example is adopted from [51].

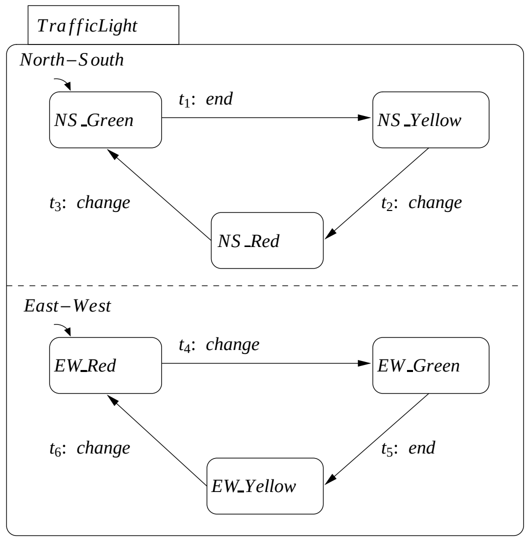


Figure 3.7: Traffic light system.

3.3.2 Small-Step Consistency

For two enabled transitions that neither is an interrupt for the other, this semantic sub-aspect specifies whether they can be taken together in a small step. In the Source/Destination Orthogonal semantic option, two transitions that are orthogonal (i.e., whose source control states and destination control states are pairwise orthogonal) can be taken together in a small step. The Arena Orthogonal option is more restrictive in that two transitions can be included in the same small step only if their arenas are orthogonal (where the arena of a transition is the lowest *Or* control state in the hierarchy of the composition tree that is a common ancestor of the source and destination control states of the transition). In comparison, the Arena Orthogonal option is simpler than the Source/Destination Orthogonal option, but it can introduce undesired non-determinism by not taking all of the enabled transitions that the Source/Destination Orthogonal option takes. The Arena Orthogonal semantic option and the Take One big-step maximality semantics are similar: The former semantic option disallows two transitions whose arenas are the same or ancestrally related to be included in a small step, while the latter disallows the two transitions to be included in a big step.

Example 5 *The model in Figure 3.8 is similar to the model in Figure 3.6, on page 39, but has the extra Or control state* Status *that specifies whether the counter is in the process of counting,*

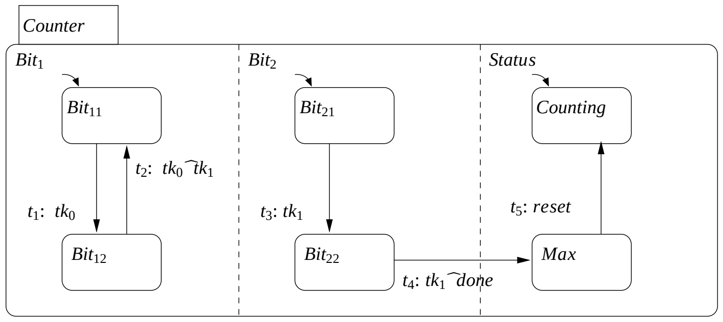


Figure 3.8: The revised two-bit counter.

or it has already counted four ticks and should be reset. Consider the snapshot where the model resides in control states, Bit_{12}, Bit_{22}*, and* Counting*, and the fourth* tk_0 *event is received. Let us choose the* MANY *concurrency semantics together with the* PRESENT IN SAME *event communication mechanism (explained in Section 3.4), in which a generated event can enable a transition in the same small step. If the* ARENA ORTHOGONAL *semantics is chosen, then only* $\{\mathrm{t}_2\}$ *can be taken, but not together with* t_4*, because the arena of* t_4 *is a parent of the arena of* t_2*. If the* SOURCE/DESTINATION ORTHOGONAL *semantics is chosen, then* $\langle\{\mathrm{t}_2, \mathrm{t}_4\}\rangle$ *can be taken, and the model behaves correctly. (As described in detail in Section 4.3, the execution of* t_4 *involves exiting the Or control state* Bit_2 *and reentering its default control state* Bit_{21}*. The destination configuration of the small step is* Bit_{11}, Bit_{21}*, and* Max*.)*

3.3.3 Preemption

The notion of *preemption* [11] is relevant for a pair of transitions when one is an *interrupt for* the other, as described in Section 2.1. Recall that a transition t is an interrupt for transition t' when the sources of the transitions are orthogonal and one of the following conditions holds: (i) the destination of t' is orthogonal with the source of t, and the destination of t is not orthogonal with the sources of either transitions (Figure 3.9(a)); or (ii) the destination of neither transition is orthogonal with the sources of the two transitions, but the destination of t is a descendant of the destination of t' (Figure 3.9(b)). The NON-PREEMPTIVE option allows such a t and t' to be executed together in the same small step, whereas the PREEMPTIVE option does not. In the NON-PREEMPTIVE option, the effect of executing such a small step $\{t, t'\}$ includes the variable assignments and event generations of both transitions, but the destination configuration of the small step is determined

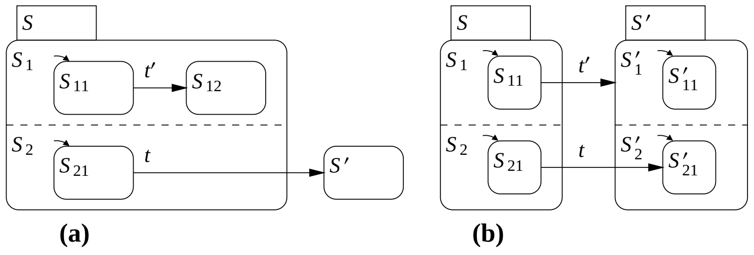


Figure 3.9: Interrupting transitions.

as if only t has been executed (i.e., the destination of t' is not relevant). As such, executing $\{t, t'\}$ in Figure 3.9(a) moves the model to control state S', and executing $\{t, t'\}$ in Figure 3.9(b) moves the model to control states S'_{11} and S'_{21}. While complex, due to its counterintuitive flow of control, the Non-Preemptive option satisfies the "last wishes" of the children of an *And* control state that is interrupted.

The Non-Preemptive semantics can be used to model the "weak preemption" semantics of `exit` and `trap` statements in Esterel [14, 40]. The concurrent execution of an `exit` command with a non-`exit` command complies with the condition (i) above of the interrupt for relation. The concurrent execution of two `exit` commands complies with the condition (ii) above of the interrupt for relation. In Argos [68], a different notion of hierarchical control state than ours is used. A transition whose source control state is a non-*Basic* control state S is an interrupt for a transition whose arena is S or a descendent of S. This notion of control state and interrupt can be translated into the normal-form syntax described here, by turning S into an *And* control state with two children: One representing S without the interrupt transition, and another having only one transition that models the interrupt transition. In Esterel [14, 40], in addition to the Non-Preemptive semantics, there is a syntax to specify Preemptive behaviour through the "strong preemption" semantics of `watching` statements. In a "`do <statements> watching(e)`" statement, the execution of "`<statements>`" is immediately aborted when event e occurs, without satisfying the "last wish" of "`<statements>`". Such a `watching` statement can be translated into the normal-form syntax here by creating a transition, t, whose source is an *And* or *Or* control state that represents the "`<statements>`", and it is triggered with event e. Transition t in the aforementioned translation is not an interrupt for any transition, but needs to be assigned a higher priority than the transitions in its source.

Example 6 *The model in Figure 3.10 is an extension of the model in Figure 3.5. This model*

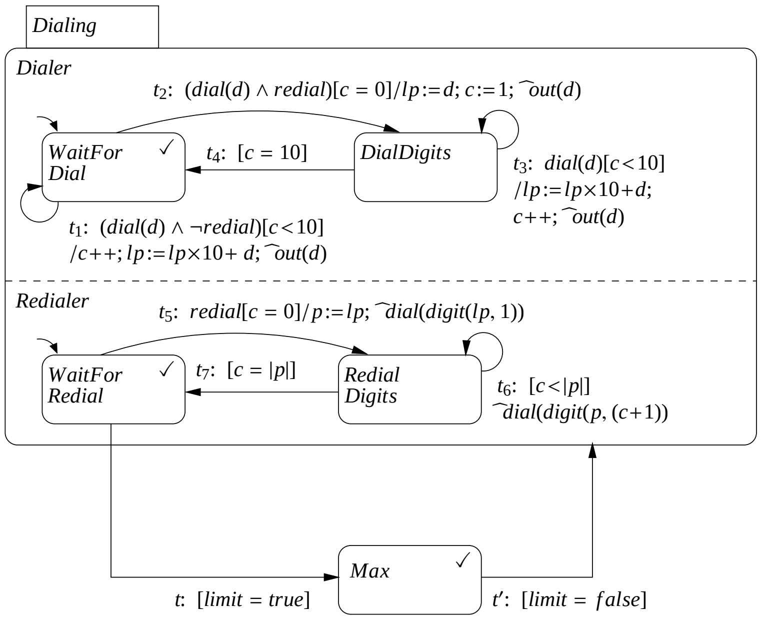

Dialing
Dialer
t_2: $(dial(d) \wedge redial)[c = 0]/lp := d; c := 1; \frown out(d)$
WaitFor Dial ✓
t_4: $[c = 10]$
DialDigits
t_3: $dial(d)[c < 10]$ $/lp := lp \times 10 + d;$ $c{+}{+}; \frown out(d)$
t_1: $(dial(d) \wedge \neg redial)[c < 10]$ $/c{+}{+}; lp := lp \times 10 + d; \frown out(d)$
Redialer
t_5: $redial[c = 0]/p := lp; \frown dial(digit(lp, 1))$
WaitFor Redial ✓
t_7: $[c = |p|]$
Redial Digits
t_6: $[c < |p|]$ $\frown dial(digit(p, (c+1))$
Max ✓
t: $[limit = true]$
t': $[limit = false]$

Figure 3.10: Interrupting transitions.

is a model of a dialer system that receives the dialed digits of a phone, through event dial(d), *and transmits these digits via output events* out(d), *to establish the connection with a destination phone number. Compared to the model in Figure 3.5, the model in Figure 3.10 additionally controls the total number of calls that can be established at each point of time. If the maximum number of concurrent calls is reached, which is determined by the boolean environmental input variable* limit, *the dialing process is aborted via transition* t. *Let us consider the snapshot where environmental input variable* limit *is true, the model resides in control states* WaitforDial *and* WaitforRedial, *the value of variable* c, *which is the number of dialed digits so far, is nine, and the environmental input* dial(d) *is received, i.e., the caller dials the last digit of a phone number. Let us choose the* SYNTACTIC *big-step maximality semantics and the* MANY *concurrency semantics. If the* PREEMPTIVE *semantic option is chosen, the system may abort the dialing process by executing* $\langle\{\mathrm{t}\}\rangle$, *and not* $\langle\{\mathrm{t}_1\}\rangle$. *But if the* NON-PREEMPTIVE *semantic option is chosen, then the call would go through by executing* $\langle\{\mathrm{t}_1, \mathrm{t}\}\rangle$, *arriving at the destination configuration* {Max}.

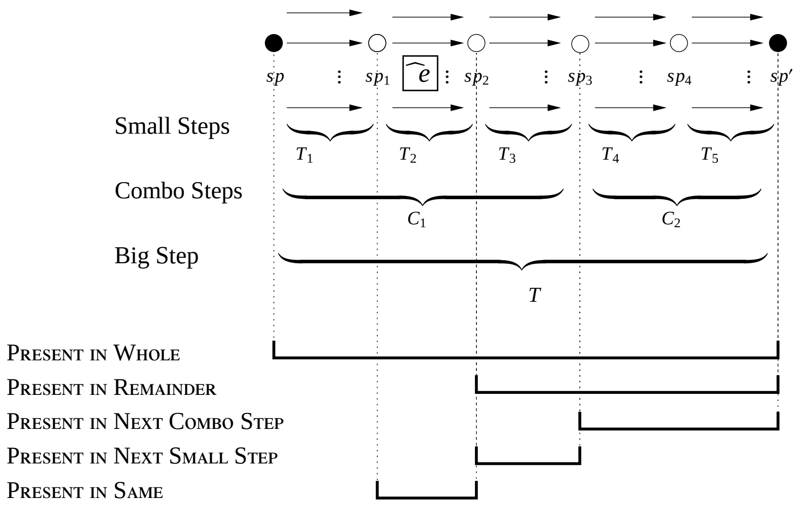


Figure 3.11: The event lifeline of the generated event *e* according to different event lifeline semantic options.

3.4 Event Lifeline

A generated event of a transition is broadcast to all parts of a model. An event's *status*, which is either *present* or *absent*, can be sensed by the event trigger of a transition. The *event lifeline* semantics of a BSML specifies the snapshots of a big step in which a generated event can be sensed as present. In this dissertation, the maximum event lifeline of an internal event is the big step in which it is generated. Interface events, describe in Section 3.4.2, provide a semantic option for a lifeline beyond the same big step in which an event is generated. Table 3.3 shows the five event lifeline semantics: (i) in the Present in Whole option, a generated event is present throughout its big step, from the beginning of its big step; (ii) in the Present in Remainder option, a generated event is present in the snapshot after it is generated and persists until the end of its big step; (iii) in the Present in Next Combo Step option, a generated event is present only during the next combo step; (iv) in the Present in Next Small Step option, a generated event is present only in the next snapshot; and (v) in the Present in Same option, a generated event is present only during the small step in which it is generated (instantaneous communication). Figure 3.11 depicts the event lifeline of the event *e* generated in small step T_2, according to the different event lifeline semantics. The name of an event lifeline semantics is followed by a line that depicts the extent of the big step in which *e* is present, according to that semantics.

Table 3.3: Event lifeline semantics.

Options	Definition	Characteristics	Examples
PRESENT IN WHOLE	A generated event in a big step is assumed to be present throughout the same big step.	(+) Modularity (+) Global consistency (-) Non-causality (-) Counterintuitive behaviour	Argos [68] and Esterel [14]
PRESENT IN REMAINDER	A generated event in a big step is sensed as present in the same big step after it is generated.	(+) Causality (-) Unorderedness (-) Global inconsistency	statecharts [45, 86]
PRESENT IN NEXT COMBO STEP	A generated event can be sensed as present only in the next combo step after it is generated.	(+) Causality (+) Partial orderedness (-) Multiple-instance events	Statemate [43] and RSML [63]
PRESENT IN NEXT SMALL STEP	A generated event can be sensed as present only in the next small step after it is generated.	(+) Causality (+) Orderedness (-) Multiple-instance events	statecharts[23]
PRESENT IN SAME	A generated event can be sensed as present only in the same small step it is generated in.	(+) Instantaneous communication (-) Non-causality (-) Multiple-instance events	statecharts [82] and used in [75]

The Present in Whole semantic option supports the "perfect synchrony hypothesis" [10, 68]. If a big step is considered as the reaction of a synchronous circuit during a "tick" of the clock, the semantics of the perfect synchrony hypothesis is similar to the signal rules of synchronous hardware. In synchronous hardware, a signal is either present or absent during a tick of a clock, but not both.

The Present in Same semantic option is different from the other semantic options in that the generated events of a small step cannot affect the enabledness of another small step, making the small steps of a big step independent of one another. In Chapter 6, this semantics is considered within the context of synchronization semantics.

Causality: A big step is *causal* if its small steps can be sequenced as: $T_1, T_2, \cdots, T_n$, such that any event that triggers a transition in small step T_i ($1 \leq i \leq n$) must be generated by some earlier small step in $T_1, T_2, \cdots, T_{i-1}$. To a modeller, the transitions of a non-causal big step may seem counterintuitive, and execute out of the blue. The Present in Whole and the Present in Same semantic options can create non-causal big steps. To avoid non-causal big steps, some BSMLs that use the Whole event lifeline semantics introduce a notion of a "correct" model, which never creates a non-causal big step [14, 16, 93]. Analysis tools can be used to detect "incorrect" models, conservatively, and reject them at compile time [16, 40]. But if a BSML supports variables, the detection of incorrect models is undecidable [40].

Orderedness: The Present in Remainder semantics lacks a "rigorous causal ordering" [63]: if event e_1 is generated earlier than event e_2, it need not be the case that transitions triggered by e_1 are executed earlier than transitions triggered by e_2. The Present in Next Combo Step semantics was devised to alleviate this problem by having a "rigorous causal ordering" between combo steps, while being insensitive to the order of event generation within a combo step [43, 63]. A disadvantage of the Present in Next Combo Step semantics is that a modeller needs to keep track of the scope of a combo step in order to consider its generated events all at once in the next combo step. The Present in Next Small Step semantics is ordered: a transition triggered by an internal event e can be executed only if e is generated by a transition in the previous small step.

Modularity: The Present in Whole option is "modular" [50] with respect to events: an event generated during a big step can be conceptually considered the same as an environmental input event because it is present from the beginning of the big step. All other event lifeline semantics are non-modular. In a non-modular event lifeline semantics, concurrent parts of a model cannot play the role of the environment for each other, because extensions of the model may change the behaviour in different ways than the environment does. As a result, a model cannot be constructed incrementally.

Multiple-instance events: An *instance* of an event in a big step is a contiguous segment of the snapshots of a big step where the event is present. Two distinct instances of an event correspond to two disjoint sets of small steps. In the Present in Next Combo Step, Present in Next Small Step, and Present in Same event lifeline semantics, multiple instances of the same event, generated by different small steps, may exist in the same big step. Thus, the status of an event can change multiple times in a big step, making it complicated for a modeller to determine whether an event is present in a certain snapshot of a big step, or not.

Global inconsistency: When negated events are included in the BSML syntax, the Present in Remainder semantic option can produce "globally inconsistent" big steps [85, 86]. A big step is globally inconsistent if it includes a transition that generates an event and a transition triggered by the absence of that event. A globally inconsistent big step is undesired because an event is sensed both as absent and present in the same big step. The Present in Remainder semantic option can achieve a variation of the original global consistency semantics [85, 86], by not taking a transition that generates an event that was sensed as absent earlier in the big step [66]. The global inconsistency problem is not relevant for other semantic options because the Present in Remainder semantic option is the only semantic option that allows maximum one instance of an event in a big step and yet allows the aforementioned inconsistency. The other lifeline semantics that allow multiple instances of an event in the same big step are globally inconsistent, but by design.

Global consistency vs. causality: Figure 3.12 shows the relationship between the big steps of the Present in Remainder semantics and the Present in Whole semantics. A big step T from a globally consistent Present in Remainder semantics also satisfies a Present in Whole semantics because T's generated events, by the definition of global consistency, are present from the beginning of the big step. Conversely, a big step T' from a causal Present in Whole semantics also satisfies a Present in Remainder semantics because, by the definition of causality, an event is

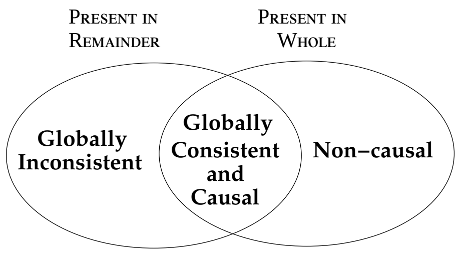


Figure 3.12: Global consistency vs. causality.

sensed as present by a transition of T' only if it is already generated in the big step. Therefore, if global consistency is guaranteed syntactically (e.g., there are no negated event triggers), then the set of big steps in the Present in Remainder semantics is a subset of the big steps of the Present in Whole semantics.

Events with parameters: An event can have a value parameter, as in Esterel [14].[5] For an event with a value parameter, the value of its parameter is determined per instance of the event. When an event instance is generated by more than one transition, the value of its parameter is determined by a "combine function" [14]. A combine function is a commutative, associative function, such as addition, that "combines" the different values of the parameter of an event that are generated by a set of transitions. In the Present in Remainder, Present in Next Combo Step, Present in Next Small Step, and Present in Same semantics, a combine function combines the values of the parameter of an event generated by transitions in the previous and current small steps, previous combo step, previous small step, and current small step, respectively. In the Present in Whole option, the value of the parameter of an event instance is fixed during a big step, and is determined by combining all of the values of the parameter of the event generated during the big step.

Implicit events: Some BSMLs use *implicit events* in their syntax, which represent events that are generated in response to a certain property of the computation of a model. For example, the implicit event `entered(`*s*`)` [85] is generated when control state *s* is entered, implicit event `@T(`*cond*`)` [46, 47] is generated when the value of boolean expression *cond* changes from false

[5]In Esterel [1], the value parameter of an event can be of type array, which means that, in effect, an event can have more than one value parameter, each of which being an element of a single array.

to true, and `assigned(v)` [85] is generated when variable v is assigned a value. Implicit events may or may not have the same semantics as the event lifeline semantics of named events.

Example 7 *In Example 3, on page 38, when considering the* TAKE ONE *big-step maximality semantics, the semantics that subscribes to the* PRESENT IN WHOLE, PRESENT IN REMAINDER, *or* PRESENT IN NEXT SMALL STEP *event lifeline semantics all yield the expected behaviour. If the* TAKE ONE *big-step maximality semantics, the* MANY *concurrency semantics, the* ARENA ORTHOGONAL *small-step consistency semantics, the* PREEMPTIVE *preemption semantics (or the* NON-PREEMPTIVE *preemption semantics) are chosen, then the* PRESENT IN SAME *semantics also yields the expected behaviour.*

Example 8 *In the model in Figure 3.5, on page 38, variable* lp *stores the last dialed phone number. Upon receiving the* redial *environmental input event,* Redialer *instructs* Dialer, *by generating the corresponding* dial *events, to dial the digits of* lp. *(The size of an integer,* x, *is denoted as* $|x|$, *and its* n*th digit as* $\mathrm{digit}(x, n)$*.) Variable* p *is necessary because once redialling starts* lp *is overwritten. Consider the snapshot where the environmental input event* redial *is received,* c *is zero, and* $|lp|$ *is 10. The environmental input event* redial *persists throughout the big step. A semantics that follows the* SYNTACTIC *big-step maximality semantics (annotating a stable control state with a "✓"), the* MANY *concurrency semantics, the* ARENA ORTHOGONAL *small-step consistency semantics, the* PREEMPTIVE *preemption semantics, the* PRESENT IN NEXT SMALL STEP *event lifeline semantics, and uses the up-to-date values of variables, can produce the big step* $\langle t_5, \{t_2, t_6\}, \{t_3, t_6\}, \cdots, \{t_3, t_6\}, \{t_4, t_7\}\rangle$, *which transmits the first digit twice and does not transmit the last digit. If the* PRESENT IN SAME *event lifeline semantics is chosen, the model produces the correct big step* $\langle\{t_5, t_2\}, \{t_3, t_6\}, \cdots, \{t_4, t_7\}\rangle$. *In both cases, if the size of the redialled number is less than 10, the model cannot stabilize, and remains in the* DialDigits *control state.*

Example 9 *The model in Figure 3.13 is a simple model of a cruise control system of a car. The system regulates the amount of power transmitted to the wheels of the car by adjusting the amount of gas that is provided to the engine, in order to maintain the speed specified by the cruise control system. If the cruise control system is on, de-acceleration does not have any effect on the amount of gas that is provided to the engine. But if the cruise control system is on and the acceleration event is received, then the cruise control system is turned off, and acceleration is processed as usual. The two Or control states of the And control state* FuelControl *process the cruise control and acceleration/de-acceleration functionalities, respectively. The environmental*

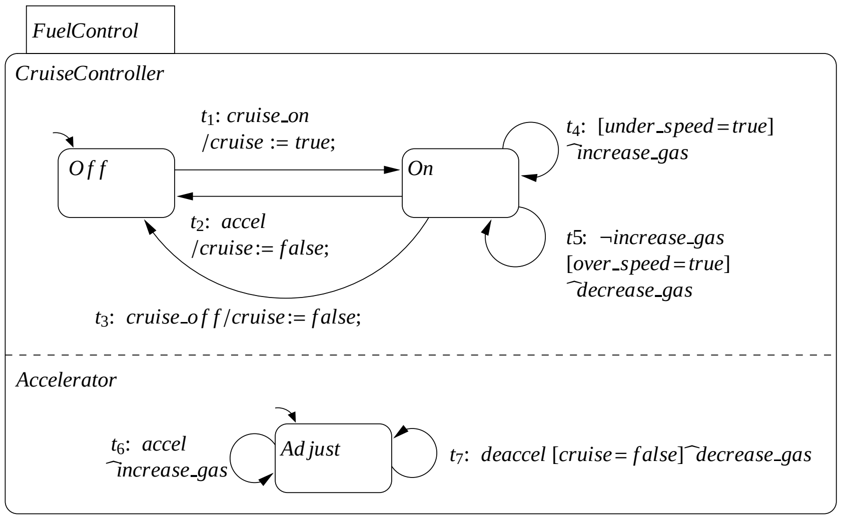

FuelControl
CruiseController
t_1: cruise_on /cruise := true;
t_2: accel /cruise:= false;
t_3: cruise_off/cruise:= false;
t_4: [under_speed=true] ⌢increase_gas
$t5$: ¬increase_gas [over_speed=true] ⌢decrease_gas
Off
On
Accelerator
Adjust
t_6: accel ⌢increase_gas
t_7: deaccel [cruise=false]⌢decrease_gas

Figure 3.13: Speed control system for a car.

input events cruise_on *and* cruise_off *turn the cruise control system on and off, respectively. The environmental input events* accel *and* deaccel *specify whether the accelerator is being pressed or de-pressed, respectively. The boolean environmental input variables* over_speed *and* under_speed *specify whether the vehicle is moving faster or slower, respectively, than the target speed set by the cruise control system. Events* increase_gas *and* decrease_gas *slightly increase and decrease the amount of fuel into the engine, respectively.*

Consider the moment when the cruise control system is on, the system is over speed, and the accelerator is pressed; i.e., when the system resides in control state On, over_speed = true, *and* accel *is received from the environment. Let us choose the* TAKE ONE *big-step maximality semantics and the* SINGLE *concurrency semantics. If the* PRESENT IN WHOLE *semantic option is chosen, then the only possible big step consists of* $\{t_6\}$ *and* $\{t_2\}$*, which results in the desired behaviour for the system. If the* PRESENT IN REMAINDER *semantic option is chosen, then additionally* $\langle\{t_5\}, \{t_6\}\rangle$ *is a valid big step, which both decreases and increases the amount of gas to the engine. The latter big step is globally inconsistent, because* increase_gas *is sensed as absent by* t_5 *and is generated by* t_6*. If the variation of global consistency semantics in [66] is chosen, then* $\langle\{t_5\}\rangle$ *is a valid big step;* t_6 *cannot be taken during the big step since it generates* increase_gas.

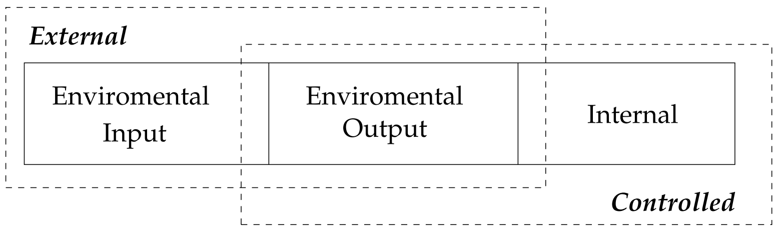


Figure 3.14: A taxonomy for events.

3.4.1 External Events

The model in Figure 3.5, on page 38, uses event *dial* in two different ways: (i) as an environmental input event initiated by a human caller, and (ii) as an internal event generated by the *Redialer*. To avoid modelling flaws, many have advocated that the interface of a system with its environment should be clearly and explicitly specified [79, 101]. A celebrated way to achieve this interface, as shown in Figure 3.14, is to distinguish between the events that the environment can control, *environmental input events*, and the events that are generated by the model, *controlled events*. A controlled event may be observable by the environment (i.e., an *environmental output event*), or not (i.e., an *internal event*). The environmental input and output events of a model together constitute the *external events* of the model.

A BSML may choose distinct event lifeline options for environmental input events, environmental output events, and internal events, as shown in the feature diagram of Figure 3.2. Often, the event lifeline semantics of the environmental input events is the X Present in Whole (or equivalently, the X Present in Remainder) semantics, meaning that an input event persists throughout a big step, and the event lifeline semantics of the environmental output events is the same as the event lifeline semantics of the internal events. The prefix "X" in the name of the semantic options, signifying e<u>x</u>ternal event, is used so that no two semantic options would have the same name.

A BSML may syntactically distinguish environmental input events and environmental output events from each other, and from internal events. Alternatively, a BSML is *non-distinguishing* if it does not distinguish syntactically between the external events and the internal events of a model. In these BSMLs, it is still possible to consider inputs received at the beginning of the big step as environmental inputs, and outputs generated in the last small step or last combo step of a big step as environmental outputs, each with distinct event lifeline choices. Table 3.4 lists the possible semantic options for differentiating environmental input events and internal events. In the Syntactic Input Events option, an environmental input event is syntactically distinguished. Thus a BSML that subscribes to this option is a "distinguishing" BSML. In the Received Events as Environmental option, an event that is received at the beginning of a big step is considered an environmental input event. In the Hybrid Input Events option, an event that is received at the beginning of a big step is considered an environmental input event only if it is a *genuine input* of a model, meaning it is not generated by any transitions in the model. While in the Syntactic Input Events and the Hybrid Input Events semantic options, the set of environmental input events of a model can be identified syntactically, in the Received Events as Environmental semantic option, the environmental input events of a model are determined per each big step.

As shown in Figure 3.2, an event lifeline semantics for the environmental input events can be chosen, regardless of the choice of the semantic option for distinguishing the input events. For example, if the semantics for environmental inputs is the Received Events as Environmental semantic option together with the X Present in Next Small Step semantic option, then an input event that is received at the beginning of a big step persists only for the first small step of the big step. Environmental output events have similar options; events generated in either the last small step or last combo step of a big step could be considered as environmental output events.

Example 10 *In Example 8, a non-distinguishing semantics was considered for the model in Figure 3.5, because event* dial *can be both received from the environment and generated, possibly in the same big step. Event* redial *is a genuine input. Both the* Received Events as Environmental *and* Hybrid Input Events *semantic options, together with the* X Present in Remainder *event lifeline semantics, yield a behaviour that matches the behaviour specified in Example 8.*

If the single-input assumption [46, 47] is assumed, which requires that dial *and* redial *are not both received from the environment in the same big step, then* dial *cannot be received from the environment at the beginning of a big step and generated in the same big step.*

3.4.2 Interface Events

Some BSMLs structure a model as a set of *components*, each of which is a compound control state. The components of a model communicate with each other through their *interface events* according to an *inter-component communication mechanism*. Figure 3.15 refines the taxonomy

Table 3.4: Differentiating environmental input events from internal events.

Options	Definition	Characteristics	Examples
SYNTACTIC INPUT EVENTS	Only syntactically distinguished events are treated as environmental inputs.	(+) Separates system from environment (-) Usually different semantics for different event types	Esterel [14]
RECEIVED EVENTS AS ENVIRONMENTAL	Any event that is received at the beginning of a big step is considered an environmental input event.	(+) Treats input and internal events uniformly (-) No boundary between system and environment	statecharts [86] and RSML [63]
HYBRID INPUT EVENTS	Only "genuine" inputs that are received from the environment at the beginning of a big step are treated as environmental inputs.	(+) Distinguishes between internal and genuine input events (-) Complex	N/A

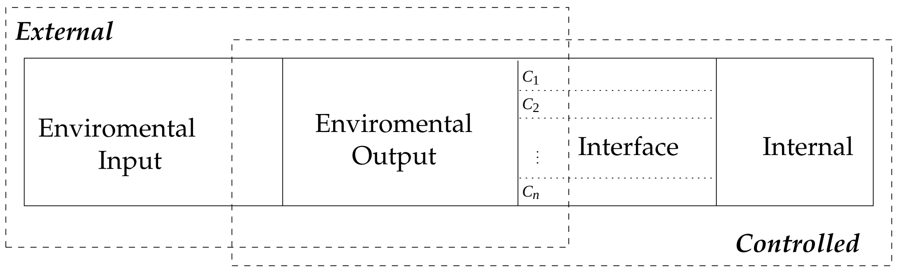


Figure 3.15: A taxonomy of events for inter-component communication.

of Figure 3.14 by including interface events as a subset of the controlled events of a model. Conventionally, an interface event is generated only by one component, called a *sending component*. A component that accesses an interface event is its *receiving component*. As such, the interface events of a model are partitioned into sets, shown by dashed lines in Figure 3.15, each of which is generated by one component.

Table 3.5 lists the three possible inter-component communication semantic options for interface events. In the STRONG SYNCHRONOUS EVENT option, a generated interface event is sensed as present throughout the big step in which it is generated, from the beginning of the big step (similar to the PRESENT IN WHOLE semantic option for internal events). In the WEAK SYNCHRONOUS EVENT option, a generated interface event is present in the big step in which it is generated, but only after it is generated (similar to the PRESENT IN REMAINDER semantic option for internal events). In the ASYNCHRONOUS EVENT option, a generated interface event is present in the next big step, from the beginning of the big step. The STRONG SYNCHRONOUS EVENT and the WEAK SYNCHRONOUS EVENT semantic options have similar advantages and disadvantages as the PRESENT IN WHOLE and PRESENT IN REMAINDER semantic options, respectively. The ASYNCHRONOUS EVENT semantic option is unique in that a generated event in a big step can influence the behaviour of the model in the next big step. A modeller or an analyst should keep track of the generated events in the previous big step to understand the behaviour of the current big step. This semantics for interface events can potentially be a source of complication for a modeller because it is at odds with the semantics of other kinds of events in a semantics, i.e., internal events and environmental input/output events, whose statuses cannot persist beyond a current big step. In the ASYNCHRONOUS EVENT semantics, a generated interface event in a big step acts similar to an environmental input event in the next big step. As such, the ASYNCHRONOUS EVENT semantics is modular with respect to interface events, because an interface event, similar to an environmental

Table 3.5: Semantic options for interface events.

Options	Definition	Characteristics	Examples
Strong Synchronous Event	A generated interface event of a big step is sensed as present from the beginning of the big step.	(+) Modularity (+) Unique status for an interface event during a big step (-) Non-causality	"Hybrid Semantics" [50]
Weak Synchronous Event	A generated interface event of a big step is sensed as present in the snapshot after it is generated.	(+) Causality (-) Unclear status of an interface event during a big step	N/A
Asynchronous Event	A generated interface event of a big step is sensed as present in the next big step after it is generated.	(+) Modularity (-) Previous big step affects current big step	"Output" events in RSML [63] and "GALS" [89]

input event, is either present from the beginning of a big step or is not present at all.

There are several BSMLs that support the notion of inter-component event communication. The "hybrid semantics" of Huizing and Gerth [50], which distinguishes between "local" and "global" events, treats the "global" events of a model according to the Strong Synchronous Event semantic option. The semantics of "output" events in RSML [63] follows the Asynchronous Event semantics; an "output" event is generated by a component via a "`SEND`" command, and can be received by a component via a "`RECEIVE`" event in the next big step. Similarly, the semantics of "registered" events in Esterel [1] follows the Asynchronous Event semantics. In "globally asynchronous locally synchronous (GALS)" languages [19, 89], the communication of events within "local" components of a system follows the semantics of the Present in Whole option, and the "global" communication of events between components follows the semantics of the ASYNCHRONOUS EVENT option.

Example 11 *The model in Figure 3.16 shows a door controller system, which is responsible for unlocking the door to an industrial area only if the temperature inside the area is not above 40°C. The system has two components,* Lock *and* Thermometer*, separated by the thick dashed line. The two components communicate via two interface events,* check_temp *and* heat*. There are three environmental input events,* lock, open, *and* reset*. Event* unlock *is the environmental output event of the model. Consider the snapshot in which the model resides in its* Idle *and* Measure *control states,* temp = 99*, and event* open *is received from the environment. If the* TAKE MANY *big-step maximality semantics, the* SINGLE *concurrency semantics is chosen together with the* STRONG SYNCHRONOUS EVENT *semantic option, then the big step* $\langle\{t_1\}, \{t_6\}, \{t_3\}\rangle$ *is the only possible big step, which, correctly, does not open the door. If the* WEAK SYNCHRONOUS EVENT *semantic option is chosen, then additionally,* $\langle\{t_1\}, \{t_2\}, \{t_6\}\rangle$ *is a valid big step, which opens the door although the temperature is 99°C. If the* ASYNCHRONOUS EVENT *semantic option is chosen, the only possible big step is* $\langle\{t_1\}, \{t_2\}, \{t_6\}\rangle$*, in which event* heat *is sensed in the next big step, after the door has already been opened.*

3.5 Enabledness Memory Protocol

The *enabledness memory protocol* of a BSML determines the values of variables that a transition *reads* for its guard condition (GC). Table 3.6 shows the three possible memory protocols: (i) in the GC BIG STEP option, a read of a variable returns its value from the beginning of the big step; (ii) in the GC SMALL STEP option, a read of a variable returns its value from the beginning of the small step; and (iii) in the GC COMBO STEP option, a read of a variable returns its value from the beginning of the current combo step.[6] As such, in the GC BIG STEP, the GC SMALL STEP, and the GC COMBO STEP semantics, the last *write* of a value to a variable, via an assignment, during the current big step, the current small step, and the current combo step, respectively, becomes the value returned by a *read* of that variable in the next big step, next small step, and next combo step, respectively.

[6]As shown in Table 3.6, in SCR [46, 47], both the GC BIG STEP and GC SMALL STEP memory protocols are used, but in different syntactic constructs of the language, namely in the "event tables" and "condition tables", respectively.

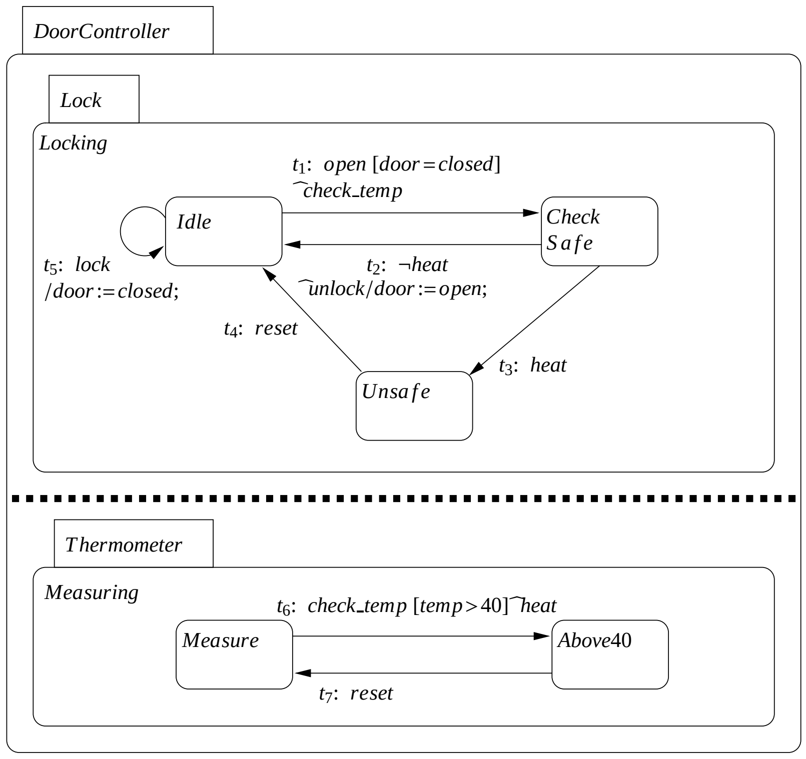


Figure 3.16: Door controller system: using interface events *heat* and *check_temp*.

Table 3.6: Enabledness memory protocols.

Options	Definition	Characteristics	Examples
GC BIG STEP	The value of a variable during a big step is obtained from the beginning of the big step.	(+) Non-interference (+) Modularity (-) Non-sequentiality in small steps	statecharts [45, 86], SCR [46, 47], and reactive modules [3]
GC SMALL STEP	The value of a variable is its up-to-date value, obtained from the beginning of the small step.	(+) Sequentiality in small steps (+) Straightforward traceability (-) Interference	Esterel [14] and SCR [46, 47]
GC COMBO STEP	The value of a variable during a combo step is obtained from the beginning of the combo step.	(+) Some non-interference (+) Some sequentiality in small steps (-) Complicated traceability	Statemate [43]

Traceability: In the GC Big Step semantics, the value of a variable at a snapshot in a big step is obtained from the beginning of the big step, but the assignments to the variable need to be traced so that its value is updated for the next big step. In the GC Small Step semantics, the value of a variable at a snapshot in a big step is determined by tracing all of the assignments to the variable since the beginning of the current big step. In the GC Combo Step semantics, the value of a variable at a snapshot in a big step is determined by tracing all of the assignments to the variable since the beginning of the current combo step in the big step. But a big step may have several combo steps, which, compared to the other memory protocols, could make the tracing of the value of a variable complicated.

Modularity with respect to variables: In general, a semantics is "modular" if it treats the behaviour of a new concurrent part of the model the same as the behaviour of the environment [50]. Originally, "modularity" was defined with respect to events [50], but, in the same spirit, I extend its definition for variables. The GC Big Step is modular with respect to variables because even if a new concurrent part of a model assigns new values to variables, the new values are visible only at the beginning of the next big step, just like new environmental values. The other semantic options are not modular because the behaviour of an addition to an existing model, unlike the environment, affects the intermediate snapshots of a big step.

Non-interference vs. sequentiality in small steps: The GC Big Step option is *non-interfering*: an earlier small step of a big step does not affect the read value of a later small step. Non-interference is useful because it relieves a modeller or an analyst from considering the accumulated effect of assignments to a variable during a big step. The GC Small Step option, which is an "interfering" semantics, is useful for specifying a sequence of computations where each small step reads the values from the previous small step. Sequentiality is useful because it enables a modeller to decompose a computation into parts that each is carried out by a separate transition. The GC Combo Step option enjoys non-interference inside a combo step and sequentiality of combo steps. In the GC Combo Step option, a big step could consist of multiple combo steps, which a modeller needs to keep track of each of their scopes.

Variable operators: A BSML may provide a *variable operator* that obtains a value of a variable that is different from its value according to its memory protocol. Table 3.7 lists some common operators together with a brief description of their semantics. It also specifies whether each

Table 3.7: Variable operators.

Operator	Obtains Value From	Memory Protocols	Total
`pre` (e.g., [63])	big-step source snapshot	GC SMALL STEP	✓
`cur` (e.g., [45])	small-step source snapshot	GC BIG STEP	✓
`new` (e.g., [3])	small-step source snapshot	GC BIG STEP and GC SMALL STEP	✗
`new_small` (e.g., [85])	small-step destination snapshot	GC SMALL STEP	✓

operator is *total* or not. A non-total operator may *block* until it can be evaluated. As specified in the table, each variable operator is relevant for certain enabledness memory protocols.

Operator `pre` returns the values of variables from the beginning of a big step, thus it is relevant for the GC SMALL STEP enabledness memory protocol. Operator `cur` returns the up-to–date values of variables, thus it is relevant for the GC BIG STEP enabledness memory protocol. Operator `new` is different from `cur` in that it can be evaluated only if its operand has already been assigned a value during the big step, which means it requires a "dataflow" order for the execution of small steps within a big step (cf., Section 3.7). Thus, operator `new` can be relevant for both the GC BIG STEP and the GC SMALL STEP enabledness memory protocols.

Operator `new_small` returns the value of its operand at the end of the current small step. It is used in the GC SMALL STEP enabledness memory protocol to look ahead the value of a variable. A variable in the GC of a transition that is prefixed with the `new_small` operator requires an *evaluation order* between the transitions of the small step, in order to obtain the newly assigned value of the variable at the end of the small step. If a variable is not assigned a value during a small step, then its value when prefixed with the `new_small` operator returns the value of the variable at the source snapshot of the small step.[7] Two transitions can create *cyclic evaluation order* by using the `new_small` operator over variables that are assigned values by one another.

Example 12 *The following sequence of arrows shows a sequence of two small steps,*

[7]It is possible to define a non-total `new_small` operator that returns a value for a variable, only if it is assigned a value in the current small step. Such an operator would be in the spirit of the "next" operator in SMV language [58], which is an input language for a family of model checkers with the same name. However in the semantics of SMV, unlike in BSMLs, even if a variable is not assigned a value during a small step, it is assigned a non-deterministic value, which makes the "next" operator a total operator.

$$\xrightarrow{t_1:/v_1:=v_3+1} \quad \xrightarrow{t_2:/v_2:=v_1+1},$$

when $v_1 = v_2 = v_3 = 0$ *at the beginning of the sequence.*

Let us also consider a third transition,

$$t_3 : [(v_1 + v_2 + v_3) \geq 2]/v_3 := v_1 + v_2,$$

which is intended to be executed after the execution of the above two small steps.

If the GC Small Step *enabledness memory protocol is chosen, after the execution of* t_1 *and* t_2*, the values of variables that are used to evaluate* $gc(t_3)$ *will be* $v_1 = 1$, $v_2 = 2$, *and* $v_3 = 0$. *Thus,* $gc(t_3)$ *is true and* t_3 *can be executed. If* t_3 *is changed to* $t_{31} : [(v_1 + v_2 + \mathtt{new\_small}(v_3)) \geq 2]/v_3 := v_1 + v_2$, $gc(t_{31})$ *will be true because of the evaluation* $v_1 = 1$, $v_2 = 2$, *and* $\mathtt{new\_small}(v_3) = 3$. *However, if* t_3 *is changed to* $t_{32} : [(\mathtt{pre}(v_1) + \mathtt{pre}(v_2) + \mathtt{pre}(v_3)) \geq 2]/v_3 := v_1 + v_2$, $gc(t_{32})$ *will be false because of the evaluation* $\mathtt{pre}(v_1) = 0$, $\mathtt{pre}(v_2) = 0$, *and* $\mathtt{pre}(v_3) = 0$.

If the GC Big Step *enabledness memory protocol is chosen, after the execution of* t_1 *and* t_2*, the values of variables that are used to evaluate* $gc(t_3)$ *will still be* $v_1 = 0$, $v_2 = 0$, *and* $v_3 = 0$*, from the beginning of the big step. Thus,* $gc(t_3)$ *cannot be executed. If* t_3 *is changed to* $t_{33} : [(\mathtt{cur}(v_1) + \mathtt{cur}(v_2) + \mathtt{cur}(v_3)) \geq 3]/v_3 := v_1 + v_2$, $gc(t_{33})$ *will be true because of the evaluation* $\mathtt{cur}(v_1) = 1$, $\mathtt{cur}(v_2) = 1$, *and* $\mathtt{cur}(v_3) = 0$. *If* t_3 *is changed to transition* $t_{34} : [(\mathtt{new}(v_1) + \mathtt{new}(v_2) + \mathtt{new}(v_3)) \geq 3]/v_3 := v_1 + v_2$, $gc(t_{34})$ *cannot be evaluated because* $\mathtt{new}(v_3)$ *could have been evaluated only if* v_3 *would have been assigned a value by the previous small steps.*

Example 13 *In Example 8, on page 51, the* GC Small Step *enabledness memory protocol was used. If the same semantic options that led to an incorrect behaviour in that example are used, but the guard condition of transition* t_6 *is changed to* "[new_small(c) < |p|]" *and its generated event to event* "dial(digit(new_small(c) + 1, p))"*, then the model behaves correctly:* $\langle \{t_5\}, \{t_2, t_6\}, \{t_3, t_6\}, \cdots, \{t_3\}, \{t_4, t_7\} \rangle$.

The operators in Table 3.7 are not relevant for the GC Combo Step memory protocol, but they can be extended to be used in the context of GC Combo Step memory protocol. For example, a version of `cur` operator for the GC Combo Step semantic option would return the current value of a variable considering all of the assignments to the variable since the beginning of the current combo step. Similarly, a `new_small` operator can be defined for the GC Big Step memory protocol.

3.5.1 External Variables

As with events, it is useful to distinguish syntactically between the variables of the model that can be modified by the environment and the variables of the model that can be modified by the system [79, 101]. Figure 3.14, which depicts a taxonomy of events, also represents the taxonomy for distinguishing environmental variables. The *environmental output variables* and *environmental input variables* of a model are the sets of the variables of the model that can be read from and written to by the environment, respectively. The *internal variables* of a model are those variables that do not communicate with environment.[8] The union of the set of environmental input variables and the set of environmental output variables of a model is its set of *external variables*. The union of the set of environmental output variables and the set of internal variables of a model is its set of *controlled variables*, which is the set of variables that can be written to by the system. Many modelling languages, including some BSMLs, provide syntax to distinguish between different types of variables [3, 46, 47, 79]. Unlike for events, the notion of "non-distinguishing BSMLs" (cf., Section 3.4.1) is not relevant with respect to variables, because most BSMLs either syntactically distinguish between environmental input variables and controlled variables, or they do not support the notion of environmental input variables at all (i.e., variables are not assigned values by the environment).

When external variables are distinct from the internal variables, the memory protocol semantic aspects described in Sections 3.5 and 3.6 specify the semantics of internal variables. The notion of memory protocol for environmental input variables is not relevant because they are never assigned a value by a transition; they keep the same value throughout the big-step. Normally, an output variable is not read by the model, therefore no option has been included for it in the feature diagram. If an output variable is read by the model, the semantics of environmental output variables can be any of the memory protocols, but it would not likely be the Big Step semantics.

3.5.2 Interface Variables in GC

Some BSMLs allow a component of a model, which is usually a physically distinct part of the model, to communicate with another component of the model via *interface variables*. Fig-

[8]Internal variables are often called "private variables". The term "internal variables" is adopted to keep the terminology of variables consistent with that for events.

ure 3.15, which depicts the taxonomy of events including interface events, can also describe the taxonomy of variables including interface variables. As for interface events, conventionally, an interface variable can be written to by only one component (the *sending component*), but can be read by multiple components (the *receiving components*). The semantics of interface variables, similar to memory protocols for internal variables, specifies when a change to an interface-variable value becomes the value returned by a read of that variable.

Table 3.8 lists the possible inter-component communication semantic options. In the GC Strong Synchronous Variable option, a write to an interface variable during a big step can be read by the GC of a transition right from the beginning of the same big step; i.e., if an interface variable is assigned a value, only this new value is read during the big step. In the GC Weak Synchronous Variable option, a write to an interface variable can be read after the variable is written to, but the variable can also be read before it is written to, in which case it returns its value from the previous big step (similar to the GC Small Step semantic option). In the GC Asynchronous Variable option, a write to an interface variable can be read by the GC of any transition in the next big step (similar to the GC Big Step semantic option).

Blocking read vs. communication delay: The GC Strong Synchronous Variable semantics is compatible with the "zero-time computation" principle of the synchrony hypothesis [10, 14]: The value of an interface variable is exchanged between two components in "zero-time". However, there should exist a "dataflow order" (cf., Section 3.7) between the small steps of a big step so that the value of an interface variable is read only after it has been assigned. A component that is waiting for the new value of an interface variable is said to be *blocking*. It is possible for two transitions to block cyclically on each other creating deadlock. In the GC Weak Synchronous Variable semantic option, a read operation on a variable never blocks, but it may return a *stale value* of the variable from the previous big step or a newly assigned value from the current big step. In the GC Asynchronous Variable semantic option, a read operation on a variable never blocks, but there is a delay of one big step between writing a new value to a variable and reading the new value.

Modularity with respect to interface variables: The GC Strong Synchronous Variable and GC Asynchronous Variable semantic options are modular with respect to interface variables because the value of an interface variable in these semantic is the same throughout the big step, similar to an environmental input variable. In these two semantics, the behaviour of a component

Table 3.8: Semantic options for interface variables.

Options	Definition	Characteristics	Examples
GC Strong Synchronous Variable	Either an interface variable is not written to during a big step, or all of its reads happen after it has been written to and it returns the newly assigned value.	(+) Modularity (-) Blocking read and cyclic dataflow order	Composition in reactive modules [3]
GC Weak Synchronous Variable	An interface variable can be read before or after it is written to; in the latter case it returns the newly assigned value.	(+) Non-blocking read (-) Stale values for interface variables	N/A
GC Asynchronous Variable	The value written to an interface variable during a big step can be read in the next big step.	(+) Non-blocking read (+) Modularity (-) Delayed read	“Output” variables in RSML [63]

that is added to an existing model is perceived as that of the environment, when it comes to the interface variables in the GC of transitions of the existing model. The GC WEAK SYNCHRONOUS VARIABLE semantic option is not modular with respect to interface variables because the value of an interface variable may change during a big step, unlike the value of an environmental input variable.

Example 14 *The model in Figure 3.17 is similar to the model in Example 11, but has been modified: (i) to use the interface variable* heat*, instead of interface event* heat*; and (ii) the functionality of* Locking *the door is separated from the functionalities of the* Controller *of the lock and the* Thermometer*, to allow for the lock to work with different controllers.*

Let us consider the snapshot where the model resides in its Idle*,* Ready*, and* Measure *control states, the door is closed,* temp = 99*,* heat = *false, and event* open *is received from the environment. Also, let us choose the* SYNTACTIC *big-step maximality semantics, the* SINGLE *concurrency semantics, the* PRESENT IN REMAINDER *event lifeline semantics, the* GC *(and* RHS*)* SMALL STEP *enabledness (assignment) memory protocols, and the* GC STRONG SYNCHRONOUS EVENT *interface event semantics. If the* GC STRONG SYNCHRONOUS VARIABLE *semantic option is chosen, then the big step* $\langle\{t_1\}, \{t_6\}, \{t_9\}, \{t_8\}, \{t_3\}\rangle$ *is the only possible big step, which correctly does not open the door. If the* GC WEAK SYNCHRONOUS VARIABLE *semantic option is chosen, then the big step* $\langle\{t_1\}, \{t_6\}, \{t_7\}, \{t_9\}, \{t_2\}\rangle$ *is also possible, which opens the door although the temperature is 99°C. Reversing the order of* $\{t_9\}$ *and* $\{t_2\}$ *yields another big step that opens the door. If the* GC ASYNCHRONOUS VARIABLE *semantic option is chosen, then the* true *value of* heat *is only sensed in the next big step, and thus the door is opened.*

3.6 Assignment Memory Protocol

The *assignment memory protocol* of a BSML determines the values of variables that a transition reads when evaluating the righthand side (RHS) of an assignment. Exactly the same semantic options as those of the enabledness memory protocol exist: RHS BIG STEP, RHS SMALL STEP, and RHS COMBO STEP. (Their names are prefixed with "RHS" instead of "GC".) The enabledness and assignment memory protocols of a BSML need not be the same (e.g., SCR [46, 47]).In SCR [46, 47], the RHS SMALL STEP assignment memory protocol is used together with a combination of the GC BIG STEP and GC SMALL STEP enabledness memory protocols. The same advantages and

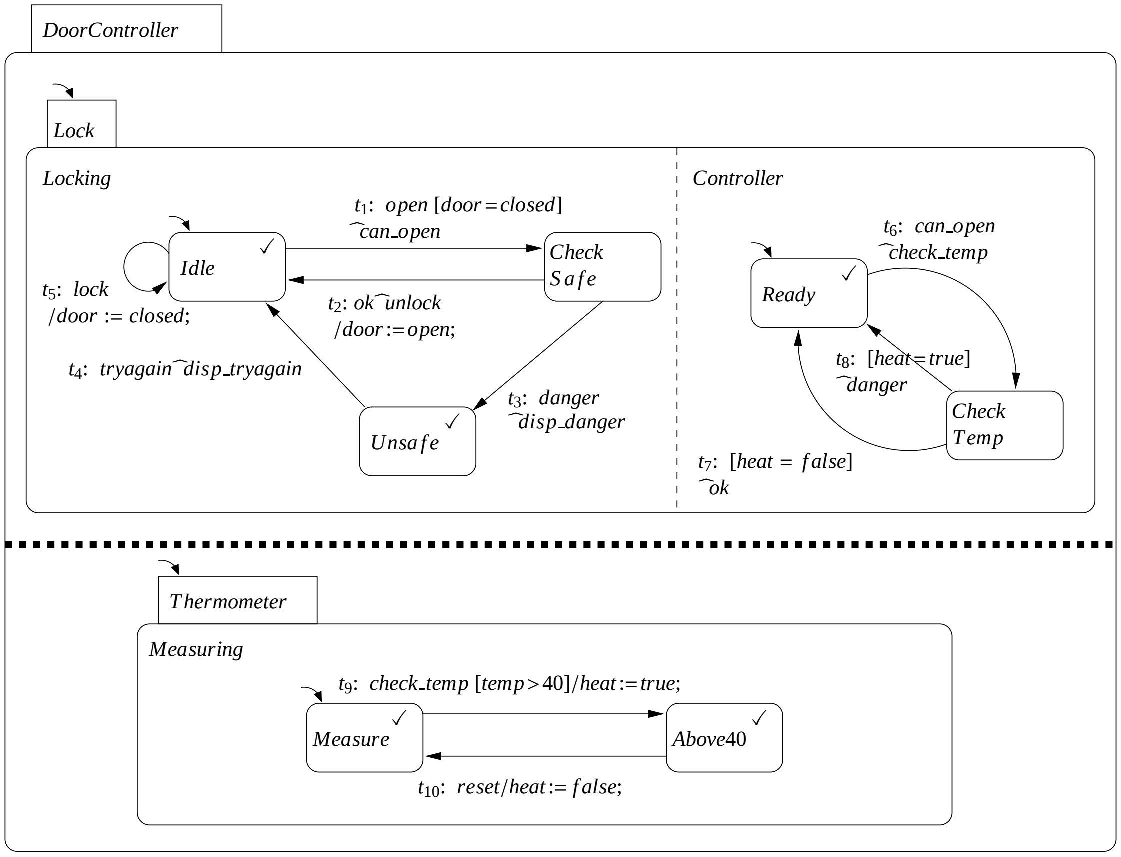
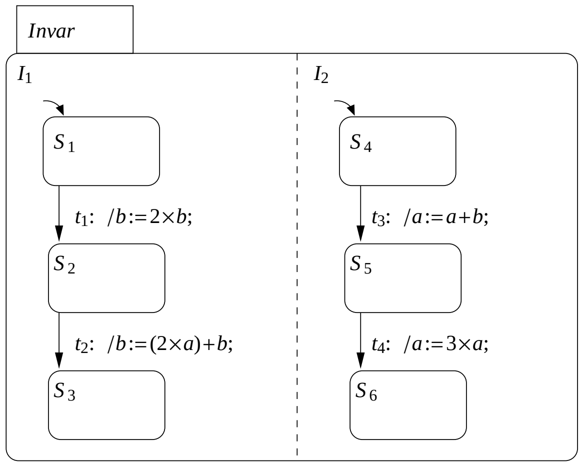


Figure 3.17: Door controller system: using interface variable *heat* and interface event *check_temp*.



Invar
I_1
I_2
S_1
S_2
S_3
S_4
S_5
S_6
t_1: $/b := 2 \times b;$
t_2: $/b := (2 \times a) + b;$
t_3: $/a := a + b;$
t_4: $/a := 3 \times a;$

Figure 3.18: A model for maintaining an invariant between *a* and *b*.

disadvantages as the semantic options of the "enabledness memory protocol", in Table 3.6, apply to the corresponding semantic options of the "assignment memory protocol" semantic aspect, so they are not repeated in this section.

Variable operators: The same four variable operators listed in Table 3.7 can be used in the RHS of assignments. However, when using the `new_small` operator in an assignment expression, it may be impossible to find an "evaluation order". For example, for two assignments, $a :=$ `new_small`$(b) - 1$ and $b :=$ `new_small`$(a) + 2$, which have a cyclic evaluation order, the value of *a* and *b* cannot be evaluated.

Example 15 *The model in Figure 3.18, which is adopted from an example in [49], is meant to specify a computation that maintains the invariant that* a−b *has the same value before and after the execution of a big step. Consider the snapshot where the model resides in its control states* S_1 *and* S_4, a = 7, *and* b = 2. *Let us choose the* SINGLE *concurrency semantics. If the* TAKE MANY *big-step maximality semantics together with the* RHS BIG STEP *assignment memory protocol are chosen, then the end result would be* a = 21 *and* b = 16, *which maintains the invariant that* a−b *has the same value before and after the big step. If the* RHS SMALL STEP *semantic option is chosen, then the model can create a big step that does not maintain the invariant; for example, the execution of the big step* $\langle\{t_1\}, \{t_2\}, \{t_3\}, \{t_4\}\rangle$ *results in* a = 75 *and* b = 18.

3.6.1 Interface Variables in RHS

Similar to the using of interface variables in the GC of transitions, as described in Section 3.5.2, interface variables can be used in the RHS of assignments of the transitions of the different components of a system. Exactly the same semantic options as those for interface variables in GC of transitions can be used for the semantics of interface variables in the RHS of assignments, but their names prefixed with "RHS" instead of "GC": RHS Strong Synchronous Variable, RHS Weak Synchronous Variable, and RHS Asynchronous Variable. The interface variables in GC semantics of a BSML need not be the same as its interface variables in RHS semantics. Similar to the GC Strong Synchronous Variable option, a cyclic dataflow order might arise when the RHS Strong Synchronous Variable semantic option is chosen. The same advantages and disadvantages as the ones for the semantic options of the inter-component variable communication, in Table 3.8, are relevant for the corresponding semantic options of the interface variables in RHS semantic aspect. Therefore, they are not repeated here.

3.7 Order of Small Steps

At a snapshot, when it is possible to execute more than one small step based on the enabledness of transitions with respect to guard conditions and event triggers, some BSMLs non-deterministically execute one (the None option), while others order their executions either by syntactic means (the Explicit Ordering option) or by *dataflow* orders (the Dataflow option), as shown in Table 3.9. Stateflow is an example of the Explicit Ordering option because the transitions of a model are executed according to the graphical, clockwise order of their arenas [22]. A dataflow order allows only those sequences of small-steps where a transition that writes to a variable is executed before a transition that reads the variable. The dataflow order of a model can be specified by an explicit partial order between its variables (e.g., SCR [46, 47]), or via variable operator `new`, as described in Section 3.5, to determine data dependencies (e.g., reactive modules [3]). In the statecharts semantics of Pnueli and Shalev [86], the boolean operator `assigned` is used in the event trigger of a transition to determine whether a variable is assigned a value during a big step or not, which, in effect, induces a dataflow order between small steps of the big step.[9] The Explicit Ordering and Dataflow options can be used to avert undesired non-determinism by disallowing

[9]The GC Strong Synchronous Variable and RHS Strong Synchronous Variable semantic options for interface variables, described in Section 3.5.2 and Section 3.6.1, respectively, can also introduce dataflow orders.

Table 3.9: Order of small steps semantic options.

Options	Definition	Characteristics	Examples
None	Small steps are not ordered.	(+) Simplicity (-) Non-determinism	statecharts [41, 45]
Explicit Ordering	Execution of small steps is ordered syntactically.	(+) Control over ordering (+) Control over non-determinism (-) Possible unintended ordering	Stateflow [22]
Dataflow	Small steps are ordered so that an assignment to a variable happens before it is being read.	(+) Natural for some domains (+) Control over non-determinism (-) Possible cyclic orders	SCR [46, 47], reactive modules [3], and statecharts [86]

the execution of the small steps that do not satisfy the ordering constraints. In the Dataflow semantic option, each big step of a model might have a different dataflow order. The Explicit Ordering option can be difficult to use because a modeller may introduce an unintended order of transitions. The Dataflow semantics can be difficult to use because a modeller might create a cyclic dataflow order, either directly or by transitivity. The Dataflow semantics is compatible with the domain of some synchronous hardware systems where there is an inherent distinction between the value of a variable at the beginning of a big step, i.e., when the clock ticks, and during a big step when a value might be assigned to a variable.

Example 16 *Consider the semantic options in Example 8, on page 51, that lead to an incorrect behaviour. One way to fix the incorrect behaviour is to modify the model by moving the* "p := lp" *assignment from* t_5 *to* t_2, *changing the GC of* t_6 *to* "$c < |new(p)| - 1$", *and its event generation to* "dial(digit(new_small(c) + 1, p))". *Such a model then behaves correctly:* $\langle\{t_5\}, \{t_2\}, \{t_6\}, \{t_3, t_6\}, \cdots, \{t_3\}, \{t_4, t_7\}\rangle$, *because the dataflow order does not allow* t_2 *and* t_6 *to be executed together.*

Example 17 *In Example 8, the* Many *concurrency semantics was chosen together with the* Present in Next Small Step *event lifeline semantics, which lead to an incorrect behaviour. If* *the* SINGLE *concurrency semantics would be chosen, then the model would create both a correct big step, and an incorrect, non-terminating big step (e.g.,* $\langle\{t_5\}, \{t_2\}, \{t_6\}, \{t_6\}, \cdots\rangle$*), non-deterministically. However, if the* EXPLICIT ORDERING *order of small steps semantics according to the graphical, clockwise order of the arena of transitions would be chosen, then the model would always behave correctly:* $\langle\{t_5\}, \{t_2\}, \{t_6\},\{t_3\}, \{t_6\}, \{t_3\}, \cdots, \{t_7\}, \{t_4\}\rangle$.

3.8 Priority

At a snapshot of a model, there could exist multiple sets of transitions that can be chosen non-deterministically to be executed as its small step. Table 3.10 shows three common ways for assigning a priority to a transition to avert non-determinism. A set of transitions T_1 has a higher priority than a set of transitions T_2, if for each pair of transitions $t_1 \in T_1$ and $t_2 \in T_2$, either t_1 has a higher priority than t_2 or they are not comparable priority wise.

The HIERARCHICAL option is a set of priority semantics that use the hierarchical structure of the control states of a model to compare the relative priority of two enabled transitions. A HIERARCHICAL priority semantics is defined by its *basis*, which is one of the three values, SOURCE, DESTINATION, SCOPE, and its *scheme*, which is either PARENT or CHILD. For example, SCOPE-PARENT is a priority semantics that gives a higher priority to a transition whose scope is the highest in the hierarchy of a composition tree.

If a BSML semantics supports neither the SCOPE-CHILD nor the SCOPE-PARENT priority semantics, the semantic option NO PRIORITY is used to characterize it. In the NO PRIORITY semantic option, all transitions of a model have the same priority. The NO PRIORITY semantic option could introduce undesired non-determinism because if more than one transitions are enabled at a snapshot, any of them can be taken. The NO PRIORITY semantic option is useful when a modeller is interested in using the hierarchy tree of a model only as a means to distinguish between the general and the specialized elements of the behaviour of a system, by using high-level and low-level control states of the model, respectively, without giving neither element of the behaviour a higher priority than the other.

The EXPLICIT PRIORITY priority option explicitly assigns priority to the transitions of a model (e.g., by assigning numbers to transitions and giving a greater number a higher priority [75]).

The NEGATION OF TRIGGERS option is not an independent way to assign priority, but uses the

Table 3.10: Priority semantic options.

Options	Definition	Characteristics	Examples
HIERARCHICAL	The source and destination control states of transitions determine priority.	(+) Simplicity (-) Incomplete prioritization	SCOPE-PARENT in Statemate [43] and SOURCE-CHILD in Rhapsody [42]
EXPLICIT PRIORITY	Each transition is given an explicit, relative priority.	(+) Exhaustive prioritization (-) Tedious to use	Used in [75]
NEGATION OF TRIGGERS	A transition is given higher priority than another by strengthening the event trigger of the second transition such that it is not enabled when the first transition is enabled.	(+) Exhaustive prioritization (+) No additional syntax (-) Tedious to use	statecharts [86], Esterel [14], and Argos [68]
NO PRIORITY	All transitions have equal hierarchical priority.	(+) Avoid unintended priority (-) Non-determinism	statecharts [41]

notion of "negation" to assign priorities: t_1 can be assigned a higher priority than t_2 by conjoining the negation of one of the positive events in the trigger of t_2 with the events in the trigger of t_1.

Exhaustiveness vs. simplicity: The Hierarchical option can be easily understood by a modeller, but may render many transitions as priority incomparable. The Explicit Priority option provides greater control over specifying the relative priority of a set of transitions, but can be tedious to use. For example, a wrong relative priority for a pair of transitions can be deduced transitively, although a modeller may not have been aware of such an indirect, transitive prioritization in her/his model. In the Negation of Triggers and Explicit Priority options, it can be difficult to identify the pair of transitions where it is necessary to assign a relative priority because whether two transitions are both enabled or not in a small step depends on the source snapshot, and not merely on the syntax of a model. But in principle, it is possible to specify a priority scheme for a model exhaustively.

Combination of priority semantics: It is possible to use more than one priority semantics in the semantics of a BSML, as shown in the feature diagram in Figure 3.2. In such a BSML, if two transitions are not comparable according to the first priority semantics, then they are compared according to the second semantics, and so on. By the definition of enabledness, if the Negation of Triggers is used in a BSML, its semantics overrides the other priority semantics.

Example 18 *In Example 6, on page 44, if the* Single *concurrency and the* Scope-Child *priority semantics are chosen together, then the model always executes* $\langle\{\mathrm{t}_1\}\rangle$ *as its big step, allowing the call to go through.*

Example 19 *In the model in Figure 3.5, on page 38,* t_2 *is assigned a higher priority than* t_1 *by conjoining the original event trigger of* t_1, dial(d), *with the negation of the event trigger of* t_2, dial(d) $\wedge$ redial, *resulting in* t_1 *having the event trigger* dial(d) $\wedge \neg$redial. *The effect is that* t_2 *will be chosen when the* redial *event occurs instead of* t_1.

Example 20 *In Example 11, on page 58, if transition* t_6 *is given a higher priority than* t_2 *explicitly, then the choice of the* Weak Synchronous Event *semantic option always yields a correct behaviour (i.e., the door is not opened when the temperature is above 40°C). Similarly, in Example 14, if transition* t_9 *is given a higher priority than* t_7 *explicitly, then the choice of the* Weak Synchronous Variable *semantic option always yields a correct behaviour.*

Remainder of thesis: In the remainder of the thesis, for the sake of brevity, only the Scope-Parent and Scope-Child hierarchical priority semantics are considered. However, the semantics of other Hierarchical priority semantics are very similar to these two. Also, the Explicit Priority semantics, which is not a common priority semantics, is not considered in the remainder of the thesis, except in Section 7.3.3, where the feasibility of formalizing this semantics is discussed. A formalization of the Explicit Priority semantics can be found in template semantics [75, 74].

3.9 Combo-Step Maximality

The combo-step maximality semantics specifies the extent of a contiguous segment of a big step where computation is carried out based on the statuses of events and/or values of the variables at the beginning of the segment. As specified in Figure 3.3, the combo-step maximality semantics is relevant for a BSML semantics only if at least one of the *combo-step semantic options*, namely, Present in Next Combo Step, GC Combo Step, or RHS Combo Step, is chosen in the semantics. These options describe how the statuses of events and values of variables change (or not) within a combo step. For example, if a BSML uses the Present in Next Combo Step and GC Combo Step options, then during a combo step (other than the first combo step of the big step) the statuses of events depend on the generated events of the previous combo step, and the values of variables in GC of transitions depend on the assignments performed in the previous combo step.

Table 3.11 shows the three semantic options for the combo-step maximality semantic aspect. These options are similar to the three semantic options for the big-step maximality semantics, but specify the scope of a combo step, instead of a big step. In the Combo Syntactic option, a BSML allows a modeller to designate a basic control state of a model as a *combo stable* control state. During a combo step, once a transition t that enters a combo stable control state is executed, no other transition whose arena overlaps with the arena of t can be taken during that combo step. In the Combo Take One option, once a transition t is executed during a combo step, no other transition whose arena overlaps with the arena of t can be executed during that combo step. As such, each *Or* control state can contribute a maximum of one transition to a combo step. The Combo Take Many option allows a sequence of small steps to continue executing until there are no more enabled transitions to be executed. In practice, the BSMLs that use the Combo Take One option for the combo step maximality semantics use the Take Many option for the big-step

Table 3.11: Combo-step maximality semantic options.

Options	Definition	Characteristics	Examples
COMBO SYNTACTIC	No two transitions with overlapping arenas that enter designated "combo stable" control states can be taken in a same combo step.	(+) Syntactic scope for combo steps (+) Sequential *Or* transitions in a combo step (-) Non-terminating combo steps	N/A
COMBO TAKE ONE	No two transitions with overlapping arenas can be taken in a same combo step.	(+) Terminating combo steps (+) Unclear, non-syntactic scope for combo steps	RSML [63] and Statemate [43]
COMBO TAKE MANY	No constraint on transitions that can be taken in a combo step.	(+) Sequential *Or* transitions in a combo step (-) Unclear, non-syntactic scope for combo steps (-) Non-terminating combo steps	N/A

maximality semantics (e.g., RSML [63] and Statemate [43]). As specified in Figure 3.3, the COMBO TAKE MANY combo-step maximality semantics cannot be chosen together with the TAKE ONE big-step maximality semantics, because a combo step cannot include more small steps than its big step. The same advantages and disadvantages as the ones for the semantic options of the big-step maximality semantic aspect are relevant for the corresponding semantic options of the combo-step maximality semantic aspect.

Scope of a combo step: In the COMBO SYNTACTIC semantic option, the end of a combo step can be traced syntactically, which can be helpful for constructing and understanding a model. The scope of a combo step when the COMBO TAKE ONE or the COMBO TAKE MANY is chosen is more difficult to determine. For example, if the COMBO TAKE MANY combo-step maximality semantics, along with the PRESENT IN NEXT COMBO STEP and GC COMBO STEP semantic options, are chosen,

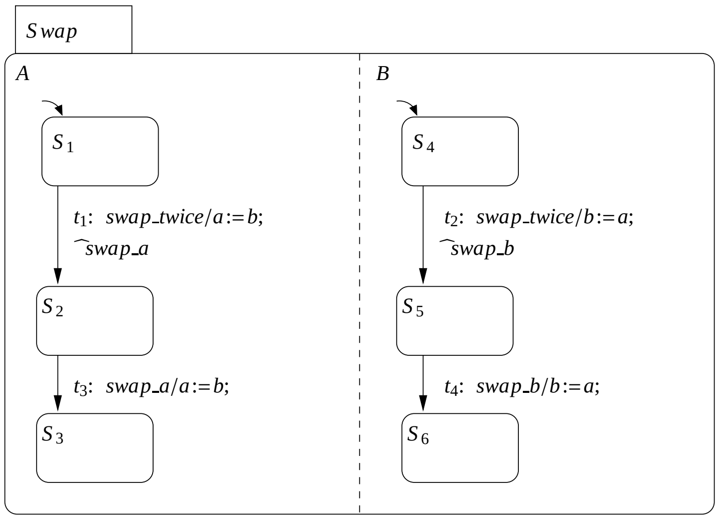

Swap
A
B
S1
S2
S3
S4
S5
S6
t1: swap_twice/a := b; ⁀swap_a
t2: swap_twice/b := a; ⁀swap_b
t3: swap_a/a := b;
t4: swap_b/b := a;

Figure 3.19: Swapping *a* and *b* twice, using combo steps.

then a combo step of a big step continues until there are no more transitions that are enabled with respect to the generated events and the assignments of the previous combo step. In such a semantics, it is far from clear what the possible combo steps, and thus big steps, of a model are, based on mere review of the syntax of the model.

Example 21 *The model in Figure 3.19 is meant to swap the values of variables* a *and* b *twice during a big step, maintaining their original values. Let us choose the* Combo Take One *option for the combo step maximality semantics, the* Take Many *option for the big-step maximality semantics, the* Single *concurrency semantics, and the semantics that the statuses of events and the values of variables are fixed during a combo step (i.e., the* RHS Combo Step, *and the* Present in Next Combo Step *semantic options). Upon receiving the environmental input event* swap_twice, *the model executes transitions* t_1 *and* t_2*, at which point the first combo step concludes. The second combo step starts by first considering the effects of the transitions of the first combo step, i.e., the effect of swapping the values of* a *and* b *and the effect of generating events* swap_a *and* swap_b, *and then executing transitions* t_3 *and* t_4*. At the end of the second combo step the big step concludes and the values of* a *and* b *are the same as their values at the beginning of the big step. If the effect of the assignments of the transitions are not hidden from one another during a combo step, the correct behaviour cannot be achieved. For example, depending on whether* t_1 *or* t_2 *is executed first, both* a *and* b *are assigned the initial value of* b *or* a*, respectively.*

Choosing the TAKE MANY *big-step maximality semantics, the* MANY *concurrency semantics, the* PRESENT IN NEXT COMBO STEP *event lifeline semantics (or the* PRESENT IN REMAINDER *event lifeline semantics), and the* RHS SMALL STEP *assignment memory protocol, also yields the correct behaviour. While such an equivalence of behaviour holds for some models, it does not always hold. For example, if there is a possibility for race conditions (e.g., in Example 22) or if it is important whether a model can reach certain configuration of control states or not, then it is not possible to replace the* SINGLE *concurrency semantics with the* MANY *concurrency semantics.*

Example 22 *The model in Figure 3.20 shows a simple model of a system that controls the operation of a chemical plant.*[10] *The operation of the plant relies on two chemical substances* A *and* B*. There are two processes, shown as two Or control states* Process_1 *and* Process_2*, which can independently increase the amounts of substances* A *and* B *by one unit or two units, respectively. The two processes may simultaneously request for increase; i.e., environmental input events* inc_one *and* inc_two *might be received at the same big step. Variables* a *and* b *represent the amount of requested increase for substance* A *and substance* B*, respectively. Environmental output event* start_process(a, b) *instructs a physical component of the plant to increase the amounts of substance* A *and* B*, by amounts* a *and* b*, respectively. Internal event* process *is meant to instruct the* Controller *to increase the amounts of the substances. Environmental input event* end_process *signifies that the requested amounts of the substances have been successfully increased by the physical component of the plant, at which point the system can process new requests.*

Consider the snapshot where the model resides in its default control states, inc_one *and* inc_two *are received, and* a *and* b *are zero. The correct behaviour is to increase the amount of* A *and* B *by three units. Let us choose the* COMBO TAKE ONE *option for the combo step maximality semantics, the* TAKE ONE *option for the big-step maximality semantics, and the* SINGLE *concurrency semantics. The only pair of semantic options that yield a correct behaviour are, the* PRESENT IN NEXT COMBO STEP *for the event lifeline semantics and the* RHS SMALL STEP *semantic option for the assignment memory protocol semantics, which produce the following two correct big steps:* $\langle\{t_1\}, \{t_3\}, \{t_5\}\rangle$ *and* $\langle\{t_3\}, \{t_1\}, \{t_5\}\rangle$*. If, for example, the* PRESENT IN NEXT COMBO STEP *event lifeline semantics is chosen together with the* RHS COMBO STEP *assignment memory protocol, the same big steps as before are produced, but the former big step increases the amounts of* A *and* B *by*

[10]This example is inspired by the motivating example in [2], where sequence diagrams are used for modelling an aspect of the operation of a nuclear power plant.

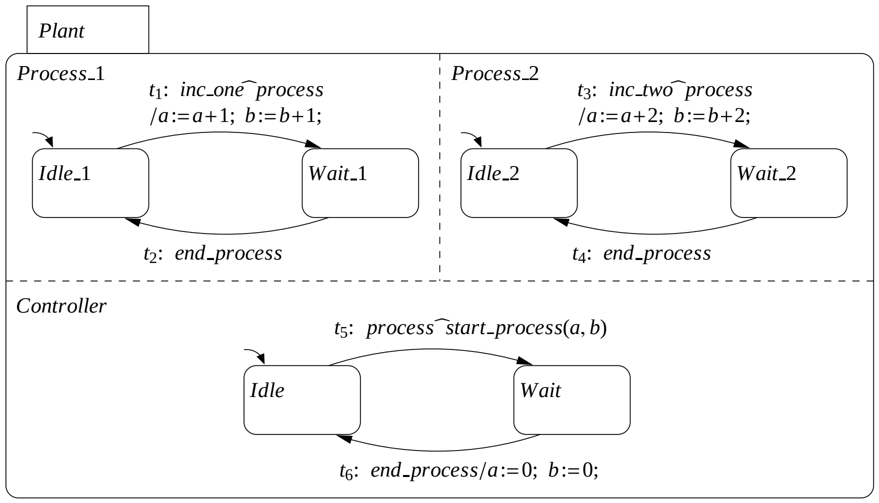


Figure 3.20: Controlling the operation of a chemical plant.

two units only, whereas the latter big step increases the amounts of A *and* B *by one unit only. If the* PRESENT IN REMAINDER *event lifeline semantics is chosen together with the* RHS SMALL STEP *assignment memory protocol, which means that there is no need to choose any semantic option for the combo-step maximality semantic aspect, the additional big step* $\langle\{t_1\}, \{t_5\}, \{t_3\}\rangle$ *is possible, which ignores the increase requested by* Process_2.

Example 23 *In Example 7, on page 51, some possible semantics to make the counter in Example 3 to behave correctly were enumerated. Another possible semantics is a semantics that subscribes to the* COMBO TAKE ONE *combo-step maximality semantics, the* TAKE ONE *big-step maximality semantics, the* SINGLE *concurrency semantics, and the* PRESENT IN NEXT COMBO STEP *event lifeline semantics.*

Example 24 *Another way to maintain the invariant in Example 15, on page 69, is to choose the* COMBO TAKE ONE *combo-step maximality semantics, the* TAKE MANY *big-step maximality semantics, and the* RHS COMBO STEP *assignment memory protocol. The execution of the first combo step,* $\{t_1\}$, $\{t_3\}$, *results in* $a = 9$ *and* $b = 4$, *and the execution of the second combo step,* $\{t_2\}$, $\{t_4\}$, *results in* $a = 27$ *and* $b = 22$. *The order of the execution of* $\{t_1\}$ *and* $\{t_3\}$, *and,* $\{t_2\}$ *and* $\{t_4\}$, *do not affect the end result. If the* COMBO TAKE MANY *combo-step maximality semantics is chosen, then the invariant would be maintained, but the big step concludes with* $a = 21$ *and* $b = 16$.

3.10 Semantic Side Effects

In this section, a few *side effects* that arise when a group of semantic options are chosen together are described. Also, it is explained how these side effects can be avoided. The choice of a group of semantic options has a side effect when it causes a semantic complication that is not due to the original design of any of the semantic options. A side effect can sometimes be tolerated because the benefit of having a set of semantic options in a BSML outweighs their caused complication.

3.10.1 Complicated Event Lifeline Semantics

Choosing the Take One big-step maximality semantics when the Present in Whole event lifeline semantics is used in a BSML semantics, achieves a less complicated semantics, as is done in Argos [68]. The Take One semantic option introduces less complication compared to the other big-step maximality semantics because the status of an event in a big step can be identified by considering at most one transition of each of the non-overlapping arenas of a model. Similarly, I recommend to choose the Take One semantic option, when choosing the Strong Synchronous Event semantic option for interface events.

3.10.2 Cyclic Evaluation Orders

To avoid a "cyclic evaluation order" when using the `new_small` operator, as described in Section 3.6, a conservative well-formedness criterion can disallow small steps whose assignments create cyclic evaluation orders. Such a well-formedness criteria depends on the choice of the semantic options for the Small-Step Consistency and Preemption semantic aspects. For example, consider a BSML that subscribes to the Arena Orthogonal small-step consistency semantics and the Preemptive preemption semantics. For such a semantics, a conservative well-formedness condition to avoid a cyclic evaluation order is to require that, for every pair of orthogonal control states S_1 and S_2, if the arena of t is S_1, or a descendent of S_1, and t uses `new_small`(u) in the RHS of its assignment a_1 and assigns a value to variable v in assignment a_2, then there is no t' whose arena is S_2, or a descendent of S_2, and uses `new_small`(v) in the RHS of its assignment a'_1, together with assigning a value to u in its assignment a'_2.

3.10.3 Ambiguous Dataflow

An ambiguity arises for a dataflow order if a variable is prefixed by the `new` operator but it is assigned values more than once during a big step. A sufficient, but not necessary, condition for an unambiguous DATAFLOW order of small-steps is to require the TAKE ONE big-step maximality semantics with each variable assigned a value only by the transitions that have the same arena, as is done in SCR [46, 47] and reactive modules [3]. Similarly, the TAKE ONE semantic option can be chosen together with the GC STRONG SYNCHRONOUS VARIABLE or the RHS STRONG SYNCHRONOUS VARIABLE semantic options for interface variables, to avoid ambiguity in obtaining the value of an interface variable.

3.10.4 Complicated Explicit Ordering

In the EXPLICIT ORDERING semantic option, when the small steps of a big step are ordered according to the order of the arenas of the transitions of the big step, being able to take two transitions with the same arena in the same big step causes complication in defining the semantics. For example, if the TAKE MANY big-step maximality semantics is chosen, a complication arises because a big step may consist of several rounds of small steps, some of the small steps belonging to the same arena. To avoid a complicated semantics, the TAKE ONE big-step maximality semantics could be required when the EXPLICIT ORDERING order of small steps semantics is chosen.

3.10.5 Partial Explicit Ordering

Frequently, the SINGLE concurrency semantics is chosen with the EXPLICIT ORDERING order of small-steps semantics when the EXPLICIT ORDERING ordering allows only one transition to be taken in each small step. However, if the ordering is partial, or hierarchically-based, then the MANY concurrency semantics can also be used.

3.10.6 Inconsistent Preemption and Priority Semantics

When the PREEMPTIVE preemption semantics is chosen, the choice of the priority semantics determines whether the interrupt transition has higher or lower priority than non-interrupt transitions.

For example, giving the highest priority to a transition whose destination control state is the lowest in the composition tree, i.e., the choice of the Destination-Child semantics, has the effect of giving interrupt transition t in Figure 2.2(b) a higher priority than t', which is an intuitive, desired behaviour. Similarly, the Scope-Parent priority semantics gives transition t in Figure 2.2(a) a higher priority than transition t'.

3.10.7 Conflicting Maximality

The choice of the Syntactic semantic option for the big-step maximality semantics together with the choice of the Combo Syntactic semantic option for the combo-step maximality semantic aspect means that a small step may move a model to a snapshot where the model resides in a pair of orthogonal control states, one being a **Stable** control state and the other a **Combo Stable** control state. In such a snapshot, it is unclear whether the current combo step has concluded, or not. Alternatively, choosing the Take Many semantic option for the big-step maximality semantic aspect and the Combo Syntactic semantic option for the combo-step maximality semantic aspect avoids this problem.

3.11 Validation: Specifying the Semantics of BSMLs

In the semantic framework in this chapter, a BSML is described by, first, describing how its syntax can be translated to the normal-form syntax, and then, enumerating its choice of semantic options. The syntactic translation to the normal-form syntax is straightforward for most BSMLs, as briefly discussed in Section 2.3. Table 3.12 shows the specification of the semantics of some of the BSMLs that were considered throughout the chapter. This table validates that the semantics of a wide range of languages can be described by enumerating the constituent semantic options of each. Chapter 4 complements this validation by formalizing most of the semantic options in my big-step semantic deconstruction.

For the sake of brevity, the External Output Events semantic aspect is not included in Table 3.12. Also, some aspects have been merged; e.g., the Enabledness Memory Protocol for Internal Variables in GC merged with Internal Variables in RHS semantic aspects.

Table 3.12: Example BSMLs and their semantic options. ([45]: Harel statecharts, [86]: Pnueli and Shalev statecharts, [63]: RSML, [43]: Statemate, [14]: Esterel, [68]: Argos, [46]: SCR, and [3]: reactive modules.)

Semantic Aspects	Semantic Options	[45]	[86]	[63]	[43]	[14]	[68]	[46]	[3]
Big-Step Maximality	Syntactic					✓			
	Take One	✓	✓				✓	✓	✓
	Take Many			✓	✓				
Concurrency	Single	✓	✓	✓	✓			✓	
	Many					✓	✓		✓
Small-Step Consistency	Source/Destination Orthogonal								
	Arena Orthogonal					✓	✓		✓
Preemption	Non-Preemptive					✓	✓		
	Preemptive								
(Internal) Event Lifeline	Present in Whole					✓	✓		
	Present in Remainder	✓	✓						
	Present in Next Combo Step			✓	✓				
	Present in Next Small Step								
	Present in Same								
Environmental Input Events	Syntactic Input Events			✓		✓	✓		
	Received Events as Env.	✓	✓		✓				
	Hybrid Input Event								
(Interface) Event Lifeline	Strong Synchronous Event								
	Weak Synchronous Event								
	Asynchronous Event					✓			
(Internal Variables) Enabledness Memory Protocol	GC/RHS Big Step							✓	✓
	GC/RHS Combo Step				✓				
	GC/RHS Small Step		✓	✓		✓		✓	
(Interface Variables) Memory Protocol	GC/RHS Strong Synch. Variable								✓
	GC/RHS Weak Synch. Variable								
	GC/RHS Asynch. Variable								
Combo-Step Maximality	Combo Syntactic								
	Combo Take One			✓	✓				
	Combo Take Many								
Order of Small Steps	None	✓		✓	✓	✓	✓		
	Explicit Ordering								
	Dataflow		✓					✓	✓
Priority	Hierarchical				✓				
	Explicit Priority								
	Negation of Triggers		✓	✓	✓	✓	✓	✓	✓

3.12 Related Work: Semantic Categorization and Comparison

Compared to the related work, my deconstruction in this chapter covers a more comprehensive class of BSMLs and range of BSML semantics. Relative to previous comparative studies of different subsets of BSMLs (e.g., statecharts variants [99, 50], Synchronous languages [40], Esterel variants [16, 93], and UML StateMachines [92]), my deconstruction isolates the essential semantic aspects in a language-independent manner and in terms of the big step as a whole.

Parts of my big-step semantic deconstruction overlap with the seminal comparison of statecharts variants by von der Beeck [99]. The difference here is that: (i) I consider a broader range of languages, in addition to statecharts; (ii) the big-step semantic deconstruction presents a decomposition of BSML semantics into semantic aspects and their corresponding semantic options that lend itself to formalization, as opposed to the comparison criteria in [99], which does not have a similar structure: it consists of a mixture of syntactic, semantic, and semantic-definition method criteria; and (iii) the big-step semantic deconstruction considers additional semantic concepts that do not have counterparts in [99]. For example, the Small-Step Consistency semantic aspect, the Present in Next Small Step and the Present in Next Combo Step semantic option of the Event Lifeline semantic aspect, the enumeration of the semantic options of the Enabledness Memory Protocol and Assignment Memory Protocol semantic aspects, etc. are not considered in [99].

Maggiolo-Schettini, Adriano Peron and Simone Tini consider three semantic variations of events according to the semantics of statecharts by Pnueli and Shalev [86], Maggiolo-Schettini, Adriano Peron and Simone Tini [66], and Philips and Scholz [82] in a structural operational semantics (SOS) framework [67]. They consider a common syntax, in the form of *statecharts terms*, adopted from the syntax proposed by Uselton and Smolka [95]. This syntax resembles the compositional syntax of process algebras; it does not support variables and does not allow interrupt transitions whose source and destinations have different parents: The source and destination control states of a transition "must be siblings in an or-state."[95] This common syntax enables them to study and compare each of these event semantics based on how different simulation relations in process algebras, e.g., ready simulation [9], are or are not a congruent with respect to composition operators in the syntax. In this dissertation, the aforementioned event semantics are considered in Section 3.4, in the context of our normal-form syntax, which is more expressive than the one in [67]. Our normal-form syntax, however, does not lend itself to the analysis of the kind of congruence and pre-congruence properties considered in [67], because it is not compositional.

Huizing and Gerth [50] compare simple BSMLs that have only events, covering most of the event lifeline semantic options and the observability of events among components. My deconstruction describes these options more concisely and places them in the context of other semantics aspects for BSMLs.

Comparable to the *parameters* and *parameter values* in template semantics [75, 76, 74], here I have introduced semantic aspects and semantic options. Template semantics is aimed at implementation parameters that all describe variations of small steps semantics. Because I consider a big step as a whole, my semantic deconstruction is presented at a higher level of abstraction, with more understandable variation points; i.e., eight semantic aspects vs. 22 parameters in template semantics. Furthermore, additional semantic aspects, such as the Concurrency and Consistency, External Input Events, and the Combo-Step Maximality semantic aspects, are considered here, which are not present in template semantics. Also, for some common semantic aspects, additional semantic options are introduced; e.g., in the Enabledness Memory Protocol, the new semantic option Present in Next Combo Step is introduced.

3.13 Summary

This chapter presented the big-step semantic deconstruction framework for the family of BSMLs. The framework consists of eight semantics aspects and an enumeration of the common semantic options of each of the semantic aspects. A BSML semantics is described in this framework by enumerating its semantic options. Table 3.12, on page 83, states the semantic variations of a set of common BSMLs in my framework. Furthermore, the chapter presented an analysis of the comparative advantages and disadvantages of the semantics options of each semantic aspect, together with many example models that describe each semantic option. Some combinations of the semantic options create new BSML semantics not found in the literature. Lastly, the chapter enumerated a few combinations of the semantic options that create BSML semantics that suffer from side effects. These side effects can make the specification and comprehension of a BSML semantics difficult.

Chapter 4

Semantic Formalization

“Frege ridiculed the formalist conception of mathematics by saying that the formalists confused the unimportant thing, the sign, with the important, the meaning.” . . . “Frege’s idea could be expressed thus: the propositions of mathematics if they were just complexes of dashes, would be dead and utterly uninteresting, where as they obviously have a kind of life.” . . . “But if we had to name anything which is the life of the sign, we should have to say it was its *use*.” [100, p.89–90]

Ludwig Wittgenstein

This chapter presents a formal, operational *semantic definition schema* for my big-step semantic deconstruction framework for BSMLs. This schema is parametric with respect to the BSML semantic aspects. A particular BSML semantic definition is obtained by instantiating the parameters of the semantic definition schema. As such, a semantic definition produced in this framework is “prescriptive” [5, 4]: The formalization of the semantic options of a semantics can be traced clearly in its semantic definition.

The formalization of the parameters covers most of the semantic options described in Chapter 3. The few semantic variations that are not covered are transition-aware semantics. These convolute the role of structural and enabledness semantic aspects. The difficulties of formalizing these semantics prescriptively are discussed.

The remainder of this chapter is organized as follows. Section 4.1 introduces the parametric semantic definition schema together with the semantic notation that are used to formalize the

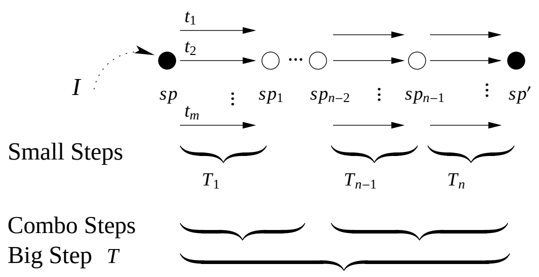


Figure 4.1: Steps.

big-step semantic deconstruction. Section 4.2 presents the syntactic notation that formalizes the syntactic concepts presented in Chapter 3. Section 4.3 formally describes how a BSML model moves from one configuration, i.e., one set of control states, to another, upon the execution of a small step. This semantics is common to all BSMLs. Sections 4.4 and 4.5 present the formalization of the structural and enabledness parameters of the semantic definition schema, respectively. Section 4.6 considers the related work.

4.1 Overview of Semantic Definition Schema

This section presents an overview of my semantic definition method by describing my semantic definition schema and its parameters. The values of the parameters are described in the subsequent sections.

As depicted in Figure 4.1, copied here from page 24 for convenience, a big step is an alternating sequence of snapshots and small steps in reaction to an environmental input. An environmental input, which is typically denoted by I, as in Figure 4.1, consists of a set of environmental input events and a set of variable assignments.

Figure 4.2 shows the parametric *semantic definition schema* that is used to define the semantics of a BSML. The highest level predicate of the semantic definition schema is N_{Big}, in line 1; it characterizes all of the big steps of a model. Predicate N_{Big} is a ternary relation consisting of tuples each of which is a big step of the model: snapshot sp^0 is the source snapshot of the big step; I is an environmental input; and snapshot sp is the destination snapshot of the big step, after a sequence of small steps are executed. A snapshot of a BSML model consists of a set of snapshot elements that together represent a moment in the execution of the model. A snapshot

$$\begin{array}{lcl}
1.\ N_{Big}(sp^0, I, sp') & \equiv & reset(sp^0, I, sp) \wedge (\exists k \geq 0 \cdot N^k(sp, sp')) \\
 & & \wedge\ executable(root, sp') = \emptyset \\
2.\ reset(sp^0, I, sp) & \equiv & \bigwedge\limits_{1 \leq i \leq n} \texttt{reset\_}el_i(sp^0, I, sp) \\
3.\ N^0(sp, sp') & \equiv & sp = sp' \\
4.\ N^{k+1}(sp, sp') & \equiv & \exists \tau, sp'' \cdot N_{Small}(sp, \tau, sp'') \wedge N^k(sp'', sp') \\
5.\ N_{Small}(sp, \tau, sp') & \equiv & \bigwedge\limits_{1 \leq i \leq n} \texttt{next\_}el_i(sp, \tau, sp') \wedge \tau \in executable(root, sp)
\end{array}$$

Figure 4.2: Semantic definition schema.

element is used to model an enabledness semantic aspect of the BSML. It is defined via its type, which permits to create a set of elements of that type, and three predicates that specify how it changes. The `reset` predicate specifies the effect of receiving an environmental input on the snapshot element; the `en` predicate specifies whether a transition is enabled with respect to the value of snapshot element in a certain snapshot; and the `next` predicate specifies how the value of snapshot element changes when a small step is executed. The snapshot elements that are used in the formalization of the semantics of a BSML depend on the enabledness semantic options that the BSML uses. Two BSMLs that subscribe to the same enabledness semantic option use the same corresponding snapshot elements for the semantic option. The formal definition of snapshots and snapshot elements are presented in Section 4.1.1.

In Figure 4.2, when an environmental input I is received at snapshot sp^0 (via predicate "*reset*" on line 2), k small steps are executed (via predicate N on lines 3 and 4), and the big step concludes at snapshot sp', when there are no further small steps to be taken, i.e., when $executable(root, sp') = \emptyset$. The term "$executable(root, sp)$" specifies the set of *potential small steps* of the model at snapshot sp that each can be *executed* as the next small step. The value of "$executable(root, sp)$" not only depends on the `en` predicates of the snapshot elements of the BSML, but also the structural semantic aspects of the BSML. The *reset* predicate in line 2, is the conjunction of the `reset` predicates of the snapshot elements of the BSML semantics; it specifies the effect of receiving the environmental input I for the set of n snapshot elements of the BSML. Line 5 specifies the operation of a small step through the predicate N_{small}. Predicate N_{small} is the conjunction of the `next` predicates of the snapshot elements of the BSML semantics. The effect of executing a small step is captured in the destination snapshot of the small step. The

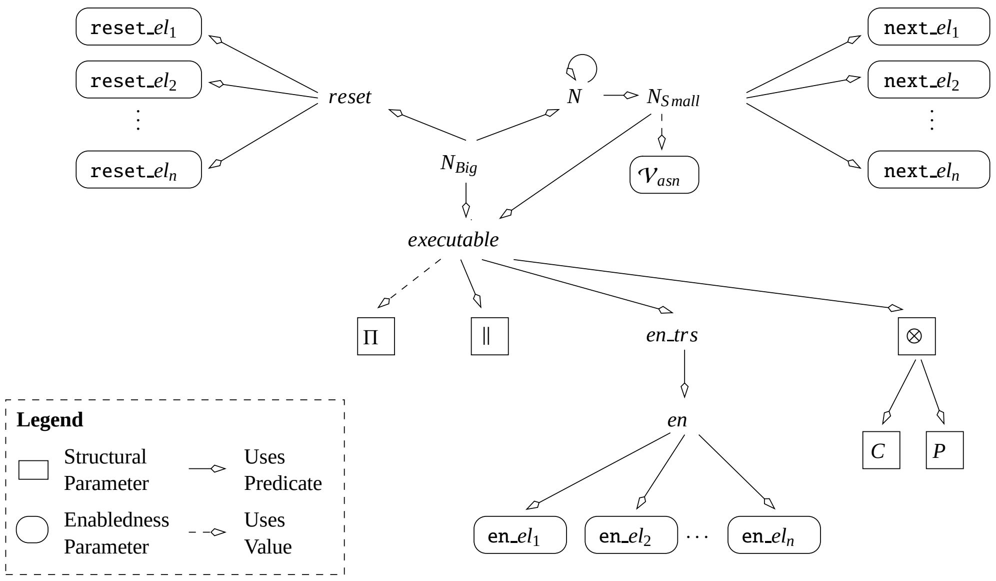


Figure 4.3: The structure of the semantic definition schema.

N_{small} predicates are chained together via the N relation to create a sequence of small steps, as shown in lines 3 and 4.

Figure 4.3 depicts the constituent predicates of the semantic definition schema in Figure 4.2. The predicates that are parameters, i.e., their definitions vary in different BSML semantics, are surrounded by a box. A rounded box corresponds to an *enabledness parameter*, which in turn corresponds to an enabledness semantic aspect. A solid box represents a *structural parameter*, which in turn corresponds to a structural semantic aspect. (Recall that semantic aspects are partitioned into structural and enabledness semantic aspects, as shown in Figure 3.2, on page 32.) A value for an enabledness parameter is a set of snapshot elements, each of which is characterized by a type, and a `reset`, an `en`, and a `next` predicate. In Figure 4.3, n snapshot elements, namely, el_1, el_2, $\cdots$, el_n, are shown together with their corresponding predicates. A value for a structural parameter is one of the following two kinds of values: (i) either it is a predicate that determines which enabled transitions can be included together in the same small step, in the case of the formalization of the Concurrency and Consistency semantic aspect; (ii) or it specifies a certain way that the hierarchy tree of a model should be traversed when creating a small step, in the case of the formalization of the Priority semantic aspect.

In Figure 4.3, a solid arrow specifies that the predicate in the source of the arrow uses the predicate in the destination of the arrow (the destination of a solid arrow must be a predicate). A dashed arrow is different from a solid arrow in that the destination of a dashed arrow is not a predicate: In the case of dynamic parameter "$\mathcal{V}_{asn}$", the parameter refers to an already-defined snapshot element that maintains the up-to-date values of the variables of a model (one of el_i's, $1 \leq i \leq n$, snapshot elements); in the case of structural parameter "Π", the parameter refers to the name of the attribute grammar that is used to compute the potential small steps.

Lastly, in Figure 4.3, function en_trs returns the set of enabled transitions in a set of given transitions; it uses predicate en, which determines whether a single transition is enabled or not. These functions are described in Section 4.1.2.

4.1.1 Snapshots and Snapshot Elements

This section presents some notation for defining and accessing snapshots and snapshot elements. A snapshot of a model is a valuation of the snapshot elements of the model. A BSML semantics uses a set of snapshot elements that are determined by the constituent enabledness semantic options of the BSML. If two BSMLs subscribe to the same semantic option of an enabledness semantic aspect, then they use the same snapshot elements to formalize that semantic option. This approach is as opposed to template semantics [75], where different semantic options could use the same "snapshot element", but with different parameters; cf., Section 4.6 for more detail. Each snapshot element represents an aspect of the behaviour of a model. For example, a BSML that uses variables has a snapshot element that keeps track of the values of variables by maintaining a set of tuples, each of which consisting of a variable name and its current value.

The following conventions are used in the formalization of snapshots and snapshot elements. The identifier sp itself, or sp with a superscript, is used as the name of a snapshot; e.g., sp and sp'. The name of a snapshot element always uses a subscript. To access a snapshot element in a snapshot, the snapshot element name is annotated with the superscript of the snapshot; e.g., S_c and S'_c access the snapshot element S_c in snapshot sp and snapshot sp', respectively.

A snapshot element el_i is characterized by its type, and three predicates:

i $\texttt{reset\_}el_i(sp^0, I, sp)$, which specifies how the value of el_i changes at the beginning of a big step, at a snapshot, sp^0, when an environmental input, I, is received, to result in snapshot sp;

ii $\texttt{next\_}el_i(sp, \tau, sp')$, which specifies how the value of el_i at a snapshot, sp, is changed when an small step, τ, is executed, to result in snapshot sp'; and

iii $\texttt{en\_}el_i(t, sp)$, which specifies the role of el_i in determining a transition, t, as enabled at a snapshot, sp.

The set of all snapshot elements that are used by a BSML semantics is denoted by $SpEl = \{el_1, el_2, \cdots, el_n\}$.

In the above predicates for snapshot element el_i, snapshots sp^0, sp, and sp' can be replaced by the snapshot elements that each predicate needs at each of these snapshots. However, I chose to pass an entire snapshot to the predicates to achieve a uniformity in dealing with the predicates of different snapshot elements; e.g., when conjoining the $\texttt{en}$ predicates of a set of snapshot elements.

The *replace operator*, $\oplus$, replaces the value of snapshot element(s) in a snapshot with a new value of the same type. For example, $sp' \oplus \{el_1, el_2, \cdots, el_m\}$ replaces el'_1, el'_2, $\cdots$, and el'_m with values el_1, el_2 , $\cdots$, and el_m, respectively.

4.1.2 Enabledness of a Transition

The enabledness of a single transition of a model is determined by the enabledness predicates of the snapshot elements that are used in the semantics of the BSML that the model is specified in. The enabledness of a transition at a snapshot does not guarantee its execution, because, for example, an enabled transition with a higher priority can replace it. The following predicate specifies whether a transition, t, is *enabled* at a snapshot, sp,

$$en(t, sp) \quad \equiv \quad \bigwedge_{1 \le i \le n} \texttt{en\_}el_i(t, sp).$$

Intuitively, transition t is enabled at sp if its source control state is in the set of current control states; its guard condition is satisfied; the events in its trigger that are in positive form are present; the events in its trigger that are in negated form are absent; and all other enabledness criteria relevant for a single transition, such as big-step/combo-step maximality criteria are satisfied.

Table 4.1: Syntactic notation for control states in BSMLs.

Notation	Description
$children(s)$	The set of control states that are *children* of s in the hierarchy tree.
$children^+(s)$	The set of control states that are *descendents* of s in the hierarchy tree either directly or by transitivity.
$children^*(s)$	$children^*(s) = children^+(s) \cup \{s\}$.
$default(s)$	If s is an *Or* control state, $default(s)$ is the *default* control state of s, otherwise it is not defined.
$bigstable(s)$	If s is a stable control state, $bigstable(s)$ is $true$, otherwise it is $false$.
$combostable(s)$	If s is a combo-stable control state, $combostable(s)$ is $true$, otherwise it is $false$.
$lca(s, s')$	The *least common ancestor* of s and s' is the lowest control state in the hierarchy tree such that $s, s' \in children^*(lca(s, s'))$.
$s \perp s'$	Control states s and s' are *orthogonal*, $s \perp s'$, if neither of s and s' is an ancestor of the other and $lca(s, s')$ is an *And* control state.
$overlap(s, s')$	Control states s and s' are *overlapping*, $overlap(s, s')$, if $s \in children^*(s')$ or $s' \in children^*(s)$.

Given a set of transitions, $transitions$, and a snapshot, sp, function en_trs specifies all of the transitions in it that are enabled. Formally,

$$en\_trs(transitions, sp) \quad \equiv \quad \{\, t : transitions \mid en(t, sp) \,\}.$$

4.2 Syntactic Notation

Tables 4.1 and 4.2 present the syntactic functions and relations defined over control states and transitions of a BSML model, respectively. Most of these definitions were discussed informally in Chapter 2. Some of these definitions are adopted from Pnueli and Shalev's work [85, 86].

4.3 The Snapshot Element for Control States

This section presents the formalization of the snapshot element that maintains the current set of control states that a model resides in at each point of execution. This snapshot element is used

Table 4.2: Syntactic notation for transitions in BSMLs.

Notation	Description
$src(t)/dest(t)$	The *source*/*destination* control state of t.
$gc(t)$	The *guard condition* of t, which is a boolean expression over the set of variables of a model.
$asn(t)$	The *set of assignments* of t, which is a set of assignments over the set of variables of a model.
$lhs(a)/rhs(a)$	The *left hand side*/*right hand side* of assignment a.
$trig(t)$	The *trigger* of t, which is a set of events and negations of events.
$pos\_trig(t)$	The set of events used in $trig(t)$ in positive form.
$neg\_trig(t)$	The set of events used in $trig(t)$ in negation form.
$scope(t)$	The *scope* of transition t is the lowest control state such that: $src(t), dest(t) \in children^{+}(scope(t))$
$arena(t)$	The *arena* of transition t is the lowest *Or* control state such that: $src(t), dest(t) \in children^{+}(arena(t))$
$t \perp t'$	Transitions t and t' are *orthogonal*, $t \perp t'$, if $src(t) \perp src(t')$ and $dest(t) \perp dest(t')$.
$t \lightning t'$	Transitions t is an *interrupt for* t', $t \lightning t'$, if $src(t) \perp src(t')$ and: either $(dest(t') \perp src(t)) \wedge (dest(t) \not\perp src(t')) \wedge (dest(t) \not\perp src(t))$, meaning that t exits the scope of t', as shown in Figure 2.2(a), on page 20; or $(dest(t') \not\perp src(t)) \wedge (dest(t) \not\perp src(t')) \wedge (dest(t) \not\perp src(t)) \wedge (dest(t') \not\perp src(t')) \wedge (dest(t) \in children^{+}(dest(t'))$, meaning that the source control states of t and t' have different *And* ancestors than their destination control states, while the destination control state of t is a descendant of the destination control state of t', as shown in Figure 2.2(b), on page 20.
$interrupted(\tau)$	For a set of transitions τ, its set of *interrupted transitions*, $interrupt(\tau) \subset \tau$, is defined as: $interrupted(\tau) = \{ t' \in \tau \mid \exists t \in \tau \cdot t \lightning t'\}$. If τ is the set of transitions of a small step, then the destination control states of the transitions in $interrupted(\tau)$ do not have any roles in determining the control states that the model would reside after executing τ.

by all BSML semantics.

Snapshot element S_c, specified below, maintains the current control states that a model currently resides in. The type of S_c is a set of control states. A BSML model uses at least one control state, thus all BSMLs use the snapshot element S_c. Initially, a BSML model resides in the default control state of its root control state. Thus S_c is initially populated with $default(root)$ and those descendant control states of it such that if the model resides in an *Or* control state, it resides in exactly one of its children, which is by default its "default" child; and if the model resides in an *And* control state, it resides in all of its children.

Next, first, snapshot element S_c is formally described, followed by a discussion about the correctness of its formalization.

$$\begin{array}{lcl}
\texttt{reset\_}S_c(sp^0, I, sp) & \equiv & S_c = sp^0.S_c \\
\texttt{next\_}S_c(sp, \tau, sp') & \equiv & S_c' = [S_c - (exited(\tau, S_c) \cup pot\_entering(\tau))] \cup \\
 & & \quad entered(\tau - interrupted(\tau)) \\
\texttt{en\_}S_c(t, sp) & \equiv & src(t) \in S_c
\end{array}$$

In predicate $\texttt{reset\_}S_c(sp^0, I, sp)$, snapshot sp^0 is the snapshot at which environmental input I is received, while snapshot sp is the snapshot that captures the effect of receiving I. In predicate $\texttt{next\_}S_c(sp, \tau, sp')$, snapshot sp and sp' are the source snapshot and the destination snapshot of small step τ; snapshot sp' captures the effect of executing τ. In predicate $\texttt{en\_}S_c(t, sp)$, snapshot sp is the snapshot that the enabledness of a transition is evaluated against to determine whether its source control state belongs to the current configuration, allowing the transition to be included in a potential small step.

Predicate $\texttt{next\_}S_c(sp, \tau, sp')$ uses functions *exited*, *pot_entering*, *entered* to determine the set of control states that small step τ *exits*, *could potentially enter*, and *enters* upon its execution, respectively. To define these functions, first, auxiliary functions $ss(t)$ and $ds(t)$ need to be defined.

The *source scope* of a transition, t, denoted by $ss(t)$, specifies the highest control state that t exits upon execution. Formally,

$$ss(t) = \begin{cases} \text{If } src(t) \in children^+(dest(t)) & ss(t) = dest(t), \\ \text{Else If } dest(t) \in children^*(src(t)) & ss(t) = src(t), \\ \text{Else} & ss(t) = s, \text{ such that } s \text{ is the highest control} \\ & \text{state such that } src(t) \in children^*(s) \text{ and} \\ & dest(t) \notin children^*(s) \end{cases}$$

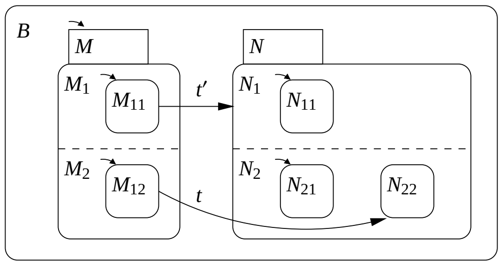


Figure 4.4: A model with interrupting transitions.

Similarly, the *destination scope* of t, denoted by $ds(t)$, specifies the highest control state that t enters upon execution. Formally,

$$ds(t) \quad = \quad \begin{cases} \text{If } dest(t) \in children^{+}(src(t)) & ds(t) = src(t), \\ \text{Else If } src(t) \in children^{*}(dest(t)) & ds(t) = dest(t), \\ \text{Else} & ds(t) = s, \text{ such that } s \text{ is the highest control} \\ & \text{state such that } dest(t) \in children^{*}(s) \text{ and} \\ & src(t) \notin children^{*}(s) \end{cases}$$

The *set of exited control states* of transition t at a snapshot, sp, is then defined as $exited(t, S_c) = children^{*}(ss(t)) \cap S_c$, which specifies the set of control states that the model exits upon the execution of t at snapshot sp. For a set of transitions, τ, $exited(\tau, S_c)$ denotes $\bigcup_{t \in \tau} exited(t, S_c)$.

Example 25 *In the model in Figure 4.4,*

$$\begin{array}{lcl} ss(\mathrm{t}) & = & \{\mathrm{M}\}, \\ ss(\mathrm{t}') & = & \{\mathrm{M}\}, \\ exited(\mathrm{t}, \{\mathrm{M}, \mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_{11}, \mathrm{M}_{12}\}) & = & \{\mathrm{M}, \mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_{11}, \mathrm{M}_{12}\}, \\ exited(\mathrm{t}', \{\mathrm{M}, \mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_{11}, \mathrm{M}_{12}\}) & = & \{\mathrm{M}, \mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_{11}, \mathrm{M}_{12}\}, \textit{ and} \\ exited(\{\mathrm{t}, \mathrm{t}'\}, \{\mathrm{M}, \mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_{11}, \mathrm{M}_{12}\}) & = & \{\mathrm{M}, \mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_{11}, \mathrm{M}_{12}\}. \end{array}$$

The *set of potentially entering control states* of a transition, t, is defined as $pot\_entering(t) = children^{*}(ds(t))$, and specifies the set of control states that the model might enter; this set is computed independently of snapshot sp.

The *set of entered control states* of a transition, t, denoted by $entered(t)$, specifies the set of control states that the model enters upon the execution of transition t. The computation of this set, however, depends on the value of the current snapshot sp.

The set of control states $entered(t)$ is defined through the following two conditions. A control state s belongs to $entered(t)$ if one of the following conditions holds.

Condition 1: This condition deals with the case when the destination control state of transition t is nested in a compound control state $ds(t)$.

A control state s belongs to $entered(t)$ if $s \in pot\_entering(t)$ and one of the following three conditions holds,

i $dest(t) \in children^*(s)$; or
[Ancestors of $dest(t)$ that belong to $ds(t)$ also belong to $entered(t)$.]

ii there exists a control state $s' \in (entered(t) \cap pot\_entering(t))$ such that,

(a) either s' is an *And* control state and $s \in children(s')$, or s' is an *Or* control state and $s = default(s')$, and

(b) $lca(s, dest(t))$ is not an *Or* control state; or

[There is already an s' in $entered(t)$, thus the appropriate children of s' that none of them is a descendant of $dest(t)$ should also belong to $entered(t)$ so that the model enters a consistent set of control states.]

iii there exists a control state $s' \in (entered(t) \cap pot\_entering(t))$ such that,

(a) either s' is an *And* control state and $s \in children(s')$, or s' is an *Or* control state and $s = default(s')$, and

(b) $s' \in children^*(dest(t))$;

[There is already an s' in $entered(t)$, thus the appropriate children of s' that each is either $dest(t)$ or a descendent of $dest(t)$ should also belong to $entered(t)$ so that the model enters a consistent set of control states.]

Condition 2: This condition deals with the cases when $scope(t)$ is an *And* control state, which requires the control states in $ss(t)$ to be not only exited but also entered.

A control state s belongs to $entered(t)$, if $scope(t)$ is an *And* control state, $s \in C$, where $C = children^*(ss(t))$, and one of the following two conditions holds,

i s is the highest control state in $ss(t)$; or
[Such an s is included as part of the next configuration because the execution of t does not leave the And control state $scope(t)$*.]*

ii there exists a control state $s' \in (entered(t) \cap C)$ such that either s' is an *And* control state and $s \in children(s')$, or s' is an *Or* control state and $s = default(s')$.
[Since $ss(t)$ *is included in* $entered(t)$ *so its appropriate children should be so that the model enters a consistent set of control states.]*

For a set of transitions, τ, $pot\_entering(\tau)$ and $entered(\tau)$ denote $\bigcup_{t\in\tau} pot\_entering(t)$ and $\bigcup_{t\in\tau} entered(t)$, respectively.

Example 26 *In the model in Figure 4.4,*

$$\begin{array}{lcl}
ds(\mathrm{t}) & = & \{\mathrm{N}\}, \\
ds(\mathrm{t}') & = & \{\mathrm{N}\}, \\
pot\_entering(\mathrm{t}) & = & \{\mathrm{N}, \mathrm{N}_1, \mathrm{N}_2, \mathrm{N}_{11}, \mathrm{N}_{21}, \mathrm{N}_{22}\}, \\
pot\_entering(\mathrm{t}') & = & \{\mathrm{N}, \mathrm{N}_1, \mathrm{N}_2, \mathrm{N}_{11}, \mathrm{N}_{21}, \mathrm{N}_{22}\}, \\
entered(\mathrm{t}) & = & \{\mathrm{N}, \mathrm{N}_1, \mathrm{N}_2, \mathrm{N}_{11}, \mathrm{N}_{22}\}, \textit{ and} \\
entered(\mathrm{t}') & = & \{\mathrm{N}, \mathrm{N}_1, \mathrm{N}_2, \mathrm{N}_{11}, \mathrm{N}_{21}\}.
\end{array}$$

If the configuration where $S_c = \{\mathrm{B}, \mathrm{M}, \mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_{11}, \mathrm{M}_{12}\}$ *is considered, when transition* t, *which is an interrupt for transition* t′*, is executed together with transition* t′*, the new value of snapshot element* S_c *is computed as follows, according to the definition of the* $\mathtt{next\_}S_c$ *parameter:*

$$\begin{array}{rcl}
S'_c & = & S_c - [exited(\{\mathrm{t}, \mathrm{t}'\}, \mathrm{S_c}) \cup pot\_entering(\{\mathrm{t}, \mathrm{t}'\})] \cup entered(\{\mathrm{t}, \mathrm{t}'\} - \{\mathrm{t}'\})] \\
& = & \{\mathrm{B}, \mathrm{M}, \mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_{11}, \mathrm{M}_{12}\} - [\{\mathrm{M}, \mathrm{M}_1, \mathrm{M}_2, \mathrm{M}_{11}, \mathrm{M}_{12}\} \cup \{\mathrm{N}, \mathrm{N}_1, \mathrm{N}_2, \mathrm{N}_{11}, \mathrm{N}_{21}, \mathrm{N}_{22}\}] \cup \\
& & \{\mathrm{N}, \mathrm{N}_1, \mathrm{N}_2, \mathrm{N}_{11}, \mathrm{N}_{22}\} \\
& = & \{\mathrm{B}, \mathrm{N}, \mathrm{N}_1, \mathrm{N}_2, \mathrm{N}_{11}, \mathrm{N}_{22}\}.
\end{array}$$

The next example demonstrates a model in which condition 2 of the *entered* function, which deals with the *entered* function for a transition whose scope is an *And* control state, is used. When the scope of a transition, t, is an *And* control state, the control states in $ss(t)$ are not only exited, but also, are entered, so that the default control states of the *Or* control states in $ss(t)$ are entered. Furthermore, the set of potential entering control states $pot\_entering(t)$ needs to be removed first, because t may enter a child of an *Or* control state other than the one that it currently resides in.

Example 27 *If the model in Figure 4.5, which is copied from the model in Figure 3.8 on page 99, resides in configuration,* $S_c = \{\text{Counter}, \text{Bit}_1, \text{Bit}_2, \text{Status}, \text{Bit}_{11}, \text{Bit}_{22}, \text{Counting}\}$*, then*

$$\begin{array}{lcl} ss(\text{t}_4) & = & \{\text{Bit}_2\}, \\ ds(\text{t}_4) & = & \{\text{Status}\}, \\ exited(\text{t}_4, sp) & = & \{\text{Bit}_2, \text{Bit}_{22}\}, \\ pot\_entering(\text{t}_4) & = & \{\text{Status}, \text{Counting}, \text{Max}\}, \; \textit{and} \\ entered(\text{t}_4) & = & \{\text{Status}, \text{Max}, \text{Bit}_2, \text{Bit}_{21}\}. \end{array}$$

Executing t_4 *would then result in a new value for* S_c*:*

$$\begin{array}{rcl} S_c' & = & S_c - [exited(\{\text{t}_4\}, S_c) \cup pot\_entering(\{\text{t}_4\})] \cup entered(\{\text{t}_4\}) \\ & = & \{\text{Counter}, \text{Bit}_1, \text{Bit}_2, \text{Status}, \text{Bit}_{11}, \text{Bit}_{22}, \text{Counting}\} - \\ & & [\{\text{Bit}_2, \text{Bit}_{22}\} \cup \{\text{Status}, \text{Counting}, \text{Max}\}] \cup \\ & & \{\text{Status}, \text{Max}, \text{Bit}_2, \text{Bit}_{21}\} \\ & = & \{\text{Counter}, \text{Bit}_1, \text{Bit}_2, \text{Status}, \text{Bit}_{11}, \text{Bit}_{21}, \text{Max}\}. \end{array}$$

Proposition 4.1 *For any BSML model, at any of its snapshots, the set of control states in snapshot element* S_c *always includes a* valid *set of control states, where a valid set of control states is defined as a set that: includes the root control state of the model; if an And control state belongs to the set, then all of its children belong to the set; and if an Or control state belongs to the set, then exactly one of its children belongs to the set.*

Proof Idea. When the model is in its initial state, the above claim holds by the definition of the initialization of a BSML model.

The root control state always belongs to S_c because by the definition of the `next`$\_S_c$ relation, copied below for convenience,

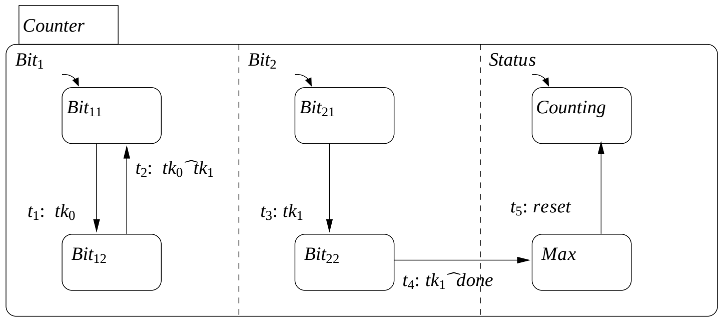


Figure 4.5: The revised two-bit counter, copied from page 43.

$$\texttt{next\_}S_c(sp, \tau, sp') \equiv S_c' = S_c - [exited(\tau, S_c) \cup pot\_entering(\tau)] \cup entered(\tau - interrupted(\tau)),$$

the root control state can be removed from S_c only if there is a small step, τ, such that the set of control states "$exited(\tau, S_c) \cup pot\_entering(\tau)$" includes the root control state; however, this is a contradiction because it is not possible for the root control state to be entered or exited.

Also, by the definition of the $\texttt{next\_}S_c$ relation, if an *And* or an *Or* control state is removed from S_c, all its children are also removed because of the definitions of the *exited* and *pot_entering* functions. Similarly, if an *And* control state is added to S_c, all its children are also added to S_c because of the items iia, iiia, and ii in the definition of function *entered*; and if an *Or* control is added to S_c, exactly one of its children is added to S_c, again, because of the items iia, iiia, and ii in the definition of function *entered*, and also because of item that would add the destination of a transition, t, to S_c, if $dest(t)$ is a child of an *Or* control state. Therefore, since an *And* or an *Or* control state is added consistently to S_c, S_c always consists of a set of valid control states. □

4.4 Structural Parameters

This section presents the structural parameters and their possible values. Each of the structural semantic (sub)aspects Concurrency, Small-Step Consistency, Preemption, and Priority corresponds to a structural parameter, which affects the definition of predicate $executable(root, sp)$ in the semantic definition schema in Figure 4.2. Figure 4.6 is the same as the feature diagram in Figure 3.2, on page 32, except that it excludes the Priority semantic options that are not included in my formalization, as described in Section 3.8. The excluded priority semantics use Source or Destination of control states as the basis of a priority semantics, instead of Scope.

This section is organized into three subsections that each describes one of the hierarchical Priority semantics. In my semantics formalization, the choice of the hierarchical priority semantics in a BSML affects the parsing mechanism used for formalizing the semantics of the BSML. As such, the structural parameter for the Priority semantic aspect consists in: (i) determining the parsing mechanism of a BSML, and (ii) specifying the mechanism by which the set of potential small steps of a model are computed. Section 4.4.4 discusses how the formal semantics presented in this chapter can be generalized to include other priority semantic options.

The BNF in Figure 4.7 represents an *abstract syntax* [69] for BSMLs.This syntax is different from the BNF in Figure 2.3, on page 21, in that it does not include all the derivations rules there, and furthermore, it associates the transitions of a model with their scopes, as opposed to having a separate set of derivation rules for transitions, as the BNF in Figure 2.3. Thus, this representation of BSML syntax is suitable for the specification of the hierarchical priority semantic options that are based on the scope of transitions: i.e., the Scope-Child and the Scope-Parent semantic options.

4.4.1 Scope-Parent Priority Semantics

Figure 4.8 shows an attribute-grammar–like formalism that computes the set of potential small steps of a model at snapshot sp according to the Scope-Parent in attribute $\mathbf{ex_P}(\text{root}, sp)$; i.e., the value of *executable*(*root*, *sp*) in the semantic definition schema in Figure 4.2 is the value of $\mathbf{ex_P}(\text{root}, sp)$ in the hierarchical computation in Figure 4.8. The structural parameter Π, shown in Figure 4.3, denotes the name of the attribute in the attribute grammar whose value in the root control state is the set of potential small steps; in case of the Scope-Parent priority semantics, the value of Π is $\mathbf{ex_P}$.

In the specification of the attributes in Figure 4.8, if a non-terminal symbol is used in both sides of a rule, such as in rule 3, I use the subscripts “0” and “1” to refer to the instance of the symbol on the left-hand–side and the instance of the symbol on the right-hand–side of the rule, respectively. For example, in line 3c, “states-o_1” refers to the right-hand–side of the rule whereas “states-o_0” refers to the left-hand–side of the rule. Therefore, line 3c means that the value of the $\mathbf{ex_P}$ attribute for the non-terminal in the left-hand–side is equal to the value of the $\mathbf{ex_P}$ attribute

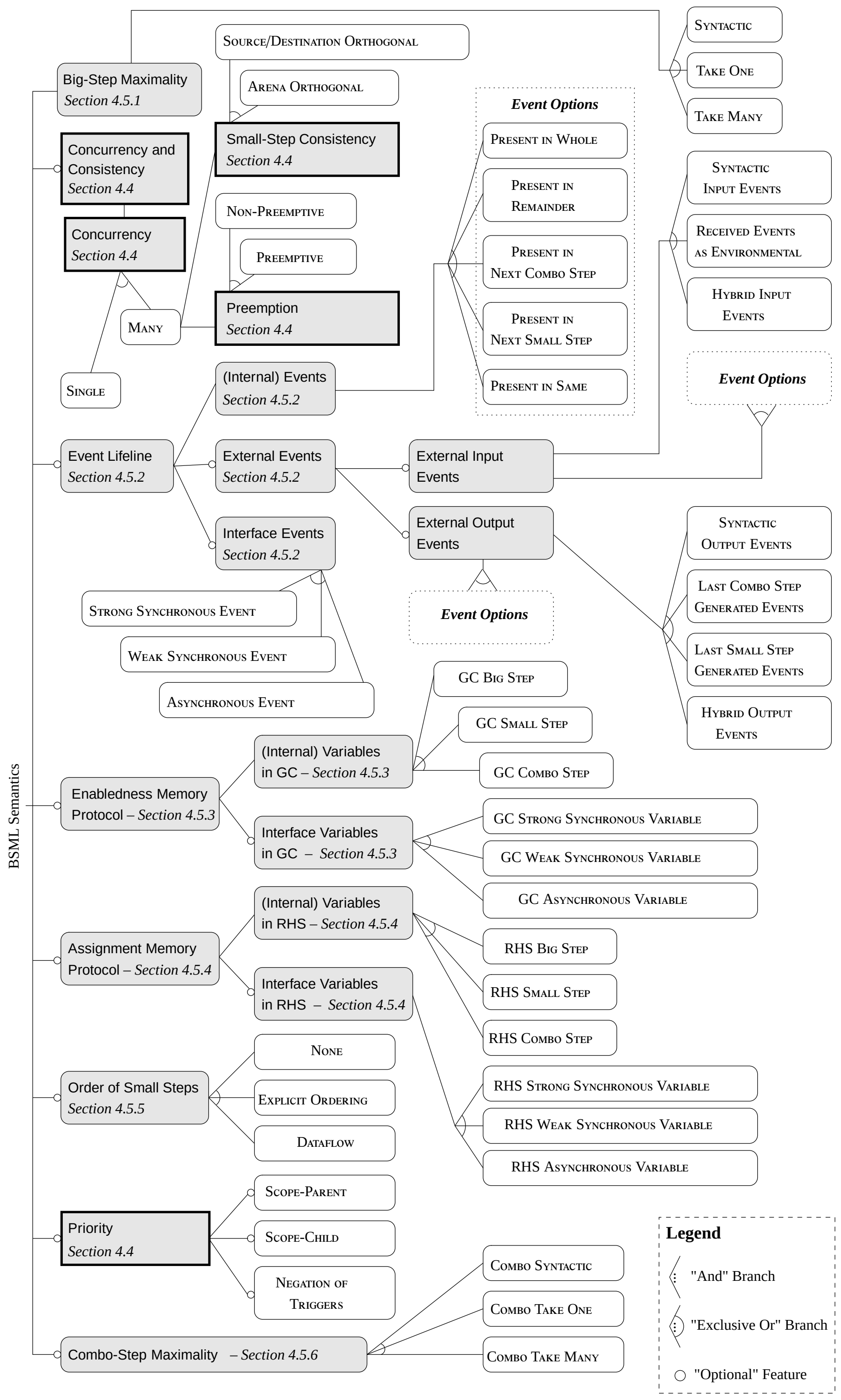

BSML Semantics
Big-Step Maximality Section 4.5.1
Syntactic
Take One
Take Many
Source/Destination Orthogonal
Arena Orthogonal
Small-Step Consistency Section 4.4
Concurrency and Consistency Section 4.4
Concurrency Section 4.4
Many
Single
Non-Preemptive
Preemptive
Preemption Section 4.4
Event Options
Present in Whole
Present in Remainder
Present in Next Combo Step
Present in Next Small Step
Present in Same
Syntactic Input Events
Received Events as Environmental
Hybrid Input Events
Event Options
(Internal) Events Section 4.5.2
Event Lifeline Section 4.5.2
External Events Section 4.5.2
External Input Events
External Output Events
Interface Events Section 4.5.2
Event Options
Syntactic Output Events
Last Combo Step Generated Events
Last Small Step Generated Events
Hybrid Output Events
Strong Synchronous Event
Weak Synchronous Event
Asynchronous Event
GC Big Step
GC Small Step
GC Combo Step
(Internal) Variables in GC – Section 4.5.3
Enabledness Memory Protocol – Section 4.5.3
Interface Variables in GC – Section 4.5.3
GC Strong Synchronous Variable
GC Weak Synchronous Variable
GC Asynchronous Variable
(Internal) Variables in RHS – Section 4.5.4
Assignment Memory Protocol – Section 4.5.4
Interface Variables in RHS – Section 4.5.4
RHS Big Step
RHS Small Step
RHS Combo Step
RHS Strong Synchronous Variable
RHS Weak Synchronous Variable
RHS Asynchronous Variable
None
Explicit Ordering
Dataflow
Order of Small Steps Section 4.5.5
Scope-Parent
Scope-Child
Negation of Triggers
Priority Section 4.4
Combo-Step Maximality – Section 4.5.6
Combo Syntactic
Combo Take One
Combo Take Many
Legend
"And" Branch
"Exclusive Or" Branch
"Optional" Feature

Figure 4.6: BSML semantic aspects and options: Solid boxes are the structural semantic aspects, while rounded boxes are the enabledness semantic aspects.

⟨root⟩	::=	⟨Orstate⟩
⟨Orstate⟩	::=	**Or** ⟨states-o⟩ ⟨transitions⟩
⟨states-o⟩	::=	⟨states-o⟩ ⟨state⟩ | ⟨state⟩
⟨Andstate⟩	::=	**And** ⟨states-a⟩ ⟨transitions⟩
⟨states-a⟩	::=	⟨states-a⟩ ⟨state⟩ | ⟨state⟩
⟨Basicstate⟩	::=	**Basic**
⟨state⟩	::=	⟨Orstate⟩ | ⟨Andstate⟩ | ⟨Basicstate⟩

Figure 4.7: The abstract syntax for BSML syntax based on the scope of transitions.

for the non-terminal in the right-hand–side unioned by the value of $\mathbf{ex_P}$ attribute for control state state.

The computation in Figure 4.8 uses two classes of attributes, **top** and $\mathbf{ex_P}$, for the non-terminal elements of a BSML model. The **top** attributes are *inherited* attributes, while the $\mathbf{ex_P}$ attributes are *synthesized* attributes [59]. To enforce that a transition with a high scope has a high priority, a control state, through its **top** attribute, passes the possible combinations of the enabled transitions that can be executed from the higher scope in the hierarchy tree to its children control states. The $\mathbf{ex_P}$ attributes collect the set of high-priority transitions in a bottom-up manner, starting from the *Basic* control states, to compute the $\mathbf{ex_P}(\text{root}, sp)$, which is the set of potential small steps of the model at snapshot sp. The computation of both the **top** attributes and the $\mathbf{ex_P}$ attributes follow the structural semantic options of the BSML. Next, the roles of structural parameters and their values in computing the attributes of control states in Figure 4.8 are described.

The binary, *concurrency* operator, $\|$, used in line 6c is responsible for collecting the contribution of the children of an *And* control state to the set of potential small steps, using their corresponding $\mathbf{ex_P}$ attributes. If the SINGLE concurrency semantics is chosen, then the concurrency operator specifies that either its first operand or its second operand can be chosen to be the next small step. If the MANY concurrency semantics is chosen, then the concurrency operator combines the set of sets of transitions in one of its operand with the set of sets of transitions in its other operand to create a new set of sets of transitions. The first part of Table 4.3 is the formalization of the values of the structural parameters $\|$. The operands of the $\|$ operator are written in a special font to denote that each is of type set of sets of transitions.

For the $\|$ operator to work correctly, the computation of the **top** attributes must consider that whether the SINGLE or the MANY concurrency semantics is chosen in a semantics, so that in the latter case only singleton sets of high-priority transitions are passed down the hierarchy

1. $\langle\text{root}\rangle$	::=	$\langle\text{Orstate}\rangle$
a. $\mathbf{top}(\text{Orstate}, sp)$	=	$\emptyset$
b. $\mathbf{ex_P}(\text{root}, sp)$	=	$\mathbf{ex_P}(\text{Orstate}, sp)$
2. $\langle\text{Orstate}\rangle$	::=	**Or** $\langle\text{states-o}\rangle\ \langle\text{transitions}\rangle$
a. $\mathbf{top}(\text{states-o}, sp)$	=	$\mathbf{top}(\text{Orstate}, sp) \otimes en\_trs(\text{transitions}, sp)$
b. $\mathbf{ex_P}(\text{Orstate}, sp)$	=	$\mathbf{ex_P}(\text{states-o}, sp)$
3. $\langle\text{states-o}\rangle$	::=	$\langle\text{states-o}\rangle\ \langle\text{state}\rangle$
a. $\mathbf{top}(\text{states-o}_1, sp)$	=	$\mathbf{top}(\text{states-o}_0, sp)$
b. $\mathbf{top}(\text{state}, sp)$	=	$\mathbf{top}(\text{states-o}_0, sp)$
c. $\mathbf{ex_P}(\text{states-o}_0, sp)$	=	$\mathbf{ex_P}(\text{states-o}_1, sp) \cup \mathbf{ex_P}(\text{state}, sp)$
4. $\langle\text{states-o}\rangle$	::=	$\langle\text{state}\rangle$
a. $\mathbf{top}(\text{state}, sp)$	=	$\mathbf{top}(\text{states-o}, sp)$
b. $\mathbf{ex_P}(\text{states-o}, sp)$	=	$\mathbf{ex_P}(\text{state}, sp)$
5. $\langle\text{Andstate}\rangle$	::=	**And** $\langle\text{states-a}\rangle\ \langle\text{transitions}\rangle$
a. $\mathbf{top}(\text{states-a}, sp)$	=	$\mathbf{top}(\text{Andstate}, sp) \otimes en\_trs(\text{transitions}, sp)$
b. $\mathbf{ex_P}(\text{Andstate}, sp)$	=	$\mathbf{ex_P}(\text{states-a}, sp)$
6. $\langle\text{states-a}\rangle$	::=	$\langle\text{states-a}\rangle\ \langle\text{state}\rangle$
a. $\mathbf{top}(\text{states-a}_1, sp)$	=	$\mathbf{top}(\text{states-a}_0, sp)$
b. $\mathbf{top}(\text{state}, sp)$	=	$\mathbf{top}(\text{states-a}_0, sp)$
c. $\mathbf{ex_P}(\text{states-a}_0, sp)$	=	$\mathbf{ex_P}(\text{states-a}_1, sp) \parallel \mathbf{ex_P}(\text{state}, sp)$
7. $\langle\text{states-a}\rangle$	::=	$\langle\text{state}\rangle$
a. $\mathbf{top}(\text{state}, sp)$	=	$\mathbf{top}(\text{states-a}, sp)$
b. $\mathbf{ex_P}(\text{states-a}, sp)$	=	$\mathbf{ex_P}(\text{state}, sp)$
8. $\langle\text{Basicstate}\rangle$	::=	**Basic**
a. $\mathbf{ex_P}(\text{Basicstate}, sp)$	=	$\mathbf{top}(\text{Basicstate}, sp)$
9. $\langle\text{state}\rangle$	::=	$\langle\text{Orstate}\rangle \mid \langle\text{Andstate}\rangle \mid \langle\text{Basicstate}\rangle$
a. $\mathbf{top}(\text{Orstate}, sp)$	=	$\mathbf{top}(\text{state}, sp)$
b. $\mathbf{top}(\text{Andstate}, sp)$	=	$\mathbf{top}(\text{state}, sp)$
c. $\mathbf{top}(\text{Basicstate}, sp)$	=	$\mathbf{top}(\text{state}, sp)$
d. $\mathbf{ex_P}(\text{state}, sp)$	=	$\mathbf{ex_P}(\text{Orstate}, sp), \mathbf{ex_P}(\text{Andstate}, sp), \mathbf{ex_P}(\text{Basicstate}, sp)$

10. $$\mathbb{T} \otimes T' = \{\, (T_1 \cup T'') \mid T_1 \in \mathbb{T} \wedge T'' \subseteq T' \wedge \\ (\forall t' : (T_1 \cup T') \cdot t' \in (T' - T'') \Leftrightarrow \exists t \in (T_1 \cup T'') \cdot \neg C(t', t) \wedge \neg P(t', t)) \,\}$$

Figure 4.8: Computing potential small steps in the SCOPE-PARENT priority semantics.

Table 4.3: Structural parameters for Concurrency, Small-Step Consistency, and Preemption semantic aspects.

Semantic Option	Parameter Value
Concurrency	
SINGLE	$\mathbb{T} \parallel \mathbb{T}' = \mathbb{T} \cup \mathbb{T}'$
MANY	$\mathbb{T} \parallel \mathbb{T}' = \{T_1 \cup T_1' \mid T_1 \in \mathbb{T} \wedge T_1' \in \mathbb{T}'\}$
Small-Step Consistency [$C(t,t') \equiv false$, when SINGLE concurrency semantics.]	
ARENA ORTHOGONAL	$C(t,t') \equiv arena(t) \perp arena(t')$
SOURCE/DESTINATION ORTHOGONAL	$C(t,t') \equiv t \perp t'$
Preemption [$P(t,t') \equiv false$, when SINGLE concurrency semantics.]	
NON-PREEMPTIVE	$P(t,t') \equiv (t \lightning t') \vee (t' \lightning t)$
PREEMPTIVE	$P(t,t') \equiv false$

tree and in the former case the high-priority transitions are combined as they are passed down the hierarchy tree. The structural parameters, C and P, which correspond to the Small-Step Consistency and the Preemption semantic aspects, respectively, enforce the above semantics. These parameters, formalized in the middle and the bottom parts of Table 4.3, respectively, by being false when the SINGLE concurrency semantics is chosen, ensure that no two transitions are allowed to be combined together.

The *merge* operator, $\otimes$, defined in line 10 of Figure 4.8, and used in lines 2a and 5a uses predicates C and P to combine a set of sets of transitions, denoted by parameter $\mathbb{T}$, with a set of enabled transitions, denoted by T', to compute the **top** attributes of compound control states. Parameter $\mathbb{T}$ is in a special font because its type is set of sets of transitions, as opposed to set of transitions, as T' is. Each set of transitions in $\mathbb{T}$ is combined with a subset of T' to create a new maximal set of transitions. The result is maximal because of the if-only-if predicate in the definition: A transition, t', is not included in the merge result iff there is a transition, t, that is already included in the merge result, and t and t' can neither be included together according to the small-step consistency semantics (parameter C) nor according to the preemption semantics of the BSML (parameter P).

In the bottom-up computation of the $\mathbf{ex_P}$ attributes, there is no need to check for the small-step consistency and preemption semantic constraints because these have already been checked in the top-down traverse. Line 9d uses "," to represent three separate equalities, each of which corresponds to one of the right-hand-side alternatives in line 9.

The formalization of the parameter values for the $\|$, C and P structural parameters follow their English descriptions in Section 3.3.

Example 28 *Figure 4.9 shows a BSML model, with its root control state shown explicitly. If the model resides in snapshot sp, where $S_c = \{\mathrm{root}, \mathrm{M}, \mathrm{A}, \mathrm{A}_1, \mathrm{A}_{11}, \mathrm{A}_2, \mathrm{A}_{21}\}$, and the BSML subscribes to the* MANY, SOURCE/DESTINATION ORTHOGONAL, *and* NON-PREEMPTIVE *concurrency and consistency semantics, together with the* SCOPE-PARENT *priority semantics, then Table 4.4 shows the values of $\mathbf{ex_P}$ and **top** attributes for each control state of the model at snapshot sp. The value of $\mathbf{ex_P}$ for root control state determines the set of potential small steps of the model.*

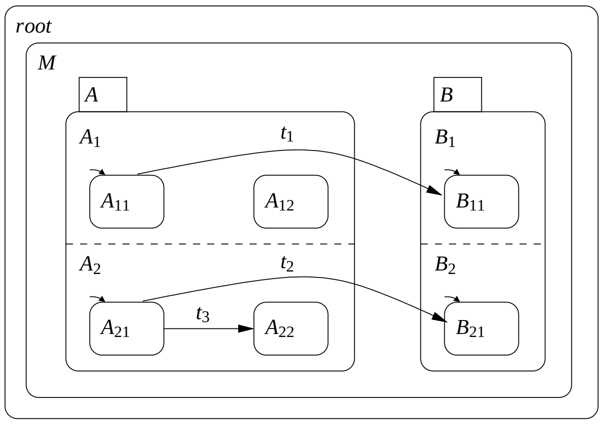


Figure 4.9: Computation of potential small steps for an example BSML model.

Table 4.4: The values of attributes for the model in Figure 4.9, according to the SCOPE-PARENT priority semantics.

Control State	**top**	$\mathbf{ex_P}$	Control State	**top**	$\mathbf{ex_P}$
root	$\emptyset$	$\{\{t_1, t_2\}\}$	A_{21}	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$
M	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$	A_{22}	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$
A	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$	B	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$
A_1	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$	B_1	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$
A_{11}	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$	B_{11}	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$
A_{12}	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$	B_2	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$
A_2	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$	B_{21}	$\{\{t_1, t_2\}\}$	$\{\{t_1, t_2\}\}$

Transition t_3 *does not have a chance to be a member of a potential small step because it has a lower priority than transition* t_2*. The reason is that, according to the computation in Figure 4.8, to compute attribute "*$\mathbf{top}([\mathrm{A}_2], sp)$*" using line 4a, where a pair of square bracket "[ ]" is used to distinguish the syntactic part of an expression, the value of "*$\mathbf{top}([\mathrm{A}_1, \mathrm{A}_2], sp)$*", using line 5a, should be computed:*

$$\begin{aligned}\mathbf{top}([\mathrm{A}_1, \mathrm{A}_2], sp) &= \mathbf{top}([\mathrm{And}\ \mathrm{A}_1, \mathrm{A}_2], sp) \otimes en\_trs(\{\mathrm{t}_3\}, sp),\\ &= \{\{\mathrm{t}_1, \mathrm{t}_2\}\} \otimes \{\mathrm{t}_3\},\\ &= \{\{\mathrm{t}_1, \mathrm{t}_2\}\},\end{aligned}$$

which does not allow t_3 *to be added to set* $\{\mathrm{t}_1, \mathrm{t}_2\}$ *because the small-step consistency would be violated.*

4.4.2 Scope-Child Priority Semantics

Figure 4.10 presents the hierarchical computation of the set of potential small steps according to the Scope-Child. Here only one attribute is necessary, which is computed in a bottom-up manner. The value of Π is $\mathbf{ex_C}$. The merge operator remains the same as in Figure 4.8. Line 9a uses "," to represent three separate equalities, each of which corresponds to one of the right-hand-side alternatives in line 9.

Example 29 *Let us consider the model in Figure 4.9 in Example 28 again, but this time with the* Scope-Child *priority semantics, instead of the* Scope-Parent *priority semantics. Table 4.5 shows the values of* $\mathbf{ex_C}$ *for each control state of the model at snapshot* sp*. The value of* $\mathbf{ex_C}$ *for root control state determines the set of potential small steps of the model.*

Transition t_2 *does not have a chance to be a member of a potential small step because it has a lower priority than transition* t_3*. The reason is that, according to the computation in Figure 4.10, the value of "*$\mathbf{ex_C}([\mathrm{M}, \{\mathrm{t}_1, \mathrm{t}_2\}], sp)$*" is computed as follows:*

$$\begin{aligned}\mathbf{ex_C}([\mathrm{M}, \{\mathrm{t}_1, \mathrm{t}_2\}], sp) &= \mathbf{ex_C}([\mathrm{A}, \mathrm{B}], sp) \otimes en\_trs(\{\mathrm{t}_1, \mathrm{t}_2\}, sp),\\ &= \{\{\mathrm{t}_3\}\} \otimes \{\mathrm{t}_1, \mathrm{t}_2\},\\ &= \{\{\mathrm{t}_1, \mathrm{t}_3\}\},\end{aligned}$$

which does not allow t_2 *to be added to the potential small step because its source is the same as* t_2 *and it is also not an interrupt for* t_3*;* t_1*, however, is added because it is an interrupt for* t_3.

1. $\langle$root$\rangle$	::=	$\langle$Orstate$\rangle$
a. $\mathbf{ex_C}$(root, sp)	=	$\mathbf{ex_C}$(Orstate, sp)
2. $\langle$Orstate$\rangle$	::=	**Or** $\langle$states-o$\rangle$ $\langle$transitions$\rangle$
a. $\mathbf{ex_C}$(Orstate, sp)	=	$\mathbf{ex_C}$(states-o, sp) $\otimes$ en_trs(transitions, sp)
3. $\langle$states-o$\rangle$	::=	$\langle$states-o$\rangle$ $\langle$state$\rangle$
a. $\mathbf{ex_C}$(states-o$_0$, sp)	=	$\mathbf{ex_C}$(states-o$_1$, sp) $\cup$ $\mathbf{ex_C}$(state, sp)
4. $\langle$states-o$\rangle$	::=	$\langle$state$\rangle$
a. $\mathbf{ex_C}$(states-o, sp)	=	$\mathbf{ex_C}$(state, sp)
5. $\langle$Andstate$\rangle$	::=	**And** $\langle$states-a$\rangle$ $\langle$transitions$\rangle$
a. $\mathbf{ex_C}$(Andstate, sp)	=	$\mathbf{ex_C}$(states-a, sp) $\otimes$ en_trs(transitions, sp)
6. $\langle$states-a$\rangle$	::=	$\langle$states-a$\rangle$ $\langle$state$\rangle$
a. $\mathbf{ex_C}$(states-a$_0$, sp)	=	$\mathbf{ex_C}$(states-a$_1$, sp) $\parallel$ $\mathbf{ex_C}$(state, sp)
7. $\langle$states-a$\rangle$	::=	$\langle$state$\rangle$
a. $\mathbf{ex_C}$(states-a, sp)	=	$\mathbf{ex_C}$(state, sp)
8. $\langle$Basicstate$\rangle$	::=	**Basic**
a. $\mathbf{ex_C}$(Basicstate, sp)	=	$\emptyset$
9. $\langle$state$\rangle$	::=	$\langle$Orstate$\rangle$ | $\langle$Andstate$\rangle$ | $\langle$Basicstate$\rangle$
a. $\mathbf{ex_C}$(state, sp)	=	$\mathbf{ex_C}$(Orstate, sp), $\mathbf{ex_C}$(Andstate, sp), $\mathbf{ex_C}$(Basicstate, sp)

10. $\mathbb{T} \otimes T' = \{\, (T_1 \cup T'') |\ T_1 \in \mathbb{T} \wedge T'' \subseteq T' \wedge$
$(\forall t' : (T_1 \cup T') \cdot t' \in (T' - T'') \Leftrightarrow \exists t \in (T_1 \cup T'') \cdot \neg C(t', t) \wedge \neg P(t', t)) \,\}$

Figure 4.10: Computing potential small steps in the Scope-Child priority semantics.

Table 4.5: The values of attributes for the model in Figure 4.9, according to the SCOPE-CHILD priority semantics.

Control State	$\mathbf{ex_C}$	Control State	$\mathbf{ex_C}$
root	$\{\{t_1, t_3\}\}$	A_{21}	$\emptyset$
M	$\{\{t_1, t_3\}\}$	A_{22}	$\emptyset$
A	$\{\{t_3\}\}$	B	$\emptyset$
A_1	$\emptyset$	B_1	$\emptyset$
A_{11}	$\emptyset$	B_{11}	$\emptyset$
A_{12}	$\emptyset$	B_2	$\emptyset$
A_2	$\{\{t_3\}\}$	B_{21}	$\emptyset$

4.4.3 NO PRIORITY Semantics

To specify the semantics that no hierarchical semantics is chosen, i.e., the NO PRIORITY, a similar computation to the one in Figure 4.10 can be used, except that the merge operator is defined as,

$$\begin{aligned}\mathbb{T} \otimes T' \quad = \quad & \{\, (T_1 - T_1') \cup T'' \mid T_1 \in \mathbb{T} \wedge T_1' \subseteq T_1 \wedge T'' \subseteq T' \wedge \\ & (\forall t' : (T_1 \cup T') \cdot t' \in (T' - T'') \Leftrightarrow \exists t \in (T_1 - T_1' \cup T'') \cdot \neg C(t', t) \wedge \neg P(t', t)) \wedge \\ & (\forall t : (T_1 \cup T') \cdot t \in T_1' \Leftrightarrow \exists t' \in T'' \cdot \neg C(t, t') \wedge \neg P(t, t')) \,\}.\end{aligned}$$

The above merge operator is different from the one used for the SCOPE-PARENT and SCOPE-CHILD semantic options because all combinations of merging should be considered, instead of giving precedence to transitions with higher or lower scopes, as in the SCOPE-PARENT and SCOPE-CHILD semantic options, respectively. Thus, the merge operator should perform two tasks. First, a maximum set of enabled transitions at the current control state should be added to each of the sets of set of transitions received from the children of the control state, in a manner that the concurrency and consistency semantics of the BSML are not violated. Second, the enabled transitions at the current control state should be considered to replace the transitions in each of the sets of set of transitions received from the children of the control state, again in a manner that the result is maximal: no more transitions can be added without violating one of the concurrency and consistency semantics of the BSML. The above two tasks are embodied in the definition of merge operator above. The second and the third lines in the definition of the merge operator above enforce the maximality of the resulting merge.

Example 30 *Let us consider the model in Figure 4.9 one last time, this time with the* No Priority *semantics. Table 4.6 shows the values of* **ex**, *which determine the set of potential small steps for the* No Priority *semantics, for each control state of the model at snapshot sp.*

Table 4.6: The values of attributes for the model in Figure 4.9, according to the No Priority semantics.

Control State	**ex**	Control State	**ex**
root	$\{\{t_1, t_3\}, \{t_1, t_2\}\}$	A_{21}	$\emptyset$
M	$\{\{t_1, t_3\}, \{t_1, t_2\}\}$	A_{22}	$\emptyset$
A	$\{\{t_3\}\}$	B	$\emptyset$
A_1	$\emptyset$	B_1	$\emptyset$
A_{11}	$\emptyset$	B_{11}	$\emptyset$
A_{12}	$\emptyset$	B_2	$\emptyset$
A_2	$\{\{t_3\}\}$	B_{21}	$\emptyset$

Transition t_2 *has a chance to be a member of a potential small step because it can replace transition* t_3. *The reason is that, according to the merge operator in Section 4.4.3, the value of "*$\mathbf{ex}([M, \{t_1, t_2\}], sp)$*" is computed as follows:*

$$\begin{aligned} \mathbf{ex}([M, \{t_1, t_2\}], sp) &= \mathbf{ex}([A, B], sp) \otimes en\_trs(\{t_1, t_2\}, sp), \\ &= \{\{t_3\}\} \otimes \{t_1, t_2\}, \\ &= \{\{t_1, t_2\}, \{t_1, t_3\}\}. \end{aligned}$$

4.4.4 Other Priority Semantics

Other hierarchical semantic options can be defined similar to the Scope-Parent and Scope-Child semantics, but using different parsing mechanisms. For example, to formalize the semantics of Source-Parent and Source-Child priority semantics, mentioned in Section 3.8, the set of transitions in "transitions" in the BNF in figure 4.7, should be the transitions whose source control states are the control state that "transitions" is associated with. The formalization of the Explicit priority semantics is not hierarchical, and is independent of a parsing mechanism. The decision that whether a potential small step, T, has a higher priority than another potential small step, T', can be only made when the two sets are entirely computed: T has a higher priority than T' if there is a transition in T that has a higher priority than all transitions in T'.[1] The formalization of the Negation of Triggers semantics is manifested in the formalization of the enabledness of a single transition, as was described at the end of Section 4.1.2.

4.5 Enabledness Parameters

In this section, the formalization of the semantic options of the enabledness semantic aspects is described.

Some of the semantic options of the enabledness semantic aspects are out of the scope of the formalization in this dissertation. These semantic options are transition-aware semantic options. The feature diagram in Figure 4.11 is the same as the feature diagram in Figure 4.6, except that the transition-aware semantic options are signified by a "✷" on their righthand sides. The Whole event lifeline semantics is an example of a transition-aware semantic option. To determine whether a transition, *t*, whose trigger includes a negated event, is enabled in a big step, it should be ensured that the negated event is not generated by any of the transitions of the big step, even the transitions that are executed after *t* is executed. The formalization of the transition-aware semantic options require, at each snapshot, being able to determine the enabledness and/or effect of the execution of other transitions in the immediate or future small steps. A transition-aware semantic option convolutes the role of enabledness semantic aspects, which are supposed to be formalized by only using snapshot elements, and the role of structural semantic aspects, which use predicates over the transitions of models.

A semantic option of an enabledness semantic aspect is formalized by a set of snapshot elements. For each semantic option, I introduce snapshot elements of varying names. I use the following naming convention for these snapshot elements: When formalizing an enabledness semantic option, the name of the semantic option, or its abbreviation, is used in the name of one of the snapshot elements that formalize the semantics. For example, $E_{\text{Remainder}}$ is the snapshot element that models the Present In Remainder event lifeline semantics. Also, by convention, if the enabledness predicate of a snapshot element is not specified, it means it is equivalent to true.

For each of the semantic aspects Event Lifeline, Enabledness Memory Protocol, and As-

[1] In Section 7.2, where the semantic definition of synchronizing big-step modelling languages (SBSMLs) is described, it is shown how the semantics of the Explicit priority semantics can be also formalized using a similar semantic definition schema as the one for the synchronizing big-step modelling languages.

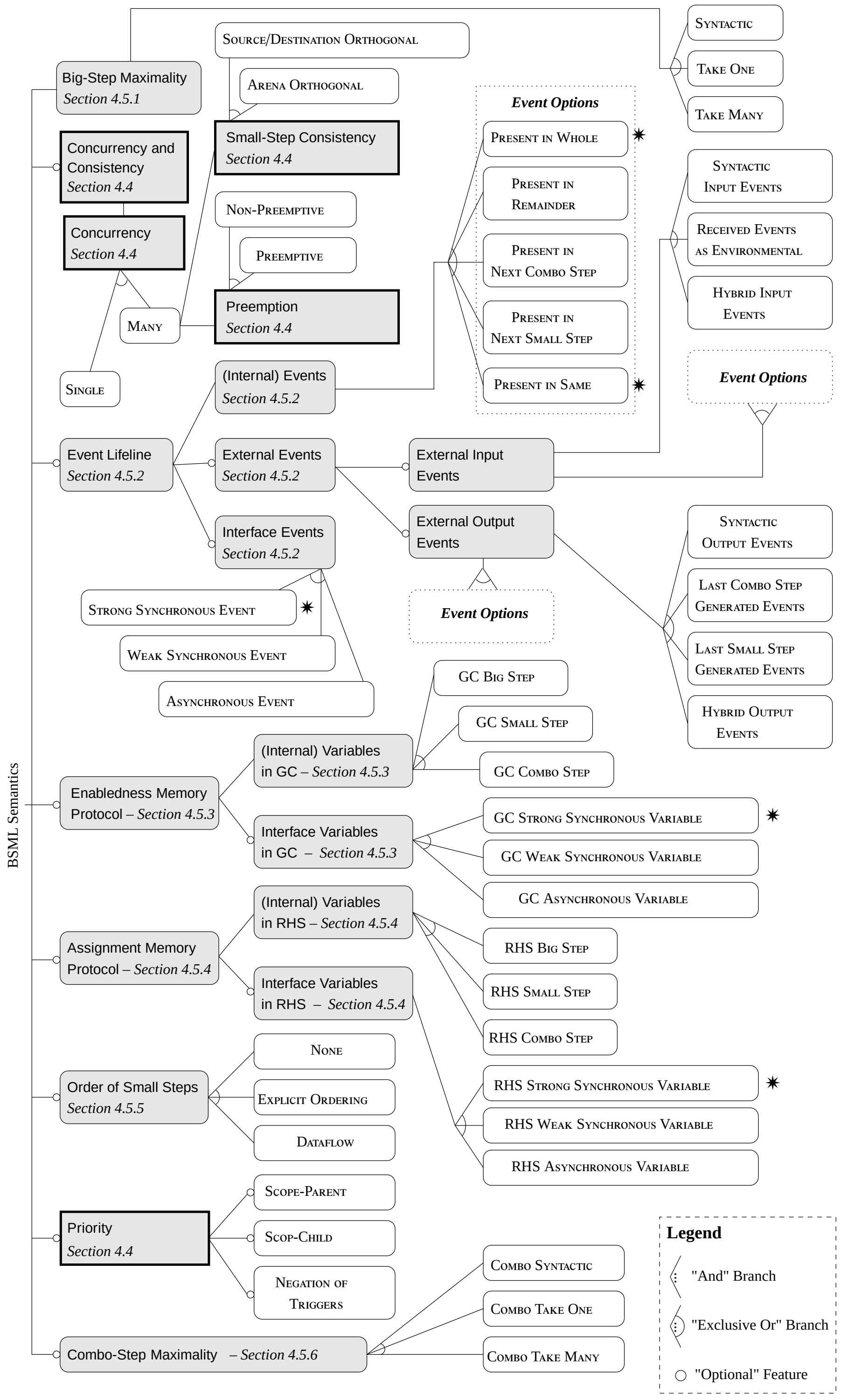

BSML Semantics
Big-Step Maximality Section 4.5.1
Syntactic
Take One
Take Many
Concurrency and Consistency Section 4.4
Concurrency Section 4.4
Single
Many
Source/Destination Orthogonal
Arena Orthogonal
Small-Step Consistency Section 4.4
Non-Preemptive
Preemptive
Preemption Section 4.4
Event Options
Present in Whole ✷
Present in Remainder
Present in Next Combo Step
Present in Next Small Step
Present in Same ✷
Syntactic Input Events
Received Events as Environmental
Hybrid Input Events
Event Options
Event Lifeline Section 4.5.2
(Internal) Events Section 4.5.2
External Events Section 4.5.2
Interface Events Section 4.5.2
External Input Events
External Output Events
Event Options
Syntactic Output Events
Last Combo Step Generated Events
Last Small Step Generated Events
Hybrid Output Events
Strong Synchronous Event ✷
Weak Synchronous Event
Asynchronous Event
Enabledness Memory Protocol – Section 4.5.3
(Internal) Variables in GC – Section 4.5.3
Interface Variables in GC – Section 4.5.3
GC Big Step
GC Small Step
GC Combo Step
GC Strong Synchronous Variable ✷
GC Weak Synchronous Variable
GC Asynchronous Variable
Assignment Memory Protocol – Section 4.5.4
(Internal) Variables in RHS – Section 4.5.4
Interface Variables in RHS – Section 4.5.4
RHS Big Step
RHS Small Step
RHS Combo Step
RHS Strong Synchronous Variable ✷
RHS Weak Synchronous Variable
RHS Asynchronous Variable
Order of Small Steps Section 4.5.5
None
Explicit Ordering
Dataflow
Priority Section 4.4
Scope-Parent
Scop-Child
Negation of Triggers
Combo-Step Maximality – Section 4.5.6
Combo Syntactic
Combo Take One
Combo Take Many
Legend
"And" Branch
"Exclusive Or" Branch
○ "Optional" Feature

Figure 4.11: Transition-aware semantic options are signified by a "✷" next to them. As in Figure 4.6, the boxes with bold, solid frames represent the structural semantic aspects.

signment Memory Protocol, first, the formalization of their semantic options for the internal events/variables is presented, followed by the formalization of the external and interface events/variables.

4.5.1 Big-Step Maximality

The semantics of each of the SYNTACTIC and TAKE ONE semantic options is specified by a snapshot element. The TAKE MANY semantic option does not introduce any snapshot elements.

SYNTACTIC

During a big step, snapshot element $M_{\text{SYNTACTIC}}$ collects the control states such that each is either the arena or a descendant of the arena of an executed transition that enters a stable control state. Predicate $\texttt{en\_}M_{\text{SYNTACTIC}}(t, sp)$ determines whether a transition, t', has already been executed during the big step that has entered a stable control state, s', such that $arena(t) \in children^*(arena(t'))$, in which case t cannot be taken in the current big step.

$$\begin{array}{lcl}
\texttt{reset\_}M_{\text{SYNTACTIC}}(sp^0, I, sp) & \equiv & M_{\text{SYNTACTIC}} = \emptyset \\
\texttt{next\_}M_{\text{SYNTACTIC}}(sp, \tau, sp') & \equiv & M'_{\text{SYNTACTIC}} = M_{\text{SYNTACTIC}} \cup \\
 & & \bigcup_{t \in \tau}\{s \mid s \in children^*(arena(t)) \;\wedge bigstable(dest(t))\} \\
\texttt{en\_}M_{\text{SYNTACTIC}}(t, sp) & \equiv & arena(t) \notin M_{\text{SYNTACTIC}}
\end{array}$$

TAKE ONE

During a big step, snapshot $M_{\text{TAKE ONE}}$ collects the control states such that each is either the arena or the child of the arena of an executed transition. Predicate $\texttt{en\_}M_{\text{TAKE ONE}}(t, sp)$ determines whether a transition, t', has already been executed during the big step such that $arena(t) \in children^*(arena(t'))$, in which case t cannot be taken in the current big step.

$$\begin{array}{lcl}
\texttt{reset\_}M_{\text{TAKE ONE}}(sp^0, I, sp) & \equiv & M_{\text{TAKE ONE}} = \emptyset \\
\texttt{next\_}M_{\text{TAKE ONE}}(sp, \tau, sp') & \equiv & M'_{\text{TAKE ONE}} = M_{\text{TAKE ONE}} \cup \\
 & & \bigcup_{t \in \tau}\{s \mid s \in children^*(arena(t))\} \\
\texttt{en\_}M_{\text{TAKE ONE}}(t, sp) & \equiv & arena(t) \notin M_{\text{TAKE ONE}}
\end{array}$$

4.5.2 Event Lifeline

This section presents the formalization of the event lifeline semantics for internal events, followed by examples of the formalization of external and interface events. The snapshot elements used in this section are all of type set of events. First, some notation for the formalization of the notion of combo step are presented.

The semantics of the PRESENT IN SAME event lifeline semantics can be formalized using a synchronization capability, as will be described in Chapter 6.

Combo-Step Semantics

A *combo-step semantic option*, or a *combo-step semantics*, is an event lifeline or memory protocol semantic option whose semantics determines the scope of the combo-steps of a model; e.g., the PRESENT IN NEXT COMBO STEP event lifeline semantics. In formalizing a combo-step semantics, the last small step of a combo step must be identified so that the values of the necessary snapshot elements are adjusted at the end of the combo step. For example, in the PRESENT IN NEXT COMBO STEP event lifeline semantics, at the end of each combo step, the statuses of events are adjusted by setting them to the collected events during the current combo step. To identify the last small step of a combo step, however, all semantic options that are combo-step semantics must be known because otherwise it is not possible to determine whether there is any transition enabled at the destination snapshot of the small step. This task can be achieved by identifying the snapshot elements that must be adjusted at the end of combo steps. The set of all snapshot elements used in the formalization of the combo-step semantic options of a BSML semantics are its *combo-step snapshot elements*, denoted by Cs. As such, the formalization of a combo-step semantics requires knowledge about the formalization of other enabledness semantic aspects, and thus indirectly depends on them. The formalizations of the combo-step semantics are the only cases that introduce such cross-cuttings in the formalization.

PRESENT IN REMAINDER

The snapshot element $E_{\text{REMAINDER}}$ collects the set of generated events of a big step. At the beginning of each big step, $E_{\text{REMAINDER}}$ is initialized to the set of environmental input events received from the environment. A transition is enabled according to $E_{\text{REMAINDER}}$, if the positive literals in its trigger are in $E_{\text{REMAINDER}}$, but not its negated literals.

$$
\begin{array}{lcl}
\texttt{reset\_}E_{\textsc{Remainder}}(sp^0, I, sp) & \equiv & E_{\textsc{Remainder}} = I.events \\
\texttt{next\_}E_{\textsc{Remainder}}(sp, \tau, sp') & \equiv & E'_{\textsc{Remainder}} = E_{\textsc{Remainder}} \cup gen(\tau) \\
\texttt{en\_}E_{\textsc{Remainder}}(t, sp) & \equiv & (pos\_trig(t) \subseteq E_{\textsc{Remainder}}) \wedge \\
 & & (neg\_trig(t) \cap E_{\textsc{Remainder}} = \emptyset)
\end{array}
$$

Function $gen(\tau)$ denotes $\bigcup_{t \in \tau} gen(t)$.

If the variation of global consistency semantics in an operational way [66], as discussed on page 49, is desired, then the following snapshot element is also needed.

$$
\begin{array}{lcl}
\texttt{reset\_}E_{GlobalConsistency}(sp^0, I, sp) & \equiv & E_{GlobalConsistency} = \emptyset \\
\texttt{next\_}E_{GlobalConsistency}(sp, \tau, sp') & \equiv & E'_{GlobalConsistency} = E_{GlobalConsistency} \cup neg\_trig(\tau) \\
\texttt{en\_}E_{GlobalConsistency}(t, sp) & \equiv & gen(t) \cap E_{GlobalConsistency} = \emptyset
\end{array}
$$

Function $neg\_trig(\tau)$ denotes $\bigcup_{t \in \tau} neg\_trig(t)$.

Present In Next Combo Step

Two snapshot elements are used to model the Present In Next Combo Step semantics. Snapshot element $E_{Collect}$ collects the generated events during a combo step, to make them available in the next combo step. Snapshot element $E_{\textsc{Next C.S.}}$ is the set of generated events collected from the previous combo step that are considered as present in the current combo step. Snapshot element $E_{Collect}$ has no role in the enabledness of a transition. When the Present In Next Combo Step semantic option is chosen, $\{E_{\textsc{Next C.S.}}, E_{Collect}\} \subseteq Cs$, where Cs is the set of combo-step snapshot elements, as described earlier in this section.

$$
\begin{array}{lcl}
\texttt{reset\_}E_{\textsc{Next C.S.}}(sp^0, I, sp) & \equiv & E_{\textsc{Next C.S.}} = I.events \\
\texttt{next\_}E_{\textsc{Next C.S.}}(sp, \tau, sp') & \equiv & E'_{\textsc{Next C.S.}} = \text{if } \ EndC \ \text{ then} \\
 & & \qquad\qquad\qquad E_{Collect} \cup gen(\tau) \\
 & & \qquad\qquad\quad \text{else} \\
 & & \qquad\qquad\qquad E_{\textsc{Next C.S.}} \\
\texttt{en\_}E_{\textsc{Next C.S.}}(t, sp) & \equiv & (pos\_trig(t) \subseteq E_{\textsc{Next C.S.}}) \land \\
 & & (neg\_trig(t) \cap E_{\textsc{Next C.S.}} = \emptyset)
\end{array}
$$

$$
\begin{array}{lcl}
\texttt{reset\_}E_{Collect}(sp^0, I, sp) & \equiv & E_{collect} = \emptyset \\
\texttt{next\_}E_{Collect}(sp, \tau, sp') & \equiv & E'_{collect} = \text{if } \ EndC \ \text{ then} \\
 & & \qquad\quad \emptyset \\
 & & \qquad \text{else} \\
 & & \qquad\quad E_{Collect} \cup gen(\tau)
\end{array}
$$

And, $EndC \equiv (\nexists \tau' \cdot \tau' \in executable(root, sp' \oplus Cs))$.

The $EndC$ predicate above identifies the last small step of a combo step. Its definition relies on the set of combo-step snapshot elements, Cs, which is the set of snapshot elements that specify the notion of combo step in a semantics. The replace operator "$\oplus$", described in Section 4.1.1, modifies a snapshot in the first operand, by replacing those of its snapshot elements that each has a corresponding new value in the second operand.

Present In Next Small Step

Snapshot element $E_{\textsc{Next S.S.}}$ is equal to the set of generated events in the previous small step, except at the beginning of a big step when $E_{\textsc{Next S.S.}}$ is equal to the set of environmental input events.

$$
\begin{array}{lcl}
\texttt{reset\_}E_{\textsc{Next S.S.}}(sp^0, I, sp) & \equiv & E_{\textsc{Next S.S.}} = I.events \\
\texttt{next\_}E_{\textsc{Next S.S.}}(sp, \tau, sp') & \equiv & E'_{\textsc{Next Small Step}} = gen(\tau) \\
\texttt{en\_}E_{\textsc{Next S.S.}}(t, sp) & \equiv & (pos\_trig(t) \subseteq E_{\textsc{Next S.S.}}) \land \\
 & & (neg\_trig(t) \cap E_{\textsc{Next S.S.}} = \emptyset)
\end{array}
$$

Transition-Aware Event Lifeline Semantics

The non-operational, globally-consistent variation of the PRESENT IN REMAINDER semantics [86], as well as the WHOLE semantics, are transition-aware semantic options because in order for the negation of an event to trigger a transition there should be a guarantee that it is not generated by any of the transitions that can generate it. Similarly, in a snapshot, an event should be sensed as present if it is generated by a transition in a future small step. Thus, transitions need to be *aware* of each others' executions in a structural way to accommodate the above scenarios. The PRESENT IN SAME semantic option is also transition-aware, but can be defined through a notion of synchronization, as described in Chapter 6. Unlike the PRESENT IN SAME semantic option, however, the non-operational, globally-consistent variation of PRESENT IN REMAINDER and the WHOLE semantics cannot be modelled by synchronization, because synchronization is relevant for the transitions within one single small step, whereas the lifeline of the events in these two semantic options is beyond a single small step.

External Events

In the above formalization of the PRESENT IN REMAINDER, PRESENT IN NEXT COMBO STEP, and PRESENT IN NEXT SMALL STEP event lifeline semantics, a non-distinguishing BSML is assumed: The input, internal, and output events are not distinguished syntactically, as described 3.4.1. Such a formalization means that the same semantics are considered for internal events and the events received and sent to the environment. For example, in the PRESENT IN NEXT SMALL STEP semantic option, the environmental input events received from the environment, i.e., $I.events$, persist for one small step. As shown in the feature diagram in Figurer 4.11, regardless of a BSML being non-distinguishing or distinguishing, the input and output events can have semantic options of their own, independent of the internal events. Next, two examples of formalizing the semantics of external events, one for distinguishing BSMLs and one for non-distinguishing BSMLs, are presented.

SYNTACTIC INPUT EVENTS The SYNTACTIC INPUT EVENTS semantic options is a semantic option for external input events for a distinguishing BSML: External events of a model are distinguished syntactically from the internal events. The following formalization assigns a PRESENT IN REMAINDER-like semantics to the environmental input events of a model and the PRESENT IN NEXT COMBO

Step semantics to the internal and output events of the model. In the following formalization, the environmental input events of a model are denoted by *Env*.

$$
\begin{array}{lcl}
\texttt{reset\_}E_{\textsc{Next C.S.[SIE]}}(sp^0, I, sp) & \equiv & E_{\textsc{Next C.S.[SIE]}} = \emptyset \\
\texttt{next\_}E_{\textsc{Next C.S.[SIE]}}(sp, \tau, sp') & \equiv & E'_{\textsc{Next C.S.[SIE]}} = \text{if } \; EndC \; \text{ then} \\
 & & \qquad\qquad E_{Collect} \cup gen(\tau) \\
 & & \qquad \text{else} \\
 & & \qquad\qquad E_{\textsc{Next C.S.[SIE]}} \\
\texttt{en\_}E_{\textsc{Next C.S.[SIE]}}(t, sp) & \equiv & ((pos\_trig(t) - Env) \subseteq E_{\textsc{Next C.S.[SIE]}}) \wedge \\
 & & ((neg\_trig(t) - Env) \cap E_{\textsc{Next C.S.[SIE]}} = \emptyset)
\end{array}
$$

$$
\begin{array}{lcl}
\texttt{reset\_}E_{Collect}(sp^0, I, sp) & \equiv & E_{collect} = \emptyset \\
\texttt{next\_}E_{Collect}(sp, \tau, sp') & \equiv & E'_{collect} = \text{if } \; EndC \; \text{ then} \\
 & & \qquad\qquad \emptyset \\
 & & \qquad \text{else} \\
 & & \qquad\qquad E_{Collect} \cup gen(\tau)
\end{array}
$$

And, $EndC \equiv (\nexists \tau' \cdot \tau' \in executable(root, sp' \oplus Cs))$.

$$
\begin{array}{lcl}
\texttt{reset\_}E_{\textsc{Remainder[Env]}}(sp^0, I, sp) & \equiv & E_{\textsc{Remainder[Env]}} = Env \\
\texttt{next\_}E_{\textsc{Remainder[Env]}}(sp, \tau, sp') & \equiv & E'_{\textsc{Remainder[Env]}} = E_{\textsc{Remainder[Env]}} \\
\texttt{en\_}E_{\textsc{Remainder[Env]}}(t, sp) & \equiv & ((pos\_trig(t) \cap Env) \subseteq E_{\textsc{Remainder[Env]}}) \wedge \\
 & & ((neg\_trig(t) \cap Env) \cap E_{\textsc{Remainder[Env]}} = \emptyset)
\end{array}
$$

In the above formalization, it should be the case that $I.events \subseteq Env$. Snapshot element $E_{Collect}$ is exactly the same as in Section 4.5.2. Snapshot element $E_{\textsc{Next C.S.[SIE]}}$ is different from snapshot element $E_{\textsc{Next C.S.}}$ in that its "`en`" predicate checks only for presence and absence of internal events. (SIE stands for "Syntactic Input Events". Similar abbreviations are used in the following formalizations.)

Hybrid Input Events

In the Hybrid Input Events semantic option, which is relevant for non-distinguishing BSMLs, an event that is received at the beginning of a big step is treated as an environmental input event only if it is a genuine input of a model, meaning that it is not generated by any transition in the model. The following formalization assigns a PRESENT IN REMAINDER-like semantics to the genuine input events and the PRESENT IN NEXT SMALL STEP semantics to internal and output events. In the formalization, the set of genuine events of a model are denoted by *Genuine*.

$$\begin{array}{lcl}
\texttt{reset\_}E_{\text{NEXT S.S.[HIE]}}(sp^0, I, sp) & \equiv & E_{\text{NEXT S.S.[HIE]}} = I.events - Genuine \\
\texttt{next\_}E_{\text{NEXT S.S.[HIE]}}(sp, \tau, sp') & \equiv & E'_{\text{NEXT S.S.[HIE]}} = gen(\tau) \\
\texttt{en\_}E_{\text{NEXT S.S.[HIE]}}(t, sp) & \equiv & ((pos\_trig(t) - Genuine) \subseteq E_{\text{NEXT S.S.[HIE]}}) \wedge \\
& & ((neg\_trig(t) - Genuine) \cap E_{\text{NEXT S.S.[HIE]}} = \emptyset)
\end{array}$$

$$\begin{array}{lcl}
\texttt{reset\_}E_{\text{REMAINDER[G]}}(sp^0, I, sp) & \equiv & E_{\text{REMAINDER[G]}} = I.events \cap Genuine \\
\texttt{next\_}E_{\text{REMAINDER[G]}}(sp, \tau, sp') & \equiv & E'_{\text{REMAINDER[G]}} = E_{\text{REMAINDER[G]}} \\
\texttt{en\_}E_{\text{REMAINDER[G]}}(t, sp) & \equiv & ((pos\_trig(t) \cap Genuine) \subseteq E_{\text{REMAINDER[G]}}) \wedge \\
& & ((neg\_trig(t) \cap Genuine) \cap E_{\text{REMAINDER[G]}} = \emptyset)
\end{array}$$

In the above formalization, *I.events* might include received input events other than the ones in *Genuine*, which are treated according to the event lifeline semantics of internal events.

Interface Events

The following snapshot elements together specify a PRESENT IN REMAINDER-like semantics for input events according to the HYBRID INPUT EVENTS semantics, the PRESENT IN NEXT SMALL STEP semantics for internal events, and the ASYNCHRONOUS EVENTS semantics for interface events. The set of genuine and interface events of a model are denoted by *Genuine* and *Interface*, respectively.

$$
\begin{array}{lcl}
\texttt{reset\_}E_{\text{NEXT S.S.[HIE-ASYN]}}(sp^0, I, sp) & \equiv & E_{\text{NEXT S.S.[HIE-ASYN]}} = I.events - Genuine \\
\texttt{next\_}E_{\text{NEXT S.S.[HIE-ASYN]}}(sp, \tau, sp') & \equiv & E'_{\text{NEXT S.S.[HIE-ASYN]}} = gen(\tau) - Interface \\
\texttt{en\_}E_{\text{NEXT S.S.[HIE-ASYN]}}(t, sp) & \equiv & (pos\_trig(t) - Genuine - Interface) \\
 & & \subseteq E_{\text{NEXT S.S.[HIE-ASYN]}} \wedge \\
 & & (neg\_trig(t) - Genuine - Interface) \cap \\
 & & E_{\text{NEXT S.S.[HIE-ASYN]}} = \emptyset
\end{array}
$$

$$
\begin{array}{lcl}
\texttt{reset\_}E_{\text{REMAINDER[G]}}(sp^0, I, sp) & \equiv & E_{\text{REMAINDER[G]}} = I.events \cap Genuine \\
\texttt{next\_}E_{\text{REMAINDER[G]}}(sp, \tau, sp') & \equiv & E'_{\text{REMAINDER[G]}} = E_{\text{REMAINDER[G]}} \\
\texttt{en\_}E_{\text{REMAINDER[G]}}(t, sp) & \equiv & ((pos\_trig(t) \cap Genuine) \subseteq E_{\text{REMAINDER[G]}}) \wedge \\
 & & ((neg\_trig(t) \cap Genuine) \cap E_{\text{REMAINDER[G]}} = \emptyset)
\end{array}
$$

$$
\begin{array}{lcl}
\texttt{reset\_}E_{\text{ASYNCHRONOUS EVENTS}}(sp^0, I, sp) & \equiv & E_{\text{ASYNCHRONOUS EVENTS}} = sp^0.E_{CollAsyn} \\
\texttt{next\_}E_{\text{ASYNCHRONOUS EVENTS}}(sp, \tau, sp') & \equiv & E'_{\text{ASYNCHRONOUS EVENTS}} = E_{\text{ASYNCHRONOUS EVENTS}} \\
\texttt{en\_}E_{\text{ASYNCHRONOUS EVENTS}}(t, sp) & \equiv & ((pos\_trig(t) - Genuine) \cap Interface) \subseteq \\
 & & E_{\text{ASYNCHRONOUS EVENTS}} \wedge \\
 & & ((neg\_trig(t) - Genuine) \cap Interface) \cap \\
 & & E_{\text{ASYNCHRONOUS EVENTS}} = \emptyset
\end{array}
$$

$$
\begin{array}{lcl}
\texttt{reset\_}E_{CollAsyn}(sp^0, I, sp) & \equiv & E_{CollAsyn} = \emptyset \\
\texttt{next\_}E_{CollAsyn}(sp, \tau, sp') & \equiv & E'_{CollAsyn} = E_{CollAsyn} \cup (gen(\tau) \cap Interface)
\end{array}
$$

In the above formalization, it is assumed that $I.events$ might include received input events other than the ones in $Genuine$, which are treated according to the event lifeline semantics of internal events. Snapshot element $E_{\text{REMAINDER[G]}}$ is defined the same as in Section 4.5.2. Snapshot element $E_{CollAsyn}$ collects the generated interface events of a big step to be used in the next big step, by $E_{\text{ASYNCHRONOUS EVENTS}}$.

Similar to the transition-aware semantic options of internal events, discussed in Section 4.5.2, on page 116, the STRONG SYNCHRONOUS EVENT lifeline semantics for interface events is a transition-aware semantic option.

4.5.3 Enabledness Memory Protocol

This section presents the formalization of the semantic options for the Enabledness Memory Protocol. First, the formalization of the memory protocols for internal variables is presented, followed by examples of formalizing the semantic options of interface variables.

The snapshot elements used in the formalization of Enabledness Memory Protocols need to know about the snapshot element that keeps track of values of variables according to the Assignment Memory Protocol, denoted by $\mathcal{V}_{asn}$, in order to adjust the values of variables in the snapshot element that the GC of transitions are checked against. At the beginning of a big step and before a small step is executed, the snapshot element that $\mathcal{V}_{asn}$ refers to is used to initialize the snapshot elements that models an enabledness memory protocol. In Section 4.5.4, as part of the formalization of the semantics of assignment memory protocols, the value of $\mathcal{V}_{asn}$ for each of the assignment memory protocols is specified.

Before presenting the formalization of the enabledness memory protocols, some notation for formalizing a notion of store are presented.

A *store* is a set of $\langle$ variable, value $\rangle$ pairs. It is a total function from the set of variables of a model to their values. The type of all of the snapshot elements used in this section is store.

The *override operator*, $\uplus$, replaces some pairs of a store with new pairs whose first element are the same as the replaced ones. For example, $x_1 \uplus \{(var_1, val_1), (var_2, val_2), \cdots, (var_n, val_n)\}$ replaces the values of variables var_1, var_2, $\cdots$, and var_n in store x_1 with values val_1, val_2, $\cdots$, and val_n, respectively.

A *variable expression* is an arithmetic or boolean expression over the variables of a BSML model, possibly with some variables being prefixed with a variable operator. All variable expressions are assumed to be *well-typed*: i.e., the operand of operators are of the expected types. Furthermore, it is assumed that the subexpressions of an expression are unambiguously parsed, which can be interpreted as the subexpressions of all expressions being parenthesized. Function *evaluate*, formalized below, receives two explicit inputs, an expression and a store, and returns the evaluation of the expression with respect to the store. If an expression uses a variable operator, then its evaluation needs extra stores each of which keeps track of the values of variables according to the semantics of the operator. Formally,

$$\begin{array}{lcl} evaluate((exp_1 * exp_2), V) & = & evaluate(exp_1, V) \circledast evaluate(exp_1, V), \\ evaluate(v, V) & \equiv & V(v), \\ evaluate(\texttt{pre}(v), V) & = & V_{\text{RHS B.S.}}(v), \\ evaluate(\texttt{cur}(v), V) & = & V_{\text{RHS S.S.}}(v), \\ evaluate(\texttt{new}(v), V) & = & V_{\text{RHS S.S.}}(v), \end{array}$$

where $*$ is the syntax for an arithmetic or a boolean operator and $\circledast$ is an operator representing the semantics of $*$. For example, if $*$ is addition over integers variables, then $\circledast$ represents the semantics of addition over integer values. Snapshot elements $V_{\text{RHS B.S.}}$ and $V_{\text{RHS S.S.}}$, used for the formalization of the RHS Big Step and the RHS Small Step semantic options, are defined in Section 4.5.4, where the formalization of assignment memory protocols are presented.

The execution of a small step of a model includes capturing the effects of the assignments of the transitions of the small step and storing them in the destination snapshot of the small step. Using the *evaluate* function, the semantics of assignment in a BSML can be defined. Relation *assign*, formalized below, has four parameters: a set of assignments, A, the snapshot element that captures the effects of assignments so far in computation, V_1, the snapshot element that the RHS of assignments are evaluated against, V_2, and the snapshot element that captures the effects of executing A, V_3. Snapshot elements V_1, V_2, and V_3 are all stores and are defined over all variables of a model; i.e., the size of each store is the size of the set of the variables of the model. In the absence of any race condition, relation *assign* is a function that receives A, V_1, and V_2, and determines V_3 deterministically. If race condition is possible, i.e., if it is possible that more than one assignments in A assign values to a variable, the *assign* is not a function, because it chooses one of the values assigned to the variable non-deterministically. Formally,

$$\begin{array}{lcl} assign(A, V_1, V_2, V_3) & \equiv & [\forall (v, val) \in V_1 \cdot v \notin lhs(A) \Rightarrow (v, val) \in V_3] \wedge \\ & & [\forall v \in lhs(A) \cdot \exists a \in A \cdot (v = lhs(a)) \wedge \\ & & \ (val = evaluate(rhs(a), V_2)) \wedge (v, val) \in V_3], \end{array}$$

where $lhs(A) = \bigcup_{a \in A} lhs(a)$.

The “`new_small`” variable operator leads to a transition-aware semantics, even in the absence of cyclic evaluation order as described in Section 3.6. The problem is that in the presence of a “`new_small(`v`)`” in the GC of a transition, t, whether t is enabled or not depends on whether v is assigned a value by another transition, t', which may or may not be enabled or executed in the current small step.

GC Big Step

Snapshot element $V_{\text{GC B.S.}}$, throughout a big step, maintains the values of the variables of a model the same as at the beginning of the big step. As described earlier, the snapshot element that $\mathcal{V}_{asn}$ refers to provides the values of variables at the beginning of a big step according to the Assignment Memory Protocol.

$$\begin{array}{lcl}
\texttt{reset\_}V_{\text{GC B.S.}}(sp^0, I, sp) & \equiv & V_{\text{GC B.S.}} = \mathcal{V}_{asn} \\
\texttt{next\_}V_{\text{GC B.S.}}(sp, \tau, sp') & \equiv & V'_{\text{GC B.S.}} = V_{\text{GC B.S.}} \\
\texttt{en\_}V_{\text{GC B.S.}}(t, sp) & \equiv & evaluate(gc(t), V_{\text{GC B.S.}})
\end{array}$$

GC Combo Step

Snapshot element $V_{\text{GC C.S.}}$ is a store for the variables of a model that maintains the same values for the variables during a combo step. At the beginning of each combo step (including the first combo step), the values of variables according to the assignment memory protocol of the BSML, which are stored in the snapshot element that $\mathcal{V}_{asn}$ refers to, are assigned to $V_{\text{GC C.S.}}$. When a small step is executed, using predicate $EndC$, it is checked whether the current combo step ends, in which case the value of $V_{\text{GC C.S.}}$ is updated. The definition of $EndC$ is the same as the definition of $EndC$ used in the formalization of the Present In Next Combo Step event lifeline semantics. When the GC Combo Step semantic option is chosen, then $V_{\text{GC C.S.}} \in Cs$.

$$\begin{array}{lcl}
\texttt{reset\_}V_{\text{GC C.S.}}(sp^0, I, sp) & \equiv & V_{\text{GC C.S.}} = \mathcal{V}_{asn} \\
\texttt{next\_}V_{\text{GC C.S.}}(sp, \tau, sp') & \equiv & V'_{\text{GC C.S.}} = \text{if } \; EndC \; \text{ then} \\
 & & \qquad\qquad V'_{Uptodate} \\
 & & \qquad \text{else} \\
 & & \qquad\qquad V_{\text{GC C.S.}} \\
\texttt{en\_}V_{\text{GC C.S.}}(t, sp) & \equiv & evaluate(gc(t), V_{\text{GC C.S.}})
\end{array}$$

And, $EndC \equiv (\nexists \tau' \cdot \tau' \in executable(root, sp' \oplus Cs))$.

$$\begin{array}{lcl}
\texttt{reset\_}V_{Uptodate}(sp^0, I, sp) & \equiv & \mathcal{V}_{asn} \\
\texttt{next\_}V_{Uptodate}(sp, \tau, sp') & \equiv & assign(\bigcup_{t \in \tau} asn(t), V_{Uptodate}, \mathcal{V}_{asn}, V'_{Uptodate})
\end{array}$$

GC Small Step

Snapshot element $V_{\text{GC S.S.}}$ is a store for the variables of a model that maintains the up-to-date values for the variables during a big step. At the beginning of each big step, the values of variables according to the assignment memory protocol of the BSML, which are stored in the snapshot element that $\mathcal{V}_{asn}$ refers to, are assigned to $V_{\text{GC S.S.}}$.

$$\begin{array}{lcl}
\texttt{reset\_}V_{\text{GC S.S.}}(sp^0, I, sp) & \equiv & V_{\text{GC S.S.}} = \mathcal{V}_{asn} \\
\texttt{next\_}V_{\text{GC S.S.}}(sp, \tau, sp') & \equiv & assign(\bigcup_{t \in \tau} asn(t), V_{\text{GC S.S.}}, \mathcal{V}_{asn}, V'_{\text{GC S.S.}}) \\
\texttt{en\_}V_{\text{GC S.S.}}(t, sp) & \equiv & evaluate(gc(t), V_{\text{GC S.S.}})
\end{array}$$

Interface Variables in GC

The following snapshot elements together specify the GC Small Step semantics for internal variables and the GC Asynchronous Variable semantics for interface variables. The set of interface variables of a model that are used in guard condition of transitions are denoted by $IntVarsGC$. Set A determines the set of assignments to the interface variables. Similar to internal variables, in order to specify the values of interface variables at the beginning of a big step, the snapshot element that maintains the values of variables according to the assignment memory protocol must be known; this snapshot element is denoted as $\mathcal{V}_{asnInt}$. Snapshot element $V_{\text{GC S.S.[Asynch]}}$ specifies the enabledness memory protocol of internal variables, whereas the $V_{\text{GC A.V.}}$ determines the enabledness memory protocol of interface variables.

$$\begin{array}{lcl}
\texttt{reset\_}V_{\text{GC S.S.[Asynch]}}(sp^0, I, sp) & \equiv & V_{\text{GC S.S.[Asynch]}} = \mathcal{V}_{asn} \\
\texttt{next\_}V_{\text{GC S.S.[Asynch]}}(sp, \tau, sp') & \equiv & assign(A, V_{\text{GC S.S.[Asynch]}}, (\mathcal{V}_{asn} \cup \mathcal{V}_{asnInt}), V'_{\text{GC S.S.[Asynch]}}) \\
\texttt{en\_}V_{\text{GC S.S.[Asynch]}}(t, sp) & \equiv & evaluate(gc(t), (V_{\text{GC S.S.[Asynch]}} \cup V_{\text{GC A.V.}}))
\end{array}$$

And, $A = \{a | \; \exists t \in \tau \cdot a \in asn(t) \wedge lhs(a) \notin IntVarsGC\}$.

$$\begin{array}{lcl}
\texttt{reset\_}V_{\text{GC A.V.}}(sp^0, I, sp) & \equiv & V_{\text{GC A.V.}} = \mathcal{V}_{asnInt} \\
\texttt{next\_}V_{\text{GC A.V.}}(sp, \tau, sp') & \equiv & V'_{\text{GC A.V.}} = V_{\text{GC A.V.}}
\end{array}$$

Similar to the Strong Synchronous Event for interface events, the GC Strong Synchronous Variable enabledness memory protocol for interface variables is a transition-aware semantics: The value of a variable assigned later in a big step should be sensed by a snapshot earlier in the big step, so that it can enable transitions according to that value, rather than a stale value.

4.5.4 Assignment Memory Protocol

This section presents the formalization of the semantic options for the Assignment Memory Protocol semantic aspect. First, the formalization of the memory protocols for internal variables is presented, followed by examples of formalizing the semantic options of interface variables.

RHS Big Step

Snapshot element $V_{\text{RHS B.S.}}$ maintains the values of the variables the same throughout a big step, and is used to evaluate the RHS of assignments according to the RHS Big Step semantics. Snapshot element $V_{Current}$ keeps track of the values of variables, as assignments are carried out through a big step, to deliver these new values to the next big step. It is initialized with $V_{\text{RHS B.S.}}$, which, in turn, is initialized by "$V^0_{Current} \uplus I.asns$". When the RHS Big Step semantics is chosen, $\mathcal{V}_{asn} = V_{\text{RHS B.S.}}$.

$$\begin{array}{lcl}
\texttt{reset\_}V_{\text{RHS B.S.}}(sp^0, I, sp) & \equiv & V_{\text{RHS B.S.}} = V^0_{Current} \uplus I.asns \\
\texttt{next\_}V_{\text{RHS B.S.}}(sp, \tau, sp') & \equiv & V'_{\text{RHS B.S.}} = V_{\text{RHS B.S.}} \\
 & & \\
\texttt{reset\_}V_{Current}(sp^0, I, sp) & \equiv & V_{Current} = V_{\text{RHS B.S.}} \\
\texttt{next\_}V_{Current}(sp, \tau, sp') & \equiv & assign(\bigcup_{t \in \tau} asn(t), V_{Current}, V_{\text{RHS B.S.}}, V'_{Current})
\end{array}$$

RHS Combo Step

Snapshot element $V_{\text{RHS C.S.}}$ maintains the values of the variables the same during a combo step, which can be used to evaluate the RHS of assignments according to the values of variables at the beginning of the current combo step. When the RHS Combo Step semantics is chosen, $\mathcal{V}_{asn} = V_{\text{RHS C.S.}}$ and $V_{\text{RHS C.S.}} \in Cs$. The definition of $EndC$ is the same as the definition of $EndC$ used in the formalization of Present In Next Combo Step event lifeline semantics.

$$\begin{array}{lcl}
\texttt{reset\_}V_{\text{RHS C.S.}}(sp^0, I, sp) & \equiv & V_{\text{RHS C.S.}} = V_{CurCombo} \\
\texttt{next\_}V_{\text{RHS C.S.}}(sp, \tau, sp') & \equiv & V'_{\text{RHS C.S.}} = \text{if } \ EndC \ \text{ then} \\
& & \qquad\qquad\quad V'_{CurCombo} \\
& & \qquad\qquad \text{else} \\
& & \qquad\qquad\quad V_{\text{RHS C.S.}}
\end{array}$$

And, $EndC \equiv (\nexists \tau' \cdot \tau' \in executable(root, sp' \oplus Cs))$.

$$\begin{array}{lcl}
\texttt{reset\_}V_{CurCombo}(sp^0, I, sp) & \equiv & V^0_{CurCombo} \uplus I.asns \\
\texttt{next\_}V_{CurCombo}(sp, \tau, sp') & \equiv & assign(\bigcup_{t \in \tau} asn(t), V_{CurCombo}, V_{\text{RHS C.S.}}, V'_{CurCombo})
\end{array}$$

RHS Small Step

Snapshot element $V_{\text{RHS S.S.}}$ keeps track of up-to-date values of variables, which can be used to evaluate the RHS of assignments according to the up-to-date values of variables. When the RHS Small Step semantics is chosen, $\mathcal{V}_{asn} = V_{\text{RHS S.S.}}$.

$$\begin{array}{lcl}
\texttt{reset\_}V_{\text{RHS S.S.}}(sp^0, I, sp) & \equiv & V_{\text{RHS S.S.}} = V^0_{\text{RHS S.S.}} \uplus I.asns \\
\texttt{next\_}V_{\text{RHS S.S.}}(sp, \tau, sp') & \equiv & assign(\bigcup_{t \in \tau} asn(t), V_{\text{RHS S.S.}}, V_{\text{RHS S.S.}}, V'_{\text{RHS S.S.}})
\end{array}$$

Interface Variables in RHS

The following snapshot elements together specify the GC Small Step semantics for variables in GC of transitions, the RHS Small Step semantics for internal variables in RHS of assignment, the GC Weak Synchronous Variable semantics for interface variables in GC of transitions, and the RHS Asynchronous Variable semantics for interface variables in RHS of assignments. The set of interface variables of a model that are used in the GC and RHS of assignments of transitions are denoted as $IntVarsGC$ and $IntVarsRHS$, respectively. Sets A and B determine the set of assignments to the non-interface and interface variables, respectively. Snapshot elements $V_{\text{GC S.S.[NonInterface]}}$ and $V_{\text{RHS S.S.[NonInterface]}}$ specify the enabledness and assignment memory protocols of internal variables, respectively. Snapshot elements $V_{\text{GC W.S.V.}}$ and $V_{\text{RHS A.V.}}$ specify the enabledness and assignment memory protocols of interface variables, respectively. In this semantics, $\mathcal{V}_{asn} = V_{\text{RHS S.S.[NonInterface]}}$ and $\mathcal{V}_{asnInt} = V_{\text{RHS A.V.}}$.

$$
\begin{aligned}
\texttt{reset\_}V_{\text{GC S.S.[NonInterface]}}(sp^0, I, sp) &\equiv V_{\text{GC S.S.[NonInterface]}} = \mathcal{V}_{asn}\\
\texttt{next\_}V_{\text{GC S.S.[NonInterface]}}(sp, \tau, sp') &\equiv assign(A, V_{\text{GC S.S.[NonInterface]}},\\
&\qquad\qquad (\mathcal{V}_{asn} \cup \mathcal{V}_{asnInt}),\\
&\qquad\qquad V'_{\text{GC S.S.[NonInterface]}})\\
\texttt{en\_}V_{\text{GC S.S.[NonInterface]}}(t, sp) &\equiv evaluate(gc(t), V_{\text{GC S.S.[NonInterface]}} \cup V_{\text{GC W.S.V.}})
\end{aligned}
$$

$$
\begin{aligned}
\texttt{reset\_}V_{\text{RHS S.S.[NonInterface]}}(sp^0, I, sp) &\equiv V_{\text{RHS S.S.[NonInterface]}} = V^0_{\text{RHS S.S.[NonInterface]}} \uplus I.asns\\
\texttt{next\_}V_{\text{RHS S.S.[NonInterface]}}(sp, \tau, sp') &\equiv assign(A, V_{\text{RHS S.S.[NonInterface]}}, (\mathcal{V}_{asn} \cup \mathcal{V}_{asnInt}),\\
&\qquad\qquad V'_{\text{RHS S.S.[NonInterface]}})
\end{aligned}
$$

And, $A = \{a | \exists t \in \tau \cdot a \in asn(t) \wedge lhs(a) \notin (IntVarsGC \cup IntVarsRHS)\}$.

$$
\begin{aligned}
\texttt{reset\_}V_{\text{GC W.S.V.}}(sp^0, I, sp) &\equiv V_{\text{GC W.S.V.}} = \mathcal{V}_{asnInt}\\
\texttt{next\_}V_{\text{GC W.S.V.}}(sp, \tau, sp') &\equiv assign(B, V_{\text{GC W.S.V.}}, (\mathcal{V}_{asnInt} \cup \mathcal{V}_{asn}), V'_{\text{GC W.S.V.}})
\end{aligned}
$$

$$
\begin{aligned}
\texttt{reset\_}V_{\text{RHS A.V.}}(sp^0, I, sp) &\equiv V_{\text{RHS A.V.}} = V^0_{\text{RHS A.V.}} \uplus I.asns\\
\texttt{next\_}V_{\text{RHS A.V.}}(sp, \tau, sp') &\equiv assign(B, V_{\text{RHS A.V.}}, (\mathcal{V}_{asnInt} \cup \mathcal{V}_{asn}), V'_{\text{RHS A.V.}})
\end{aligned}
$$

And, $B = \{a \mid \exists t \in \tau \cdot a \in asn(t) \wedge lhs(a) \in (IntVarsGC \cup IntVarsRHS)\}$.

The RHS Strong Synchronous Variable assignment memory protocol for interface variables, similar to the GC Strong Synchronous Variable enabledness memory protocol for interface variables, is a transition-aware semantics: The value of a variable assigned later in a big step should be sensed by a snapshot earlier in the big step, so that the evaluation of the RHS of an assignment is done using the new value.

4.5.5 Order of Small Steps

This section presents the formalization of the Explicit Ordering and Dataflow semantic options. The None semantic option does not require any snapshot elements.

Explicit Ordering

The Explicit Ordering order of small steps is relevant for the transitions within the scope of an *And* control state. The execution of transitions whose scopes are within an *And* control state are ordered according to their graphical order. As discussed in Section 3.10, the Explicit Ordering semantic option should be chosen together with the Single concurrency semantics and the Take One big-step maximality semantics, otherwise, the notion of ordering according to a graphical order of control states does not make sense. In the formalization below, it is assumed that the Single and the Take One semantic options are chosen together with the Explicit Ordering semantic option in a BSML semantics, as it is the case in Stateflow [22], which subscribes to the Explicit Ordering semantics. Before presenting the formal semantics of the Explicit Ordering semantic option, some notation are introduced.

For each *And* control state, s, the *graphical order* of its compound children is denoted by $go(s)$, which is a sequence of control states, $\langle s_1, \cdots, s_m \rangle$. If the scope of a transition, t, is nested inside more than one *And* control states, then it will be ordered by all those *And* control states. A transition, t_1, *graphically precedes* another transition, t_2, if according to the *And* control states that order the two transitions t_1 must execute before t_2. Because of the Single and the Take One semantic options, however, such two transitions need to be compared only according to the graphical order of the lowest *And* control state that is an ancestor of the scopes of both transitions. For a transition, t, its *graphical predecessors*, denoted by $gpre(t)$, is the set of all transitions t' that graphically precede t.

Snapshot element $O_{\textsc{Explicit}}$ declares a transition, t, as enabled if it is its turn to be executed according to the graphical order of the *And* control states it belongs to; i.e., either the graphical predecessors of t have been executed already, or they are not enabled. The type of snapshot element $O_{\textsc{Explicit}}$ is a set of transitions each of which specifies whether a transition of the model, discarding the Explicit Ordering semantics, is enabled or not. The set of snapshot elements of a BSML is denoted by $SpEl = \{el_1, el_2, \cdots, el_n\}$.

$$
\begin{aligned}
\texttt{reset\_}O_{\textsc{Explicit}}(sp^0, I, sp) &\equiv O_{\textsc{Explicit}} = \{t \mid \bigwedge_{\substack{1 \le i \le n \\ el_i \ne O_{\textsc{Explicit}}}} \texttt{en\_}el_i(t, sp)\} \\
\texttt{next\_}O_{\textsc{Explicit}}(sp, \tau, sp') &\equiv O'_{\textsc{Explicit}} = \{t \mid \bigwedge_{\substack{1 \le i \le n \\ el_i \ne O_{\textsc{Explicit}}}} \texttt{en\_}el_i(t, sp')\} \\
\texttt{en\_}O_{\textsc{Explicit}}(t, O_{\textsc{Explicit}}) &\equiv O_{\textsc{Explicit}} \cap gpre(t) = \emptyset
\end{aligned}
$$

The $\texttt{en\_}O_{\text{EXPLICIT}}$ predicate checks whether any of the graphic predecessors of a transition are enabled. If a graphical predecessor transition of the transition has already been executed, it cannot be enabled at the current snapshot, because of the TAKE ONE semantics.

The above formalization would work only if the definitions of the snapshot elements in Cs are adjusted so that they do not refer to snapshot O'_{EXPLICIT} in their "`next`" predicates. Instead, for example, these definitions could refer to the value of snapshot element O_{EXPLICIT} at sp, instead of sp', or entirely discard the role of snapshot element O_{EXPLICIT}. Both solutions are fine since the EXPLICIT ORDERING semantic option has no role in determining the end of a big step or a combo step; instead, when there are enabled transitions to be executed, it orders them.

DATAFLOW

Snapshot element O_{DATAFLOW} declares a transition, t, as enabled if all variables in $prefix\_new(t)$ are assigned values during the current big step, where $prefix\_new(t)$ returns the set of variables prefixed by `new` that are used in $gc(t)$ or in the RHS of an assignment in $asn(t)$.

$$\begin{array}{lcl}
\texttt{reset\_}O_{\text{DATAFLOW}}(sp^0, I, sp) & \equiv & O_{\text{DATAFLOW}} = \emptyset \\
\texttt{next\_}O_{\text{DATAFLOW}}(sp, \tau, sp') & \equiv & O'_{\text{DATAFLOW}} = \bigcup_{t \in \tau} \bigcup_{a \in asn(t)} lhs(a) \cup O_{\text{DATAFLOW}} \\
\texttt{en\_}O_{\text{DATAFLOW}}(t, sp) & \equiv & prefix\_new(t) - O_{\text{DATAFLOW}} = \emptyset
\end{array}$$

4.5.6 Combo-Step Maximality

The formalization of the semantic options of the Combo-Step Maximality semantics is related to the formalization of the combo-step semantics, as described in Section 4.5.2. As such, the snapshot elements involved in formalizing the Combo-Step Maximality semantics belong to the combo-step snapshot elements Cs. The predicate $EndC$ used in this section, which determines the end of a combo step, is the same as in the formalization of other combo-step semantics. Similar to the Take Many big-step maximality semantics, the COMBO TAKE MANY semantic option does not introduce any snapshot elements.

COMBO SYNTACTIC

During a combo step, snapshot $C_{\text{COMBO SYNTACTIC}}$ collects the control states that each is the arena or a child of the arena of a transition that enters a combo stable control state. The predicate

$\mathtt{en\_}C_{\textsc{Combo Syntactic}}(t, sp)$ determines whether a transition, r, has already been executed during the combo step that has entered a combo stable control state, s such that $arena(t) \in children^*(arena(r)$, in which case t cannot be taken in the current big step. When the COMBO SYNTACTIC semantic option is chosen $C_{\textsc{Combo Syntactic}}, C_{ArenaCollectSyn} \in Cs$.

$$
\begin{array}{lcl}
\mathtt{reset\_}C_{\textsc{Combo Syntactic}}(sp^0, I, sp) & \equiv & C_{\textsc{Combo Syntactic}} = \emptyset \\
\mathtt{next\_}C_{\textsc{Combo Syntactic}}(sp, \tau, sp') & \equiv & C'_{\textsc{Combo Syntactic}} = \text{if } EndC \text{ then} \\
 & & \qquad\qquad C_{ArenaCollectSyn} \cup A \\
 & & \qquad \text{else} \\
 & & \qquad\qquad C_{\textsc{Combo Syntactic}} \\
\mathtt{en\_}C_{\textsc{Combo Syntactic}}(t, sp) & \equiv & arena(t) \notin C_{\textsc{Combo Syntactic}} \\
 & & \\
\mathtt{reset\_}C_{ArenaCollectSyn}(sp^0, I, sp) & \equiv & C_{ArenaCollectSyn} = \emptyset \\
\mathtt{next\_}C_{ArenaCollectSyn}(sp, \tau, sp') & \equiv & C'_{ArenaCollectSyn} = \text{if } EndC \text{ then} \\
 & & \qquad\qquad \emptyset \\
 & & \qquad \text{else} \\
 & & \qquad\qquad C_{ArenaCollectSyn} \cup A
\end{array}
$$

And, $A = \bigcup_{t \in \tau}\{s \mid s \in children^*(arena(t)) \ \wedge combostable(dest(t))\}$, and $EndC \equiv (\nexists \tau' \cdot \tau' \in executable(root, sp' \oplus Cs))$.

COMBO TAKE ONE

During a big step, snapshot $M_{\textsc{Combo Take One}}$ collects the control states that each is the arena or the child of the arena an executed transition. The predicate $\mathtt{en\_}M_{\textsc{Combo Take One}}(t, sp)$ determines whether a transition, r, has already been executed during the big step such that $arena(t) \in children^*(arena(r)$, in which case t cannot be taken in the current big step. When the COMBO TAKE ONE semantic option is chosen $C_{\textsc{Combo Take One}}, C_{ArenaCollectOne} \in Cs$.

$$
\begin{array}{lcl}
\texttt{reset\_}C_{\text{Combo Take One}}(sp^0, I, sp) & \equiv & C_{\text{Combo Take One}} = \emptyset \\
\texttt{next\_}C_{\text{Combo Take One}}(sp, \tau, sp') & \equiv & C'_{\text{Combo Take One}} = \text{if } \; EndC \; \text{ then} \\
 & & \qquad\qquad C_{ArenaCollectOne} \cup B \\
 & & \qquad \text{else} \\
 & & \qquad\qquad C_{\text{Combo Take One}} \\
\texttt{en\_}C_{\text{Combo Take One}}(t, sp) & \equiv & arena(t) \notin C_{\text{Combo Take One}} \\
 & & \\
\texttt{reset\_}C_{ArenaCollectOne}(sp^0, I, sp) & \equiv & C_{ArenaCollectOne} = \emptyset \\
\texttt{next\_}C_{ArenaCollectOne}(sp, \tau, sp') & \equiv & C'_{ArenaCollectOne} = \text{if } \; EndC \; \text{ then} \\
 & & \qquad\qquad \emptyset \\
 & & \qquad \text{else} \\
 & & \qquad\qquad C_{ArenaCollectOne} \cup B
\end{array}
$$

And, $B = \bigcup_{t \in \tau} \{ s \mid s \in children^*(arena(t)) \}$, and
$EndC \equiv (\nexists \tau' \cdot \tau' \in executable(root, sp' \oplus Cs))$.

The Structure of a BSML Semantics A complete BSML semantics can be instantiated by choosing the desired semantic options of the semantic aspects of interest. The chosen semantic options of the structural semantic aspects of the BSML semantics determine a parsing mechanism (when a hierarchical semantic option is chosen), together with values for the corresponding predicates of structural parameters. The chosen semantic options of the enabledness semantic aspects determine a set of snapshot elements that implement those semantic options.

4.6 Related Work: Semantic Formalization Methods

My semantic formalization is influenced by the formalization in template semantics [75, 74]. In particular, (i) lines 1-5 in Figure 4.2 are adopted from the definition of *macro step* in template semantics; and (ii) the notion of snapshot elements in template semantics is adapted to model the enabledness parameters of BSML semantics. In template semantics, a snapshot element has a type and a set of three *parameters*, *reset*, *next*, and *enabled*, which can be instantiated with a *value*, from a pre-determined, extensible set of values. There are a fixed, but extensible, number of snapshot elements that can be instantiated to obtain a semantics. While the snapshot elements in template semantics are the semantic variation points by themselves, in my framework, the semantic variation points are semantic aspects and semantic options; snapshot elements are a mechanism to formalize these semantic variations. A semantic option may require multiple snapshot elements for its formalization.

Template semantics has a notion of composition operator, which I do not need, because: first, some of the characteristics of the composition operators can be modelled using different structural semantic options; and second, as Chapter 6 will describe, the semantics of many composition operators can be specified using the synchronization mechanism introduced in Chapter 6. I also introduce the notion of combo step, which was not considered in template semantics. The main divergence in my semantic definition framework from template semantics is that my proposed framework produces a semantic definition whose elements corresponds to the semantic aspects and the semantic options. To the best of my knowledge, the approach presented in formalizing the semantic aspects of the deconstruction is the first one that defines disjoint parameters in a manner that matches the factoring into the structural semantic aspects from the high-level big-step semantic deconstruction.

My work is comparable with *tool-support generator frameworks* (TGFs), which by accepting the definition of a notation, including its semantics, as input, generate tool support, such as model checking and simulation capability, as output [81, 24, 25, 28, 65, 6, 38]. TGFs differ in the *semantic input formats* (SIF) they use, and the procedure by which they obtain tool support for a notation. An SIF, by its function, is a semantic definition language, and thus can be potentially compared with our semantic definition framework. Some TGFs adopt an existing formalism as their SIF; for example, higher-order logic [24, 25], structural operational semantics [28], graph grammars [6], and forwarding attribute grammars [38]; others devise their own SIFs; for example, *execution rules* [81], which defines a semantics via its *enabling*, *matching*, and *firing rules*, and template semantics [65], which defines a semantics by instantiating values for *semantic parameters* and choosing or defining a set of *composition operators*. While TGFs strive for flexibility and extensibility, to accommodate new notations, I have strived to create a systematic semantic definition framework that clearly defines a BSML semantics.

A precise comparison of my semantic definition framework with the SIF of a TGF requires knowledge about the range of semantics that the SIF can express, or is meant to express. However, the range of semantics that an SIF can express is usually left as unspecified, or underspecified. I think that this is not accidental, and is a result of ambiguity about the domain of notations that a TGF is designed for. I argue that for a family of notations, a task similar to what

I undertook for BSMLs should precede the attempt to develop a TGF for them. Otherwise, it is not possible to determine the range of the family of notations that the TGF supports, unless a TGF aims for universality. Furthermore, the users of the TGF cannot fully benefit from it, unless they independently discover the expressiveness of the SIF of the TGF. However, discovering this is not straightforward: SIFs are designed with flexibility and extensibility goals in mind, rather than systematicness and clarity.

The mere choice of a SIF will not likely address the difficulty of reconciling the flexibility and extensibility of a TGF with the systematicness and clarity of its SIF, as described above. For example, choosing a general SIF, such as structural operational semantics [28] or forwarding attribute grammars [38], might seem a good idea because it provides a certain level of systematicness and clarity, and hopefully flexibility and extensibility would follow. But I observe that researchers either report about supporting a limited set of notations (e.g., variations of "Lotos subset", without variables [28]), or report about difficulties with extensibility (e.g., difficulty in modelling the semantics of "events", because the semantic definition is "not trivial" and becomes "verbose" [38]). Conversely, devising a specific SIF, such as "execution rules" [81] and "template semantics" [65], might seem a good idea because it provides flexibility and extensibility, and hopefully systematicness and clarity would follow. But I observe that the flexibility and extensibility in such a framework is with respect to its SIF, and does not necessarily translate to clarity and/or systematicness for users. As an example, the ability to define a semantics that is a mixture of the semantics of statecharts and Petri nets in an extensible way [81], does not necessarily mean that a user of the TGF would perceive its SIF as systematic, and a resulting semantics as clear.

The *dynamic* semantic concepts in the graph-transformational semantic definition approach for UML statecharts by Varró [98] is similar to the notions of enabledness semantic aspects and enabledness parameters in my semantic definition schema. Varró's approach, however, is only considered for one language with a simple syntax that supports a simple kind of control states and asynchronous events. The *static* semantic concepts in his approach are comparable to the syntactic helper functions in Table 4.1 and Table 4.2. His graph-transformational semantic definition approach can be considered as a prescriptive semantic definition method, because each of the dynamic and static semantic concepts correspond to distinct graph transformation rules. However, this method is applied to a single semantics, and thus the scope of the languages that it can support is not clear. In particular, in the presence of variables and our structural semantic aspects, it would be interesting to investigate whether these rules can be extended to cover a range of different semantics, and yet maintain a prescriptive semantic definition method.

My semantic definition framework shares the same goals as other general semantic definition methods that advocate clarity and systematicness [73, 49]. In *action semantics* [73], a semantic definition can be organized as a hierarchy of *modules* and *sub-modules*. Furthermore, conceptually, a semantic definition can be decomposed across two axes: *types of information*, which distinguish between *transient*, *scoped*, *stable*, and *permanent* data; and *facets of actions*, which distinguish between *basic*, *functional*, *declarative*, *imperative*, and *communicative* processing modes, each of which is designated to process a specific *type of information*. Similar to action semantics, my definition framework is *modular* in that a semantics can be incrementally defined by specifying its different aspects. The *unifying theory of programming* aims for the vision of the unification of different paradigms of programming languages and their semantics, which "can be described at different levels of abstraction" [49]. A notion of *theory* describes an aspect of computation, such as non-determinism or recursion. A theory is described in terms of its alphabets, signatures, relations, functions, and axioms. More primitive theories are *refined* to derive more specific ones. The link between different theories can be defined through *linking theories*. The unifying theory of programming is a general vision to understand the different paradigms of computation and their relationships, rather than a particular method for semantic definition. My proposed semantic definition framework is consistent with the vision of a unifying theory of programming in that it introduces semantic aspects that lend themselves to the kind of analysis advocated in the unifying theory of programming. Also, at a high level, perhaps, the basic BSML semantics could be considered as a theory, which can be related to various BSML semantics through the semantic aspects/options linkings.

Lastly, my work is comparable to that of Huizing and Gerth [50]. Huizing and Gerth categorize and specify the semantics of simple BSMLs that only have events. In comparison, my semantic definition framework considers a more advanced normal-form syntax, resulting in considering a wide range of semantic aspects and options, in addition to the event lifeline semantics.

4.7 Summary

This chapter introduced a formal semantic definition method that uniformly formalizes most of the BSML semantics of the big-step semantic deconstruction. This semantic definition method is bases on a semantic definition schema that is parametric with respect to the semantic aspects of the big-step semantic deconstruction. A semantic definition of a BSML produced in this method is prescriptive in that the manifestation of the constituent semantic options of the BSML, according to the big-step semantic deconstruction, can be clearly identified in the semantic definition. The semantic definition schema can define most of the BSML semantics of the big-step semantic deconstruction, except those that are transition aware. In a transition-aware semantics, the enabledness of a transition depends on the execution of the semantics in the current or future small steps.

Chapter 5

Semantic Quality Attributes of BSMLs

"Languages differ essentially in what they *must* convey and not in what they *may* convey." [53, p.141]

Roman Jakobson

While a BSML provides a modeller with the convenience of describing the reaction of a system to an environmental input as the execution of a set of transitions, facilitating the decomposition of a model into concurrent components, it also introduces the complexity of dealing with the semantic intricacies related to the *ordering* of these transitions. In this chapter, three *semantic quality attributes* for BSMLs are introduced, each of which identifies a desirable semantic characteristic for BSML semantics that exempts a modeller from worrying about some of the complications of ordering in the sequence of the small steps of a big step.

For each semantic quality attribute, the necessary and sufficient constraints over the choices of the BSML semantic options are specified so that the resulting semantics each has the semantic quality attribute. As opposed to the advantages and disadvantages of each semantic option, which were discussed in Chapter 3, the characterization of a semantic quality attribute is a cross-cutting concern over different semantic aspects.

The remainder of this chapter is organized as follows. Section 5.1 presents the terminology that is used throughout the chapter. Section 5.2 formally presents the three semantic quality attributes for BSMLs, together with examples that describe the role of each semantic quality attribute. Section 5.3 specifies the set of BSML semantics that satisfy each of the semantic quality attributes, together with proofs of the correctness of each specification. Section 5.4 describes how a semantic quality attribute can be achieved through the choice of a set of semantic options together with a set of syntactic well-formedness criteria in a language. Section 5.5 discusses related work.

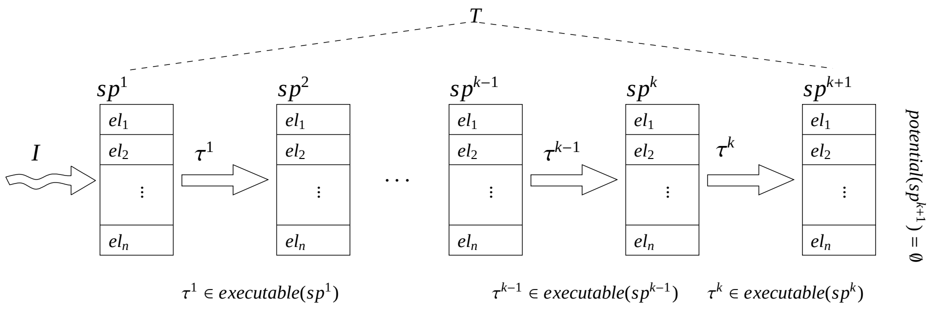
5.1 Quantification over Big Steps

This section describes terminology to quantify over the set of big steps of a BSML model and to declaratively access parts of a big step. This terminology facilitates the specification of the semantic quality attributes, as well as the proofs of the correctness of the characterization of the semantic quality attributes.

Figure 5.1, similar to Figure 4.1, on page 87, depicts the structure of a big step: After receiving an environmental input, I, k small steps are executed to arrive at snapshot sp^{k+1}. In this section, a big step is represented formally as a tuple, $\langle b, length, b' \rangle$, where b is the *beginning* snapshot of the $\underline{b}$ig step, *length* is the number of small steps in the big step, and b' is the destination snapshot of the big step, as usual. As an example, for the big step in Figure 5.1, $T.b = sp^1$, $T.length = k$, and $T.b' = sp^{k+1}$, where the operator "." is used to access an element of a tuple.

Compared to the formal semantics in Chapter 4, the formal representation of a big step used in this chapter adopts two different conventions. First, unlike in Figure 4.1, the beginning snapshot of a big step includes the effect of receiving the environmental input of the big step, as opposed to the "source" snapshot of the big step in the previous chapter, which needs to be "reset" with environmental inputs. And second, unlike the formal semantics of combo-step semantics in Chapter 4, here it is assumed that once a combo step ends, it is transitioned explicitly to a new snapshot that is the start of the new combo step. This approach is as opposed to the formal semantics of combo-steps in the previous chapter where upon detecting the last small step of a combo step, by using predicate *EndC*, the adjustments to start the new combo step happens together with the execution of the last small step of the current combo step. The above two conventions are adopted to simplify the formalization and the presentation of semantic quality attributes. It is straightforward to rephrase and re-formalize the content of this chapter if these conventions are not assumed.

The set of potential small steps of a model at a snapshot sp is denoted by $executable(root, sp)$, as usual. For the sake of brevity, I write "$executable(sp)$" instead of "$executable(root, sp)$"



T
sp^1
sp^2
sp^{k-1}
sp^k
sp^{k+1}
el_1
el_2
el_n
I
τ^1
τ^{k-1}
τ^k
$potential(sp^{k+1}) = \emptyset$
$\tau^1 \in executable(sp^1)$
$\tau^{k-1} \in executable(sp^{k-1})$
$\tau^k \in executable(sp^k)$

Figure 5.1: Big step $T = \langle sp^1, k, sp^{k+1} \rangle$.

because the first parameter of this function is not relevant to the formalization presented in this chapter. The ith small step of a big step, T, where $1 \leq i \leq T.length$, is denoted by T^i. Each small step itself is represented as a tuple, $\langle s, \tau, s' \rangle$, where s and s' are the source and destination snapshots of the small step, respectively, and τ is the set of transitions that are executed by the small step. For example, the destination snapshot of the i th small step of big step T is obtained by $T^i.s'$. For all T^i, $1 \leq i \leq T.length$, $T^i.\tau \in executable(T^i.s)$. Also, $T^j.s' = T^{j+1}.s$ for $1 \leq j < T.length$. For a BSML model M, the set of all its possible big steps is denoted as $bigsteps(M)$. This set includes all big steps in response to all environmental inputs at all possible snapshots. As usual, in examples, a big step is referred to by the sequence of the sets of transitions of its small steps, which is surrounded by a "$\langle\ \rangle$" pair.

The set of big steps at a snapshot is determined by the eight semantic aspects of BSMLs, as described in Figure 5.2. Recall that in the previous chapters these aspects were partitioned into two categories:

- Enabledness semantic aspects deal with the semantics of how a single transition can be included in a big step and what the effect of its execution; and
- Structural semantic aspects deal with how a set of enabled transitions can be taken together in a small step.

To describe the semantic quality attributes, the set of enabledness semantic aspects are partitioned further into two subcategories:

- *Transition-based* semantic aspects, which determine how the semantics of a BSML uses

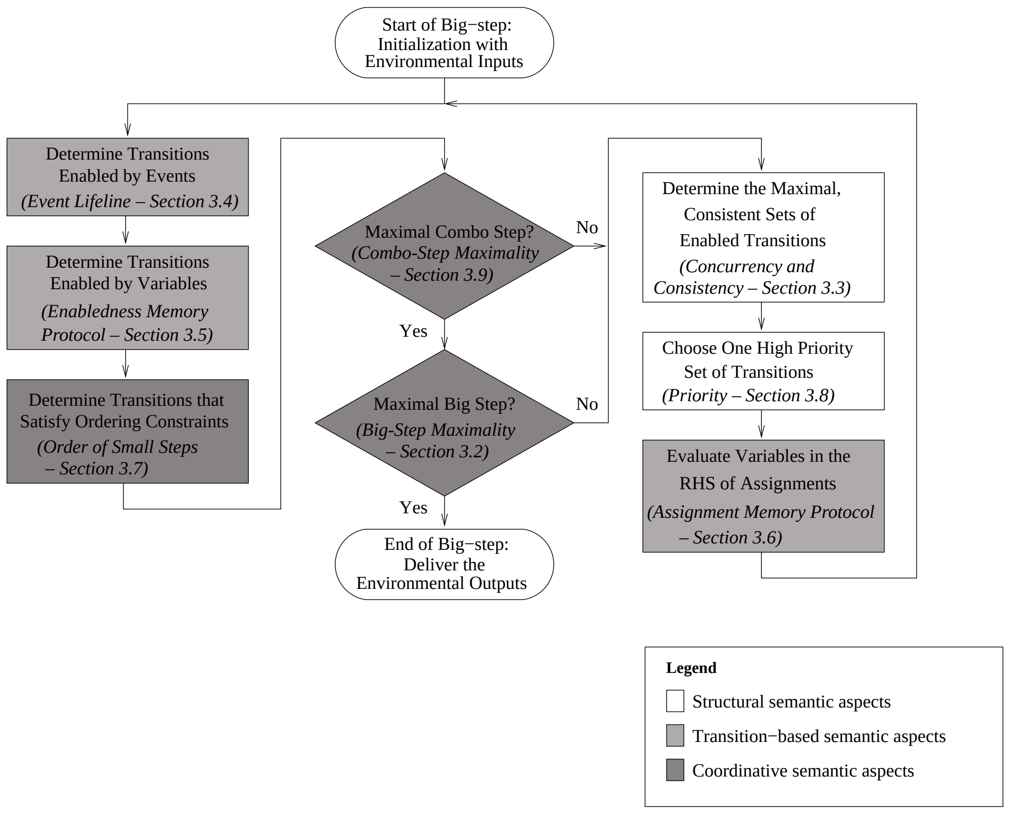

Start of Big−step: Initialization with Environmental Inputs
Determine Transitions Enabled by Events (Event Lifeline – Section 3.4)
Determine Transitions Enabled by Variables (Enabledness Memory Protocol – Section 3.5)
Determine Transitions that Satisfy Ordering Constraints (Order of Small Steps – Section 3.7)
Maximal Combo Step? (Combo-Step Maximality – Section 3.9)
No
Yes
Maximal Big Step? (Big-Step Maximality – Section 3.2)
No
Yes
End of Big−step: Deliver the Environmental Outputs
Determine the Maximal, Consistent Sets of Enabled Transitions (Concurrency and Consistency – Section 3.3)
Choose One High Priority Set of Transitions (Priority – Section 3.8)
Evaluate Variables in the RHS of Assignments (Assignment Memory Protocol – Section 3.6)
Legend
Structural semantic aspects
Transition−based semantic aspects
Coordinative semantic aspects

Figure 5.2: Operation of a big step through its structural, transition-based, and coordinative semantic aspects.

the modelling constructs of the language, namely its variables, events, and control states; and

- *Coordinative* semantic aspects, which determine how the execution of the transitions of a big step of a model are ordered and grouped across a big step.

From a modelling point of view, a transition-based semantic aspect is different from a coordinative semantic aspect in that the corresponding snapshot elements of a transition-based semantic option maintain information about the values of the syntactic elements of the transitions, whereas the corresponding snapshot elements of a coordinative semantic option maintain information about the history of the execution of the transitions in a big step.

The flowchart in Figure 5.2 is similar to the one in Figure 3.1, on page 30, except that it shows the partitioning of aspects into these categories. The stages of the flowchart with clear elements represent the structural semantic aspects of the operation of a big step; the other stages represent the enabledness semantic aspects of the operation of a big step. The light gray elements of the flowchart represent the transition-based, enabledness semantic aspects of the operation of a big step. The dark gray elements of the flowchart represent the coordinative, enabledness semantic aspects of the operation of a big step. The Event Lifeline, Enabledness Memory Protocol, and Assignment Memory Protocol semantic aspects are transition-based semantic aspects. The Big-Step Maximality, Combo-Step Maximality, and Order of Small Steps semantic aspects are coordinative semantic aspects. As an example, the Event Lifeline semantic aspect is a transition-based semantic aspect in that it determines how a generated event of a transition, i.e., a syntactic element of a transition of a model, triggers the transitions of the model, while the Big-Step Maximality semantic aspect is a coordinative semantic aspect in that it determines the limit on the number of transitions in a big step.

With this partitioning, the definition of the enabledness of a transition, from Section 4.1.2, on page 91, is divided into two parts, one for each set of enabledness semantic aspects,

$$\begin{aligned}
en(t, sp) &\equiv ready(t, sp) \wedge fireable(t, sp), \\
ready(t, sp) &\equiv \bigwedge_{x \in \mathcal{TransitionBased}(SpEl)} \mathsf{en\_}x(t, sp), \text{ and} \\
fireable(t, sp) &\equiv \bigwedge_{x \in \mathcal{Coordinative}(SpEl)} \mathsf{en\_}x(t, sp),
\end{aligned}$$

where $SpEl$, as before, is the set of snapshot elements used in the semantics of a BSML, and $\mathcal{TransitionBased}(SpEl)$ and $\mathcal{Coordinative}(SpEl)$ are the sets of *transition-based* and *coordinative* snapshot elements that are used in the definition of the transition-based and the coordinative semantic aspects, respectively. By definition, the snapshot element that maintains the current set of control states that a model resides in, i.e., snapshot element S_c described on page 92, belongs to $\mathcal{TransitionBased}(SpEl)$.

For a transition, t, at a snapshot sp, if $ready(t, sp)$ is true, it is called a *ready* transition, and otherwise an *unready* transition. Similarly, if $fireable(t, sp)$ is true, t is called a *fireable* transition, and otherwise an *unfireable* transition. If both $ready(t, sp)$ and $fireable(t, sp)$ are true, t is called an enabled transition, as usual.

A transition, t, at a snapshot, sp, is called *executable*, denoted by $executable(t, sp)$, if it belongs to at least one potential small step in that snapshot. Formally,

$$executable(t, sp) \equiv \exists \tau \in executable(sp) \wedge t \in \tau.$$

Figure 5.3, which depicts the structure of a semantic definition schema from the previous chapter, is the same as the one in Figure 4.3, on page 89, except that it is annotated to show the partitioning of the parameters of the formal semantic definition schema. Function *en_trs*, described in Section 4.1.2, receives a set of transitions and uses predicate *en* to return the set of enabled transitions in the set. By definition, in Chapter 4, and as can be traced in the figure, if $executable(t, sp)$ is true, so is $ready(t, sp)$ and $fireable(t, sp)$; i.e.,

$$executable(t, sp) \Rightarrow ready(t, sp) \wedge fireable(t, sp), \tag{5.1}$$

but, in general, not vice versa, because of the Priority semantic aspect, which corresponds to the "Π" parameter. A transition might be ready and fireable, but have a lower priority compared to another transition that can replace it in all potential small steps. The three sub-aspects of the Concurrency and Consistency semantic aspect, which correspond to parameters "$\|$", "C", and "P', each has a role in determining the set of potential small steps, by combining a set of ready, fireable transitions into a small step. However, these semantic sub-aspects do not have any role in determining whether a ready, fireable transition is executable or not: The priority semantics eventually determines that. If the NO PRIORITY priority semantics is chosen, however, i.e., if neither the SCOPE-PARENT nor the SCOPE-CHILD semantics is chosen, then

$$executable(t, sp) \Leftrightarrow ready(t, sp) \wedge fireable(t, sp), \tag{5.2}$$

which means that if a transition is both ready and fireable it belongs to at least one potential small step, and vice versa.

If a transition is not executable, it is called *unexecutable*.

Lastly, a transition, t, is *priority-ready*, at a snapshot, sp, denoted by, $priority\_ready(t, sp)$, if: (i) t is ready, and (ii) discounting the coordinative semantic aspects, t would belong to a potential small step. By definition, if $priority\_ready(t, sp)$ is true, then $executable(t, sp)$ is true only if $fireable(t, sp)$ is also true. Conversely, by the definition of an executable transition, if $executable(t, sp)$ is true, it should be the case that both $priority\_ready(t, sp)$ and $fireable(t, sp)$ are true. Formally,

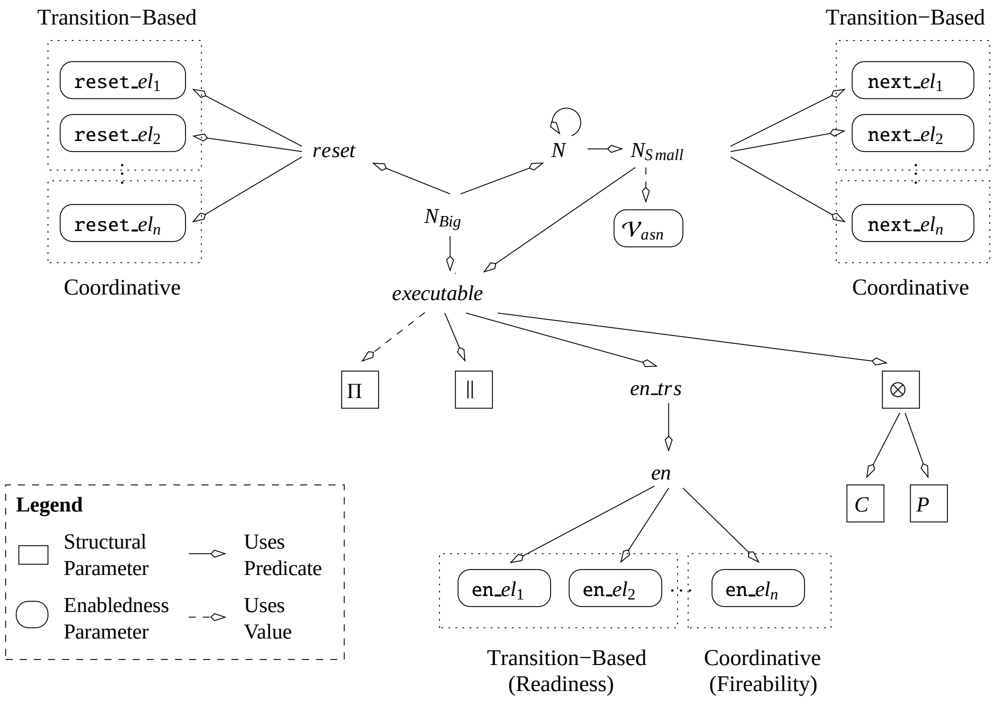


Figure 5.3: The structure of a semantic definition schema, from Chapter 4.

$$executable(t, sp) \Leftrightarrow priority\_ready(t, sp) \wedge fireable(t, sp). \qquad (5.3)$$

Predicate 5.3, as opposed to predicate 5.2, which is true for all BSML semantics without a hierarchial priority semantics, is true for all BSML semantics regardless of their priority semantics.

Table 5.1 summarizes the terminology presented so far in this section.

Table 5.1: Summary of terminology for semantic aspects.

Structural semantic aspects deal with how a set of transitions can be executed together to form a small step.	
Transition-based, enabledness semantic aspects deal with how a BSML uses the syntactic elements of a transition.	
$\mathcal{TransitionBased}(SpEl)$	The set of snapshot elements that model the transition-based, enabledness semantic aspects of a BSML.
$ready(t, sp)$	Transition t is ready at sp, and can be taken according to the snapshot elements in $\mathcal{TransitionBased}(SpEl)$.
Coordinative, enabledness semantic aspects deal with how the execution of transitions are coordinated across a big step.	
$\mathcal{Coordinative}(SpEl)$	The set of snapshot elements that model the coordinative, enabledness semantic aspects of a BSML.
$fireable(t, sp)$	Transition t is fireable at sp, and can be taken according to the snapshot elements in $\mathcal{Coordinative}(SpEl)$.
Enabledness	
$en(t, sp)$	Transition t is enabled at sp if and only if both $ready(t, sp)$ and $fireable(t, sp)$ are true.
Executability	
$executable(t, sp)$	Transition t is executable at sp: it is enabled and has a high priority.
Priority Readiness	
$priority\_ready(t, sp)$	Transition t is priority-ready at sp, if t is ready and discounting the coordinative semantic aspects, it would be executable.

5.1.1 Priority-Related Definitions

This section presents notation for comparing the priority of transitions as well as the priority of sets of transitions. Two transitions, t and t', are *priority comparable*, if they can be compared with respect to the priority ordering of the semantics, in which case their priorities can be compared by prefixing the name of each transition with *pri* and using the normal comparison operators "$>$", "$<$", and $=$. If two transitions are not comparable, they are called *priority incomparable*, which can be expressed using the "$<>$" operator.

When the Scope-Parent priority semantics is used in an SBSML, $pri(t) > pri(t')$, $pri(t) < pri(t')$, and $pri(t) = pri(t')$ mean, respectively, that the scope of t is higher, lower, or the same as the scope of t' in the hierarchy tree. If the scopes of t and t' are not comparable (i.e., they belong to different branches of the hierarchy tree), then $pri(t) <> pri(t')$. Similar definitions for the Scope-Child priority semantics can be defined, by swapping the descriptions of the comparison operators "$<$" and "$>$".

When the Negation of Triggers priority semantics is used in a BSML, $pri(t) > pri(t')$, $pri(t) < pri(t')$, and $pri(t) = pri(t')$ mean that the trigger of t' is conjoined with some of the positive literals in the trigger of t, the trigger of t is conjoined with some of the positive literals in the trigger of t', and neither of the transitions has any of the positive literals of the trigger of the other in its trigger in the negated form, respectively. If both t and t' have some of the positive literals of one another's triggers in the negated form in their triggers, they are priority incomparable; i.e., $pri(t) <> pri(t')$.

In this chapter, as discussed in Section 3.8, only the Scope-Parent, the Scope-Child, and the Negation of Triggers priority semantics are considered. If a BSML semantics subscribes both to a hierarchical priority semantics and the Negation of Triggers priority semantics, then the Negation of Triggers priority semantics overrides the hierarchical priority semantics: First, t and t' are compared according to the Negation of Triggers priority semantics; if they have an equal priority or they are not priority comparable, then the hierarchical semantics is used as the secondary criterion to compare their priorities.

Priority Comparison Between Sets of Transitions. For two sets of transitions T and T', the operands "$\gtrdot$", "$\lessdot$", and "$\doteq$" compare the priority of the transitions of the two sets. The following definitions formalize these operators:

$$\begin{aligned}
T \gtrdot T' &\equiv (\exists t \in T \cdot \exists t' \in T' \cdot pri(t) > pri(t')) \wedge \neg(\exists t' \in T' \cdot \exists t \in T \cdot pri(t') > pri(t)),\\
T \lessdot T' &\equiv (\exists t \in T \cdot \exists t' \in T' \cdot pri(t) < pri(t')) \wedge \neg(\exists t' \in T' \cdot \exists t \in T \cdot pri(t') < pri(t)), \text{ and}\\
T \doteq T' &\equiv \neg(T \lessdot T') \wedge \neg(T \gtrdot T').
\end{aligned}$$

If $T \gtrdot T'$, $T \lessdot T'$, or $T \doteq T'$, it is said that T has a *higher priority*, *lower priority*, or *equal priority*, respectively, compared to T'. Intuitively, $T \doteq T'$, if it is not the case that T has a transition that has a higher priority than a transition in T' without T' having such a transition, and also not vice versa. As opposed to the comparison of the priority of individual transitions, two sets of transitions are always "priority comparable".

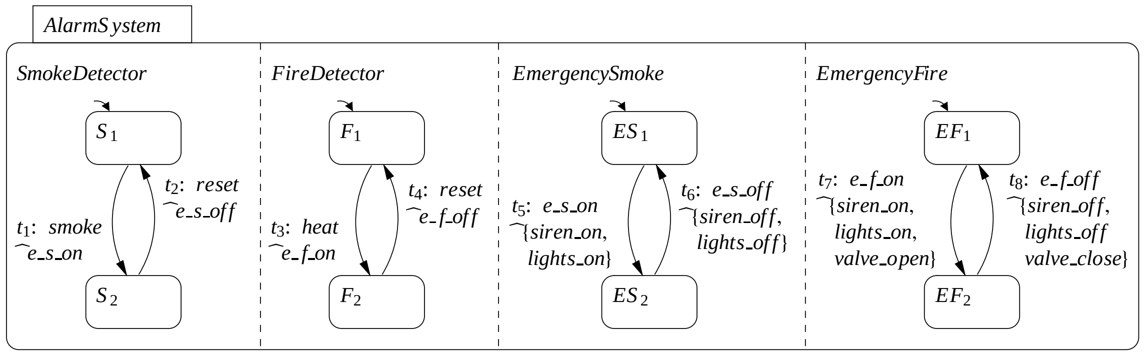
By definition, all potential small steps at a snapshot of a BSML model that uses a hierarchical priority semantics have an equal priority. This is because of the formal semantics of the Scope-Parent and Scope-Child semantic options, specified in Figure 4.8, on page 103, and Figure 4.10, on page 107, respectively, which, by definition, each gives a precedence to include a higher-priority transition than a lower-priority transition in a small step.

5.2 Semantic Quality Attributes for BSMLs

In this section, the three semantic quality attributes for BSMLs are introduced. The *non-cancelling* semantic quality attribute guarantees that if a transition becomes executable during a big step, it does not become mistakenly disabled. The *priority consistency* semantic quality attribute guarantees that higher-priority transitions are chosen over lower-priority transitions. The *determinacy* semantic quality attribute guarantees that all possible orders of small steps in a big step have the same result. Various modelling examples are presented that exhibit the presence and the absence of each semantic quality attribute.

5.2.1 Non-Cancelling

In a *non-cancelling* BSML semantics, once a transition of a model becomes executable in a big step, it remains executable during the big step, unless: it is taken by the next small step, it remains priority-ready unless it becomes unfireable, or its scope is entered or exited by a taken transition in the next small step. The second case requires that if a transition becomes executable, it cannot become unexecutable unless it also becomes unfireable. A BSML semantics that is not non-cancelling is *cancelling*. A non-cancelling BSML semantics is useful since it exempts a modeller from worrying about an enabled transition of interest mistakenly becoming disabled.



AlarmSystem
SmokeDetector
FireDetector
EmergencySmoke
EmergencyFire
S1
S2
F1
F2
ES1
ES2
EF1
EF2
t1: smoke ⌒e_s_on
t2: reset ⌒e_s_off
t3: heat ⌒e_f_on
t4: reset ⌒e_f_off
t5: e_s_on ⌒{siren_on, lights_on}
t6: e_s_off ⌒{siren_off, lights_off}
t7: e_f_on ⌒{siren_on, lights_on, valve_open}
t8: e_f_off ⌒{siren_off, lights_off valve_close}

Figure 5.4: A fire alarm system.

Example 31 *Figure 5.4 shows a model of a fire alarm system. The system performs two tasks: (i) when it detects smoke, it turns on the emergency lights and the danger sirens; and (ii) when it detects excessive heat, in addition to the actions in (i), it opens the valves of the extinguishing fountains. The model consists of four Or control states. Control states* SmokeDetector *and* FireDetector *model the interaction of the system with the smoke and fire detection devices, respectively. Control states* EmergencySmoke *and* EmergencyFire *control the operation of the emergency devices. The environmental input events* smoke *and* heat *specify the detection of smoke and excess heat. The output events* siren_on, lights_on, *and* valve_open *turn on the danger sirens on, turn on the emergency lights, and open the valves of the extinguishing fountains, respectively. The output events* siren_off, lights_off, *and* valve_close *do the opposites. The internal events* e_s_on *(emergency smoke on) and* e_f_on *(emergency fire on) activate the emergency operations of the* EmergencySmoke *and* EmergencyFire*, respectively. The internal events* e_s_off *and* e_f_off *do the opposites upon receiving* reset.

When the model resides in its default control states, $\{S_1, F_1, ES_1, EF_1\}$, *and both environmental input events* smoke *and* heat *are present, if the* SINGLE *concurrency semantics and the* PRESENT IN NEXT SMALL STEP *event lifeline semantics are chosen, then only the following two big steps carry out the intended behaviour of the system (i.e., turning on the danger sirens and the flashing lights, and opening the extinguishing valves):* $\langle t_1, t_5, t_3, t_7\rangle$ *and* $\langle t_3, t_7, t_1, t_5\rangle$.[1] *Additionally, the model can create an incorrect big step,* $\langle t_3, t_1, t_5\rangle$, *in which transition* t_7, *which opens the extinguishing valves, is not executed because* e_f_on *persists only one small step and is absent after* t_1 *is executed. The last possible big step,* $\langle t_1, t_7, t_3\rangle$, *although it does not execute transition* t_5,

[1]The system behaves correctly even if the output events siren_on and lights_on are generated more than once.

luckily behaves correctly because gen(t_5) $\subset$ *gen*(t_7). *This semantics, which is cancelling, would have been a suitable semantics only if* smoke *and* heat *could not be received together.*

If the MANY *concurrency semantics is chosen instead of the* SINGLE *concurrency semantics, the resulting BSML semantics is non-cancelling. The only possible big step would be* $\langle\{t_1, t_3\}, \{t_5, t_7\}\rangle$, *which carries out the intended behaviour of the system.*

Definition 5.1 *A BSML semantics is* non-cancelling *if for any BSML model, M,*

$$\forall T \in bigsteps(M) \cdot \forall i\ (1 \leq i < T.length) \cdot \forall t \cdot executable(t, T^i.s) \Rightarrow$$
$$(t \in T^i.\tau) \vee (fireable(t, T^i.s') \Rightarrow priority\_ready(t, T^i.s')) \vee (\exists t' \in T^i.\tau \cdot conflict(t, t')),$$

where

$$\begin{aligned} conflict(t, t') \quad \equiv \quad & [src(t) \in exited(t') \vee src(t) \in entered(t')] \vee \\ & [dest(t) \in exited(t') \vee dest(t) \in entered(t')]. \end{aligned}$$

The above predicate requires that if a transition, t, is executable in the source snapshot of a small step, T^i, *(i.e., it is in a potential small step at* $T^i.s$*): it is either taken in* $T^i.\tau$ *(the first disjunct), or if it is still fireable in* s'*, it is also priority-ready in* s' *(the second disjunct), or there is a transition* $t' \in T^i.\tau$ *that cannot possibly be taken together with t (the third disjunct).*

Two explanations about Definition 5.1 are in order.

First, the second disjunct, i.e., "$fireable(t, T^i.s') \Rightarrow priority\_ready(t, T^i.s')$", cannot be replaced with "$executable(t, T^i.s')$". This is because if an executable transition, t, in a big step becomes unfireable, that transition is not of interest in that big step any more. Therefore, the second disjunct requires only a transition to remain executable if it is still fireable. (Note that if both $fireable(t, T^i.s')$ and $priority\_ready(t, T^i.s')$ are true, according to predicate 5.3, on page 142, $executable(t, T^i.s')$ is also true. Thus, the second disjunct can be replaced with "$fireable(t, T^i.s') \Rightarrow executable(t, T^i.s')$".)

Second, the third disjunct recognizes the case that an executable transition, t, cannot be taken together with a transition, t' in the same small step. Such a t' either enters, exits, or both enters and exits control state $src(t)$ or control state $dest(t)$, where *entered* and *exited* functions are defined on page 92. In such a case, it is natural to consider the executability of t anew. An example of such a t' is a self transition where $src(t') = dest(t') = src(t)$. The execution of such a t' could make t disabled, and the third disjunct exempts t to remain enabled after such a t' is

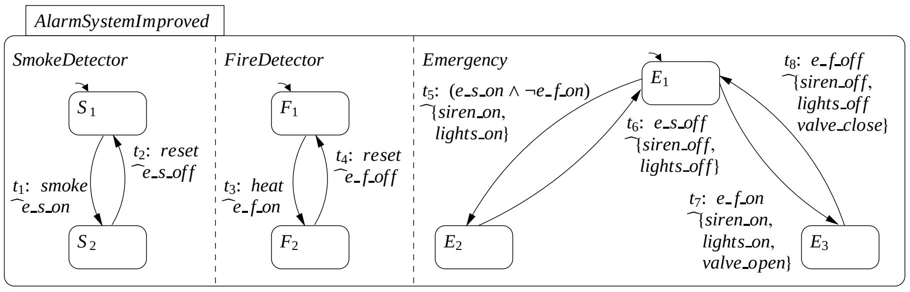

AlarmSystemImproved
SmokeDetector
FireDetector
Emergency
S1
S2
t1: smoke ⌒e_s_on
t2: reset ⌒e_s_off
F1
F2
t3: heat ⌒e_f_on
t4: reset ⌒e_f_off
E1
E2
E3
t5: (e_s_on ∧ ¬e_f_on) ⌒{siren_on, lights_on}
t6: e_s_off ⌒{siren_off, lights_off}
t7: e_f_on ⌒{siren_on, lights_on, valve_open}
t8: e_f_off ⌒{siren_off, lights_off valve_close}

Figure 5.5: An improved fire alarm system, compared to the one in Figure 5.4.

executed. Other examples of such a t' are a transition where $src(t) = src(t')$, a transition where $src(t) = dest(t')$, and a transition where $dest(t) = src(t')$. Again, t cannot be taken together with neither of these transitions, while its source control state is exited, in the first case, its source control state is entered, in the second case, and its destination control state is entered in the third case. (In the last case, the source and the destination of t' are children of two orthogonal control state, otherwise t could not have been executable in the first place.)

5.2.2 Priority Consistency

In a *priority-consistent* BSML semantics, higher-priority transitions must be chosen to execute over lower-priority transitions. The set of big steps of the model cannot include two big steps, T and T', where T includes transitions that are all of lower or incomparable priority than T'. A semantics that is not priority consistent is *priority inconsistent*.

Example 32 *The model in Figure 5.5 is similar to the model in Figure 5.4 except that control states* EmergencySmoke *and* EmergencyFire *in Figure 5.4 are represented by only one control state in Figure 5.5, namely, the* Emergency *control state. The new model, as opposed to the model in Figure 5.4, generates at most one instance of each of the* siren_on, lights_on, siren_off, *and* lights_off *events during a big step, to avoid any damage to the emergency devices.*

When the model resides in its default control states, $\{S_1, F_1, E_1\}$, *and both environmental input events* smoke *and* heat *are present, the intended behaviour of the system is that the danger sirens and the flashing lights should be turned on and the extinguishing valves should be opened; i.e.,* t_7 *should be executed and not* t_5. *Transition* t_7 *has a higher priority than* t_5 *according to the*

Negation of Triggers *priority semantics because trig*(t_5) *consists of* e_s_on *plus the negation of trig*(t_7). *If a BSML semantics is chosen that subscribes to the* Single *concurrency semantics and the* Present In Remainder *event lifeline semantics, then only the following three big steps carry out the intended behaviour of the system:* $\langle t_1, t_3, t_7 \rangle$, $\langle t_3, t_7, t_1 \rangle$, $\langle t_3, t_1, t_7 \rangle$. *Additionally, the model can create an incorrect big step,* $\langle t_1, t_5, t_3 \rangle$, *in which transition* t_7, *which opens the extinguishing valves, is not executed because* t_5 *is executed before* t_3. *This semantics is priority inconsistent because* t_7, *which has a higher priority than* t_5, *is not included in all big steps.*

If events e_f_on, e_f_off, e_s_on, *and* e_s_off *are interface events and follow the* Asynchronous Event *semantics for interface events, the resulting BSML semantic would be priority consistent. Two big steps are possible:* $\langle t_1, t_3 \rangle$ *and* $\langle t_3, t_1 \rangle$, *each of which can be taken non-deterministically. The execution of the second big step,* $\langle t_7 \rangle$, *carries out the intended behaviour of the system.*

Definition 5.2 *A semantics is* priority consistent *if for any BSML model M,*

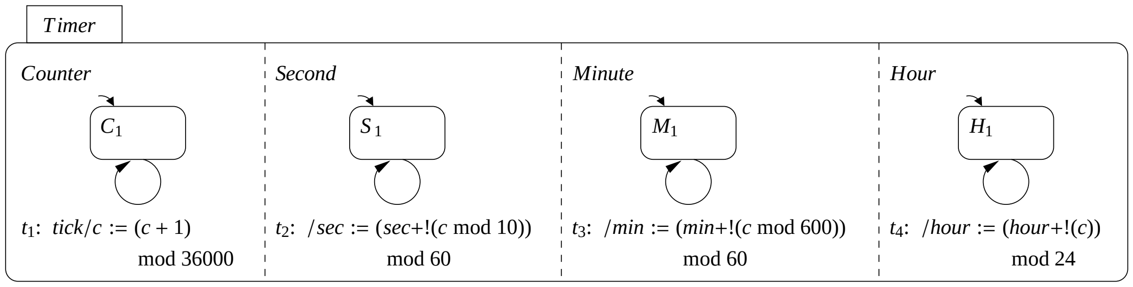
$$\forall T_1, T_2 \in bigsteps(M) \cdot (T_1.b = T_2.b) \Rightarrow \\ (\bigcup_{1 \le i_1 \le T_1.length} T_1^{i_1}.\tau) \doteq (\bigcup_{1 \le i_2 \le T_2.length} T_2^{i_2}.\tau),$$

Where "$\doteq$" operand, defined in Section 5.1.1, requires that it is not the case that T_1 executes a transition that has a higher priority than a transition in T_2 without T_2 having such a transition, and also not vice versa.

A big step may include the execution of the same transition more than once, but it suffices to consider one representative of them (i.e., no need to use multisets). The relative priority of two transitions is independent of the number of times they are executed.

5.2.3 Determinacy

In a *determinate* BSML semantics, in response to the same environmental input, if two big steps of a BSML model execute the same (multi) set of transitions in different orders, their destination snapshots are *equivalent*. An equivalence relation, denoted by "$\equiv$", can be defined with respect to any subset of the snapshot elements, but it is usually defined over the corresponding snapshot elements of the transition-based semantic aspects. A BSML semantics that is not determinate is *non-determinate*.



Timer
Counter
C_1
t_1: $tick/c := (c + 1)$ mod 36000
Second
S_1
t_2: $/sec := (sec+!(c \bmod 10))$ mod 60
Minute
M_1
t_3: $/min := (min+!(c \bmod 600))$ mod 60
Hour
H_1
t_4: $/hour := (hour+!(c))$ mod 24

Figure 5.6: A timer.

Example 33 *The model in Figure 5.6 is a clock that keeps track of the current time by using the frequency of a timer signal* tick, *which is received as an environmental input event every 10th of a second. The variables* sec, min, *and* hour *represent the second, minute and hour of the current time, respectively. There are three control states that update these variables, using the number of times the signal* tick *is received, which is maintained by variable* c*. Initially, all variables are 0. The unary operator "*!*" returns 0 if its operand is a non-zero integer and 1 otherwise. The binary operator "*mod*" returns the remainder of the division of the first operand by the second one. Every hour, i.e., when signal* tick *is received 36000 times, variable* c *is reset to 0.*

At the snapshot where the environmental input event tick *is received,* $c=35999$, $sec=59$, $min=59$, *and* $hour=18$, *the expected behaviour of the system after executing a big step is to reach the snapshot where* $c=0$, $sec=0$, $min=0$, *and* $hour=19$. *If the BSML semantics that subscribes to the* Single *concurrency semantics, the* Take One *big-step maximality semantics, the* RHS Small Step *assignment memory protocol is chosen, 24 big steps are possible, by permutating the order of the execution* t_1, t_2, t_3, *and* t_4. *However, only those big steps that start with* t_1, *such as* $\langle t_1, t_2, t_3, t_4\rangle$, *yield the expected behaviour. For example, executing* $\langle t_2, t_1, t_3, t_4\rangle$ *results in* $c=0$, $sec=59$, $min=0$, *and* $hour=19$.

Example 34 *Let us consider the model in Fig. 5.6, so that:*

$$t_1 : tick/c := ((c + 1) \bmod 36001) + (!(c \bmod 36000)) \text{ and}$$
$$t_4 : hour := (hour + (c \bmod 36000)) \bmod 24$$

In this new model: first, t_1 *resets* c *to 1, instead of resetting it to 0, and also, when* c *is 36000, instead of when* c *is 35999; and second,* t_4 *increments* hour *when* c *is 36000, instead of when when* c *is 0.*

In the new model, similar to Example 33, the snapshot where the environmental input event tick *is received,* c = 35999, sec = 59, min = 59, *and* hour = 18 *is considered. However, this time, instead of the* RHS SMALL STEP *assignment memory protocol, the* RHS BIG STEP *assignment memory protocol is chosen. Again, there are 24 big steps possible, but this time all of them behave similarly, reaching the snapshot where* c = 36000, sec = 59, min = 59, *and* hour = 18. *In the next big step, when* tick *is received, again 24 big steps are possible, all of which reach the snapshot where* c = 1, sec = 0, min = 0, *and* hour = 19. *Using this determinate semantics, the model behaves correctly if variable* c *is initialized with value 1, instead of 0 as in Example 33.*

Definition 5.3 *A BSML semantics is* determinate *if for any model, M,*

$$\forall T_1, T_2 \in bigsteps(M) \cdot [(T_1.b = T_2.b) \wedge (\biguplus_{1 \le i_1 \le T_1.length} T_1^{i_1}.\tau = \biguplus_{1 \le i_2 \le T_2.length} T_2^{i_2}.\tau)] \Rightarrow T_1.b' \equiv T_2.b',$$

where "$\biguplus$" is the multiset sum operator. Each of the two multiset sum terms collects the transitions of the small steps of one of the two big steps in the predicate. A transition may be executed more than once, by different small steps of a big step. Determinacy is relevant for two big steps only if the multisets representing their transitions are the same.

To have determinacy, a BSML must allow only *single assignment* models.

Definition 5.4 *A big step,* T, *is* single assignment *if there are no two transitions in the big step that assign values to the same variable. Formally,*

$$\forall t_1, t_2 \in (\biguplus_{1 \le i \le T_1.length} T^i.\tau) \cdot \forall a_1 \in asn(t_1) \cdot \forall a_2 \in asn(t_2) \cdot t_1 \not\simeq t_2 \Rightarrow lhs(a_1) \neq lhs(a_2),$$

where "$\not\simeq$" is the multiset inequality operator. Note that if t_1 and t_2 refer to the execution of the same transition in two different small step, then $t_1 \not\simeq t_2$.

A BSML model, M, is single assignment *if all big steps $T \in bigsteps(M)$ are single assignment.*

A crude condition to guarantee single assignment models is to require that: (i) only one transition of a model assigns a value to each variable; and (ii) no two transitions with overlapping arenas are executed in different small steps of a big step (i.e., the TAKE ONE maximality semantics), there by ensuring that a transition is not executed more than once in a big step.

5.3 Semantic Instantiation for Quality Attributes

In this section, for each of the three semantic quality attributes, all possible combinations of the semantic options that satisfy the semantic quality attribute are enumerated. (I do not include transition-aware semantic options because, as was discussed in 4.5, these semantic options convolute the role of structural and transition-based semantic aspects.[2] Thus, these semantics were not formalized in Chapter 4.)

Figure 5.7 once again shows the deconstruction of BSML semantics into structural and enabledness semantic aspects, but this time with the transition-based and coordinative semantic aspects distinguished by a "$\mathcal{T}ransition-\mathcal{B}ased$" and a "$\mathcal{C}oordinative$" on top of them, respectively. The transition-aware semantic options are not included in this feature diagram.

In the formalization for the semantic instantiation of the semantic quality attributes, the name of a semantic option is used as a proposition that specifies all BSML semantics that support the semantic option. For example, "Present In Next Small Step", as a proposition, specifies the set of all BSML semantics that subscribe to the Present In Next Small Step event lifeline semantics. The fact that only one semantic option of a semantic aspect can be chosen in a BSML semantics is implicit in the formalization. The only exception is that in the Priority semantic aspect, the Negation of Trigger semantics option can be chosen together with one of the Scope-Child or the Scope-Parent semantic options.

The logical connectives, such as conjunction, "$\wedge$", are used to create a predicate that specifies a set of BSML semantics. For example, the predicate "Present In Next Small Step $\wedge$ Take One" specifies all BSML semantics that subscribe to the Present In Next Small Step event lifeline semantics *and* the Take One big-step maximality semantics. The negation of a proposition has the usual meaning: prefixing the name of a semantic option is a predicate that specifies the set of all BSML semantics that do *not* subscribe to that semantic option. For example, the predicate "$\neg$Take One $\Rightarrow$ Source/Destination Orthogonal" specifies all BSML semantics that each, if it does not subscribe to the Take One big-step maximality semantics, then it subscribes to the Source/Destination Orthogonal small-step consistency semantics.

If a BSML does not support the related syntax for the corresponding semantic option of a proposition, the semantics of that BSML is not included in the set of BSML semantics that

[2]Note that the difference between the term "transition-based" semantic aspects (cf., the discussion on page 138) and the term "transition-aware" semantic options (cf., the discussion on page 110).

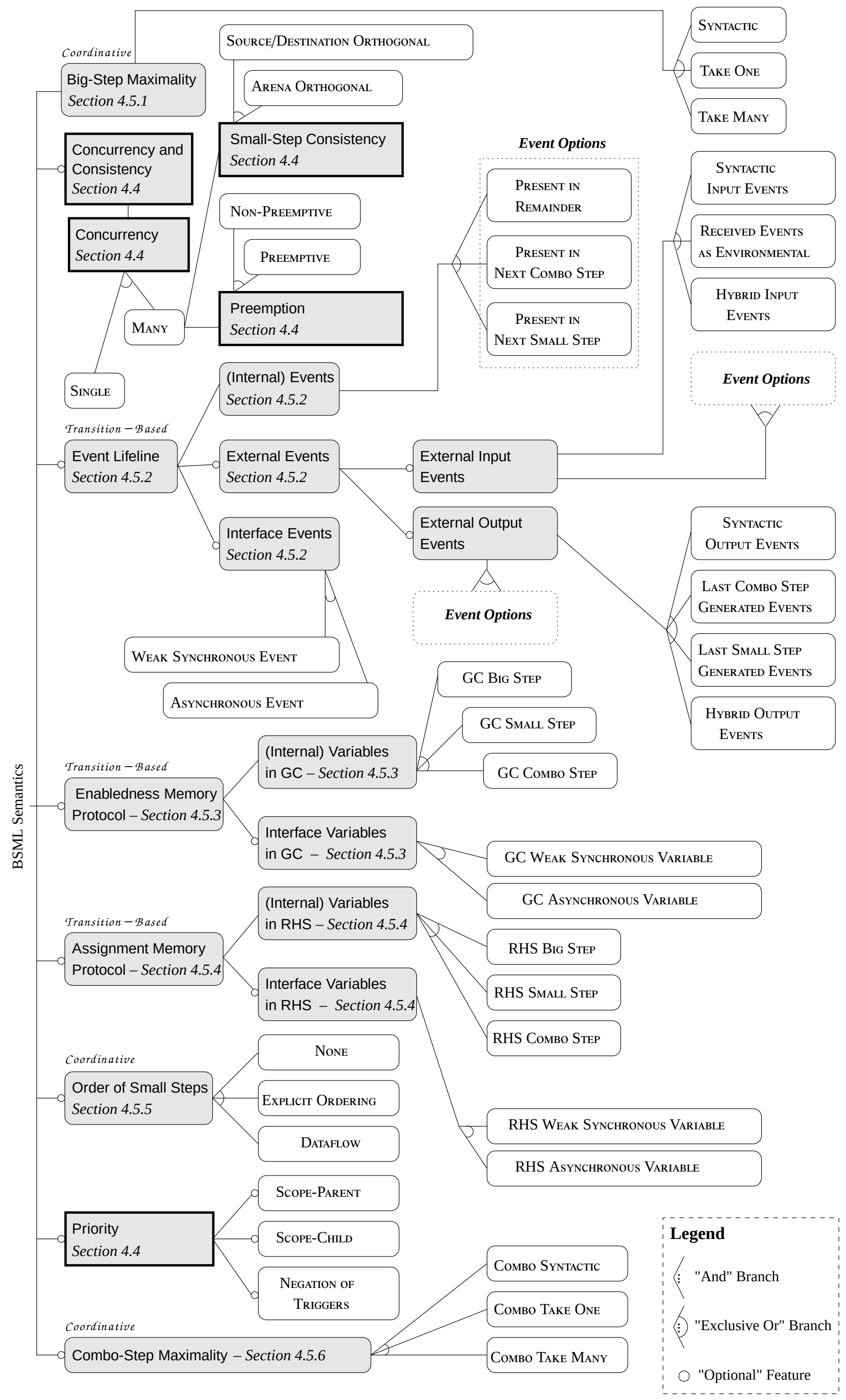

BSML Semantics
Coordinative
Big-Step Maximality Section 4.5.1
Syntactic
Take One
Take Many
Source/Destination Orthogonal
Arena Orthogonal
Small-Step Consistency Section 4.4
Concurrency and Consistency Section 4.4
Concurrency Section 4.4
Many
Single
Non-Preemptive
Preemptive
Preemption Section 4.4
Event Options
Present in Remainder
Present in Next Combo Step
Present in Next Small Step
Syntactic Input Events
Received Events as Environmental
Hybrid Input Events
Event Options
(Internal) Events Section 4.5.2
Transition – Based
Event Lifeline Section 4.5.2
External Events Section 4.5.2
External Input Events
Interface Events Section 4.5.2
External Output Events
Event Options
Syntactic Output Events
Last Combo Step Generated Events
Last Small Step Generated Events
Hybrid Output Events
Weak Synchronous Event
Asynchronous Event
GC Big Step
GC Small Step
GC Combo Step
Transition – Based
Enabledness Memory Protocol – Section 4.5.3
(Internal) Variables in GC – Section 4.5.3
Interface Variables in GC – Section 4.5.3
GC Weak Synchronous Variable
GC Asynchronous Variable
Transition – Based
Assignment Memory Protocol – Section 4.5.4
(Internal) Variables in RHS – Section 4.5.4
Interface Variables in RHS – Section 4.5.4
RHS Big Step
RHS Small Step
RHS Combo Step
Coordinative
Order of Small Steps Section 4.5.5
None
Explicit Ordering
Dataflow
RHS Weak Synchronous Variable
RHS Asynchronous Variable
Priority Section 4.4
Scope-Parent
Scope-Child
Negation of Triggers
Coordinative
Combo-Step Maximality – Section 4.5.6
Combo Syntactic
Combo Take One
Combo Take Many
Legend
"And" Branch
"Exclusive Or" Branch
"Optional" Feature

Figure 5.7: Solid boxes and rounded boxes are structural and enabledness semantic aspects, respectively. A transition-based and a coordinative enabledness semantic aspect are shown by a "*Transition – Based*" and a "*Coordinative*" on top of them, respectively.

the proposition represents. For example, proposition "Present In Next Small Step" does not include the semantics of a BSML that does not support the **Events** syntax. The syntactic feature of BSMLs were presented in Chapter 2, on page 21. Using a negation prefix before a syntactic feature specifies the set of all BSMLs that do not support that syntax. For example, "¬**Event Triggers**" specifies the set of all BSML semantics that do not support event triggers in transitions.

As specified in the dependencies in Figure 3.3, on page 34, the choice of a semantic option of a semantic aspect for a language could depend on the syntactic features used in the language. For example, according to the first dependency in Figure 3.3, i.e., "**Events** ⇔ Event Lifeline", the semantic options of the Event Lifeline semantic aspect can be chosen in a language only if the **Events** syntactic feature is also chosen, and vice versa. All of the dependencies in Figure 3.3 are implicitly conjoined with any predicate specified in this section. In this section, the syntactic features are used only in a negated form and only to preclude their corresponding semantic aspects from a predicate and not to enforce a well-formedness criterion.

To avoid long predicates, if neither the name of any of the semantic options of a semantic aspect nor the name of the corresponding syntactic feature of the semantic aspect in the negated form are used in a predicate, the predicate admits any BSML semantics that satisfies the explicit constraints of the predicate, and additionally, (i) either subscribes to one of the semantic options of the semantic aspect, or (ii) does not support the corresponding syntactic feature of the semantic aspect. As an example, the predicate "Present In Next Small Step ∧ Take One" refers to all BSML semantics that each subscribes to the Present In Next Small Step event lifeline semantics and the Take One big-step maximality semantics, and, for example, to either one of the semantic options of the Enabledness Memory Protocol semantic aspect, or provide no syntax for GC in transitions.

Next, for each of the semantic quality attributes, initially, the semantic specification of all BSMLs that subscribe to it are presented, without considering the role of external and interface events and variables. Each of these specifications is then extended by considering the role of external and interface events and variables.

5.3.1 Non-Cancelling Semantics

Recall from Definition 5.1 that in a non-cancelling BSML semantics, an executable transition, *t*, does not become disabled or low-priority, unless it is taken or has conflict with another transition,

t', in the immediate small step. Formally, a BSML semantics is non-cancelling if for any BSML model, M,

$$\forall T \in bigsteps(M) \cdot \forall i \; (1 \; \leq \; i \; < \; T.length) \cdot \forall t \cdot executable(t, T^i.s) \Rightarrow$$
$$(t \in T^i.\tau) \vee (fireable(t, T^i.s') \Rightarrow priority\_ready(t, T^i.s')) \vee (\exists t' \in T^i.\tau \cdot conflict(t, t')).$$

The first disjunct in the above predicate states that such a t is taken by the immediate small step. The challenge, however, is to achieve a non-cancelling BSML semantics when dealing with the cases that t is not taken by the small step, for example, because of non-determinism. In these cases, at least one of the two remaining disjuncts in the above predicate must be true to achieve a non-cancelling semantics.

This section presents necessary and sufficient constraints over the choices of the semantic options of a BSML that guarantee that if the first disjunct above is not true for an executable transition, at least one of the other two disjuncts is true. These constraints are organized into two sets. The first set corresponds to the BSML semantics that achieve a non-cancelling BSML semantics because the execution of a transition t' cannot possibly make an executable transition t disabled or low priority. The second set corresponds to the BSML semantics that achieve a non-cancelling BSML semantics by forcing such a t and t' to be executed together in the same small step, unless there is a conflict between them. The first and the second sets of constraints above correspond to the second and the third disjuncts in the predicate above, respectively.

Next, a formal specification of these two sets of constraints are presented. Initially, for the sake of clarity, the roles of the external and interface events and variables are not considered.

For BSMLs that do not support external and interface events and variables, the disjunction of the following two predicates determine the class of non-cancelling BSML semantics:

$$\begin{aligned}
\mathbf{N_{Steady}} \quad &\equiv \quad Big\_Semantics \vee Combo\_Semantics,\\
\mathbf{N_{Maximizer}} \quad &\equiv \quad \textsc{Many} \wedge\\
&\quad [\neg(\textsc{Take One} \vee \textsc{Combo Take One}) \Rightarrow \textsc{Source/Destination Orthogonal}] \wedge\\
&\quad [(\neg(\textsc{Take One} \vee \textsc{Combo Take One}) \wedge \textsc{No Priority}) \Rightarrow \textsc{Non-Preemptive}],
\end{aligned}$$

where,

$$
\begin{aligned}
\textit{Big\_Semantics} \quad &\equiv \quad [(\text{GC Big Step} \lor \neg\textbf{Guard Conditions}) \land \neg\textbf{Event Triggers}] \land \\
&\qquad [(\text{Take One} \lor \text{No Priority}) \land \neg\text{Dataflow}], \text{ and} \\
\textit{Combo\_Semantics} \quad &\equiv \quad [\ (\neg\text{GC Small Step} \lor \neg\textbf{Guard Conditions}) \land \\
&\qquad\ (\text{P.I. Next Combo Step} \lor \neg\textbf{Event Triggers})\ ] \land \\
&\qquad [(\text{Combo Take One} \lor \text{No Priority})].
\end{aligned}
$$

For the sake of brevity, instead of prefix “Present In”, “P.I.” is used in the formalization above, and in the rest of this chapter. These predicates do not refer to the semantic options of the Assignment Memory Protocol semantic aspect because these semantic options, as will be shown later in the section, do not have any effect on determining a BSML semantics as non-cancelling.

Predicates $\mathbf{N_{Steady}}$ and $\mathbf{N_{Maximizer}}$ correspond to the second and third disjunct in Definition 5.1, respectively. Predicate $\mathbf{N_{Steady}}$ ensures that if an executable transition is not taken in the immediate small step, it does not become unready or low priority unless it also becomes unfireable, as required in the second disjunct. Predicate *Big_Semantics* corresponds to the semantics in which the statuses of events and the values of variables remain the same, thus an executable transition remains ready and high priority. Predicate *Combo_Semantics* corresponds to similar semantics, but in the context of combo steps. Predicates *Big_Semantics* and *Combo_Semantics* characterize mainly disjoint sets of BSML semantics. Predicate $\mathbf{N_{Maximizer}}$ specifies the necessary constraints on the choices of the semantic options of the Concurrency and Consistency semantic aspects to ensure that as many executable transitions as *necessary* are taken together in a small step: The second and third conjuncts of the $\mathbf{N_{Maximizer}}$ predicate only require the more inclusive semantic options of the small-step consistency and preemption semantic aspects, respectively, if not requiring these semantic options can leave an executable transition unready or low priority, but still fireable at the destination snapshot of the small step. Predicate $\mathbf{N_{Maximizer}}$ does not enforce a constraint over the choices of the semantic options for the small-step consistency or preemption semantic aspect if the second disjunct of Definition 5.1 is guaranteed to be true. These will be discussed in more detail in the examples and the proofs later in the section.

The two examples below show how the above predicates are evaluated for a BSML. If a BSML semantics subscribes to the GC Small Step enabledness memory protocol, the P.I. Next Combo Step event lifeline semantics, the Take One big-step maximality semantics, the Combo Take One combo-step maximality semantics, and the Many concurrency semantics, then predicate $\mathbf{N_{Maximizer}}$ will be true, and thus the BSML has a non-cancelling semantics. Note that only the MANY semantic option is necessary to achieve a non-cancelling BSML semantics, according to the $\mathbf{N_{Maximizer}}$ predicate, but not the SOURCE/DESTINATION ORTHOGONAL or the NON-PREEMPTIVE semantic options. The reason that these semantic options are not necessary is that the COMBO TAKE ONE combo-step maximality semantics ensures that any executable transition that is left out of the small step, becomes unfireable at the destination of small step, satisfying the second disjunct of Definition 5.1. An example of a BSML semantics that is non-cancelling through satisfying the predicate $\mathbf{N_{Steady}}$ is a BSML semantics that subscribes to the GC BIG STEP enabledness memory protocol, the TAKE MANY big-step maximality semantics, and the NO PRIORITY semantics, and does not support events in triggers of transitions, i.e., "¬**Event Triggers**" is true. In this semantics, an executable transition never becomes unready or low priority.

Example 35 *Figure 5.8 shows examples of how if the constituent semantic options of a BSML violate predicate* $\mathbf{N_{Steady}} \vee \mathbf{N_{Maximizer}}$*, a cancelling behaviour results. For all three models in Figure 5.8, they reside in their default control states; environmental input event* i *is present in the second and the third model; and* $\mathrm{x} = \mathrm{y} = 0$ *in the third model.*

In the BSML model in Figure 5.8(a), if the BSML is a non-combo–step semantics that subscribes to the SINGLE *concurrency semantics, the* TAKE MANY *maximality semantics, and the* SCOPE-PARENT *priority semantics, transition* t_1 *and* t_3 *are initially executable, but if the first small step executes* t_2*,* t_4 *becomes executable and* t_1 *and* t_3 *become unexecutable, because* $pri(\mathrm{t}_4) > pri(\mathrm{t}_1)$ *and* $pri(\mathrm{t}_4) > pri(\mathrm{t}_3)$*, which is a cancelling behaviour. The constituent semantic options of the BSML do not satisfy* $\mathbf{N_{Steady}} \vee \mathbf{N_{Maximizer}}$*. First, Big_Semantics and Combo_Semantics are both false because their second conjuncts are false. Second, Maxmizer is false because its first conjunct, i.e., "*MANY*", is false.*

Let us adjust the BSML model in Figure 5.8(a) so that:

$$\begin{aligned} &\mathrm{t}_1 : /\mathrm{v} := 1, \\ &\mathrm{t}_2 : [\mathit{new}(\mathrm{v}) = 1], \text{ and} \\ &\mathrm{t}_3 : /\mathrm{v} := 2, \end{aligned}$$

with transition t_4 *being removed from the model; the BSML subscribes to the* DATAFLOW *semantic option. If the same semantic options as above, plus the* GC BIG STEP *enabledness memory protocol are considered, when the model resides in its default control states, big step* $\langle \mathrm{t}_1, \mathrm{t}_3 \rangle$ *is one of the possible big steps. In this big step, after the execution of* t_1*,* t_2 *is executable, but once* t_3 *is*

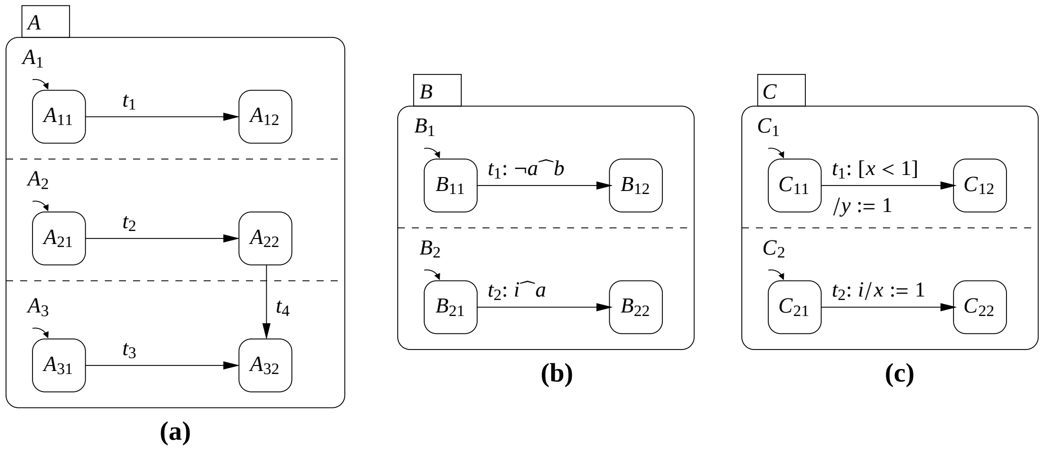


Figure 5.8: Examples of cancelling behaviour.

executed, t_2 *becomes disabled, which is a cancelling behaviour. Again,* $\mathbf{N}_{\mathbf{Steady}} \vee \mathbf{N}_{\mathbf{Maximizer}}$ *is not true for this BSML because of the same reasons as above.*

In the BSML model in Figure 5.8(b), if the BSML subscribes to the Single *concurrency semantics, the* P.I. Remainder *event lifeline semantics, and the* Take One *maximality semantics, transition* t_1 *is initially executable, but if the first small step executes* t_2*,* t_1 *becomes unready, which is a cancelling behaviour that is confirmed by the fact that predicate* $\mathbf{N}_{\mathbf{Steady}} \vee \mathbf{N}_{\mathbf{Maximizer}}$ *is false. However, if the* P.I. Next Combo Step *event lifeline semantics together with the* Combo Take One *combo-maximality semantics are chosen instead of the* P.I. Remainder *event lifeline semantics, a non-cancelling semantics is achieved: Two big steps,* $\langle (\!| \mathrm{t}_1, \mathrm{t}_2 |\!) \rangle$ *and* $\langle (\!| \mathrm{t}_2, \mathrm{t}_1 |\!) \rangle$*, are possible, where the scope of a combo step is identified by a surrounding "*$(\!|\ |\!)$*". This latter BSML semantics is non-cancelling because* $\mathbf{N}_{\mathbf{Steady}}$ *is true through Combo_Semantics being true.*

In the model in Figure 5.8(c), if the Single *concurrency semantics, the* GC Small Step *enabledness memory protocol, and the* Take One *big-step maximality semantics are chosen, transition* t_1 *is initially executable, but executing* t_2 *makes* t_1 *unready. If the* GC Next Combo Step *enabledness memory protocol together with the* Combo Take One *combo-maximality semantics are chosen instead of the* GC Small Step *enabledness memory protocol, a non-cancelling semantics is achieved:* t_1 *and* t_2 *are executed in the same combo step. The latter semantics is non-cancelling because its semantic options satisfy predicate Combo_Semantics.*

The Role of External Communication

The role of the External Input Events semantic sub-aspect in determining a BSML semantics as non-cancelling is similar to the role of the Event Lifeline semantic aspect. As shown in the feature diagram in Figure 5.7, an External Input Events semantics is instantiated by an option that determines which events are considered as input events and by an option belonging to the "*Event Options*" that determines the extent that an environmental input event persists in a big step. It is only the second option, which belongs to the "*Event Options*", that has a role in determining a BSML semantics as non-cancelling or not. The first option, by itself, does not have any effect on determining the enabledness of a transition: It only specifies which events in the trigger of a transition should be considered as environmental input events, in a given big step. As such, to extend the class of non-cancelling BSML semantics to include external input events, it suffices to adjust predicates *Big_Semantics* and *Combo_Semantics* by conjoining them with the following two predicates, respectively:

$$\begin{aligned} \mathit{XBig\_Semantics} &\equiv (\text{X.P.I. R}{\small\text{EMAINDER}} \lor \neg\textbf{Environmental Input Events}), \text{ and} \\ \mathit{XCombo\_Semantics} &\equiv (\neg\text{X.P.I. S}{\small\text{MALL}}\text{ S}{\small\text{TEP}} \lor \neg\textbf{Environmental Input Events}), \end{aligned}$$

where the prefix "X" for semantic options above refers to the event lifeline semantic options of external input events.

In the X.P.I. Remainder semantic option, as opposed to the P.I. Remainder semantic option, an environmental event is either present or absent throughout a big step. Thus, while the P.I. Remainder semantic option cannot be used in the $\mathbf{N_{Steady}}$ predicate, the X.P.I. Remainder semantic option can be used.

The Role of Interface Communication

The roles of the Interface Events and the Interface Variables in GC semantic aspects in determining a BSML semantics as non-cancelling are similar to the Event Lifeline and the Enabledness Memory Protocols semantic aspects, respectively. A major difference is that interface events and variables do not have combo-step semantic options. As such, to extend the class of non-cancelling BSML semantics to include interface events and interface variables, it suffices to adjust predicate *Big_Semantics* by conjoining it with the following predicate:

$$\begin{aligned} IBig\_Semantics \equiv\ & [\textsc{Asynchronous Event} \vee \neg \textbf{Interface Events}] \wedge \\ & [\text{GC } \textsc{Asynchronous Variable} \vee \neg \textbf{Interface Variables in GC}]. \end{aligned}$$

Similar to the Assignment Memory Protocol semantic aspect, the Interface Variables in RHS semantic aspect does not have any role in determining a BSML semantics as non-cancelling.

Proofs

Next, after presenting a few lemmas, a proposition about the correctness of the above characterization of the non-cancelling BSML semantics is presented.

Lemma 5.1 *The choice of a semantic option for each of the* External Output Events *and the* Assignment Memory Protocol *semantic aspects of a BSML has no effect in determining it as non-cancelling.*

Proof Idea. These semantic aspects are not relevant because they do not affect the readiness, fireability, or the priority of a transition. The External Output Events determines the lifeline of external output events, and not the triggering events of the transitions. The Assignment Memory Protocol specifies the values of variables on the RHS of an assignment, but not the values of the variables used in the GC of a transition. □

Lemma 5.2 *If in a BSML semantics an executable transition in a snapshot can become unready or low-priority but fireable after the execution of the immediate small step, requiring predicate Maximizer is the weakest constraint over the choices of its* Concurrency and Consistency *semantic options to guarantee a non-cancelling BSML semantics.*

Proof Idea. The $\mathbf{N_{Maximizer}}$ predicate is copied below for convenience,

$$\begin{aligned} \mathbf{N_{Maximizer}} \equiv\ & \textsc{Many} \wedge \\ & [\neg(\textsc{Take One} \vee \textsc{Combo Take One}) \Rightarrow \textsc{Source/Destination Orthogonal}] \wedge \\ & [(\neg(\textsc{Take One} \vee \textsc{Combo Take One}) \wedge \textsc{No Priority}) \Rightarrow \textsc{Non-Preemptive}]. \end{aligned}$$

To prove this claim, two points should be shown. First, requiring predicate $\mathbf{N_{Maximizer}}$ can modify a cancelling BSML semantics to a non-cancelling BSML semantics. And second, none of the constraints in predicate $\mathbf{N_{Maximizer}}$ can be relaxed.

To prove the first part of the claim, it will be shown that given a cancelling behaviour in a BSML model, if the concurrency and consistency semantic options of the BSML are changed so that they satisfy the $\mathbf{N_{Maximizer}}$ predicate, then a cancelling behaviour does not arise. According to Definition 5.1, a cancelling behaviour arises in a model when there is an executable transition, t, at a snapshot and the effect of taking the immediate small step that does not include t makes t unready or low-priority, but still fireable, although t does not have any conflict with any of the transitions in the small step. To avoid such a cancelling behaviour, predicate $\mathbf{N_{Maximizer}}$, through predicate Many, its first conjunct, tries to force such a t to be taken by the immediate small step. If t is included in the small step, a non-cancelling behaviour is achieved, according to the first disjunct in Definition 5.1. However, the Many concurrency semantics in a BSML does not guarantee that such a t will be taken by the small step. If that is the case, however, then the second and the third conjuncts of the $\mathbf{N_{Maximizer}}$ predicate ensure that the third disjunct of Definition 5.1 is true; i.e., the immediate small step includes a transition, t', such that $conflict(t, t')$ is true.

If both the Source/Destination Orthogonal and Non-Preemptive semantic options are chosen, such a t' is guaranteed to exist: If such a t' does not exist, then t could have been taken by the small step, which is a contradiction. The antecedents of the second and the third conjuncts, however, recognize the cases that requiring the Source/Destination Orthogonal and Non-Preemptive semantic options are not necessary to achieve a non-cancelling BSML semantics. If the antecedent of the second conjunct is false, it means that either the Combo Take One combo-step maximality semantics, the Take One big-step maximality semantics, or both, have been chosen in a BSML semantics, in which case even if t is left out of the small step, it becomes unfireable, making the second disjunct of Definition 5.1 true, through its antecedent being false. Similarly, if the antecedent of the third conjunct is false, t will become unfireable. The antecedent of the third conjunct of the $\mathbf{N_{Maximizer}}$ predicate has an extra conjunct compared to the antecedent of the second conjunct that does not require the Non-Preemptive semantic option if one of the hierarchical semantic options are chosen: If a hierarchical priority semantics is chosen, an interrupted and an interrupt transition need not be taken together through the Non-Preemptive semantic option because one has a higher priority than the other.

So far, the first part of the proof has been presented: it has been shown that the constraints of the $\mathbf{N_{Maximizer}}$ predicate together are *sufficient* to ensure a non-cancelling BSML semantics.

However, it should also be shown that these constraints are *necessary*: A concurrency and consistency semantic option is not unnecessarily required to be chosen by the $\mathbf{N_{Maximizer}}$ predicate. To show this, it is enough to inspect the role that other semantic aspects could have in relaxing the $\mathbf{N_{Maximizer}}$ predicate. The transition-based, enabledness semantic aspects need not be considered because they correspond to the readiness of a transition, which is not relevant in this claim (the lemma already assumes that a transition could become disabled). The coordinative enabled semantic aspects are of interest so far as they have an effect in making *t* unfireable in the destination of the immediate small step, to make the second disjunct of Definition 5.1 true. The $\mathbf{N_{Maximizer}}$ predicate already considers the roles of the Big-Step Maximality and the Comb-Step Maximality semantic aspects. The semantic options of the Order of Small Step semantic aspect order the execution of the transition of a model, however, none of them can make an executable transition unfireable unless it is executed in the small step. Thus, the Order of Small Step cannot relax any of the constraints of the predicate $\mathbf{N_{Maximizer}}$. Lastly, the NEGATION OF TRIGGERS priority semantics is not relevant in the $\mathbf{N_{Maximizer}}$ predicate since its choice affects the readiness of a transition, which is not relevant in this claim. Thus, the $\mathbf{N_{Maximizer}}$ predicate is not only a sufficient condition for turning a cancelling BSML semantics, as characterized in the lemma, into a non-cancelling one, but also is a necessary condition. □

Proposition 5.3 *A BSML semantics is non-cancelling if and only if its constituent semantic options satisfy the predicate* $\mathbf{N} \equiv \mathbf{N'_{Steady}} \vee \mathbf{N_{Maximizer}}$, *where,*

$$\begin{aligned} \mathbf{N'_{Steady}} \equiv\ & (\mathit{Big\_Semantics} \wedge \mathit{XBig\_Semantics} \wedge \mathit{IBig\_Semantics}) \vee \\ & (\mathit{Combo\_Semantics} \wedge \mathit{XCombo\_Semantics}). \end{aligned}$$

Proof Idea. To prove this claim, it should be shown that predicate **N** characterizes all non-cancelling BSML semantics and only them. First, it will be shown that any BSML semantics whose semantic options satisfy predicate **N** is a non-cancelling BSML semantics. And second, it will be shown that the semantic options of any non-cancelling BSML semantics satisfies predicate **N**.

If the semantic options of a BSML satisfy predicate **N**, then either $\mathbf{N'_{Steady}}$, $\mathbf{N_{Maximizer}}$, or both are true. If $\mathbf{N'_{Steady}}$ is true, then either the $(\mathit{Big\_Semantics} \wedge \mathit{XBig\_Semantics} \wedge \mathit{IBig\_Semantics})$ predicate or the $(\mathit{Combo\_Semantics} \wedge \mathit{XCombo\_Semantics})$ predicate is true,[3] which means that

[3]If the BSML neither supports events nor variables syntax, then both predicates can be true.

an executable transition remains priority-ready during the current big step or combo step, respectively. Thus, a non-cancelling BSML semantics is achieved. This is because if the first predicate is true, the statuses of events and the values of variables remain the same throughout the big step, thus the transition remains ready; since the No Priority semantics is chosen, then the transition is priority-ready. The Dataflow semantic option should not be chosen, because a ready transition can become unready if a variable is assigned more than once during a big step, as described in Example 35, on page 156, where the original model in the example is changed to use the `new` operator. Similarly, if the second predicate is chosen, a non-cancelling BSML semantics is achieved. If predicate $\mathbf{N}_{\mathbf{Maximizer}}$ is true, regardless of whether $\mathbf{N}'_{\mathbf{Steady}}$ is true or not, a non-cancelling BSML semantics is achieved, according to Lemma 5.2. If both $\mathbf{N}'_{\mathbf{Steady}}$ and $\mathbf{N}_{\mathbf{Maximizer}}$ are true, then the BSML is non-cancelling because each predicate separately attempts to satisfy one of the disjuncts of Definition 5.1, and these attempts never cancel each other.

Conversely, if a BSML semantics is non-cancelling, then it satisfies predicate $\mathbf{N}$. Two cases are considered based on whether the Many concurrency semantics is chosen or not.

If the BSML does not support the Many concurrency semantics, i.e., it subscribes to the Single concurrency semantics, and an executable transition does not become unready or low priority, then predicate $\mathbf{N}'_{\mathbf{Steady}}$ must be true. If $\mathbf{N}'_{\mathbf{Steady}}$ is not true, it is always possible to create a counter example model with a cancelling behaviour, similar to the ones in Example 35: A model can be constructed in which the guard condition or the event trigger of an executable transition, t, is forced to become false after the execution of the immediate small step, which executes a single transition because of the Single concurrency semantics.

If the BSML supports the Many concurrency semantics, and would have not been a non-cancelling BSML semantics if it would have supported the Single concurrency semantics, then the original BSML semantics must satisfy predicate $\mathbf{N}_{\mathbf{Maximizer}}$, according to Lemma 5.2. Lastly, if the BSML would have been a non-cancelling BSML semantics even if it would have supported the Single concurrency semantics, then the original BSML semantics should also satisfy predicate $\mathbf{N}'_{\mathbf{Steady}}$, as described in the previous paragraph. $\square$

5.3.2 Priority-Consistent Semantics

First, the BSML semantics that subscribe to either the Scope-Parent or the Scope-Child semantic options are considered, followed by the ones that subscribe to the Negation of Triggers. Lastly, the BSML semantics that subscribe to both a hierarchical semantic option and the Negation of Triggers semantic option are considered.

Hierarchical Priority Semantics

Any priority-consistent BSML semantics according to one of the hierarchical semantic options, i.e., the Scope-Parent or the Scope-Child semantic option, must subscribe to the Take One maximality semantics. Otherwise, no constraints over the choice of the other semantic options can result in a priority-consistent behaviour. For example, when the Take Many maximality semantics and the Scope-Parent priority semantics are chosen together, it is not possible to choose the transitions of a current small step in such a way that a model always reaches a control state that is the source of a transition with the highest scope. Formally, the following predicate should hold,

$$\mathbf{P_{Hierarchical}} \quad \equiv \quad \text{Take One}.$$

Example 36 *Figure 5.9(a) shows a model that demonstrates an example of how the violation of predicate* $\mathbf{P_{Hierarchical}}$ *results in a priority-inconsistent behaviour. The model is considered when it resides in its default control states and environmental input event* i *is present. If a BSML semantics that subscribes to the* Many *concurrency semantics, the* Take Many *maximality semantics (which violates* $\mathbf{P_{Hierarchical}}$*), the* Scope-Parent *priority semantics, and the* GC Small Step *enabledness memory protocol is considered, then two big steps,* $\langle\{t_1, t_4\}, t_2, t_6\rangle$ *and* $\langle\{t_1, t_4\}, t_3, t_5\rangle$ *are possible. The former big step includes transition* t_6*, which has a higher priority than transition* t_5 *in the latter big step.*

It is possible to create a similar model that neither uses events nor uses variables but exhibits a similar priority-inconsistent behaviour. The model in Figure 5.9(b) shows a BSML model that has a priority-inconsistent behaviour, when it resides in control state B_1*, and a BSML semantics is chosen that subscribes to the* Take Many *maximality semantics and the* Scope-Parent*. Two big steps are possible:* $\langle t_1, t_2, t_3\rangle$ *and* $\langle t_1, t_2, t_4\rangle$*, with the latter big step including* t_4*, which has a higher priority than transition* t_3 *in the former big step.*

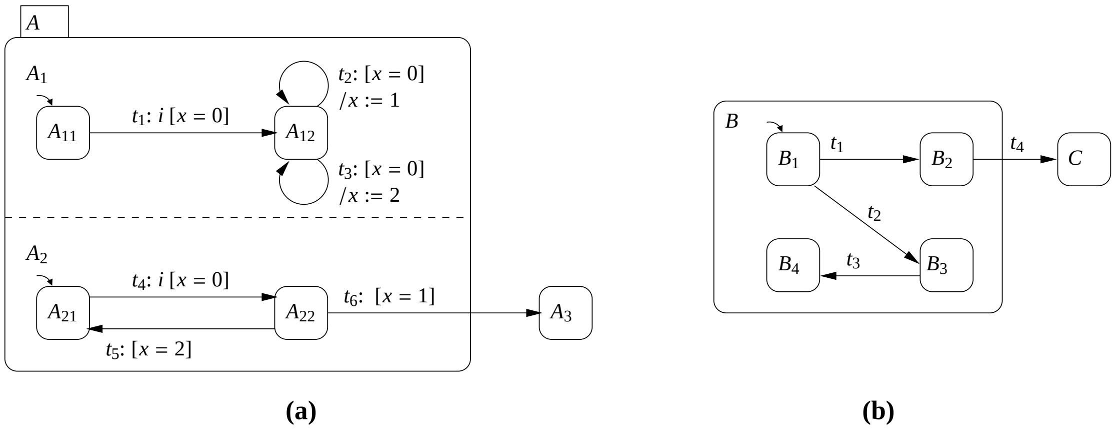


Figure 5.9: Examples of priority-inconsistent behaviour for the Scope-Parent or Scope-Child priority semantics.

Proposition 5.4 *A BSML semantics that subscribes to the priority semantics* Scope-Parent *or* Scope-Child, *but not the* Negation of Triggers, *is priority consistent if and only if it satisfies predicate* $\mathbf{P_{Hierarchical}}$.

Proof Idea. If a BSML semantics subscribes to the Take One big-step maximality semantics, a priority-inconsistent behaviour cannot arise when the Scope-Parent or the Scope-Child semantic option is chosen. This is because, by virtue of allowing each *Or* child of an *And* control state to take maximum one transition during a big step, the possibility of a model to arrive at different configurations to have the choice to execute high or low-priority transitions in a priority-inconsistent manner is precluded.

Conversely, if a BSML semantics is priority consistent, then it should subscribe to the Take One big-step maximality semantics. Otherwise, if the BSML semantics subscribes to the Take Many or the Syntactic semantic option, it is always possible to create a BSML model similar to the one in Figure 5.9(b) that has a priority-inconsistent behaviour. □

Negation of Triggers Priority Semantics

None of the transition-aware Event Lifeline semantics for internal events, i.e., none of the event lifeline semantics for internal events that are considered in the scope of the formalization in

Chapter 4, support a priority-consistent behaviour according to the NEGATION OF TRIGGERS priority semantics. Thus, the following predicate is needed to guarantee a priority-consistent behaviour,

$$\mathbf{P_{Negation}} \quad \equiv \quad \neg\mathbf{Negated\ Events},$$

where "¬**Negated Events**" predicate refers to all BSML semantics that do not support a syntax for negated events in the trigger of a transition, which according the constraint 5 in Figure 3.3, on page 34, refers to all BSML semantics that do not support the NEGATION OF EVENTS priority semantics.

Variables have no role in determining the class of priority-consistent BSML semantics above, because, unlike events that are used in the NEGATION OF TRIGGERS priority semantics, variables are used only to determine the readiness of a transition.

Example 37 *Figure 5.10 shows an example of how the violation of predicate* $\mathbf{P_{Negation}}$ *results in a priority-inconsistent behaviour according to the* NEGATION OF TRIGGERS *priority semantics. The model in Figure 5.10 is considered when it resides in its default control states and the environmental input event i is present. If a BSML semantics that subscribes to the* SINGLE *concurrency semantics, the* NEGATION OF TRIGGERS *priority semantics, and the* P.I. REMAINDER *event lifeline semantics, which violates* $\mathbf{P_{Negation}}$, *is considered, then four big steps are possible:* $\langle t_1, t_2, t_4\rangle$, $\langle t_2, t_1, t_4\rangle$ $\langle t_1, t_4, t_2\rangle$*, and* $\langle t_2, t_3, t_1\rangle$*. However, this is a priority-inconsistent behaviour because the last big step executes transition* t_3*, although* $pri(t_3) < pri(t_4)$*. Similar priority-inconsistent behaviour arise when the* P.I. NEXT SMALL STEP *semantic option is chosen. Again, four big steps are possible:* $\langle t_1, t_2, t_3\rangle$, $\langle t_2, t_3, t_1\rangle$*,* $\langle t_2, t_1, t_4\rangle$*, and* $\langle t_1, t_4, t_2\rangle$*. And again, a priority-inconsistent behaviour arises: The first two big steps include the transition* t_3 *whereas the last two big steps include the transition* t_4*, while* $pri(t_3) < pri(t_4)$*.*

If the MANY *concurrency semantics is chosen, instead of the* SINGLE *concurrency semantics, for both the* P.I. REMAINDER *and the* P.I. NEXT SMALL STEP *event lifeline semantics, the only possible big step would have been,* $T_1 = \langle\{t_1, t_2\}, t_4\rangle$*, which is a priority-consistent behaviour. However, in general, the* MANY *concurrency semantics cannot resolve this priority inconsistency problem. For example, if transition* t'_2*, such that* $src(t'_2) = A_{21}$*,* $dest(t'_2) = A_{22}$*, and* $gen(t'_2) = c$*, is added to the model, an additional big step,* $T_2 = \langle\{t_1, t'_2\}, t_3\rangle$*, is possible, which results in a priority-inconsistent behaviour:* T_2 *includes* t_3 *instead of* t_4 *in* T_1*, while* $pri(t_3) < pri(t_4)$*.*

Similar priority-inconsistent behaviour arise when the P.I. NEXT COMBO STEP *semantic op-*

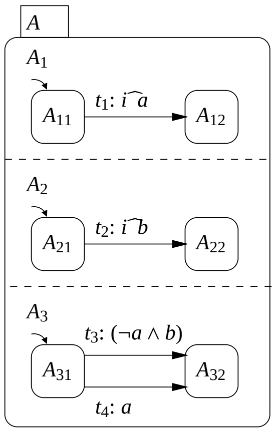


Figure 5.10: Examples of priority-inconsistent behaviour for the Negation of Triggers priority semantics.

tion is chosen. For example, consider a BSML model similar to the model in Figure 5.10, except that it has an extra transition t'_1, *such that* $src(t'_1) = A_{11}$, $dest(t'_1) = A_{12}$, *and* $gen(t'_1) = c$. *Again, the model is considered when it resides in its default control states and the environmental input event i is present. If a BSML semantics that subscribes to the* Single *concurrency semantics, the* Negation of Triggers *priority semantics, and the* P.I. Next Combo Step *event lifeline semantics is considered, then four big steps are possible:* $\langle (\!| t_1, t_2 |\!), (\!| t_4 |\!) \rangle$, $\langle (\!| t_2, t_1 |\!), (\!| t_4 |\!) \rangle$, $\langle (\!| t'_1, t_2 |\!), (\!| t_3 |\!) \rangle$, *and* $\langle (\!| t_2, t'_1 |\!), (\!| t_3 |\!) \rangle$, *where the scope of a combo step is identified by a surrounding "*$(\!|\ |\!)$*". This behaviour is priority inconsistent because the last two big steps include* t_3 *and the first two big steps include* t_4, *while* $pri(t_3) < pri(t_4)$. *If the* Many *concurrency semantics is considered instead of the* Single *concurrency semantics, then two big steps are possible:* $\langle (\!| \{t_1, t_2\} |\!), (\!| t_4 |\!) \rangle$ *and* $\langle (\!| \{t'_1, t_2\} |\!), (\!| t_3 |\!) \rangle$, *where the former big step includes includes* t_4 *instead of* t_3 *in the latter big step, while* $pri(t_3) < pri(t_4)$.

The Role of External Communication A BSML semantics that supports an External Input Events semantics with the X.P.I. Remainder event lifeline semantics can accommodate for a priority-consistent behaviour, regardless of the semantic option that determines how an external event is distinguished from an internal event, as specified in Table 3.4 on page 55. When a BSML semantics subscribes to the X.P.I. Remainder semantics, an input event that is received from the environment at the beginning of a big step persists throughout the big step, thus priority inconsistency according to the Negation of Triggers priority semantics cannot happen. The following predicate characterizes the constraint over the semantics of the external input event,

$$PXEvent \quad \equiv \quad \text{X.P.I. R}\textsc{emainder} \vee \neg\textbf{Negated External Events},$$

where the "¬**Negated External Events**" predicate refers to all BSML semantics that do not support a syntax for negated external events in the trigger of a transition, precluding the possibility of implementing the NEGATION OF TRIGGERS priority semantics using external events.

Example 38 *The model in Figure 5.11 shows a BSML model that uses environmental input events* i_1, i_2, *and* i_3. *Transition* t_3 *has a higher priority than transition* t_1 *and* t_5, *according to the* NEGATION OF TRIGGERS *priority semantics. Next, the behaviour of the model is analyzed when it resides in its default control states,* A_{11}, A_{21}, *and* A_{31}, *and* i_1, i_2, *and* i_3 *are present.*

If a BSML semantics is used that subscribes to the X.P.I REMAINDER *event lifeline semantics for external events and the* SINGLE *concurrency semantics then the following four big steps are possible:* $\langle \mathrm{t}_3, \mathrm{t}_6 \rangle$, $\langle \mathrm{t}_6, \mathrm{t}_3 \rangle$, $\langle \mathrm{t}_4, \mathrm{t}_6 \rangle$, *and* $\langle \mathrm{t}_6, \mathrm{t}_4 \rangle$, *which exhibit a priority-consistent behaviour.*

If a BSML semantics is used that subscribes to the X.P.I NEXT COMBO STEP *event lifeline semantics for external events, instead of the* X.P.I REMAINDER *semantics, then the following eight big steps are possible:* $T_1 = \langle (\!| \mathrm{t}_3, \mathrm{t}_6 |\!), (\!| \mathrm{t}_1 |\!) \rangle$, $T_2 = \langle (\!| \mathrm{t}_6, \mathrm{t}_3 |\!), (\!| \mathrm{t}_1 |\!) \rangle$, $T_3 = \langle (\!| \mathrm{t}_3, \mathrm{t}_6 |\!), (\!| \mathrm{t}_2 |\!) \rangle$, $T_4 = \langle (\!| \mathrm{t}_6, \mathrm{t}_3 |\!), (\!| \mathrm{t}_2 |\!) \rangle$, $T_5 = \langle (\!| \mathrm{t}_4, \mathrm{t}_6 |\!), (\!| \mathrm{t}_1 |\!) \rangle$, $T_6 = \langle (\!| \mathrm{t}_6, \mathrm{t}_4 |\!), (\!| \mathrm{t}_1 |\!) \rangle$, $T_7 = \langle (\!| \mathrm{t}_4, \mathrm{t}_6 |\!), (\!| \mathrm{t}_2 |\!) \rangle$, *and* $T_8 = \langle (\!| \mathrm{t}_6, \mathrm{t}_4 |\!), (\!| \mathrm{t}_2 |\!) \rangle$. *This behaviour is priority inconsistent because, for example,* $T_3 \gtrdot T_5$, *since* $pri(\mathrm{t}_3) > pri(\mathrm{t}_1)$.

If a BSML semantics is used that subscribes to the X.P.I NEXT COMBO STEP *event lifeline semantics for external events, instead of the* X.P.I REMAINDER *semantics, and the* MANY *concurrency semantics, instead of the* SINGLE *concurrency semantics, then the following four big steps are possible:* $T_1 = \langle (\!| \{\mathrm{t}_3, \mathrm{t}_6\} |\!), (\!| \mathrm{t}_1 |\!) \rangle$, $T_2 = \langle (\!| \{\mathrm{t}_3, \mathrm{t}_6\} |\!), (\!| \mathrm{t}_2 |\!) \rangle$, $T_3 = \langle (\!| \{\mathrm{t}_4, \mathrm{t}_6\} |\!), (\!| \mathrm{t}_1 |\!) \rangle$, *and* $T_4 = \langle (\!| \{\mathrm{t}_4, \mathrm{t}_6\} |\!), (\!| \mathrm{t}_2 |\!) \rangle$, *which exhibit a priority-inconsistent behaviour, because,* $T_2 \gtrdot T_3$, *since* $pri(\mathrm{t}_3) > pri(\mathrm{t}_1)$.

The Role of Interface Communication A BSML semantics that supports an Interface Events semantics with the ASYNCHRONOUS EVENT event lifeline semantics can accommodate for a priority-consistent behaviour: A generated interface event in the current big step will be only present in the next big step right from the beginning, similar to the X.P.I. REMAINDER event lifeline semantics for external events. The following predicate states this semantic characterization,

$$PIEvent \quad \equiv \quad \text{A}\textsc{synchronous} \ \text{E}\textsc{vent} \vee \neg\textbf{Negated Interface Events},$$

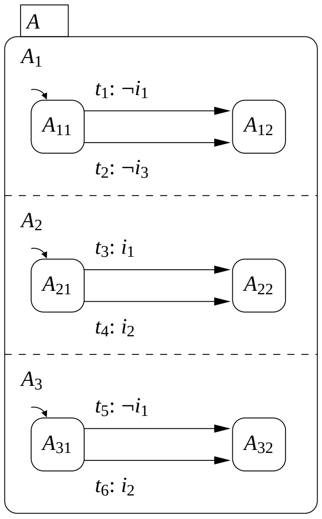


Figure 5.11: Priority consistency and the semantics of external events.

where the "¬**Negated Interface Events**" predicate refers to all BSML semantics that do not support a syntax for negated interface events in the trigger of a transition, precluding the possibility of implementing the NEGATION OF TRIGGERS priority semantics using interface events.

The Interface Variables in GC semantic aspect, similar to the Enabledness Memory Protocols semantic aspect, is not relevant for the priority-consistency semantics, because, unlike events, interface variables do not correspond to a priority semantics.

Proposition 5.5 *A BSML semantics that subscribes to the* NEGATION OF TRIGGERS *priority semantics, but not to the* SCOPE-PARENT *or the* SCOPE-CHILD *priority semantics, is priority consistent if and only if it satisfies* $\mathbf{P}'_{\mathbf{Negation}} \equiv \mathbf{P}_{\mathbf{Negation}} \wedge PXEvent \wedge PIEvent$.

Proof Idea. If a BSML semantics satisfies predicate $\mathbf{P}'_{\mathbf{Negation}}$, it is priority consistent according to the NEGATION OF TRIGGERS priority semantics. First, since it does not support internal events, because of $\mathbf{P}_{\mathbf{Negation}}$, only the roles of external and interface events, represented by predicates *PXEvent* and *PIEvent*, respectively, need to be considered. Predicate *PXEvent* characterizes priority-consistent BSML semantics for external events: An environmental input event is either present throughout a big step or is not present at all. Thus, at a snapshot of a model, either a lower-priority transition or a higher-priority transition of a model, but not both, can be included in different big steps of the model that are initiated from that snapshot. Similarly, predicate

PIEvent characterizes priority-consistent BSML semantics for interface events: An interface event is either present throughout a big step or is not present at all, precluding the possibility of a priority-inconsistent behaviour. Finally, the conjunction of the predicates $\mathbf{P}_{\mathbf{Negation}}$, *PXEvent*, and *PIEvent* effectively determines all priority-consistent BSML semantics that use different kinds of events.

Conversely, if a BSML semantics is priority consistent with respect to the NEGATION OF TRIGGERS priority semantics, it satisfies $\mathbf{P}'_{\mathbf{Negation}}$. Otherwise, at least one of the $\mathbf{P}_{\mathbf{Negation}}$, *PXEvent*, and *PIEvent* predicates does not hold. However, if any of these predicates does not hold, an example model can be constructed, as shown in Example 37 and Example 38, that has a priority inconsistent behaviour. Thus, $\mathbf{P}'_{\mathbf{Negation}}$ holds in a priority consistent semantics. □

Hierarchical and NEGATION OF TRIGGERS Priority Semantics

A BSML semantics might subscribe to both a hierarchical semantic option, i.e., one of the SCOPE-PARENT or the SCOPE-CHILD priority semantics, and the NEGATION OF TRIGGERS priority semantics. As described in Section 5.1.1, in such a BSML semantics, as described in Section 5.1.1, when the priority of two transitions can be compared both according to the NEGATION OF TRIGGERS priority semantics and according to the hierarchical priority semantics, the comparison according to the NEGATION OF TRIGGERS priority semantics has precedence.

Proposition 5.6 *A BSML semantics that subscribes to a hierarchical priority semantics together with the* NEGATION OF TRIGGERS *priority semantics is priority consistent if and only if it satisfies* $\mathbf{P} \equiv \mathbf{P}_{\mathbf{Hierarchical}} \wedge \mathbf{P}'_{\mathbf{Negation}}$.

Proof Idea. If a BSML semantics satisfies $\mathbf{P}_{\mathbf{Hierarchical}}$ and $\mathbf{P}'_{\mathbf{Negation}}$, in order for it to be priority inconsistent, it should be the case that a model specified in this BSML could create two big steps T_1 and T_2 such that $T_1 \succ T_2$. But that means that there exists a t_1 executed by T_1 and a t_2 executed by T_2 such that $pri(t_1) > pri(t_2)$. However, such a t_1 and t_2 cannot exist. If t_1 and t_2 are priority comparable according to the hierarchical priority semantics, but not the NEGATION OF TRIGGERS semantics, then t_1 should have been executed in the first small step of T_1, or otherwise the TAKE ONE big-step maximality semantics would not have allowed it to be executed (since its scope is a parent of the transitions in the first small step). But if so, then the first small step of T_2, which is initiated from the same snapshot as the first small step of T_1, should have included t_1, either instead of t_2 or together with t_2 (if the concurrency and consistency semantics allows that). In either case, it cannot be the case that $T_1 \gtrdot T_2$: In the former case, both big steps execute the high-priority transition t_1, while in the latter case, T_1 should also include t_2, or a similar transition of the same hierarchical priority, in addition to t_1. If t_1 and t_2 are priority comparable according to the Negation of Triggers priority semantics, then if t_1 could have been taken, t_2 could not have been taken because the trigger of t_2 would have been present throughout the big step. Thus, if a BSML semantics satisfies $\mathbf{P_{Hierarchical}} \wedge \mathbf{P'_{Negation}}$, it is priority consistent.

Conversely, if such a BSML semantic is priority consistent, it should also satisfy $\mathbf{P_{Hierarchical}} \wedge \mathbf{P'_{Negation}}$. If any of the conjunct is not satisfied, say $\mathbf{P_{Hierarchical}}$ is not satisfied, a counter-example model can be constructed that has a priority-inconsistent behaviour according to the hierarchical priority semantics, as described earlier in the chapter in Example 36. Thus, the BSML semantics, indeed, satisfies $\mathbf{P_{Hierarchical}} \wedge \mathbf{P'_{Negation}}$. □

5.3.3 Determinate Semantics

First, the determinate BSML semantics with respect to variables are identified, followed by the ones that are determinate with respect to events. Lastly, the BSML semantics that subscribe to both semantic options are considered.

Determinate with Respect to Variables

A BSML semantics is determinate with respect to variables for single-assignment models if it either follows the RHS Big Step assignment memory protocol, or follows the Take One big-step maximality semantics and the Many concurrency semantics, or does not support variable assignments at all. Formally,

$$\begin{array}{rcl} \mathbf{D_{Variables}} & \equiv & [\neg\textbf{Variable Assignments} \vee \text{RHS Big Step}] \vee \\ & & [(\text{RHS Small Step} \vee \text{RHS Combo Step}) \Rightarrow (\text{Take One} \wedge \text{Many})]. \end{array}$$

Example 39 *Figure 5.12 shows an example model of how the violation of predicate* $\mathbf{D_{Variables}}$ *results in a non-determinate behaviour. The model is considered when it resides in its default control states. It is meant to do two things: First, it should swap the values of integer variables* x *and* y*; and second, it should compute the values of the sum and the difference of* x *and* y

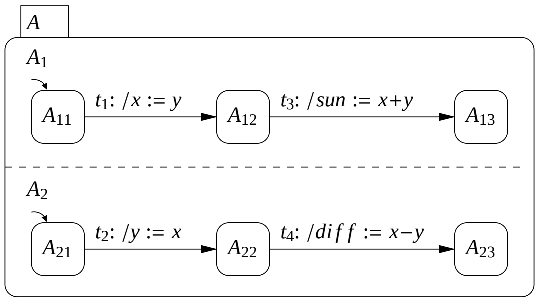


Figure 5.12: Examples of (non-) determinate behaviour with respect to variables.

according to their initial values at the beginning of the big step. If a semantics that subscribes to the SINGLE *concurrency semantics, the* TAKE MANY *big-step maximality semantics, and the* RHS SMALL STEP *assignment memory protocol, which violates* $\mathbf{D_{Variables}}$*, 6 big steps are possible, with two different outcomes, none of which achieves the intended behaviour. For example, big step* $\langle t_1, t_3, t_2, t_4 \rangle$ *assigns the value of* y *to* x *but not vice versa, and furthermore,* $\mathrm{sum} = 2 \times \mathrm{y}$ *and* $\mathrm{diff} = 0$*. If the* RHS BIG STEP *assignment memory protocol, instead of the* RHS SMALL STEP *assignment memory protocol, is chosen, again there are 6 big steps possible, all of which achieve the intended behaviour.*

In the model in Example 39, if a BSML that subscribes to the TAKE MANY maximality semantics, the RHS SMALL STEP assignment memory protocol, and the MANY concurrency semantics is chosen, which violates $\mathbf{D_{Variables}}$, only big step $\langle \{t_1, t_2\}, \{t_3, t_4\} \rangle$ is possible, which exhibits a determinate behaviour but calculates the wrong difference, $diff = y - x$, instead of $diff = x - y$. It might be tempting to replace the consequent of predicate $\mathbf{D_{Variables}}$ with only "MANY", but the new consequent does not always result in a determinate behaviour. The next example demonstrates this problem.

Example 40 *The model in Figure 5.13 shows a model of a system that controls the operation of a chemical plant. The environmental input events* inc_one *and* inc_two *indicate that the amount of a chemical substance should be incremented by one or two, respectively. If the two events are received simultaneously, the intended behaviour is to increment the amount of the chemical substance three units. The model is considered when: it resides in its default control states,* inc = inc_1 = inc_2 = 0*, and the environmental input events* inc_one *and* inc_two *are received simultaneously. The model is single-assignment only if environmental input event* reset *cannot*

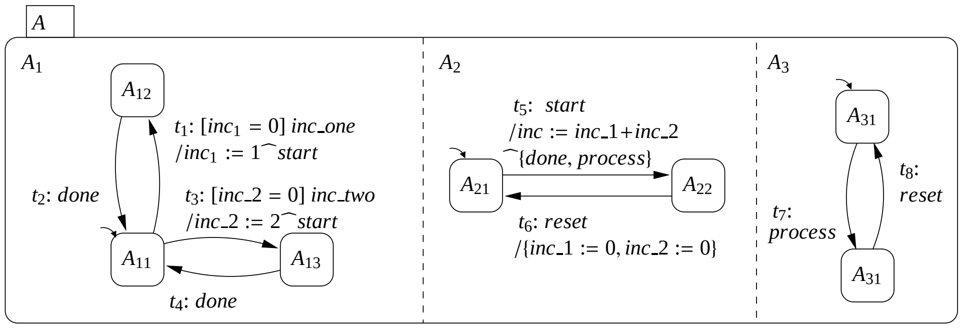


Figure 5.13: An example of a non-determinate behaviour.

be received neither together with inc_one *nor together with* inc_two. *If a BSML semantics that subscribes to the* Take Many *maximality semantics, the* RHS Small Step *assignment memory protocol, the* P.I. Remainder *event lifeline semantics for internal events, the* X.P.I. Next Combo Step *event lifeline semantics for external events, and the* Many *concurrency semantics is chosen, two big steps are possible:* $\langle t_1, t_5, \{t_2, t_7\}, t_3, t_4\rangle$ *and* $\langle t_3, t_5, \{t_4, t_7\}, t_1, t_2\rangle$, *with the value of* inc *being 1 in the former big step and 2 in latter big step, which is a non-determinate behaviour.*

The following lemma explains why the semantics in the above example is not determinate, as opposed to when the Take One maximality semantics is chosen.

Lemma 5.7 *In a BSML semantics that subscribes to the* Take One *maximality semantics and the* Many *concurrency semantics, if two big steps, T and T', of a single-assignment model consist of the same sets of transitions, then they are the same.*

Proof Idea. The above claim can be proved by inductively arguing over the small steps of such two big steps T and T'. Starting from snapshots $T.b$ and $T'.b$, which are the source snapshots of T and T', and are the same, their first small steps, T^1 and T'^1, should be the same. If not, let us assume that there exists a t, such that $t \in (T^1.\tau - T'^1.\tau)$, meaning that t is executed by the first small step of T but not the one of T'. However, such a t does not exist: Transition t can only be not taken by T'^1 if it is replaced by a $t' \in T'^1$ such that $conflict(t, t')$. But if that is true, T' can never execute t because the Take One big-step maximality semantics disallows such a t to be taken after t' has been taken, and thus it is not possible that T and T' have the same set of transitions, which is contradiction. Thus, it should be the case that $T^1.\tau = T'^1.\tau$. Similarly, it should be the case that all T^i's and T'^i's, such that $1 < i \leq T.length$ and $1 < i \leq T'.length$, are the same. Therefore T and T' are the same. □

The Role of Interface Variables The role of the Interface Variables in RHS semantic aspect in determining a BSML semantics as determinate is similar to that of the Assignment Memory Protocol, described by predicates $\mathbf{D}_{\mathbf{Variables}}$. The below predicate specifies the corresponding constraints over the choice of the semantic options of the Interface Variables in RHS:

$$\begin{aligned} DIAssign \quad \equiv \quad & [\neg\textbf{Interface Variables in RHS} \vee \text{RHS ASYNCHRONOUS VARIABLE}] \vee \\ & [\text{RHS WEAK SYNCHRONOUS VARIABLE} \Rightarrow (\text{TAKE ONE} \wedge \text{MANY})]. \end{aligned}$$

Proposition 5.8 *A BSML semantics is determinate with respect to variables if and only if its constituent semantic options satisfy the predicate* $\mathbf{D}'_{\mathbf{Variables}} \equiv \mathbf{D}_{\mathbf{Variables}} \wedge DIAssign$.

Proof Idea. If a BSML semantics satisfies $\mathbf{D}'_{\mathbf{Variables}}$, it is determinate because each of its constituent assignment semantic options falls into one of the following categories: (i) the semantic option uses the values of variables at the beginning of a big step for assignments, i.e., the RHS BIG STEP and RHS ASYNCHRONOUS VARIABLE semantic options, meaning that the order of the assignments in two big steps with the same set of transition does not affect their final outcomes; or (ii) the semantic option is used in a BSML semantics that satisfies the "TAKE ONE ∧ MANY" predicate, which, by Lemma 5.7, means that two big steps consisting of the same transitions are indeed the same. In both cases, however, the BSML semantics is determinate.

Conversely, if a BSML semantics is determinate with respect to variables it must satisfy $\mathbf{D}'_{\mathbf{Variables}}$. Otherwise, a counter example model, similar to the ones in Example 39 and Example 40, can be constructed that has a non-determinate behaviour. □

Determinate with Respect to Events

A BSML semantics is determinate with respect to events, if the following predicate is true about it,

$$\begin{aligned} \mathbf{D}_{\mathbf{Events}} \quad \equiv \quad & [\neg\textbf{Generated Events} \vee \text{P.I. REMAINDER}] \vee \\ & [(\text{P.I. NEXT SMALL STEP} \vee \text{P.I. NEXT COMBO STEP}) \Rightarrow (\text{TAKE ONE} \wedge \text{MANY})]. \end{aligned}$$

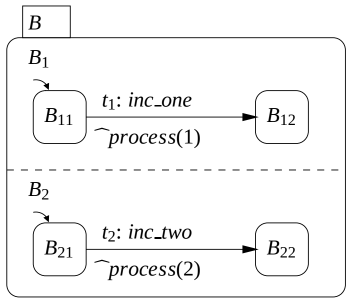


Figure 5.14: Examples of (non-) determinate behaviour with respect to events.

If events with parameters are considered, then it is also required that the combine function for parameters, which determines the value of a parameter of an event when it is generated more than once during a big step, to be both commutative and associative.

Example 41 *Figure 5.14 shows an example of how the violation of predicate* $\mathbf{D}_{\mathbf{Events}}$ *results in a non-determinate behaviour with respect to events. The model is considered when it resides in its default control states. The model in Figure 5.14 is similar to the model in Figure 5.13 in Example 40. It represents a system that controls the operation of a chemical plant.*[4] *There are two processes, modelled by control states* B_1 *and* B_2*, which increment the amount of a chemical substance in the plant by one or two units, respectively. Again, if the environmental input events* inc_one *and* inc_two *are received simultaneously, the intended behaviour is to increment the amount of the chemical substance three units. If a BSML semantics that subscribes to the* SINGLE *concurrency semantics together with the* P.I. NEXT SMALL STEP *event lifeline semantics is chosen, which violates* $\mathbf{D}_{\mathbf{Events}}$*, there are two big steps possible:* $\langle \mathrm{t}_2, \mathrm{t}_1 \rangle$ *and* $\langle \mathrm{t}_1, \mathrm{t}_2 \rangle$*, with the former big step resulting in* process(1) *while the latter big step resulting in* process(2)*, at the end of their corresponding big steps. If the* P.I. REMAINDER *event lifeline semantics is chosen, instead of the* P.I. NEXT SMALL STEP*, the same two big steps are possible but the result would always be* process(3)*. If the* MANY *concurrency semantics is chosen, instead of the* SINGLE *concurrency semantics, together with the* P.I. NEXT SMALL STEP*, only one big step is possible,* $\langle \{\mathrm{t}_1, \mathrm{t}_2\} \rangle$*, which results in* process(3)*.*

The Role of External Events The role of the External Output Events semantic sub-aspect is similar to the role of the Event Lifeline semantic aspect for internal events in determining

[4]This model is adapted from a model in [36], which in turn is inspired by the motivating example in [2].

a BSML semantics as determinate. As shown in the feature diagram in Figure 5.7, an External Output Events semantics is instantiated by an option that determines which events are considered as output events and by an option belonging to "*Event Options*", which are exactly the same set of options as for the Event Lifeline semantics for internal events but with different names, and determine the extent that an environmental output event persists in a big step. It is only the second option, belonging to the "*Event Options*", that has a role in determining a BSML semantics as determinate or not. To extend the class of determinate BSML semantics to include external output events, it suffices to conjoin predicate $\mathbf{D}_{\mathbf{Events}}$ above with the below predicate,

$$\begin{aligned} \textit{DOEvent} \quad \equiv \quad & [\neg \textbf{External Output Events} \vee \text{O.P.I. }\textsc{Remainder}] \vee \\ & [(\text{O.P.I. }\textsc{Next Small Step} \vee \text{O.P.I. }\textsc{Next Combo Step}) \Rightarrow (\textsc{Take One} \wedge \textsc{Many})]. \end{aligned}$$

where the prefix "O" for semantic options above refers to the event lifeline semantic options of external output events, which are shown as "*Event Options*" in Figure 5.7.

The External Events semantic aspect is not relevant in determining a BSML semantics as determinate because it specifies the semantics of input events, rather than the events that are generated during a big step.

The Role of Interface Events The role of the Interface Events semantic aspect in determining a BSML semantics as determinate is similar to that of the Event Lifeline semantic aspect. The below predicate specifies the corresponding constraints over the choice of the semantic options of the Interface Events semantic aspect:

$$\textit{DIEvent} \quad \equiv \quad \textit{true},$$

where, "*true*" here means any non-transition–aware semantic option of the Interface Events semantic aspect.

The Interface Variables in GC semantic aspect has no role in determining a BSML semantics as determinate because, similar to the Enabledness Memory Protocol for internal variables, it only determines the readiness of a transition but not the values of variables.

Proposition 5.9 *A BSML semantics is determinate with respect to variables if and only if its constituent semantic options satisfies the predicate* $\mathbf{D}'_{\mathbf{Events}} \equiv \mathbf{D}_{\mathbf{Events}} \wedge \textit{DOEvent}$.

Proof Idea. If a BSML semantics satisfies $\mathbf{D}'_{\mathbf{Events}}$, it is determinate because each of its constituent event lifeline semantic options falls into one of the following categories: (i) the event lifeline semantics accumulates events throughout a big step, meaning that if two big steps have the same sets of transitions, they accumulate the same sets of transitions; and (ii) the event lifeline semantics is used in a BSML semantics that satisfies the "Take One ∧ Many" predicate, which, by Lemma 5.7, means that two big steps consisting of the same transitions are indeed the same. In both cases, however, the BSML semantics is determinate. In the latter case, the set of generated events of a BSML model at the end of each of its big step is equal to the set of generated events by its last combo step or small step, based on the choice of the event lifeline semantics for a particular kind of event. If events with parameters are used, as long as a commutative, associative combination function is used to combine the values of events, the BSML semantics will be determinate.

Conversely, if a BSML semantics is determinate with respect to variables it must satisfy $\mathbf{D}'_{\mathbf{Variables}}$. Otherwise, a counter example model, similar to the ones in Example 41, can be constructed that has a non-determinate behaviour □

Determinate with Respect to Variables and Events

The following proposition states the constraints over the choices of the semantic options of the class of determinate BSML semantics, with respect to both variables and events.

Proposition 5.10 *A BSML semantics is determinate with respect to variables and events if and only if its constituent semantic options satisfies the predicate* $\mathbf{D} \equiv \mathbf{D}'_{\mathbf{Variables}} \wedge \mathbf{D}'_{\mathbf{Events}}$.

Proof Idea. If a BSML semantics satisfies **D**, in order for it to be non-determinate, it should be the case that a model specified in this BSML could create two big steps T_1 and T_2, from the same source snapshot, that have the same set of transitions, but they have different values for variables and/or have different statuses of events at their corresponding destination snapshots. But such a pair of big steps cannot exist: The values of variables cannot be different at their destination snapshots because the BSML satisfies $\mathbf{D}'_{\mathbf{Variables}}$ and because of Proposition 5.8; also, the statuses of events cannot be different at their at their destination snapshots because the BSML satisfies

$\mathbf{D'_{Events}}$ and because of Proposition 5.9. Thus, the BSML semantics is determinate with respect to variables and events.

Conversely, if a BSML semantics is determinate, its constituent semantic options should satisfy **D**. Otherwise, depending on whether it violates $\mathbf{D'_{Variables}}$ and/or $\mathbf{D'_{Events}}$, counter example models similar to the ones in Example 40, and Example 41, respectively, can be constructed, which show the semantics is not determinate with respect to variables and/or events. Thus, the BSML semantics satisfies **D**. □

5.4 Quality Attributes and Syntactic Well-formedness

Sections 5.3.1, 5.3.2, and 5.3.3 specified the semantic characteristics that each enumerated the BSML semantics that satisfy one of the three semantic quality attributes. It is, however, also possible to use a combination of syntactic and semantic criteria to specify such classes of BSML semantics. This section presents two examples of such characterizations. A language designer or a modeller, based on an application or a domain, can create similar syntactic, semantic characterization of a set of BSML semantics that satisfy a certain semantic quality attribute.

5.4.1 A Syntactic Well-Formedness Criterion for Non-Cancelling

In Proposition 5.3, on page 161, it was shown that a BSML semantics is non-cancelling if and only if its semantic options satisfy predicate $\mathbf{N} \equiv \mathbf{N'_{Steady}} \vee \mathbf{N_{Maximizer}}$, where

$$\begin{array}{rcl}\mathbf{N'_{Steady}} & \equiv & (\mathit{Big\_Semantics} \wedge \mathit{XBig\_Semantics} \wedge \mathit{IBig\_Semantics}) \vee \\ & & (\mathit{Combo\_Semantics} \wedge \mathit{XCombo\_Semantics})\end{array}$$

This section shows that if a BSML model is single assignment, as described in Definition 5.4, on page 150, then predicate *Big_Semantics*, copied below for convenience,

$$\begin{array}{rcl}\mathit{Big\_Semantics} & \equiv & [(\text{GC B}\textsc{ig} \text{ S}\textsc{tep} \vee \neg\textbf{Guard Conditions}) \wedge \neg\textbf{Event Triggers}] \wedge \\ & & [(\text{T}\textsc{ake} \text{ O}\textsc{ne} \vee \text{N}\textsc{o} \text{ P}\textsc{riority}) \wedge \neg\text{D}\textsc{ataflow}],\end{array}$$

can be relaxed by removing the "DATAFLOW" term, resulting in predicate *Big_Semantics′*:

$$\begin{aligned} Big\_Semantics' \quad \equiv \quad & [(\text{GC BIG STEP} \vee \neg \textbf{Guard Conditions}) \wedge \neg \textbf{Event Triggers}] \wedge \\ & [(\text{TAKE ONE} \vee \text{NO PRIORITY})]. \end{aligned}$$

A single-assignment model is one that it does not produce any big step such that two transitions in the big step assign values to the same variable.

Example 42 *The model in Figure 5.15 is the model characterized in the third paragraph in Example 35, on page 156, which showed a cancelling behaviour. This model is not single assignment because when the model resides in its default control states, big step* $\langle t_1, t_3 \rangle$ *is possible, which assigns values twice to v. Furthermore, this model has a cancelling behaviour because in the above big step after the execution of* t_1, t_2 *is executable, but once* t_3 *is executed,* t_2 *becomes disabled.*

If transition t_3 *in model in Figure 5.15 is changed so that,*

$$t_3 : /v' := 2,$$

then the model is a single-assignment model, and a cancelling behaviour cannot happen.

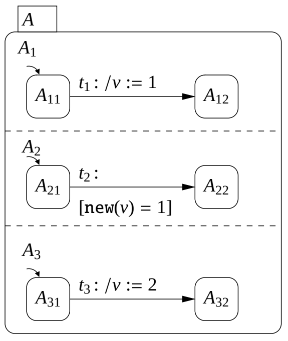


Figure 5.15: An example model with dataflow over variable *v*.

Proposition 5.11 *A BSML that only allows single-assignment BSML models is non-cancelling if and only if it satisfies predicate* $\mathbf{N} \equiv \mathbf{N}''_{\mathbf{Steady}} \vee \mathbf{N}_{\mathbf{Maximizer}}$, *where*

$$\begin{aligned} \mathbf{N}''_{\mathbf{Steady}} \quad \equiv \quad & (Big\_Semantics' \wedge XBig\_Semantics \wedge IBig\_Semantics) \vee \\ & (Combo\_Semantics \wedge XCombo\_Semantics). \end{aligned}$$

Proof Idea. The proof is the same as the proof for Proposition 5.3, on page 161, except that the part that considers the role of the DATAFLOW semantic option needs to be removed; i.e., the part that says, "The DATAFLOW semantic option should not be chosen, because a ready transition can become unready if a variable is assigned more than once during a big step, as described in Example 35, on page 156." This part is not relevant for single-assignment models because for a transition, t, that uses the `new` operator as a prefix of at least one of the variables in $gc(t)$, if it becomes executable, it cannot become disabled through its $gc(t)$: First, the values of variables in $gc(t)$ that are prefixed by `new` cannot change because the model is single assignment; and second the values of variables in $gc(t)$ that are not prefixed by `new` cannot change because of the GC BIG STEP enabledness memory protocol. □

5.4.2 A Syntactic Well-Formedness Criterion for Priority Consistency

As stated in Proposition 5.5, on page 168, a BSML semantics is priority consistent with respect to the NEGATION OF TRIGGERS priority semantics, if and only if its constituent semantic options satisfy predicate $\mathbf{P'_{Negation}} \equiv \mathbf{P_{Negation}} \wedge PXEvent \wedge PIEvent$, where

$$\begin{array}{lcl} \mathbf{P_{Negation}} & \equiv & \neg\textbf{Negated Events} \\ PXEvent & \equiv & \text{X.P.I. R}\textsc{emainder} \vee \neg\textbf{Negated External Events}, \\ PIEvent & \equiv & \text{A}\textsc{synchronous} \text{ E}\textsc{vent} \vee \neg\textbf{Negated Interface Events}. \end{array}$$

Next, a syntactic well-formedness condition is introduced that relaxes the *PXEvent* predicate above to allow more event lifeline semantic options for external events to be considered in the characterization of the class of priority-consistent BSML semantics. First, some needed definitions are presented.

Definition 5.5 *For a BSML model and a set of its transitions, T, T is* neighbouring *if for each pair of distinct transitions, t_1 and t_2, in T, their scopes are the same; i.e., $scope(t_1) = scope(t_2)$.*

Lemma 5.12 *For a BSML model, its set of transitions, T, can be partitioned into a unique set of* neighbourhood *sets of transitions $T_G = \{T_1, \cdots, T_m\}$, where $m \geq 1$, such that each of T_i's, $1 \leq i \leq m$, is a maximal set of neighbouring transitions.*

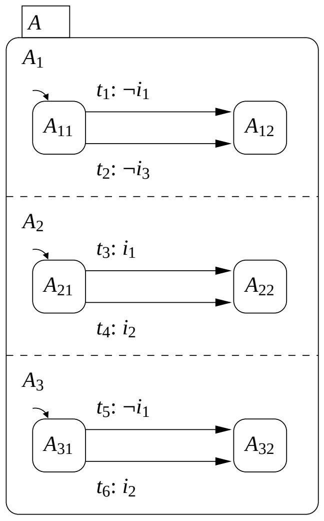


Figure 5.16: A BSML model that is not priority clustered.

Proof Idea. The set of sets of transitions T_G can be created by an algorithm that iterates through all transitions in T and assigns a transition, t, to a set of transitions whose scopes are the same as t's; if such a set of transitions does not exist, a new set of transitions is created in T_G and t is assigned to it. Once all transitions are visited, the algorithm ends with a set of sets of neighbouring transitions, T_G: By definition, the set of transitions in each set are the transitions whose scopes are pairwise the same. T_G is unique because each of the T_i's, $1 \leq i \leq m$, is maximal. $\square$

Definition 5.6 *For a BSML model, its set of transitions, T, and its set of neighbourhood sets of transitions, $T_G = \{T_1, \cdots, T_m\}$, the model is* priority clustered, *if for each distinct pairs of sets of neighbourhood transitions $T_i, T_j \in T_G$, their transitions do not share any positive or negated literals in their triggers. Formally, if*

$$\forall t_i \in T_i \cdot \forall t_j \in T_j \cdot (pos\_trig(t_i) \cup neg\_trig(t_i)) \cap (pos\_trig(t_j) \cup neg\_trig(t_j)) = \emptyset.$$

Example 43 *The model in Figure 5.16, which is the same model as in Figure 5.11, on page 168, copied here for convenience, is not priority clustered. For example, $(pos\_trig(\mathrm{t}_1) \cup neg\_trig(\mathrm{t}_1)) \cap (pos\_trig(\mathrm{t}_3) \cup neg\_trig(\mathrm{t}_3)) = \{\mathrm{i}_1\} \neq \emptyset$, although t_1 and t_3 are not neighbouring transitions.*

For the class or priority-clustered BSML models, the *PXEvent* predicate can be relaxed to the following predicate,

$$\begin{aligned} \mathit{PXEvent}' \equiv\ & (\text{X.P.I. }\textsc{Remainder} \vee \neg\textbf{Negated External Events}) \vee \\ & [\text{X.P.I. }\textsc{Next Combo Step} \wedge ((\textsc{Take One} \wedge \textsc{Many}) \vee \mathit{XGC})] \vee \\ & [\text{X.P.I. }\textsc{Next Small Step} \wedge \textsc{Take One} \wedge \textsc{Many}], \end{aligned}$$

where

$$\begin{aligned} \mathit{XGC} \equiv\ & (\neg\text{GC }\textsc{Small Step} \vee \neg\textbf{Guard Conditions}) \wedge \\ & (\text{GC }\textsc{Asynchronous Variable} \vee \neg\textbf{Interface Variables in GC}). \end{aligned}$$

Example 44 *The model in Example 5.17 shows a priority-clustered BSML model. The model is considered when it resides in its default control states, and when environmental input events* i_1, i_2, i_3, *and* i_4 *are all present, and variable* $\mathrm{c} = \mathit{true}$. *If the BSML semantics that subscribes to the* X.P.I. Next Combo Step *event lifeline semantics for environmental input events, the* GC Small Step *enabledness memory protocol for internal variables, and the* Single *concurrency semantics is chosen, which violates* $\mathit{PXEvent}'$, *then three big steps are possible:* $T_1 = \langle(\!|\mathrm{t}_1, \mathrm{t}_4|\!)\rangle$, $T_2 = \langle(\!|\mathrm{t}_2, \mathrm{t}_4|\!)\rangle$, *and* $T_3 = \langle(\!|\mathrm{t}_4|\!)(\!|\mathrm{t}_3|\!)\rangle$. *However, this behaviour is priority inconsistent:* T_1 *executes* t_1 *while* T_3 *executes* t_3, *although* $\mathit{pri}(\mathrm{t}_1) > \mathit{pri}(\mathrm{t}_3)$. *If the* GC Combo Step *is chosen, instead of the* GC Small Step *semantic option, which satisfies* $\mathit{PXEvent}'$, *then the following four big steps are possible:* $\langle(\!|\mathrm{t}_1, \mathrm{t}_4|\!)\rangle$, $\langle(\!|\mathrm{t}_2, \mathrm{t}_4|\!)\rangle$, $\langle(\!|\mathrm{t}_4, \mathrm{t}_1|\!)\rangle$, *and* $\langle(\!|\mathrm{t}_4, \mathrm{t}_2|\!)\rangle$, *which is a priority consistent behaviour. Similar counter examples can be shown to exist for the model when the* $\mathit{PXEvent}'$ *predicate is violated, for example, through its third conjunct.*

The next example demonstrates the necessity of the priority clustered well-formedness criteria in establishing a priority-consistent BSML semantics using the $\mathit{PXEvent}'$ predicate.

Example 45 *The model in Example 5.18 is similar to the model in Figure 5.17, in Example 44, except that it has an extra Or control state* A_3 *and an extra transition* t_6. *This new model is not priority clustered.*

Again, the model is considered when it resides in its default control states, and when environmental input events i_1, i_2, i_3, *and* i_4 *are all present, and variable* $\mathrm{c} = \mathit{true}$. *If the BSML semantics that subscribes to the* X.P.I. Next Combo Step *event lifeline semantics for environmental input events, the* GC Combo Step *enabledness memory protocol for internal variables,*

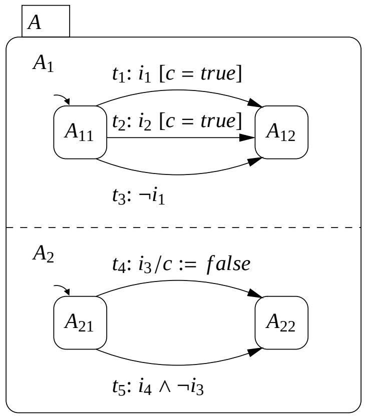


Figure 5.17: Priority consistency in a priority-clusterred BSML model.

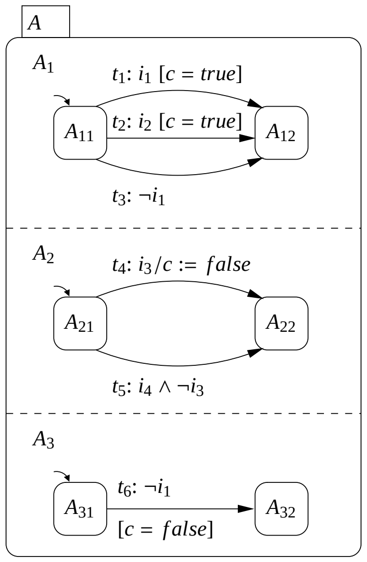


Figure 5.18: Priority inconsistency in a model that is not priority clusterred.

and the SINGLE *concurrency semantics is chosen, which satisfies predicate* $PXEvent'$*, then the following four big steps are possible:* $T_1 = \langle (\!| \mathrm{t}_1, \mathrm{t}_4 |\!), (\!| \mathrm{t}_6 |\!) \rangle$, $T_2 = \langle (\!| \mathrm{t}_2, \mathrm{t}_4 |\!), (\!| \mathrm{t}_6 |\!) \rangle$, $T_3 = \langle (\!| \mathrm{t}_4, \mathrm{t}_1 |\!), (\!| \mathrm{t}_6 |\!) \rangle$, *and* $T_4 = \langle (\!| \mathrm{t}_4, \mathrm{t}_2 |\!), (\!| \mathrm{t}_6 |\!) \rangle$*. However, this behaviour is priority inconsistent because, for example,* T_1 *executes* t_1*, while* T_4 *does not execute it, but executes* t_6*, although* $pri(\mathrm{t}_1) > pri(\mathrm{t}_6)$*.*

Similarly, if a BSML semantics that subscribes to the X.P.I. NEXT SMALL STEP *event lifeline semantics for environmental input events, the* GC SMALL STEP *enabledness memory protocol for internal variables, and the* MANY *concurrency semantics is chosen, which satisfies predicate* $PXEvent'$*, then the following four big steps are possible:* $T'_1 = \langle \{\mathrm{t}_1, \mathrm{t}_4\}, \mathrm{t}_6 \rangle$ *and* $T'_2 = \langle \{\mathrm{t}_2, \mathrm{t}_4\}, \mathrm{t}_6 \rangle$*. Again this behaviour is priority inconsistent because* $pri(\mathrm{t}_1) > pri(\mathrm{t}_6)$*, and* T'_1 *executes* t_1*, while* T'_2 *executes* t_6*.*

Proposition 5.13 *For a BSML that only allows priority-clustered BSML models, it is priority consistent if and only if it satisfies predicate* $\mathbf{P}_{\mathbf{Negation}} \wedge PXEvent' \wedge PIEvent$*.*

Proof Idea. Using the same arguments as in the proof of Proposition 5.5, it can be shown that if a BSML semantics for priority-clustered models is priority consistent, then it satisfies predicates $\mathbf{P}_{\mathbf{Negation}}$ and $PIEvent$. It remains to show that it also satisfies the $PXEvent'$ predicate.

According to the first disjunct of $PXEvent'$, an environmental input event is either present throughout a big step or is not present at all. Thus, at a snapshot of a model, either a lower-priority transition or a higher-priority transition of a model, but not both, can be included in different big steps of the model that are initiated from that snapshot.

According to the second disjunct, an environmental input event that is present in the first combo step of a big step becomes absent in the second combo step. However, because only priority-clustered BSML models are allowed, meaning that the reaction of the model to an environmental input event is modelled only by a set of neighbouring transitions, and because either the MANY concurrency semantic has been chosen or the GC of transitions remain the same during the combo-step, because of the XGC predicate, the model has a chance to react to the environmental inputs in a manner that respects the priority consistency criteria. If the MANY concurrency semantics is chosen, the highest priority transitions each belonging to a neighbourhood set of transitions are all executed during the first small step together. The next small steps and the next combo steps do not cause a priority inconsistent behaviour because if a big step executes a high-priority transition, t, in its first small step, then any other big steps would either execute the same high-priority transition or a transition that has the same priority as t, which precludes the possibility of executing a transition with a lower priority than t, because of the Take One big-step maximality semantics. Similarly, if XGC is true, the highest priority transitions each belonging to a neighbourhood set of transitions are all executed during the first combo step, possibly sequentially. However, since XGC is true, a high-priority transition remains ready during the first combo step. Again, a low-priority transition cannot execute unless a high-priority transition is not ready in the first combo step.

According to the third disjunct, an environmental input event that is present in the first small step of a big step becomes absent in the second small step. Again, a priority-inconsistent behaviour is not possible because a lower-priority transition can be taken in the small step after the first one, only if a higher-priority transition has not been ready in the first small step, which means it cannot become enabled in later small steps either.

Conversely, if a BSML semantics is priority consistent with respect to the Negation of Triggers priority semantics, it satisfies $\mathbf{P}_{\mathbf{Negation}} \wedge PXEvent' \wedge PIEvent$. Otherwise, at least one of the $\mathbf{P}_{\mathbf{Negation}}$, $PXEvent$, and $PIEvent$ predicates does not hold. However, if any of these predicates does not hold, an example model can be constructed, similar to the ones shown in Example 37, Example 44, and Example 45, that has a priority inconsistent behaviour. Thus, $\mathbf{P}_{\mathbf{Negation}} \wedge PXEvent' \wedge PIEvent$ holds in a BSML that allows only priority clustered BSML model and is priority consistent. □

5.5 Related Work: Semantic Properties

Huizing and Gerth identified the three semantic quality attributes of *responsiveness*, *modularity*, and *causality* only for Single concurrency semantics and events [50]. Their responsiveness criterion requires that the reaction of a model to an environmental input be observed in the same big step that the input is received. The semantics in their framework that is not responsive is semantics A, which corresponds to the the Asynchronous Event interface event semantics in this dissertation. Their modularity criterion requires that a generated event by a model is treated the same as an event received from the environment, as described in Chapter 3. The two semantics in their framework that are modular, namely, semantics A and D, can be easily shown to be also non-cancelling. Semantics D corresponds to the Take One maximality semantics together with the Whole event lifeline semantics. Their causality criterion for events has been considered in Chapter 3; the Whole semantic option is the only event-lifeline semantics that is not causal.

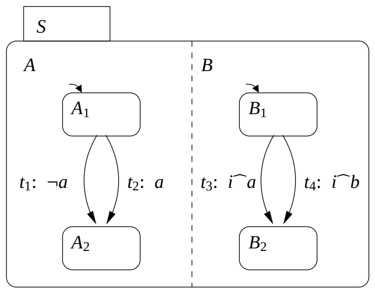


Figure 5.19: Global consistency vs. priority consistency .

Pnueli and Shalev introduced a *globally consistent* event semantics [86], as described on page 49, which is the same as the P.I. Remainder event lifeline semantics except that if the absence of an event has made a transition enabled in an early small step, that event is not generated later. This semantics introduces a notion of priority consistency with respect to the Negation of Triggers priority semantics, but at the scope of individual big steps: It is not possible for a big step to take a lower-priority transition earlier in the big step while taking a higher-priority transition later in the big step. A globally-consistent semantics is not a priority-consistent BSML semantics. For example, in the model in Figure 5.19, if the model resides in its default control states, environmental input *i* is present and persists throughout a big step, and a globally-consistent event semantics is considered, the following three big steps are possible: $T_1 = \langle \mathrm{t}_1, \mathrm{t}_4 \rangle$, $T_2 = \langle \mathrm{t}_4, \mathrm{t}_1 \rangle$, and $T_3 = \langle \mathrm{t}_3, \mathrm{t}_2 \rangle$. Big step $\langle \mathrm{t}_1, \mathrm{t}_3 \rangle$ is not a possible big step because event *a* is both generated and its absence triggers a transition. These three possible big steps, however, exhibit a priority-inconsistent behaviour, for example, because: T_1 and T_2 execute t_1, while T_3 executes t_1, although $pri(t_1) < pri(t_2)$. Global consistency semantics is not relevant for the BSML semantics that are priority consistent because by predicate $\mathbf{P}_{\mathbf{Negation}}$, described on page 164, priority-consistent BSML semantics do not support internal events.

Synchronous languages are used to model/program reactive systems that are meant to behave deterministically [40]. In the deconstruction in Chapter 3, the un-clocked variations of synchronous languages, such as Esterel [14] and Argos [68], are categorized as BSMLs that support the Whole event lifeline semantics. A model is deterministic if its reaction to an environmental input as a big step always results in a unique destination snapshot. Determinism is related to determinacy: A deterministic semantics is by definition determinate, but not vice versa. A determinate semantics does not preclude the possibility of a model reacting to a single environmental input via two big steps with different sets of transitions. In the presence of variables, determinism can be only considered as the property of a model but not a semantics, because, as opposed to events, variables can have infinite, or large, ranges, precluding the possibility of handling determinism at the level of the description of a semantics. In the absence of variables, for example, in pure Esterel, a *constructive* [13] and a *globally deterministic* [93] semantics have been developed. Similar semantics has been developed for Argos [68].

Similar concepts as our semantic quality attributes have been considered in different models of computation, but at the level of models instead of semantics. For example, in Petri nets, the notion of *persistence* [62], which requires a transition to remain enabled until it is taken, is similar to our non-cancelling semantic quality attribute. In asynchronous circuits, the notions of *semi-modularity* and *quasi semi-modularity* are similar to our non-cancelling semantic quality attribute, and the notion of *speed independence* is analogous to our determinacy semantic quality attribute [17, 90]. Janicki and Koutny introduce the notion of *disabling* in the context of a relational model of concurrency [54], which is similar to our priority consistency semantic quality attribute. If the execution of a low-priority transition, t_1, disables the enabledness of a higher-priority transition, t_2, which is in parallel with t_1, a disabling invariant for the system can be specified that executes t_1 only if t_2 is not enabled. Lastly, the notions of *persistence* and *determinacy*[5] for *program schemata* [57] are analogous to our non-cancelling and determinacy semantic quality attributes, respectively. A program schemata is a formalism to model parallel computation of programs declaratively. In general, compared to the aforementioned concepts, (i) our semantic quality attributes are defined for semantics, rather than individual models; and (ii) they are aimed at practical requirements modelling languages, instead of models of computation.

A syntactic approach, as opposed to our semantic approach, to compare the properties of BSMLs is considered by Eshuis [30], where three BSMLs, namely, statecharts by Pnueli and Shalev [86], Statemate by Harel and Naamad [43], and UML StateMachines [78], are investigated. A total of 17 syntactic constraints, including the single-input assumption [46, 47] for Statemate models, are introduced, and it is shown that that models that satisfy these constraints behave the *same* in all three semantics. Two models have the "same" behaviour if they satisfy the same *linear, stuttering-closed, separable properties* [30]: These are properties in LTL [84], do not use the "next" operator, and are *separable*[80], meaning that, "it is a boolean combination of temporal formulas each of which only refers to a sequential component of the statechart." [30] Some of the insights of this work could be perhaps useful in identifying meaningful syntactic well-formedness for achieving a semantic quality attribute.

[5] I have adopted the name of my semantic quality attribute from this work.

5.6 Summary

This chapter presented three semantic quality attributes that make it possible to compare the semantics of two BSMLs. A semantic quality attribute of a language is a desired property that is common to all models specified in that languages. Each of the semantic quality attribute specifies a desired property about the way the sequence of small steps of a big step should be formed. The set of all BSML semantics that support each of the three semantic quality attributes is characterized. These characterization are achieved systematically by specifying the combinations of the semantic options that satisfy each of the semantic quality attributes. For each specification, proof of its correctness is presented. Also, two syntactic well-formedness criteria are formally introduced that each can be used together with a set of semantic options to achieve a semantic quality attribute.

Chapter 6

Synchronization in BSMLs

"I believe that no single theory will serve all purposes." [72, p.4]

Robin Milner

This chapter introduces a formal, systematic way to adopt synchronization mechanisms for BSMLs, which traditionally have not been equipped with synchronization capability. Synchronization is only relevant for BSML semantics that subscribe to the Many concurrency semantics, in which multiple transitions can be taken within a small step. This chapter introduces 16 *synchronization types* for a synchronization mechanism that is based on two complementary roles. The 16 synchronization types arise based on the number of interactions a transition can take part in, i.e., one vs. many, and the arity of the interaction mechanisms, i.e., exclusive vs. shared. The chapter also introduces the *synchronizer* syntax that can be associated with a compound control state. A synchronizer uses a synchronization type to synchronize a set of transitions according to that synchronization type. Adopting the synchronizer syntax together with the synchronization types for BSMLs result in the class of *synchronizing big-step modelling languages*(SBSMLs). SBSMLs are useful because they facilitate the specification of the patterns of computation in which a set of transitions must be either taken together in the same small step or must not be taken at all.

The remainder of the chapter is organized as follows. Section 6.1 presents a motivating example, based on the Committee Coordination problem [18], which demonstrates the application of synchronization in modelling. Section 6.2 introduces the synchronizer syntax. Section 6.3

informally describes the semantics of the 16 synchronization types. Section 6.4 demonstrates the various applications of synchronizers and synchronization types in modelling the semantics of many existing modelling constructs.

Chapter 7 presents the formal semantics of SBSMLs, together with proofs of how the semantics of existing modelling constructs are implemented in SBSMLs.

6.1 A Motivating Example

The model in Figure 6.1 is an SBSML model that is inspired by the English description of the "Committee Coordination" problem [18]. The committee coordination problem asks for a scheme to schedule the meetings of different committees of a university. The members of each committee are faculty members, each of them can be a member of more than one committee. A committee can convene when all of its members are ready to meet. In the model in Figure 6.1, I have extended this problem to include that each faculty member is either carrying out research, teaching, or attending exactly one meeting. The model in Figure 6.1 is a *specification* of the scheduling problem for the case of four faculty members, modelled by F_i ($1 \leq i \leq 4$), and three committees, modelled by C_i ($1 \leq i \leq 3$). For example, the members of committee C_1 are F_1, F_2, and F_3. A meeting of C_1 convenes when transitions t_2, t_{12}, t_{22}, and t_{41} are executed together. The model in Figure 6.1 is specified in a synchronizing big-step modelling language (SBSML). The transitions of the model are annotated with the labels of the synchronizers, using the labels in a normal or a complementary role (shown by over bars). For example, transition t_2 uses label f_1 in a normal role, whereas transition t_{41} uses labels $\overline{f_1}$, $\overline{f_2}$, and $\overline{f_3}$ in complementary roles. A transition might use a label in a normal role and a different label in a complementary role; e.g., transition t_{52}. The *And* control state in the model is annotated with three *synchronizers*. A synchronizer, such as `UPEE`(f_1, f_2, f_3, f_4), consists of a *synchronization type*, e.g., `UPEE`, and a set of labels, e.g., $\{f_1, f_2, f_3, f_4\}$. As an example, the labels of synchronizer `UPEE`(f_1, f_2, f_3, f_4) are used by transitions t_2, t_{12}, t_{22}, and t_{41}. These transitions, according to the semantics of `UPEE`(f_1, f_2, f_3, f_4), are either executed together in a synchronized manner, or none of them can be executed.[1]

In the model in Figure 6.1, each faculty member is initially busy with research and writing (e.g., F_1 is initially in R_1 doing research, represented by transition t_1, generating event, *writing*$_1$),

[1]Unless the synchronization requirements of a transition are satisfied through synchronization with a different set of transitions, which is not the case in this example.

but may have to give lectures (e.g., F_1 may have to give a lecture when the guard condition of t_1, $class_1$, becomes true), or may attend a committee meeting (e.g., F_1 may attend a meeting of C_1 or C_2, by taking transition t_2 or transition t_4, respectively). Sometimes, a faculty member leaves a committee meeting before the meeting normally ends to give a lecture, in which case the committee meeting ends abruptly. For example, transition t_7 specifies that F_1 needs to leave a meeting of C_1, when condition $class_1$ is satisfied, requiring transitions t_{44}, t_{13}, and t_{23} to be executed with it according to the synchronizers $\texttt{UUEE}(leave_1, leave_2, leave_3)$ and $\texttt{UUES}(end_1, end_2, end_3)$. The set of transitions $\{t_7, t_{13}, t_{23}, t_{44}\}$ constitute a small step, and are synchronized according to two synchronizers. Committee C_3 is designed to give a higher priority to ending a meeting when a faculty member needs to give a lecture than to continuing the meeting. If the Scope Parent priority semantics is chosen, t_{52} would have a higher priority than t_{50}, thus the small step that includes t_{52} would have precedence over the small step that includes t_{50}.

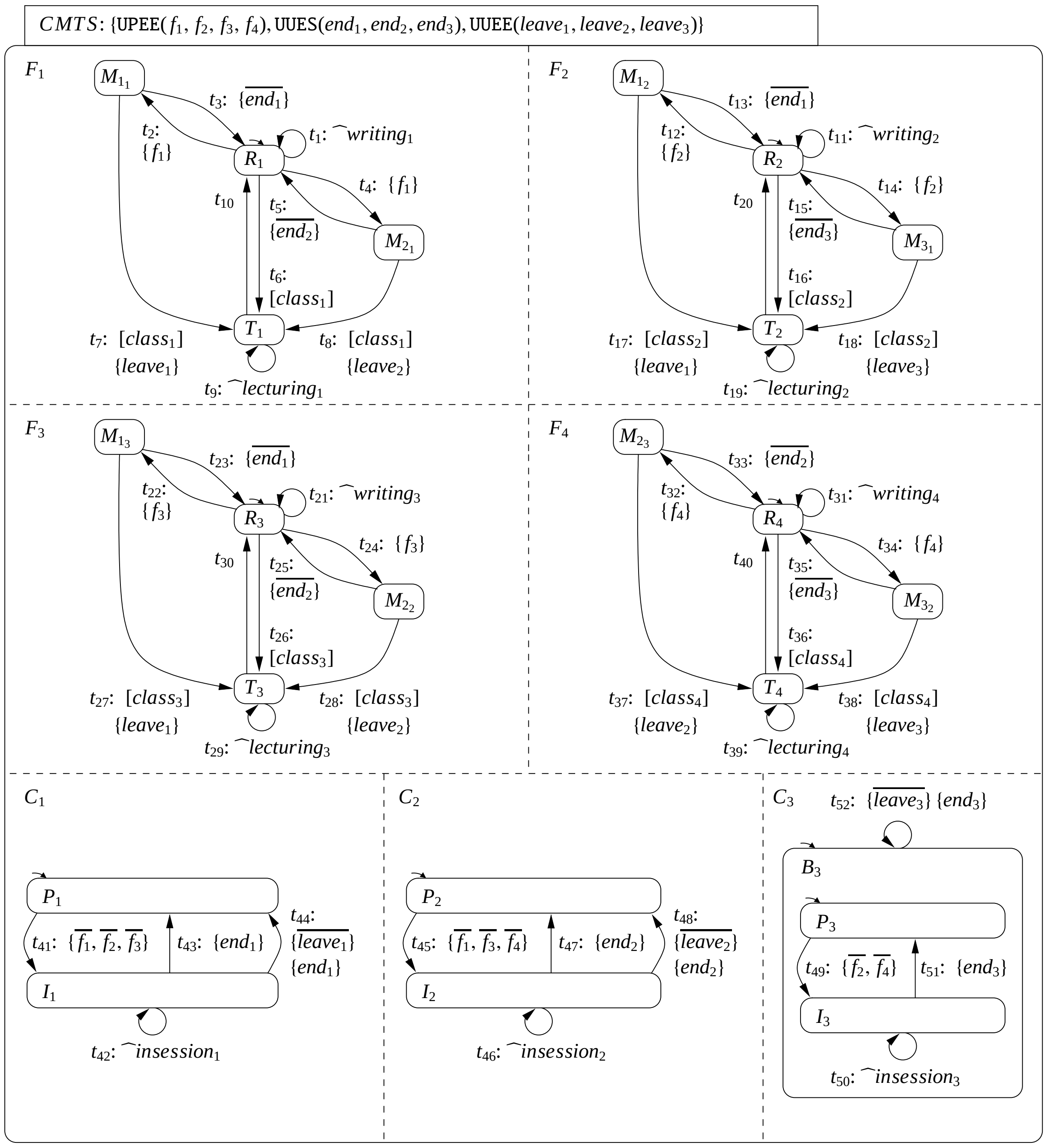
6.2 Synchronization Syntax

Similar to a BSML model, an SBSML model consists of a hierarchy tree of control states and a set of transitions between these control states. Additionally, the normal-form syntax of SBSMLs include syntax for synchronization. This section describes only the synchronization syntax of SBSMLs. The syntax of BSMLs can be found in Section 2.1. The formal BNF of SBSMLs is presented in Section 7.1.

The model in Figure 6.2 is used to describe the syntactic and semantic concepts of SBSMLs. The model shows an SBSML model that characterizes a set of simple synchronized ice skating programs. Initially, all skaters are together, represented by the *Basic* control state *Together*. During the program the skaters can split into three groups to perform the *intersection* maneuver(s), represented by the *And* control state *Intersection*.[2] To avoid a clash, at each point of time, only one of the three groups can initiate an intersection maneuver. The skaters can merge back to a group, but the program can only end, by going to the *End* control state, when the skaters are split. The environmental input events *Line* and *Circle* specify a line and circle maneuver in a program, respectively.[3] The environmental input events *Split* and *Merge* specify that the skaters split to three groups to perform intersection maneuver(s) and that the three groups merge back

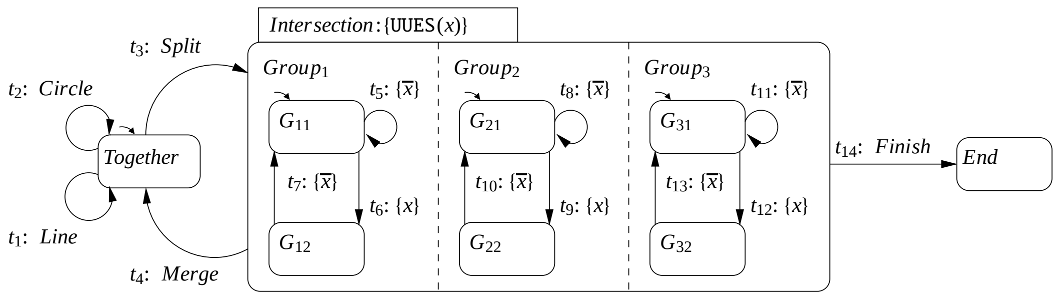
[2]In the intersection maneuver, the skaters in one group skate between the skaters of another.

[3]In the line and circle maneuvers, the skaters of a team create a formation in a line or circle pattern, respectively.



$CMTS$: {UPEE(f_1, f_2, f_3, f_4), UUES(end_1, end_2, end_3), UUEE($leave_1, leave_2, leave_3$)}
F_1
M_{1_1}
R_1
M_{2_1}
T_1
t_1: $\frown writing_1$
t_2: $\{f_1\}$
t_3: $\{\overline{end_1}\}$
t_4: $\{f_1\}$
t_5: $\{\overline{end_2}\}$
t_6: $[class_1]$
t_7: $[class_1]$ $\{leave_1\}$
t_8: $[class_1]$ $\{leave_2\}$
t_9: $\frown lecturing_1$
t_{10}
F_2
M_{1_2}
R_2
M_{3_1}
T_2
t_{11}: $\frown writing_2$
t_{12}: $\{f_2\}$
t_{13}: $\{\overline{end_1}\}$
t_{14}: $\{f_2\}$
t_{15}: $\{\overline{end_3}\}$
t_{16}: $[class_2]$
t_{17}: $[class_2]$ $\{leave_1\}$
t_{18}: $[class_2]$ $\{leave_3\}$
t_{19}: $\frown lecturing_2$
t_{20}
F_3
M_{1_3}
R_3
M_{2_2}
T_3
t_{21}: $\frown writing_3$
t_{22}: $\{f_3\}$
t_{23}: $\{\overline{end_1}\}$
t_{24}: $\{f_3\}$
t_{25}: $\{\overline{end_2}\}$
t_{26}: $[class_3]$
t_{27}: $[class_3]$ $\{leave_1\}$
t_{28}: $[class_3]$ $\{leave_2\}$
t_{29}: $\frown lecturing_3$
t_{30}
F_4
M_{2_3}
R_4
M_{3_2}
T_4
t_{31}: $\frown writing_4$
t_{32}: $\{f_4\}$
t_{33}: $\{\overline{end_2}\}$
t_{34}: $\{f_4\}$
t_{35}: $\{\overline{end_3}\}$
t_{36}: $[class_4]$
t_{37}: $[class_4]$ $\{leave_2\}$
t_{38}: $[class_4]$ $\{leave_3\}$
t_{39}: $\frown lecturing_4$
t_{40}
C_1
P_1
I_1
t_{41}: $\{\overline{f_1}, \overline{f_2}, \overline{f_3}\}$
t_{42}: $\frown insession_1$
t_{43}: $\{end_1\}$
t_{44}: $\{\overline{leave_1}\}$ $\{end_1\}$
C_2
P_2
I_2
t_{45}: $\{\overline{f_1}, \overline{f_3}, \overline{f_4}\}$
t_{46}: $\frown insession_2$
t_{47}: $\{end_2\}$
t_{48}: $\{\overline{leave_2}\}$ $\{end_2\}$
C_3
t_{52}: $\{\overline{leave_3}\}$ $\{end_3\}$
B_3
P_3
I_3
t_{49}: $\{\overline{f_2}, \overline{f_4}\}$
t_{51}: $\{end_3\}$
t_{50}: $\frown insession_3$

Figure 6.1: Modelling faculty members and their responsibilities, using synchronization.



$Intersection : \{\texttt{UUES}(x)\}$
t_3: Split
t_2: Circle
t_1: Line
t_4: Merge
Together
$Group_1$
$Group_2$
$Group_3$
t_5: $\{\overline{x}\}$
t_8: $\{\overline{x}\}$
t_{11}: $\{\overline{x}\}$
G_{11}
G_{21}
G_{31}
t_7: $\{\overline{x}\}$
t_{10}: $\{\overline{x}\}$
t_{13}: $\{\overline{x}\}$
t_6: $\{x\}$
t_9: $\{x\}$
t_{12}: $\{x\}$
G_{12}
G_{22}
G_{32}
t_{14}: Finish
End

Figure 6.2: A model for a set of synchronized ice skating programs.

into a single group, respectively. The environmental input event *Finish* specifies the end of an ice skating program.

A compound control state of an SBSML (both *And* and *Or* control states) can have a set of *synchronizers*, which are graphically positioned at the top of the control state. For example, the control state *Intersection* in the model in Figure 6.2 has one synchronizer: $\texttt{UUES}(x)$. Each synchronizer $Y(L)$ has: (i) a *synchronization type*, Y; and (ii) a *label set*, L, surrounded by parentheses, instead of curly brackets. There are 16 synchronization types, each of which is a string of four letters, where a letter represents an aspect of the semantics of the synchronization type. The label set of a synchronizer *declares* a unique set of *identifiers* (labels) that are *used* by transitions that are to be synchronized by the synchronizer. In the model in Figure 6.2, synchronizer $\texttt{UUES}(x)$ has synchronization type $\texttt{UUES}$, and declares the identifier x in its label set $\{x\}$.

A transition in an SBSML model can have: (i) a set of *role sets*, and (ii) a set of *co-role sets*. Each role set is a set of *labels*, each of which is an identifier. Each co-role set is a set of *co-labels*, each of which is an over-lined identifier. For example, in the model in Figure 6.2, the set of role sets of t_6 is $\{\{x\}\}$ and the set of co-role set of t_8 is $\{\{\overline{x}\}\}$. The well-formedness criteria of SBSMLs, summarized at the end of Section 6.3.1, require that all of the labels (co-labels) of a role set (co-role set) are associated with the identifiers of the same synchronizer. When the set of role sets or the set of co-role sets of a transition is a singleton, its curly brackets are dropped. A role set is called *uni-role* if it is a singleton and *poly-role* otherwise. Similarly, a co-role set is called *uni-co–role* or *poly-co–role*. For example, the only role set of t_6 is a uni-role. Transitions t_6, t_8, and t_{11} can execute together because synchronizer $\texttt{UUES}(x)$ match their role and co-role sets.

Table 6.1: Synchronization types and their parameters, when considered for synchronizer $Y(L)$.

Index	Parameter Purpose	Values for Synchronizer $Y(L)$
1	How an identifier can be used in the role sets of transitions	U: The identifiers in L can be used only in uni-roles
		P: The identifiers in L can be used in poly-roles
2	How an identifier can be used in the co-role sets of transitions	U: The identifiers in L can be used only in uni-co-roles
		P: The identifiers in L can be used in poly-co-roles
3	How many instances of a label can appear in the role sets of transitions in a small step	E: One, exclusively
		S: Many, in a shared manner
4	How many instances of a co-label can appear in the co-role sets of transitions in a small step	E: One, exclusively
		S: Many, in a shared manner

6.3 Synchronization Types

A synchronization type consists of a sequence of four letters, each of which is a value for one of the four parameters that together create the set of 16 synchronization types. Table 6.1 describes the role of each parameter and its corresponding two possible values, when considered for an arbitrary synchronizer $Y(L)$. The "Index" column relates the position of a letter in the synchronization type with its corresponding parameter.

Next, the semantics of synchronization types is described in detail by specifying their role in determining the potential small steps of an SBSML model. The role of structural semantic aspects, however, need not be considered. First, the semantic sub-aspects of Concurrency and Consistency, as will be shown in Section 6.4, are not relevant for SBSMLs because they are forms of synchronization themselves. And second, for the sake of clarity, in this chapter, only the No Priority semantics is considered, but Chapter 7 considers the roles of the other two hierarchical semantic options. Thus, in this chapter, I will talk about the synchronization of "enabled transitions", which are transitions that could be taken in a small step if only the role of enabledness semantic aspects are considered, instead of talking about "executable transitions", which consider the role of all semantic aspects, including the structural semantic aspects. The formal semantics in Chapter 7 describes the semantics of SBSMLs in a uniform way, considering all the semantic aspects as a whole.

From a set of enabled transitions, T, determined by the enabledness semantic aspects of a BSML, a potential small step, X, $X \subseteq T$, must not only satisfy the constraints of all of the synchronizers that control transitions in T.

In a synchronizer $Y(L)$, the first two letters of its synchronization type, Y, indicate how the identifiers in L can be used in transitions within the scope of $Y(L)$. A `U` in the first position means that for all identifiers $l \in L$, all transitions in X that have l in their role sets, l must belong to a uni-role (i.e., a singleton role set). A `U` in the second position means that for all identifiers $l \in L$, all transitions in X that have $\overline{l}$ in their co-role sets, $\overline{l}$ must belong to a uni-co-role set. A `P` in the first or second position of the synchronization type places no such constraints but only has a different meaning from a `U` if there are multiple identifers in L. The constraints of the first two indices in the synchronization type can be checked syntactically by well-formedness constraints, described later in this section.

As in some process algebras, such as CCS [72], a label in a role set, e.g., m, is *matched* with a co-label in a co-role set that has the same identifier, i.e., $\overline{m}$. For every transition, t, included in X, the labels in all its role sets and the co-labels in all its co-role sets *must* participate in a match: For every label, m, in a role set, there must be a matching co-label, $\overline{m}$, from another transition included in X, and vice-versa for every co-label, $\overline{n}$, in its co-role sets. The third and fourth indices of the synchronization type indicate how many transitions can participate in this match: Effectively, how many labels, m, can match an $\overline{m}$ and vice-versa, amongst the role sets and co-role sets of the transitions in X. For a synchronizer with label set L and a synchronization type whose third letter is `E`, i.e., one of the `**E*` synchronization types, every identifier, $l \in L$, can appear at most once in the role sets of all transitions in X. For synchronization types `***E`, every over-lined identifier of L, $\overline{l}$, can appear at most once in the co-role sets of all transitions in X. For synchronization types `**S*` (and `***S`), an identifier $l \in L$ can appear multiple times in the role sets (and co-role sets) of the transitions in X.

In summary, after collecting the role sets and co-role sets of all the transitions within X that use identifiers of L, we have a set of role sets and a set of co-role sets:

$$R = \{R_1, R_2, \cdots\} \text{ and}$$
$$C = \{C_1, C_2, \cdots\}.$$

These sets should satisfy all of the following conditions:

- Every label $r \in R_i$, where $R_i \in R$, must have a corresponding co-label $c \in C_j$, such that

Table 6.2: Examples of synchronizing transitions.

Synchronizer	Synchronizing Transitions (Non-Exhaustive)
UUEE(m)	t_1: $\{m\}$, t_2: $\{\overline{m}\}$
UUSE(m)	t_1: $\{m\}$, t_2: $\{m\}$, t_3: $\{\overline{m}\}$
UUSS(m)	t_1: $\{m\}$, t_2: $\{m\}$, t_3: $\{\overline{m}\}$, t_4: $\{\overline{m}\}$
UPEE(m,n)	t_1: $\{m\}$, t_2: $\{n\}$, t_3: $\{\overline{m},\overline{n}\}\}$
UPSE(m,n)	t_1: $\{m\}$, t_2: $\{m\}$, t_3: $\{n\}$, t_4: $\{n\}$, t_5: $\{\overline{m},\overline{n}\}$
UPES(m,n)	t_1: $\{m\}$, t_2: $\{m\}$, t_3: $\{\overline{m},\overline{n}\}$, t_4: $\{\overline{m},\overline{n}\}$
UPSS(m,n)	t_1: $\{m\}$, t_2: $\{m\}$, t_3: $\{n\}$, t_4: $\{n\}$, t_5: $\{\overline{m},\overline{n}\}$, t_6: $\{\overline{m},\overline{n}\}$
PPEE(m,n,p,q)	t_1: $\{m,n\}$, t_2: $\{p,q\}$, t_3: $\{\overline{m},\overline{p}\}$, t_4: $\{\overline{n},\overline{q}\}$
PPSE(m,n,p,q)	t_1: $\{m,n,p,q\}$, t_2: $\{m,n\}$, t_3: $\{p,q\}$, t_4: $\{\overline{m},\overline{p}\}$, t_5: $\{\overline{n},\overline{q}\}$
PPSS(m,n,p,q)	t_1: $\{m,n,p,q\}$, t_2: $\{m,n\}$, t_3: $\{p,q\}$, t_4: $\{\overline{m},\overline{n},\overline{p},\overline{q}\}$, t_5: $\{\overline{m},\overline{p}\}$, t_6: $\{\overline{n},\overline{q}\}$

$C_j \in C$, and $\overline{r} = c$; and vice versa for every co-label;

- If the synchronization type is `**E*`, for every co-label $c \in C_j$, where $C_j \in C$, there is exactly one corresponding label $r \in R_i$, such that $R_i \in R$, and $c = \overline{r}$;

- If the synchronization type is `***E`, for every label $r \in R_i$, where $R_i \in R$, there is exactly one corresponding label $c \in C_j$, such that $C_j \in C$, and $\overline{r} = c$; and

- Finally, the set X must be maximal, i.e., it is not possible to add one or more transition in T and to satisfy the above constraints of the synchronization type.

Table 6.2 shows examples of *synchronizing transitions* according to 10 synchronizers of distinct types. The transitions in each row are enabled transitions of a model. Intuitively, the first two letters of a synchronization type specify the number of interactions, i.e., the number of matchings over distinct identifiers, that a transition can take part in, i.e., *biparty* vs. *multiparty* interaction. The last two letters of a synchronization type specify the arity of the interaction mechanism, i.e., *exclusive* vs. *shared* interaction.

In the model in Figure 6.2, when the model resides in G_{11},G_{21}, and G_{31}, the set of transitions $\{t_5, t_9, t_{11}\}$ is a potential small step of the model, which satisfies the constraints of synchronizer UUES(x): (1) only uni-roles use x; (2) only uni-co-roles use x; (3) only t_9 has a role set including x; and (4) both t_5 and t_{11} have co-role sets including $\overline{x}$. The other two potential small steps are: $\{t_6, t_8, t_{11}\}$ and $\{t_5, t_8, t_{12}\}$. The model neither allows two groups to initiate an intersection maneuver simultaneously, nor a group to initiate two intersection maneuvers consecutively.

Each pair of synchronization types `UPEE` and `PUEE`, `UUSE` and `UUES`, `UPSE` and `PUES`, `PUSE` and `UPES`, `PPSE` and `PPES`, and `UPSS` and `PUSS` are symmetric. A synchronizer with one of these types can be replaced with a synchronizer with the same label set but the symmetric type, with the role sets and co-role sets of the transitions within its scope swapped.

If a model has more than one synchronizer, the constraints of their corresponding synchronization types should be considered together; i.e., the set X above should satisfy the synchronization requirements of all of the synchronizers together.

6.3.1 Well-formedness Criteria for SBSML Models

Lastly, in the semantics described above, some well-formedness conditions are assumed. An SBSML model is *well-formed* if all of the following seven conditions hold,

i Each label uniquely belongs to the label set of exactly one synchronizer.

ii For each label, l, if there is at least one transition with a poly-role that includes l, then the synchronization type of its corresponding synchronizer should be a synchronization type whose first letter is `P` (i.e., `P***`), otherwise it must be one of the `U***` synchronization types. Similarly, the second letter of a synchronization type is specified based on the characteristics of the co-role sets of transitions.

iii No two synchronizers of a control state have the same synchronization type.

iv Two labels that are associated with the same synchronization type do not belong to two different role sets or two different co-role sets of the same transition.

v For each label, l, of a synchronizer, y, and each transition, t, l is associated with at most one of the role sets or co-role sets of t.

vi A synchronizer, y, is associated with the least common ancestor of the source and destination control states of the transitions that use the labels of y in their role sets or co-role sets.

vii A synchronizer, y, of a control state, s, cannot be split into two synchronizers, y_1 and y_2, such that y_1 is assigned to s but y_2 is assigned to a descendant of s.

Hereafter, by an SBSML model, I mean a well-formed SBSML model.

6.4 Applications

This section, through examples, describes how the semantics of different modelling constructs and different semantic concepts can be modelled succinctly using synchronization in SBSMLs. Section 6.4.1 describes how the semantics of *multi-source, multi-destination transitions* [41, 86] can be described using regular transitions and synchronizers. Section 6.4.2 describes how some of the semantic options of the concurrency and consistency semantic aspect can be described by other semantic options and synchronizers, thereby allowing multiple BSML semantics to exist in different components of a model. Section 6.4.3 shows how the semantics of the Present In Same event lifeline semantics, whose semantics was not included in the semantic formalization of Chapter 4, can be modelled by using synchronizers. Section 6.4.4 describes how the semantics of some of the *composition operators* in template semantics [75] can be described using synchronizers in SBSMLs. Lastly, Section 6.4.5 shows how the essence of some of the *workflow patterns* [96] can be captured succinctly using synchronization in SBSMLs. The formal treatment of the discussions in this section are presented in Section 7.4.

6.4.1 Modelling Multi-source, Multi-destination Transitions

Multi-source, multi-destination transitions embody a form of concurrent, synchronized execution: When a multi-source, multi-destination transition is executed, it exits all of its source control states and enters all of its destination control states [41, 86]. A multi-source, multi-destination transition of a model can be replaced with a set of regular transitions that are always taken together by synchronizing via a synchronizer of type `UPEE`. As an example, the SBSML model in Figure 6.3(b) is equivalent to the model in Figure 6.3(a), which has two multi-source, multi-destination transitions x and y. Transition x is replaced by transitions x_1, x_2, and x_3, and transition y is replaced by transitions y_1, y_2, and y_3. From the set of regular transitions that model a multi-source, multi-destination transition, e.g., x, one of the transitions, e.g., x_1, adopts the guard and trigger conditions, the actions, and the possible role sets and co-role sets of x, along with a new co-role set with new co-labels each representing one of the other transitions. The other transitions each has one singleton role set, to match the new co-role set of the first transition. The number of control states in the source and destination of a multi-source, multi-destination transition need not be the same, in which case new dummy control states are introduced to make the number of source control states and destination control states equal. For example, in the model

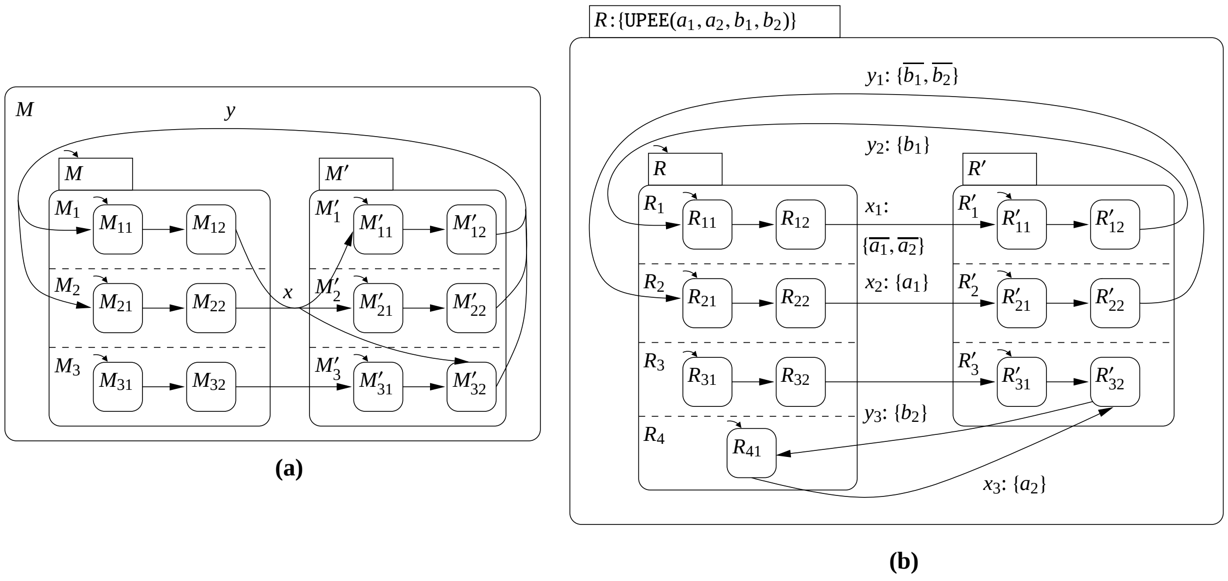


Figure 6.3: Modelling multi-source, multi-destination transitions using regular transitions.

in Figure 6.3(b), control state R_4 is introduced to accommodate for the source control state of x_3 and the destination control state of y_3. This new control state does not change the behaviour of the model compared to the original model because R_{41} is the only control state of R_4.

Henceforth, for convenience, I use the syntax of multi-source, multi-destination transition as part of the normal-form syntax of SBSMLs.

6.4.2 Modelling BSML Semantic Options

In the presence of synchronization, the BSML semantic options for the Concurrency, Small-Step Consistency and Preemption structural semantic sub-aspects are not needed. Next, through examples, it is shown how a more inclusive semantic option of each of these semantic sub-aspects can be used to implement a less inclusive one. Thus a single SBSML can include a combination of the semantic options of these semantic aspects.

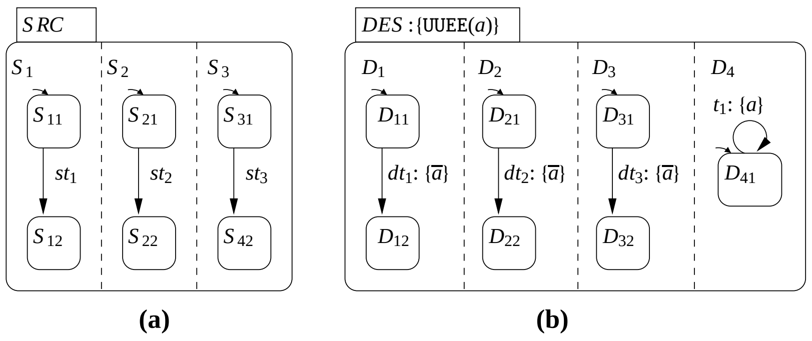


Figure 6.4: Deriving the Single semantics using the Many concurrency semantics.

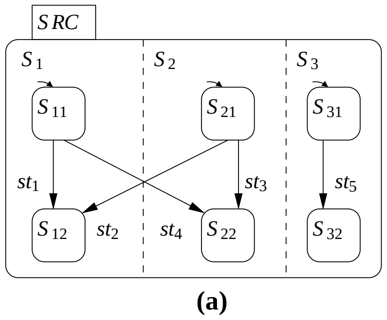
Concurrency

Using the Many semantic option together with a synchronizer of synchronization type UUEE, an *And* control state can be constrained to execute at most one of its transitions at each small step: Every transition within the *And* control state is modified to synchronize with a new self transition, t_1: {$\overline{a}$}.[4] As an example, the model in Figure 6.4(a) can take all of its three transitions together in one small step, while the model in Figure 6.4(b) can take only one of the transitions in each small step. Thus, using synchronization, the Many semantic option covers the Single semantic option.

Small-Step Consistency

Using the Source/Destination Orthogonal semantic option together with synchronizers of type PUEE, the Arena Orthogonal semantic option can be enforced. The model in Figure 6.5(b), which is specified in the Source/ Destination Orthogonal semantics, has an equivalent behaviour to the model in Figure 6.5(a), which is specified in the Arena Orthogonal small-step consistency semantics. If transition st_2 is included in a small step, then according to the Arena Orthogonal semantics, neither st_4 nor st_5 should be included in the small step. In the model in Figure 6.5(b), this is achieved by using a synchronizer of type PUEE that does not allow dt_2 to be taken together with dt_4 or dt_5. Similarly, transition dt_4 cannot be included in the same small step that dt_2 or dt_5 belong to. It is possible to have a hybrid semantics. For example, changing the role set of transitions dt_2 and dt_4 both to {a_2} and {a_4}, respectively, means that each of dt_2

[4]This transformation is analogues to how *asynchrony* can be derived from *synchrony* in SCCS [71, 72].



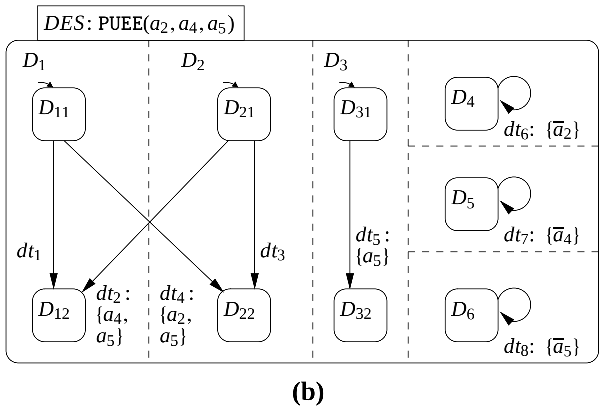


Figure 6.5: Deriving the Arena Orthogonal semantics using the Source/Destination Orthogonal semantics.

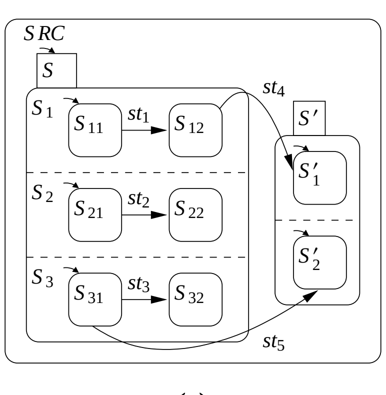
and dt_4 can be taken with dt_5 in the same small step, but dt_2 and dt_4 cannot be taken together in the same small step. Such a hybrid small-step consistency semantics disallows transitions that graphically cross each other to be included in the same small step.

Preemption

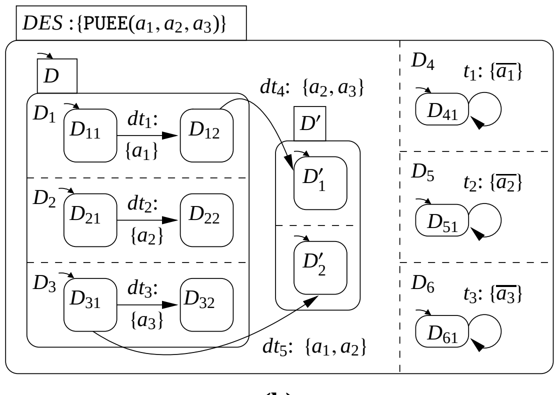
Using the Non-Preemptive semantics together with synchronizers of type PUEE, similar to the previous section, a Preemptive semantics can be derived. For example, in the model in Figure 6.6(a), transition st_4, which is an interrupt transition, can be taken together with transitions st_2 and st_3 by the Non-Preemptive semantics. Similarly, transition st_5 can be taken together with transitions st_1 and st_2. These transitions, however, cannot be taken together if the Preemptive semantics is chosen. The model in Figure 6.6(b), which is specified in the Non-Preemptive semantics, has an equivalent behaviour to the model in Figure 6.6(a), when it is specified in the Preemptive semantics. In the model in Figure 6.6(b), for example, transitions dt_4 and dt_2 cannot be taken together because only one of them can synchronize with t_2.

6.4.3 Modelling the Present In Same Event Lifeline Semantics

The Present In Same event lifeline semantics, a generated event in a small step can trigger transitions only in the same small step. This semantics can be modelled using synchronizers. Some



(a)



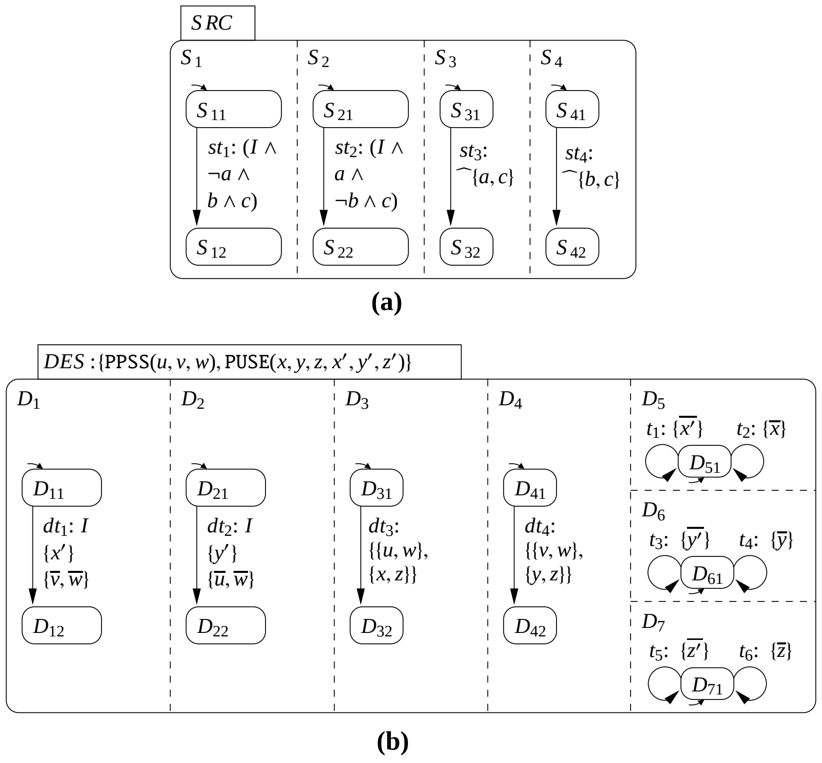
(b)

Figure 6.6: Deriving PREEMPTIVE semantics using a NON-PREEMPTIVE semantics.

BSMLs, such as μ-Charts [82], support a notion of a *signal* that when generated in a small step of a model can be sensed as present by all of the transitions of the model in the same small step, which is the same as the PRESENT IN SAME event lifeline semantics if signals are considered as events.

The PRESENT IN SAME event lifeline semantics can be modelled by using synchronizers of type PPSS. A set of signals generated by a transition corresponds to a poly-role. The conjunction of signals in the trigger of a transition corresponds to a poly-co-role. Since more than one instance of an event can be generated in the same small step, the third letter of the synchronization type is S. Since a generated event can trigger more than one transitions, the fourth letter of the synchronization type is S (shared). To model the negation of a signal, a synchronizer of type PUSE and two labels can be used to disallow both a signal to be generated by a transition and its negation to be the trigger of a transition, in the same small step.

As an example, in the model in Figure 6.7(a), when the model is initialized and input signal I is received from the environment, either of the potential small steps $\{st_1, st_4\}$ and $\{st_2, st_3\}$ can be taken, non-deterministically in the PRESENT IN SAME semantics. The SBSML model in Figure 6.7(b) has the same behaviour as the one in Figure 6.7(a). Labels u, v, and w correspond to events/signals a, b, and c, respectively. Input event I is maintained in the model in Figure 6.7(b). Labels x, y, z, together with labels x', y', z', ensure that each of the signals a, b, and c can be exclusively either generated or its negation can trigger a transition, respectively. For example, transitions dt_1 and dt_3 cannot be taken together because t_1 and t_2 cannot be taken together, but



SRC
S_1
S_2
S_3
S_4
S_{11}
st_1: $(I \wedge \neg a \wedge b \wedge c)$
S_{12}
S_{21}
st_2: $(I \wedge a \wedge \neg b \wedge c)$
S_{22}
S_{31}
st_3: $\frown\{a,c\}$
S_{32}
S_{41}
st_4: $\frown\{b,c\}$
S_{42}
(a)
DES : {PPSS(u,v,w), PUSE(x,y,z,x',y',z')}
D_1
D_2
D_3
D_4
D_5
D_{11}
dt_1: I $\{x'\}$ $\{\overline{v},\overline{w}\}$
D_{12}
D_{21}
dt_2: I $\{y'\}$ $\{\overline{u},\overline{w}\}$
D_{22}
D_{31}
dt_3: $\{\{u,w\},\{x,z\}\}$
D_{32}
D_{41}
dt_4: $\{\{v,w\},\{y,z\}\}$
D_{42}
t_1: $\{\overline{x'}\}$
t_2: $\{\overline{x}\}$
D_{51}
D_6
t_3: $\{\overline{y'}\}$
t_4: $\{\overline{y}\}$
D_{61}
D_7
t_5: $\{\overline{z'}\}$
t_6: $\{\overline{z}\}$
D_{71}
(b)

Figure 6.7: Modelling the PRESENT IN SAME event lifeline semantics.

dt_1 needs to synchronize with t_1 over label x' and dt_3 needs to synchronize with t_2 over label x.

6.4.4 Modelling Composition Operators

In template semantics [75], a set of composition operators are introduced, each of which represents a useful execution pattern in modelling. This section describes how the behaviour of the *rendezvous*, *environmental synchronization*, and *interleaving* composition operators can be modelled using synchronizers. For each of the remaining composition operators in template semantics, there is a similar *workflow pattern* [96], whose semantics is considered in the next section.

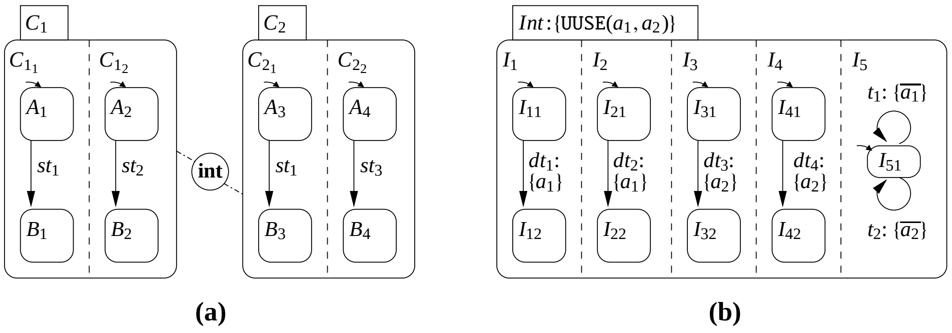


Figure 6.8: Modelling the interleaving composition operator in SBSMLs.

Interleaving:

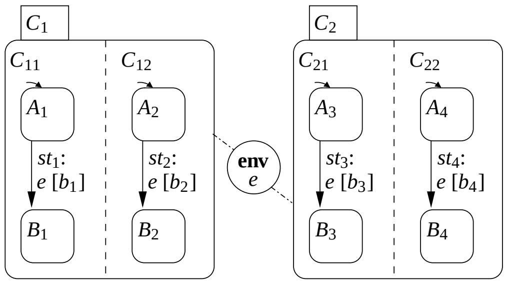
Composing two components via the interleaving composition operator has the effect that in each small step only one of the concurrent components can contribute transitions to the small step. As an example, in the model in Figure 6.8(a), in each small step, either transitions of C_1 or those of C_2 should be executed. The SBSML model in Figure 6.8(b) uses a synchronizer of synchronization type `UUSE` and an additional controlling component to enforce the interleaving semantics of the model in Figure 6.8(a), by executing either t_1 or t_2 in a small step, but not both.

Rendezvous

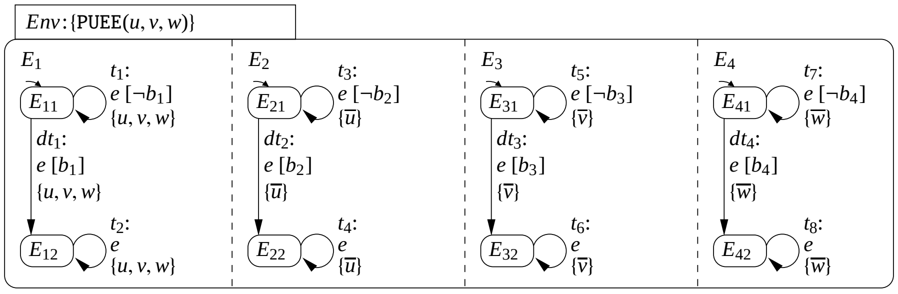
The rendezvous binary composition operator, analogous to the CCS Composition operator [72], requires one of the transitions of one of its operands to generate a synchronization event and one of the transitions of the other operand to consume it as a trigger, in the same small step. If a pair of synchronizing transitions are not enabled, the non-synchronizing transitions can be executed in an interleaving manner. Such a semantics of the rendezvous composition operator, in the context of CCS-like models, can be modelled using synchronizers of type `UUEE`.

Environmental Synchronization

The environmental synchronization composition operator, analogous to the CSP Concurrency operator [48], requires its two operands to synchronize over transitions that are triggered with the same *synchronization event* received from the environment. At each snapshot, it is possible



(a) Environmental synchronization composition operator.


Env: {PUEE(u, v, w)}
E_1
E_{11}
t_1: e [$\neg b_1$] $\{u, v, w\}$
dt_1: e [b_1] $\{u, v, w\}$
E_{12}
t_2: e $\{u, v, w\}$
E_2
E_{21}
t_3: e [$\neg b_2$] $\{\overline{u}\}$
dt_2: e [b_2] $\{\overline{u}\}$
E_{22}
t_4: e $\{\overline{u}\}$
E_3
E_{31}
t_5: e [$\neg b_3$] $\{\overline{v}\}$
dt_3: e [b_3] $\{\overline{v}\}$
E_{32}
t_6: e $\{\overline{v}\}$
E_4
E_{41}
t_7: e [$\neg b_4$] $\{\overline{w}\}$
dt_4: e [b_4] $\{\overline{w}\}$
E_{42}
t_8: e $\{\overline{w}\}$

(b) Equivalent SBSML model for the model in Fig 6.9(a).

Figure 6.9: Modelling the environment synchronization composition operator in SBSMLs.

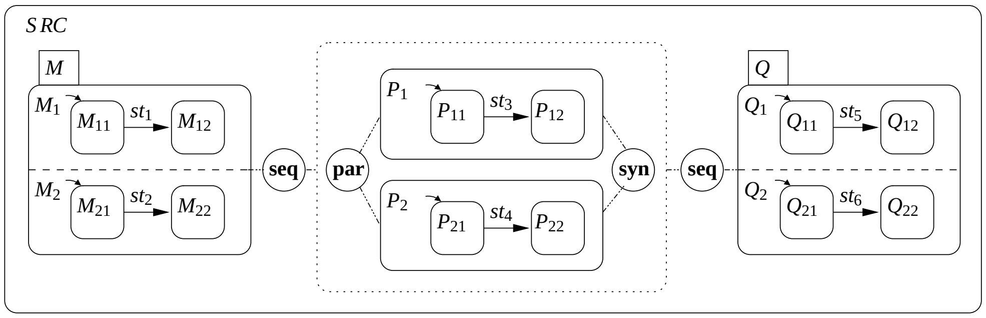
that no, one, or more than one transition in each operand is enabled and triggered with the synchronization event. When all concurrent parts of the operands have enabled transitions that are triggered with the synchronization event, a synchronizer of type PUEE can be used to execute all of them together in the same small step. Otherwise, when there is an arbitrary number of enabled transitions that are triggered with the synchronization event, it should be ensured that a maximal set of such transitions are taken together in the same small step.

Figure 6.9(a) uses the environmental synchronization composition operator over event e to coordinate the execution of C_1 and C_2. Each of the transitions in the model has a guard condition on a boolean variable that is assigned a value by the environment. Figure 6.9(b) is an SBSML model that has the same behaviour as the model in Figure 6.9(a). Each control state of the model in Figure 6.9(b) has a self transition to accommodate for the execution of synchronization transitions when not all of the synchronizing transitions are ready to execute, either because the guard condition of a synchronizing transition is false, or because it has already been executed. This transformation, however, does not consider the possibility for a synchronizing transition to be taken together with other synchronizing transitions or a non-synchronizing transition. Section 7.4.4, on page 257, presents a transformation scheme that takes into account these possibilities.

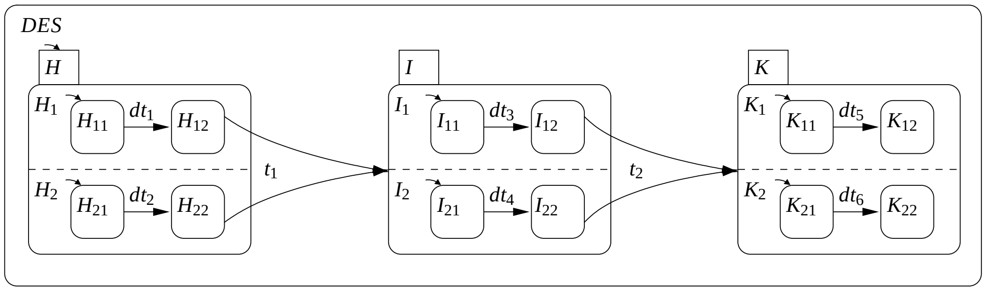
6.4.5 Modelling Workflow Patterns

Workflow patterns [96] are used in business process modelling, as well as Web services choreography languages, such as BPEL [77]. A workflow pattern distills a recurring pattern of execution that is useful in modelling a wide range of systems. There are five *basic workflow patterns* [96]: (i) `sequence`, which executes two activities sequentially: Once the first activity finishes, the second activity starts; (ii) `parallel split`, which starts executing multiple activities in parallel; (iii) `synchronization`, which merges multiple parallel activities into a single activity; (iv) `exclusive choice`, which non-deterministically chooses to execute one activity from a set of possible activities; and (v) `simple merge`, which requires exactly one of the alternative activities to finish before a next activity starts. The semantics of many of the workflow patterns inherently deal with a notion of synchronization. For example, a task should start only after an earlier task has finished; i.e., the `sequence` pattern.

Next, through examples, it is shown how simple workflow patterns can be modelled using the expressiveness of the synchronizers. Workflow patterns are in general considered as abstract modelling constructs that are manifested in different languages differently. I use a BSML-like syntax to specify workflow models. The following discussions about the semantics of workflow patterns in the context of BSMLs are also relevant in a different setting, because the proposed translations focus on the "control flow" aspects of these patterns that remain more or less the same in different frameworks. I use multi-source, multi-destination transitions in my transformations. The model in Figure 6.10(a) uses special syntax to represent the `sequence`, `parallel split`, and `synchronization` workflow patterns, denoted by `seq`, `par`, `syn` in a circle, respectively. Intuitively, the model in Figure 6.10(a) requires M to be executed followed by the parallel executions of P_1 and P_2, their synchronization once they are done, and lastly, followed by the execution of Q. I consider entering a control state that has no outgoing transition as the end of the activity that corresponds to that control state; a parallel activity ends when all of its constituent parts end. For example, in Figure 6.10(a), the end of M is when both M_1 and M_2 have been entered. The SBSML model in Figure 6.10(b) is equivalent to the intended behaviour of the model in Figure 6.10(a). The transformation from the workflow patters in the model in Figure 6.10(a) to the SBSML model in Figure 6.10(b) follows the above informal description of



(a) A workflow model using the `sequence`, `parallel split`, and `synchronization` workflow patterns.



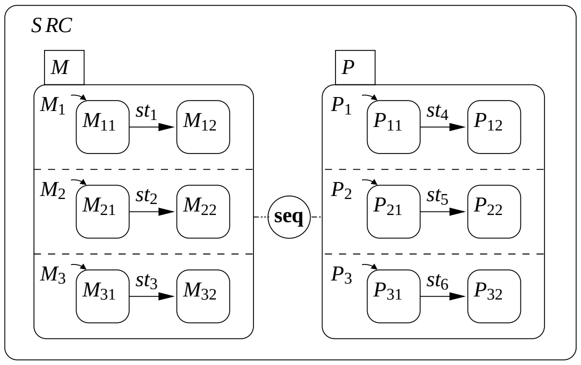
(b) Equivalent SBSML model for the model in Fig. 6.10(a).

Figure 6.10: Modelling the `sequence`, the `parallel split`, and the `synchronization` workflow patterns in SBSMLs.

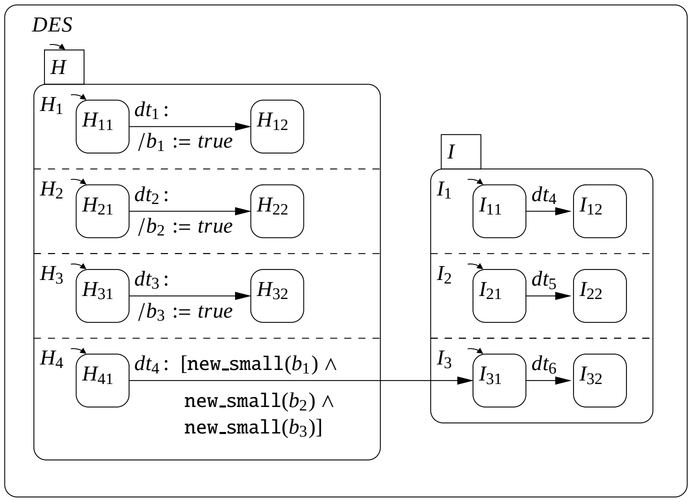
the semantics of the workflow patterns. The transformation is straightforward, thanks to being able to use multi-source, multi-destination transitions.

In interpreting and modelling the semantics of the sequence pattern above, in Figure 6.10(b), an additional idle small step is introduced between the last small step of the first activity and the first small step of the second activity. This extra small step can be avoided by using an non-preemptive semantics and an interrupt transition that transfers the control flow to the second activity simultaneously when the last small step of the first activity is being executed.

Figure 6.11(a) shows a model that uses workflow pattern `sequence` to execute P after M. The control states H and I, in the SBSML model in Figure 6.11(b), correspond to control states M and P in the model in Figure 6.11(a), respectively. The SBSML model in Figure 6.11(b) uses three boolean variables, namely b_1, b_2, and b_3, to determine the last small step at the end of which the last transitions of all of the children of H are executed. This last small step also executes new transition t, because its guard condition will be satisfied, moving the model to

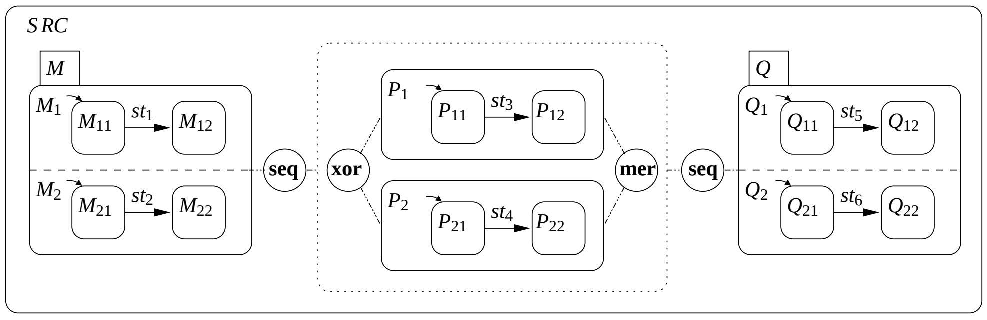


(a) Using `sequence` workflow pattern.

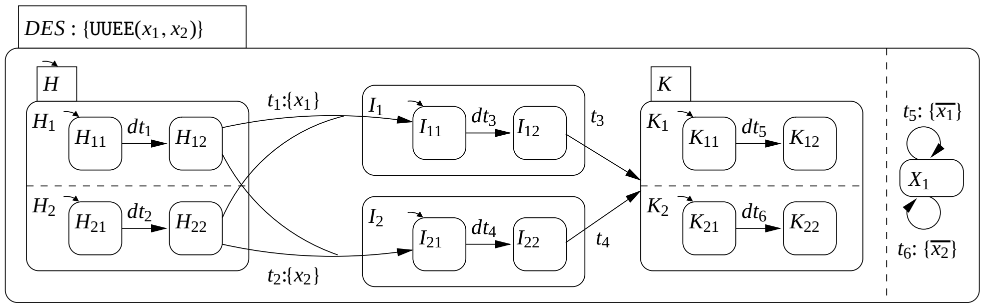


(b) Equivalent SBSML model as in Figure 6.11(a) , without introducing any extra small step.

Figure 6.11: An alternative modelling of the `sequence` workflow pattern without introducing any additional small step.



(a) A workflow model using the `sequence`, `exclusive choice`, and the `simple merge` workflow patterns.



(b) Equivalent SBSML model for the model in Fig. 6.12(a).

Figure 6.12: Modelling the `exclusive choice` and the `simple merge` workflow patterns in SBSMLs.

control state I. Although not shown here, variables b_1, b_2, and b_3 need to be reset to false, upon entering control state H. This translation, however, relies on using the `new_small` that in general is a transition-aware BSML semantics.

The model in Figure 6.12(a) shows example usage of the other two basic workflow patterns that are not used in the model in Figure 6.10(a), namely the `exclusive choice` and `simple merge` workflow patterns. The model also uses the `sequence` workflow pattern. The model in Figure 6.12(b) shows an SBSML model that has an equivalent behaviour as the model in Figure 6.12(a); the `sequence` workflow pattern is modelled using the first approach outlined above; i.e., using multi-source, multi-destination transitions together with introducing an extra small step. Control states M, P_1, P_2, and Q correspond to control states H, I_1, I_2, and K, respectively.

The semantics of the `exclusive choice` workflow pattern is modelled using a synchronizer of type `UUEE` with label set $\{x_1, x_2\}$. The size of the label set depends on the number of choices that the `exclusive choice` workflow pattern can choose from; in this case it is the non-deterministic choice of executing I_1 or I_2. In the model in Figure 6.12(b) either transition t_1 or transition t_2 could synchronize with transition t_5 or transition t_6, respectively, in effect, implementing the semantics of the `exclusive choice`. Since t_5 and t_6 cannot be executed together, exactly one choice is made. Each of the multi-source, multi-destination transitions t_1 and t_2 does not require `UUSE` for their synchronization, since only one of the constituent regular transitions of each needs to synchronize, as described in Section 6.4.1.

The semantics of the `simple merge` workflow pattern is based on the completion of exactly one of the alternative activities involved in this workflow pattern. By the definition of the pattern, only one of the activities can be executed at each point of time. For example, in the model in Figure 6.12(a), the `simple merge` requires that either st_3 or st_4, but not both, to be executed. In the model in Figure 6.12(b), two regular transitions, t_1 and t_2, are used to indicate the end, and *merge*, of the activities in I_1 and I_2, respectively. Transitions t_1 and t_2 cannot possibly be executed together.

6.5 Related Work: Taxonomies for Synchronization

The different synchronization types that are used in SBSMLs are inspired by various process algebras [9, 37], such as ACP [8], CCS [72], and CSP [48], and CIRCAL [70].

My classification of synchronization types overlaps with the classification of *multiparty interaction mechanisms* by Joung and Smolka [55]. They present a novel classification for synchronization mechanisms based on four criteria, which, in my terminology, can be described as: (i) whether or not the role sets and co-role sets of all transitions are singletons; (ii) whether or not a transition, in addition to specifying its role sets and co-role sets, can specify a particular transition (or transitions in a part of the hierarchy tree) with which it wishes to synchronize; (iii-a) whether or not the number of role sets and co-role sets of a transition together can be more than one; (iii-b) whether or not a control state can be the source control state of more than one transitions that can synchronize; and (iv) whether only a minimal set of synchronizing transitions are taken at each small step or a maximal set of all synchronizing transitions should be taken at each small step. Criterion (i) corresponds to the first two letters of my synchronization types, with my criteria being more distinguishing. Criterion (ii) is not relevant for my framework since it can be modelled by a naming convention for labels (cf., [55, p. 85]). Criterion (iii), called *conjunctive vs. disjunctive parallelism* [55], is meant to distinguish between process algebras such as SCCS (synchronous CCS) [71], which can perform multiple handshakes in one small step, and CCS, which can do at most one handshake; this criterion is closely related to the criterion (i) [55, p.83]. Part (a) of the criterion is not relevant in my framework since multiple role sets, or multiple co-role sets, related to the same synchronizer must be merged into one role set, or co-role set, respectively. Part (b) of criterion (iii) is not considered as a parameter since it corresponds to a syntactic constraint in my framework, rather than a semantic concept. Lastly, criterion (iv) is not considered, focusing on semantics in which a maximal set of synchronizing transitions is always taken, in the spirit of the semantics of BSMLs, where a maximal, consistent set of enabled transitions is always taken as a small step.

Compared to Joung and Smolka's taxonomy, my framework additionally considers the role of the third and fourth letters of my synchronization types. Also, additionally, my framework allows multiple synchronization types in one language. In general, the taxonomy of Joung and Smolka "is based on issues that may affect the complexity of scheduling multiparty interaction" [55, p.78], whereas my framework is based on issues relevant for designing suitable modelling languages for requirements specification.

Bliudze and Sifakis introduce a semantic definition framework to define and compare the meaning of *glue operators* (composition/synchronization operators) in structural operational semantics (SOS) [15], in the context of component-based development frameworks. Their work does not intend to present a design space of glue operators, but instead presents a general way for how different glue operators can be compared in terms of expressiveness; e.g., comparing two set of glue operators according to the *weak expressiveness* and *strong expressiveness* pre-orders. In comparison, my goal is to create a parameterized framework of synchronization mechanisms based on relevant semantic criteria for modelling, independent of any semantic definition mechanism.

The results by Bliudze and Sifakis [15] and Joung and Smolka [55], however, could be useful for my work when designing tool support for SBSMLs.

My synchronizer syntax is inspired by the *encapsulation operator* in Argos [68]. The encapsulation operator specifies the scope in which a signal can be used. My syntax is different in that multiple synchronizers can be attached to the same control state.

A class of BSMLs called synchronous languages [40], which includes languages such as Esterel[14] and Argos [68], have been successful in the specification of deterministic behaviour:

"In contrast with traditional thread- or event-based concurrency models that embed no precise or deterministic scheduling policy in the design formalism, synchronous language semantics take care of all scheduling decisions." [93] The main characteristic of the synchronous languages is that the statuses of *signals* of a model are constant throughout a big step, thus a transition is either enabled or disabled with respect to the statuses of events, deterministically. Synchronous semantics, however, introduce semantic difficulties such as non-causality and global inconsistency [68, 40, 14, 86]. Using the synchronization capability of SBSMLs, it is possible to simulate the subset of the responsibilities of signals in synchronous languages that deal with the coordination of the simultaneous execution of transitions. As such, when signal-like artifacts are not available in a domain, e.g., UML StateMachines [78] uses a buffered events mechanism, synchronization could be used to achieve determinism in a model, by constructing the model such that each of its snapshots has a unique potential small step.

SBSMLs, however, as opposed to synchronous languages, do not guarantee determinism as an inherent property of their semantics.[5] When a deterministic behaviour is desired in an SBSML model, care should be taken when using a synchronizer that has a synchronization type with its third and/or fourth letter being S, which allows synchronization with an arbitrary number of transitions. Similarly, care should be taken when using multiple synchronizers in a model, which could allow multiple sets of transitions to synchronize, according to different synchronizers, thereby creating different potential small steps. As an example, in the model in Fig. 6.7(b), if labels x, y, and z are removed from the model, replacing them in the co-role sets of t_2, t_4, t_6 with u, v, and w, respectively, the model can create a wrong small step that would include dt_1, dt_2, dt_3, and dt_4. The wrong small step is possible because label x' in dt_1 can match its corresponding label in t_1, while label u of dt_3 can match its corresponding label in dt_2. Similarly, dt_2 and dt_4 can match their corresponding labels in t_3 and dt_1, respectively.

6.6 Summary

This chapter introduced a synchronization mechanism for the family of BSMLs. Syntactically, transitions are extended with role sets, each of which is a set of labels, and co-role sets, each of which is a set of co-labels; and control states are extended with synchronizers, each of which has

[5]A model in a synchronous language with a possible nondeterministic behaviour is conservatively rejected at compile time.

one of the 16 introduced synchronization types and a set of labels. Semantically, the transitions of a model synchronize via their role sets and co-role sets according to the synchronizers that control them. The result is the new family of synchronizing big-step modelling languages (SBSMLs). The chapter showed that SBSMLs not only have applications in modelling but also can be used to model different semantic variations of the big-step semantic deconstruction, as well as, the semantics of many common modelling constructs.

Chapter 7

Formal Semantics for SBSMLs

"A unifying theory is usually complementary to the theories that it links, and does not seek to replace them." [48, p.1]

Tony Hoare and He Jifeng

This chapter presents a formal semantics for the synchronization syntax described in Chapter 6. The semantics of synchronization is entirely orthogonal to the enabledness semantic aspects, thus this chapter relies on the previously-stated formalization of the enabledness semantic aspects in Section 4.5. The Concurrency and Consistency semantic aspect is not relevant in the presence of synchronization: For each of its sub-aspects, using one of its semantic options and synchronizers, the other semantic option can be modelled. Therefore, the Concurrency and Consistency semantic aspect is not needed in this chapter.

The remainder of this chapter is organized as follows. Section 7.1 presents the formal syntax of synchronizing big-step modelling languages (SBSMLs) in the form of a BNF. Section 7.2 presents the formal semantics of SBSMLs, which is based on a semantic definition schema similar to the one for BSMLs. Section 7.3 presents a succinct formalization of the synchronization types that is plugged into the definition in Section 7.2, to derive a complete semantic definition. Section 7.4 formally describes the transformations of modelling constructs, such as multi-source, multi-destination transitions and composition operators, to SBSMLs. It also describes the transformations of one semantic option of the Concurrency and Consistency semantic aspect to another using SBSMLs. The proofs of the correctness of these transformations are also presented.

$$
\begin{array}{lll}
\langle\text{root}\rangle & ::= & \langle\text{Orstate}\rangle \\
\langle\text{Orstate}\rangle & ::= & \textbf{Or}\ \langle\text{states-o}\rangle\ \langle\text{transitions}\rangle\ \langle\text{synchronizers}\rangle \\
\langle\text{states-o}\rangle & ::= & \langle\text{states-o}\rangle\ \langle\text{state}\rangle \mid \langle\text{state}\rangle \\
\langle\text{Andstate}\rangle & ::= & \textbf{And}\ \langle\text{states-a}\rangle\ \langle\text{transitions}\rangle\ \langle\text{synchronizers}\rangle \\
\langle\text{states-a}\rangle & ::= & \langle\text{states-a}\rangle\ \langle\text{state}\rangle \mid \langle\text{state}\rangle \\
\langle\text{Basicstate}\rangle & ::= & \textbf{Basic} \\
\langle\text{state}\rangle & ::= & \langle\text{Orstate}\rangle \mid \langle\text{Andstate}\rangle \mid \langle\text{Basicstate}\rangle \\
\langle\text{synchronizers}\rangle & \rightarrow & \langle\text{synchronizer}\rangle \mid \langle\text{synchronizers}\rangle\ \langle\text{synchronizer}\rangle \\
\langle\text{synchronizer}\rangle & \rightarrow & \texttt{""} \mid \langle\text{synchtype}\rangle\ \langle\text{labelset}\rangle \\
\langle\text{synchtype}\rangle & \rightarrow & \texttt{UUEE} \mid \texttt{UPEE} \mid \texttt{PUEE} \mid \texttt{PPEE} \mid \texttt{UUSE} \mid \texttt{UPSE} \mid \cdots \mid \texttt{PPSS}
\end{array}
$$

Figure 7.1: SBSML abstract syntax in BNF.

Section 7.5 discusses the relevance of the semantic quality attributes, which were introduced for BSMLs in Chapter 5, for SBSMLs.

7.1 Formal Syntax

This section presents the syntax of SBSMLs formally, followed by introducing syntactic notation needed for the formalization of SBSML semantics.

7.1.1 Synchronization-Related Definitions

Figure 7.1 presents the BNF for the syntax of SBSMLs. Similar to the BNF of BSMLs, in Figure 2.3, an SBSML model is a hierarchy tree of *And*, *Or*, and *Basic* control states, together with transitions over these control states. Additionally, the BNF in Figure 7.1 allows each compound control state to have a set of synchronizers. A control state, a transition, or a synchronizer has a unique name, similar to the ones defined for BSMLs in Section 2.1.3, on page 20. For the sake of brevity and clarity, I have not included these names in the abstract syntax in Figure 7.1, however, it is always possible to ascribe a unique name to each of these elements of a model to identify it (e.g., by labelling the nodes of the hierarchy tree of the model according to an order of traversal).

Table 7.1 presents the accessor functions and relations for the elements of the syntax of an SBSML model. These definitions were discussed informally in Chapter 6. In addition to these

Table 7.1: Syntactic notation for SBSMLs.

Notation	Description
syn(s)	The set of *synchronizers* of control state s.
syntype(y)	The *synchronization type* of synchronizer y.
synlabels(y)	The *label set* of synchronizer y.
rolesets(t)	The set of *role sets* of transition t, each of which is a set of labels.
corolesets(t)	The set of *co-role sets* of transition t, each of which is a set of co-labels.

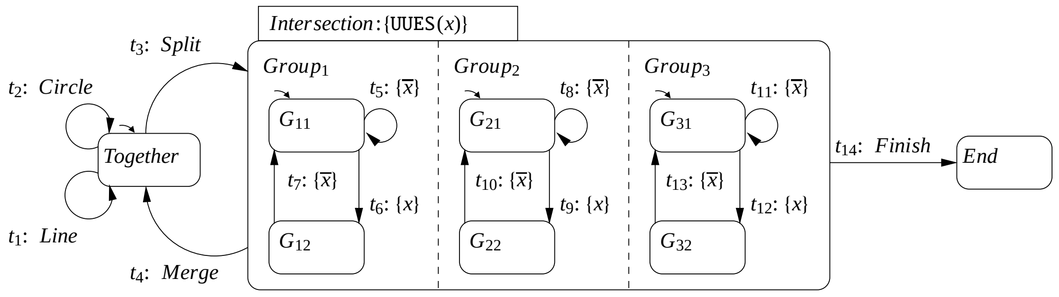


Figure 7.2: A model for a set of synchronized ice skating programs (the same as the model in Figure 6.2).

definitions, the notation defined in Table 4.1 and Table 4.2 for BSMLs, on pages 92 and page 93, respectively, are also used for SBSMLs.

Example 46 *In the SBSML model in Figure 7.2, which is the same model as in Figure 6.2 copied here for convenience, syn*(Intersection) = {UUES(x)} , *syntype*(UUES(x)) = UUES*, and synlabels*(UUES(x)) = {x}. *Also, rolesets*(t$_5$) = ∅*, corolesets*(t$_5$) = $\{\{\overline{\text{x}}\}\}$*, rolesets*(t$_6$) = {{x}}*, and corolesets*(t$_6$) = ∅*.*

As usual, the outer pair of curly brackets of a singleton set is dropped; e.g., instead of *corolesets*(t_5) = $\{\{\overline{x}\}\}$ in Figure 7.2, *corolesets*(t_5) = $\{\overline{x}\}$.

7.1.2 Well-Formed SBSML Models

As informally described in Section 6.3.1, an SBSML model must be well-formed, by following seven well-formedness conditions. Next, these conditions are presented formally.

i Each label uniquely belongs to the label set of exactly one synchronizer. Formally,

$$\forall l \cdot \exists y_1, y_2 \cdot (l \in \mathit{synlabels}(y_1)) \wedge (l \in \mathit{synlabels}(y_2)) \Rightarrow y_1 = y_2.$$

ii For each label, *l*, if there is at least one transition with a poly-role that includes *l*, then the synchronization type of its corresponding synchronizer should be a synchronization type whose first letter is `P` (i.e., `P***`), otherwise it must be one of the `U***` synchronization types. Similarly, the second letter of a synchronization type is specified based on the characteristics of the co-role sets of transitions. Formally,

$$\forall s \cdot \forall y \in \mathit{syn}(s) \cdot \forall l \in \mathit{synlabels}(y) \cdot$$
$$(\exists t \cdot \exists r \in \mathit{rolesets}(t) \cdot (l \in r) \wedge (|r| > 1) \Leftrightarrow \mathit{syntype}(y) \in \texttt{P***}),$$

where `P***` represents the set of synchronization types whose first letters is `P`. A similar predicate for the second letter of the synchronization type of a synchronizer is defined that checks for the size of co-role sets of transitions, instead of role sets in the above predicate.

iii No two synchronizers of a control state have the same synchronization type. Formally,

$$\forall s \cdot \forall y_1, y_2 \in \mathit{syn}(s) \cdot \mathit{syntype}(y_1) = \mathit{syntype}(y_2) \Rightarrow y_1 = y_2.$$

iv Two labels that are associated with the same synchronization type do not belong to two different role sets or two different co-role sets of the same transition. Formally,

$$\forall s \cdot \forall y \in \mathit{syn}(s) \cdot \forall l_1, l_2 \in \mathit{synlabels}(y) \cdot$$
$$(\forall t \cdot \forall r_1, r_2 \in \mathit{rolesets}(t) \cdot (l_1 \in r_1) \wedge (l_2 \in r_2) \Rightarrow r_1 = r_2) \wedge$$
$$(\forall t \cdot \forall c_1, c_2 \in \mathit{corolesets}(t) \cdot (\overline{l_1} \in c_1) \wedge (\overline{l_2} \in c_2) \Rightarrow c_1 = c_2).$$

v For each label, *l*, of a synchronizer, *y*, and each transition, *t*, *l* is associated with at most one of the role sets or co-role sets of *t*. Formally,

$$\forall s \cdot \forall y \in \mathit{syn}(s) \cdot \forall l \in \mathit{synlabels}(y) \cdot \forall t \cdot$$
$$\neg[(\exists r \in \mathit{rolesets}(t) \cdot (l \in r)) \wedge (\exists c \in \mathit{corolesets}(t) \cdot (\bar{l} \in c))].$$

vi A synchronizer, *y*, is associated with the least common ancestor of the source and destination control states of the transitions that use the labels of y in their role sets or co-role sets. Formally,

$$\begin{gathered}\forall s \cdot \forall y \in syn(s) \cdot \neg[\forall l \in synlabels(y) \cdot \exists s' \in children^{+}(s) \cdot \forall t \cdot \\ \forall r \in rolesets(t) \cdot (l \in r) \Rightarrow lca(src(t), dest(t)) \in children^{*}(s') \;\; \wedge \\ \forall c \in corolesets(t) \cdot (\bar{l} \in c) \Rightarrow lca(src(t), dest(t)) \in children^{*}(s')].\end{gathered}$$

vii A synchronizer, y, of a control state, s, cannot be split into two synchronizers, y_1 and y_2, such that y_1 is assigned to s but y_2 is assigned to a descendant of s. Formally,

$$\begin{gathered}\forall s \cdot \forall y \in syn(s) \cdot \neg[\exists L \subset synlabels(y) \cdot \exists s' \in children^{+}(s) \cdot \forall t \cdot \\ \forall r \in rolesets(t) \cdot (r \subseteq L) \Rightarrow lca(src(t), dest(t)) \in children^{*}(s') \;\; \wedge \\ \forall c \in corolesets(t) \cdot (c \subseteq (\textstyle\bigcup_{l \in L} \bar{l})) \Rightarrow lca(src(t), dest(t)) \in children^{*}(s')].\end{gathered}$$

The above well-formedness constraints together ensure that scoping problems, such as name clash between the labels of two synchronizers, do not arise in the semantic definition of SBSMLs.

7.2 Semantic Definition for SBSMLs

Similar to the formalization of BSML semantics, SBSML semantics are defined using a semantic definition schema. Figure 7.3 is the schema that is used to define SBSML semantics, and is the same as the one for BSMLs in Figure 4.2, on page 88. As usual, the set of potential small steps of a model at snapshot sp is denoted as $executable(root, sp)$.

7.2.1 Semantics of SBSMLs vs. Semantics of BSMLs

The formalization of the enabledness semantic aspects for SBSMLs are exactly the same as for BSMLs in Section 4.5, because they do not have any role in the semantics of synchronization. SBSMLs all use the fixed semantic options of MANY for the concurrency semantic aspect, SOURCE/DESTINATION ORTHOGONAL for the small-step consistency semantic aspect, and the NON-PREEMPTIVE for the preemption semantic aspect. The alternative semantic options for these semantic aspects can be modelled using synchronizers, as described in Section 6.4.2 informally,

$$
\begin{array}{lcl}
1.\ N_{Big}(sp^0, I, sp') & \equiv & reset(sp^0, I, sp) \wedge (\exists k \geq 0 \cdot N^k(sp, sp')) \\
 & & \wedge\ executable(root, sp') = \emptyset \\
2.\ reset(sp^0, I, sp) & \equiv & \bigwedge_{1 \leq i \leq n} \texttt{reset\_}el_i(sp^0, I, sp) \\
3.\ N^0(sp, sp') & \equiv & sp = sp' \\
4.\ N^{k+1}(sp, sp') & \equiv & \exists \tau, sp'' \cdot N_{Small}(sp, \tau, sp'') \wedge N^k(sp'', sp') \\
5.\ N_{Small}(sp, \tau, sp') & \equiv & \bigwedge_{1 \leq i \leq n} \texttt{next\_}el_i(sp, \tau, sp') \wedge \tau \in executable(root, sp)
\end{array}
$$

Figure 7.3: Semantic definition schema for SBSMLs.

and as will be formalized in Section 7.4.2. Thus, the Concurrency and Consistency semantic aspect is not considered in the semantic formalization of SBSMLs.

The semantics of SBSMLs, however, differ from the semantics of BSMLs in how the potential small steps of a model are created. In a BSML model, a potential small step of the model at a snapshot is a maximal set of enabled, high-priority transitions that can be taken together according to the concurrency and consistency semantics of the BSML. In an SBSML model, such a maximal set of enabled, high-priority transitions is not a potential small step of the model if the synchronization requirements of the transitions in the set are not satisfied.

For example, for an SBSML with a hierarchical priority semantics, a naive approach to enforce the synchronization requirements of the transitions of a potential small step of a model is to keep track of the synchronizing transitions of the model as its hierarchy tree, and its synchronizers, are traversed hierarchically. This approach would be similar to the hierarchical computation for BSMLs, presented in Section 4.4.1 and Section 4.4.2. However, such an approach fails to consider that: (i) the synchronizing transitions of a model do not necessary have the same scope; and (ii) a transition can be controlled by more than one synchronizers that are associated with different control states. Implementing the above two requirements in a hierarchical semantic definition, however, is not straightforward. For example, let us consider adapting the hierarchical computation in Figure 4.10, on page 107, for an SBSML with the SCOPE-CHILD priority semantics. Let us consider the hierarchical computation when control state, *s*, of an SBSML model, and a pair of transitions, t_1 and t_2, that synchronize according to the synchronization requirements of a synchronizer in a lower control state and are passed to the attribute of *s* that keeps track of executable transitions. If one of the transitions, e.g., t_1, has an extra synchronization requirement enforced by a synchronizer at the current level, but that requirement cannot be satisfied by any enabled transition with a scope at the current level, then t_1 should be removed from the set of executable transitions. Furthermore, removing t_1 means that: (a) t_2 should also be removed from the set of executable transitions; and (b) t_1 and t_2 could possibly be replaced by some enabled transitions from the lower levels of the hierarchy tree. However, this means that the computation in Figure 4.10 should be changed in a way that is not quite hierarchical any more. Similar problems arise when the Scope-Parent or the No Priority priority semantics is considered for SBSMLs.

Next, a method for computation of potential small steps in SBSMLS is presented, in which the synchronization and priority constraints are only considered at the root of a model, and only once.

7.2.2 Computing the Potential Small Steps

Figure 7.4 is an attribute-grammar–like formalism to compute the set of potential small steps of an SBSML model. Similar to the formalization of the semantics of BSMLs, an attribute-grammar–like formalism is used to compute the set of potential small steps of a model at a snapshot, sp. The value of $executable(root, sp)$ is equal to the value of attribute $\mathbf{ex}(\text{root}, sp)$ in Figure 7.4. Attribute $\mathbf{ex}(\text{root}, sp)$ computes the set of potential small steps of a model at snapshot sp based on a chosen priority semantics and the set of synchronizers of the model, $\mathbb{Y}$. Lines 2a to 9a in the schema compute the possible potential small steps of an SBSML model incrementally, as if it is a BSML model and the No Priority semantics is chosen, similar to lines 2a-9a of Figure 4.10, as described in Section 4.4. The semantics of synchronization and priority are considered in line 1a. Line 10 in Figure 7.4 is the definition of the merge operator, denoted by "$\otimes$", which is similar to the definitions of the merge operators in Figure 4.8 and Figure 4.10, except that the Source/Destination Orthogonal and the Non-Preemptive semantics are hard-coded as consistency criteria in the last two conjuncts.

In the formalization of the semantics of BSMLs, for each of the hierarchical semantic options, Scope-Parent and Scope-Child, a separate hierarchical computation for $executable(root, sp)$ is needed, as shown in Figure 4.8 and Figure 4.10, respectively. For SBSMLs, however, only one hierarchical computation is used: The priority semantics is enforced at the root of a hierarchy tree. This approach is adopted because the enabledness of a transition and its being a high-priority transition does not mean that it belongs to a potential small step if its synchronization

1. ⟨root⟩	::=	⟨Orstate⟩
a. **ex**(root, sp)	=	$PRI[MAX(SYN($**ex**(Orstate, sp)$))]$
2. ⟨Orstate⟩	::=	**Or** ⟨states-o⟩ ⟨transitions⟩ ⟨synchronizers⟩
a. **ex**(Orstate, sp)	=	**ex**(states-o, sp) $\otimes$ en_trs(transitions, sp)
3. ⟨states-o⟩	::=	⟨states-o⟩ ⟨state⟩
a. **ex**(states-o$_0$, sp)	=	**ex**(states-o$_1$, sp) $\cup$ **ex**(state, sp)
4. ⟨states-o⟩	::=	⟨state⟩
a.**ex**(states-o, sp)	=	**ex**(state, sp)
5. ⟨Andstate⟩	::=	**And** ⟨states-a⟩ ⟨transitions⟩ ⟨synchronizers⟩
a. **ex**(Andstate, sp)	=	**ex**(states-a, sp) $\otimes$ en_trs(transitions, sp)
6. ⟨states-a⟩	::=	⟨states-a⟩ ⟨state⟩
a.**ex**(states-a$_0$, sp)	=	$\{T_1 \cup T_2 \mid T_1 \in$ **ex**(states-a$_1$, sp) $\wedge\ T_2 \in$ **ex**(state, sp)$\}$
7. ⟨states-a⟩	::=	⟨state⟩
a. **ex**(states-a, sp)	=	**ex**(state, sp)
8. ⟨Basicstate⟩	::=	**Basic**
a. **ex**(Basicstate, sp)	=	$\emptyset$
9. ⟨state⟩	::=	⟨Orstate⟩ | ⟨Andstate⟩ | ⟨Basicstate⟩
a. **ex**(state, sp)	=	**ex**(Orstate, sp), **ex**(Andstate, sp), or **ex**(Basicstate, sp)

10. $\mathbb{T} \otimes T'$	=	$\{ (T_1 \cup T'') \mid T_1 \in \mathbb{T} \wedge T'' \subseteq T' \wedge \forall t' : (T_1 \cup T') \cdot t' \in (T' - T'') \Leftrightarrow \exists t \in (T_1 \cup T'') \cdot (t \not\perp t') \wedge \neg((t \lightning t') \vee (t' \lightning t)) \}$
11. $SYN(\mathbb{T})$	=	$\Big\{ T - T' \mid (T \in \mathbb{T}) \wedge (T' \subset T) \wedge \bigwedge_{1 \le i \le 4} SYN_i(T - T') \wedge \forall X : 2^{T'} \cdot (X \neq \emptyset) \Rightarrow \neg\Big[\bigwedge_{1 \le i \le 4} SYN_i((T - T') \cup X)\Big] \Big\}$
12. $MAX(\mathbb{T})$	=	$\{ T \mid T \in \mathbb{T} \wedge (\nexists T' \in \mathbb{T} \cdot (T' \supset T)) \}$
13. $PRI(\mathbb{T})$	=	$\{ T \mid T \in \mathbb{T} \wedge (\nexists T' \in \mathbb{T} \cdot (T' \gtrdot T)) \}$

Figure 7.4: Computing potential small steps of SBSML models.

requirements cannot be satisfied by other transitions in any potential small step. As such, the priority and synchronization semantics cannot be considered independently: They need to be considered only once at the root of the hierarchy tree.[1]

Line 1a in Figure 7.4 uses three functions, namely, PRI, MAX, and SYN, to enforce a priority semantics, to ensure that a potential small step cannot be extended with further transitions, and to incorporate the roles of the synchronizers of an SBSML model, respectively. Line 11 in Figure 7.4 specifies the SYN function, whose detailed definition is presented in Section 7.3. For each set of transitions, T, the SYN function computes a set of transitions, T', that can be removed from T so that the result consists of transitions that are synchronizing according to all of the synchronizers of the model. The fourth conjunct of the definition, on the second line, ensures that T' is a minimal set. Line 12 specifies the MAX function, which discards all computed sets of transitions by the SYN function that are a subset of another computed set of transitions. Function MAX is necessary, despite the fourth conjunct of the SYN function, because although a set of transition $T - T'$ computed by the SYN function is maximal, there could exist another set of transitions $R - R'$ such that $(R - R') \supset (T - T')$. For example, if $T = \{t_1, t_2, t_3\}$, $T' = \{t_3\}$, $R = \{t_1, t_2, t_4\}$, $R' = \emptyset$, and $t_3 \not\perp t_4$, then $(R–R') \supset (T–T')$. The fourth conjunct of the SYN function is not necessary, but I mention it to distinguish between the two types of maximality involved in the computation of the set of potential small steps. Line 13 specifies the PRI function. The PRI function allows a set of transitions to be a potential small step if there is not any set of transitions T' that has a higher priority than T. As described in Section 5.1.1, the semantics of the "$\gtrdot$" operator depends on the choice of a priority semantics. If the No Priority priority semantics is chosen, then $PRI(\mathbb{T}) = \mathbb{T}$.

7.3 Formalization of Synchronization Types

This section formally specifies the semantics of the synchronization types, which, in turn, are used to specify the formal definition of the SYN function, used in the computation in Figure 7.4. Recall that a synchronization type is a four-letter sting that specifies how a synchronizer that uses it can coordinate the execution of the transitions that use the labels in the label set of the

[1] A similar semantic definition approach for hierarchical priority semantics to the one used for SBSMLs in this chapter could have been adopted for BSMLs. However, I found the semantic definition approach in Chapter 4 more prescriptive than the alternative because the formalization of a hierarchical priority semantics is more clearly manifested in a BSML semantic definition.

Table 7.2: Synchronization types and their parameters, when considered for synchronizer $Y(L)$.

Index	Parameter Purpose	Values for Synchronizer $Y(L)$
1	How an identifier can be used in the role sets of transitions	`U`: The identifiers in L can be used only in uni-roles
		`P`: The identifiers in L can be used in poly-roles
2	How an identifier can be used in the co-role sets of transitions	`U`: The identifiers in L can be used only in uni-co-roles
		`P`: The identifiers in L can be used in poly-co-roles
3	How many instances of a label can appear in the role sets of transitions in a small step	`E`: One, exclusively
		`S`: Many, in a shared manner
4	How many instances of a co-label can appear in the co-role sets of transitions in a small step	`E`: One, exclusively
		`S`: Many, in a shared manner

synchronizer. Table 7.2, copied here from page 193 for convenience, summarizes the meaning of each letter of a synchronization type.

This section presents a formalization of the synchronization types that formalizes the meaning of each letter of a synchronization type as a separate predicate, in a prescriptive way. This formalization is designed based on the observation that a set of transitions are synchronizing according to a synchronizer if the conjunction of four statements that each represents one of the letters of the synchronization type of the synchronizer is true. For example, let us consider a synchronizer `PUES`(L) and a set of transitions, τ, that synchronize according to `PUES`(L). Also, let us denote the subset of labels in L that have been used in the role sets and co-role sets of the transitions in τ as M. Then, for each label $m \in M$, the following four statements must be all true: (i) if a transition $t \in \tau$ uses a label $m \in M$ in one of its role sets, r, then for any other label $m' \in r$, $m' \in M$; (ii) a transition $t \in \tau$ can use a label $m \in M$ only in a uni-co–role set; (iii) a label $m \in M$ must be used in exactly one of the role sets of one of the transitions in τ, and furthermore, it should match the over-lined label of one uni-co–role set of at least one transition in τ; and (iv) a label $m \in M$ must be used in at least one of the uni-co–role sets of one of the transitions in τ, and furthermore, it should match the label of one of the role sets of exactly one transition in τ. Each of the above four statements corresponds to the meaning of one of the letters of synchronization type “`PUES`”. Furthermore, each statement refers to a particular set of transitions and the labels over which they synchronize.

In the formalization of the synchronization types, to abstract away from the particularities of a certain set of synchronizing transitions, *relation types* are used that each represents one of the letters of one of the 16 synchronization types. If two synchronization types use the same letter in the same index, then it is not necessarily the case that their relation types are the same, because of the interdependencies between different letters of a synchronization type. A relation type that represents one of the letters of a synchronization type can be instantiated with a particular label set and a particular set of transitions. The result will be a set of sets of tuples that enumerate the acceptable patterns of interaction between the transitions over the labels of the label set, according to the letter of the synchronization type that the relation type represents. Thus, if a set of transitions is synchronizing according to one of the letters of the synchronization type of a synchronizer, then the pattern of interaction of the transitions must be one of the acceptable patterns derived from the corresponding relation type of the letter.

Next subsection describes how the relation types are defined and how they can be integrated into the semantic definition schema presented in the previous section to derive a complete semantics for an SBSML.

7.3.1 Formalization

For a synchronizer y and a set of transitions τ, τ_y and $\overline{\tau}_y$ denote the set of transitions in τ that has a role set using at least one of the labels in *synlabels*(y) and the set of transitions in τ that has a co-role set using at least one of the labels in *synlabels*(y), respectively. Conversely, L^{τ}_{y} and $\overline{L}^{\tau}_{y}$ denote the set of labels in *synlabels*(y) that are used in at least one of the role sets of the transitions in τ and the set of labels in *synlabels*(y) that are used in at least one of the co-role sets of the transitions in τ, respectively.

Example 47 *In the model in Figure 7.2, if* $\tau = \{\mathrm{t}_6, \mathrm{t}_8, \mathrm{t}_{11}\}$ *and* $y = \mathrm{UUES(x)}$*, then* $\tau_y = \{\mathrm{t}_6\}$, $\overline{\tau}_y = \{\mathrm{t}_8, \mathrm{t}_{11}\}$, $L^{\tau}_{y} = \{\mathrm{x}\}$ *and* $\overline{L}^{\tau}_{y} = \{\mathrm{x}\}$.

It is important to note that $\overline{L}^{\tau}_{y}$ *is equal to* $\{\mathrm{x}\}$*, and not* $\{\overline{\mathrm{x}}\}$*, as is the case for a co-role set.*

Next, some notation are introduced that capture the interactions amongst a set of transitions, τ, through the labels of a synchronizer, y. Four relations are introduced that each captures these interactions from the perspective of one of the letters of the synchronization type of the synchronizer.

Relations $\lambda_3(\tau, y) : (\tau_y \times L^\tau_y) \times \overline{\tau}_y$ and $\lambda_4(\tau, y) : (\overline{\tau}_y \times \overline{L}^\tau_y) \times \tau_y$ correspond to the third and fourth letters of *syntype*(y), respectively. Relation λ_3 specifies how the transitions that each has at least a role set using *synlabels*(y) interact with transitions that each has at least a co-role set using *synlabels*(y). Relation λ_4 does the opposite: It specifies how the transitions that each has at least a co-role set using *synlabels*(y) interact with transitions that each has at least a role set using *synlabels*(y). Formally,

$$\begin{array}{rcl} \lambda_3(\tau, y) & = & \{(t, l), t' \mid l \in \bigcup_{r \in rolesets(t)}(r) \land \overline{l} \in \bigcup_{c \in corolesets(t')}(c) \land l \in synlabels(y)\}, \text{ and} \\ \lambda_4(\tau, y) & = & \{(t, l), t' \mid \overline{l} \in \bigcup_{c \in corolesets(t)}(c) \land l \in \bigcup_{c \in roleset(t')}(r) \land l \in synlabels(y)\}. \end{array}$$

Relations $\lambda_1(\tau, y) : \tau_y \times (L^\tau_y \cup \overline{L}^\tau_y)$ and $\lambda_2(\tau, y) : \overline{\tau}_y \times (L^\tau_y \cup \overline{L}^\tau_y)$, except for their types, are derived from λ_3 and λ_4 relations, respectively. They correspond to the first and second letters of *syntype*(y), respectively. Relation λ_1 specifies how the transitions that each has at least a role set using *synlabels*(y) interact through the labels used by either the role sets or the co-role sets of the transitions in τ. Relation λ_2 does the opposite: It specifies how the transitions that each has at least a co-role set using *synlabels*(y) interact through the labels used by either the role sets or the co-role sets of the transitions in τ. Formally,

$$\begin{array}{rcl} \lambda_1(\tau, y) & = & \text{dom}(\lambda_3(\tau, y)), \text{ and} \\ \lambda_2(\tau, y) & = & \text{dom}(\lambda_4(\tau, y)). \end{array}$$

The types of λ_1 and λ_2 are not derived from the types of λ_3 and λ_4. Instead, they require their ranges to be $(L^\tau_y \cup \overline{L}^\tau_y)$, instead of L^τ_y and $\overline{L}^\tau_y$, respectively, in order to specify the labels used by both the role sets and the co-role sets of the transitions, as specified above.

Example 48 *In the model in Figure 7.2, for* $\tau = \{\mathrm{t}_6, \mathrm{t}_8, \mathrm{t}_{11}\}$ *and* $y = \mathrm{UUES(x)}$,

i $\lambda_3(\tau, y) = \{((\mathrm{t}_6, \mathrm{x}), \mathrm{t}_8), ((\mathrm{t}_6, \mathrm{x}), \mathrm{t}_{11})\}$,
(The two tuples together specify that t_6 *uses label* x *in its role set to interact with both* t_8 *and* t_{11}*, which both have a co-role set that uses* x*.)*

ii $\lambda_4(\tau, y) = \{((\mathrm{t}_8, \mathrm{x}), \mathrm{t}_6), ((\mathrm{t}_{11}, \mathrm{x}), \mathrm{t}_6)\}$,
(The first and second tuples specify that transitions t_8 *and* t_{11} *each uses label* x *in one of their co-role sets to interact with* t_6*, which has a role set that uses* x*.)*

iii $\lambda_1(\tau, y) = \{(\mathrm{t}_6, \mathrm{x})\}$*, and*
(The relation specifies that transition t_6 *uses label* x *in one of its role sets to interact with the other transitions.)*

iv $\lambda_2(\tau, y) = \{(\mathrm{t}_8, \mathrm{x}), (\mathrm{t}_{11}, \mathrm{x})\}$.
(The relation specifies that transitions t_8 *and* t_{11} *each uses label* x *in one of its co-role sets to interact with the other transitions.)*

The λ_i's relations, $1 \le i \le 4$, are used to check whether a set of transitions, τ, satisfies the semantics of a synchronizer, y. For each of the four letters of a synchronization type, a relation type is introduced that describes the pattern of interactions of the synchronization type according to that letter. Table 7.3 specifies these relation types for each synchronization type for an arbitrary *syntype*(y) and τ. For a set of transitions τ and a synchronizer y, depending on *synchtype*(y), Λ_1, Λ_2, Λ_3, and Λ_4 determine the type of relations $\lambda_1(\tau, y)$, $\lambda_2(\tau, y)$, $\lambda_3(\tau, y)$, and $\lambda_4(\tau, y)$, respectively. In fact, Table 7.3 specifies 16 *families* of relation types, each family representing all of the synchronizers with the same synchronization type, each of the synchronizers considered with its all possible sets of enabled transitions. The symbols used in Table 7.3, which specify the type of the relations, are similar to the ones used in Z notation to denote the type of a relation or function [91]; Table 7.4 specifies the meaning of each symbol.[2] A non-empty set of transitions, τ, which could be chosen as a potential small step, satisfy the synchronization requirements of a synchronizer y, if

$$\begin{gathered}[\,[\lambda_1(\tau, y) \in \Lambda_1(\tau, y)] \wedge [\lambda_2(\tau, y) \in \Lambda_2(\tau, y)] \wedge [\lambda_3(\tau, y) \in \Lambda_3(\tau, y)] \wedge [\lambda_4(\tau, y) \in \Lambda_4(\tau, y)]\,] \vee \\ [\,(\lambda_1(\tau, y) = \emptyset) \wedge (\lambda_2(\tau, y) = \emptyset) \wedge (\lambda_3(\tau, y) = \emptyset) \wedge (\lambda_4(\tau, y) = \emptyset)\,].\end{gathered}$$

In the above predicate, the first disjunct specifies the case that at least two transitions in τ participate in a synchronization according to the labels in y. The second disjunct considers the case for the transitions in τ that do not participate in any synchronization according to the labels in y; e.g., τ could contain transitions without any role sets or co-role sets.

Example 49 *In the model in Figure 7.2, for* $\tau = \{\mathrm{t}_6, \mathrm{t}_8, \mathrm{t}_{11}\}$ *and* $y = \mathrm{UUES}(\mathrm{x})$*, the following conditions hold and thus* τ *satisfies* y*'s synchronization requirements (where a pair of* "[ ]" *is used to specify the set that represents a relation type):*

[2] I have also added a few extra symbols, in the same spirit as the symbols used in Z.

Table 7.3: Invariants of synchronization types for a set of transition τ.

Type	$\Lambda_3(\tau, y)$	$\Lambda_4(\tau, y)$	$\Lambda_1(\tau, y)$	$\Lambda_2(\tau, y)$
y = UUEE(L)	$(\tau_y \times L_y^\tau)$ ⤗ $\overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau)$ ⤗ τ_y	τ_y ⤖ $(L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y$ ⤖ $(L_y^\tau \cup \overline{L}_y^\tau)$
y = UPEE(L)	$(\tau_y \times L_y^\tau)$ ⤀ $\overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau)$ ⤗ τ_y	τ_y ⤖ $(L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadleftarrow (L_y^\tau \cup \overline{L}_y^\tau)$
y = PUEE(L)	$(\tau_y \times L_y^\tau)$ ⤗ $\overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau)$ ⤀ τ_y	$\tau_y \twoheadleftarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y$ ⤖ $(L_y^\tau \cup \overline{L}_y^\tau)$
y = PPEE(L)	$(\tau_y \times L_y^\tau)$ ⤀ $\overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau)$ ⤀ τ_y	$\tau_y \twoheadleftarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadleftarrow (L_y^\tau \cup \overline{L}_y^\tau)$
y = UUSE(L)	$(\tau_y \times L_y^\tau)$ ⤀ $\overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau) \leftarrow \tau_y$	$\tau_y \twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y$ ⤖ $(L_y^\tau \cup \overline{L}_y^\tau)$
y = UPSE(L)	$(\tau_y \times L_y^\tau)$ ⤀ $\overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau) \leftarrow \tau_y$	$\tau_y \twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadleftarrow (L_y^\tau \cup \overline{L}_y^\tau)$
y = PUSE(L)	$(\tau_y \times L_y^\tau)$ ⤀ $\overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \tau_y$	$\tau_y \twoheadleftarrow\!\!\!\!\twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y$ ⤖ $(L_y^\tau \cup \overline{L}_y^\tau)$
y = PPSE(L)	$(\tau_y \times L_y^\tau)$ ⤀ $\overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \tau_y$	$\tau_y \twoheadleftarrow\!\!\!\!\twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadleftarrow (L_y^\tau \cup \overline{L}_y^\tau)$
y = UUES(L)	$(\tau_y \times L_y^\tau) \leftarrow \overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau)$ ⤀ τ_y	τ_y ⤖ $(L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$
y = UPES(L)	$(\tau_y \times L_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau)$ ⤀ τ_y	τ_y ⤖ $(L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadleftarrow\!\!\!\!\twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$
y = PUES(L)	$(\tau_y \times L_y^\tau) \leftarrow \overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau)$ ⤀ τ_y	$\tau_y \twoheadleftarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$
y = PPES(L)	$(\tau_y \times L_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau)$ ⤀ τ_y	$\tau_y \twoheadleftarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadleftarrow\!\!\!\!\twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$
y = UUSS(L)	$(\tau_y \times L_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \tau_y$	$\tau_y \twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$
y = UPSS(L)	$(\tau_y \times L_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \tau_y$	$\tau_y \twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadleftarrow\!\!\!\!\twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$
y = PUSS(L)	$(\tau_y \times L_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \tau_y$	$\tau_y \twoheadleftarrow\!\!\!\!\twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$
y = PPSS(L)	$(\tau_y \times L_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \overline{\tau}_y$	$(\overline{\tau}_y \times \overline{L}_y^\tau) \leftarrow\!\!\!\!\twoheadrightarrow \tau_y$	$\tau_y \twoheadleftarrow\!\!\!\!\twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$	$\overline{\tau}_y \twoheadleftarrow\!\!\!\!\twoheadrightarrow (L_y^\tau \cup \overline{L}_y^\tau)$

Table 7.4: Relations and functions types.

Symbol	Meaning
$\leftrightarrow$	Relation
$\twoheadleftarrow\!\!\!\!\rightarrow$	Left-total relation
$\leftarrow\!\!\!\!\twoheadrightarrow$	Right-total relation
$\twoheadleftarrow\!\!\!\!\twoheadrightarrow$	Bi-direction total relation
⇸	Partial function
⤔	Injective, partial function
⤀	Surjective, partial function
⤗	Injective, surjective, partial function
$\rightarrow$	Total function
$\rightarrowtail$	Injective, total function
$\twoheadrightarrow$	Surjective, total function
⤖	Injective, surjective, total function

i $\lambda_3(\tau, y) \in [(\{t_6\} \times \{x\}) \leftarrow \{t_8, t_{11}\}]$,

ii $\lambda_4(\tau, y) \in [(\{t_8, t_{11}\} \times \{x\}) \twoheadrightarrow\!\!\!\!\!\!\!\twoheadrightarrow \{t_6\}]$,

iii $\lambda_1(\tau, y) \in [\{t_6\} \rightarrowtail\!\!\!\!\!\rightarrow \{x\}]$, *and*

iv $\lambda_2(\tau, y) \in [\{t_8, t_{11}\} \twoheadrightarrow \{x\}]$.

Where the values of $\lambda_1(\tau, y)$, $\lambda_2(\tau, y)$, $\lambda_3(\tau, y)$, *and* $\lambda_4(\tau, y)$ *were specified in Example 48.*

If $\tau' = \{t_6, t_9, t_{11}\}$ *and* $y = \mathrm{UUES}(x)$ *are chosen, the first three conditions do not hold and thus* τ' *does not satisfy y's synchronization requirements. Formally,*

i $\lambda_3(\tau', y) = \{((t_6, x), t_{11}), ((t_9, x), t_{11})\} \notin [(\{t_6, t_9\} \times \{x\}) \leftarrow \{t_{11}\}]$,

ii $\lambda_4(\tau', y) = \{((t_{11}, x), t_6), ((t_{11}, x), t_9)\} \notin [(\{t_{11}\} \times \{x\}) \twoheadrightarrow\!\!\!\!\!\!\!\twoheadrightarrow \{t_6, t_9\}]$,

iii $\lambda_1(\tau', y) = \{(t_6, x), (t_9, x)\} \notin [\{t_6, t_9\} \rightarrowtail\!\!\!\!\!\rightarrow \{x\}]$, *and*

iv $\lambda_2(\tau', y) = \{(t_{11}, x)\} \in [\{t_{11}\} \twoheadrightarrow \{x\}]$.

Relations λ_1 and λ_2, which correspond to the semantics of the first two letters of a synchronization type, are necessary to ensure that all the labels and co-labels of a set of synchronizing transitions engage in a synchronization. Otherwise, a set of transitions can be mistakenly, vacuously, considered as synchronizing.

Example 50 *In the model in Figure 7.2, for* $\tau = \{t_6\}$ *and* $y = \mathrm{UUES}(x)$, *the following two conditions hold for* τ *and y,*

i $\lambda_3(\tau, y) = \emptyset \in [(\{t_6\} \times \{x\}) \leftarrow \emptyset]$,

ii $\lambda_4(\tau, y) = \emptyset \in [(\emptyset \times \{x\}) \twoheadrightarrow\!\!\!\!\!\!\!\twoheadrightarrow \{t_6\}]$.

However, τ *is not meant to satisfy the requirements of* UUES(x), *because,*

i $\lambda_1(\tau, y) = \emptyset \notin [\{t_6\} \rightarrowtail\!\!\!\!\!\rightarrow \{x\}]$, *and*

ii $\lambda_2(\tau, y) = \emptyset \notin [\emptyset \twoheadrightarrow\!\!\!\!\!\!\!\twoheadrightarrow \{t_6\}]$.

Table 7.3 consists of 64 relation types: Each of the 16 synchronization types is represented by four relation types that each represents one of the letters of the synchronization type. Each synchronization type is uniquely identified by its set of four relation types. As mentioned earlier, if two synchronization types use the same letter in the same index, then it is not necessarily the case that their relation types are the same, because of the interdependencies between different letters of a synchronization type. The formalization of one letter of a synchronization type factors in the effect of other letters of the synchronization type. Next, as an example, it is explained how the formalization of the relation types for the synchronization type PUES can be derived from the English description of PUES.

Example 51 *Let us consider the earlier English description of the meaning of synchronization type PUES, when considered for a synchronizer $y = PUES(L)$ and a set of synchronizing transitions τ. The meaning of synchronization type PUES can be described through the following four statements that each corresponds to one of the letters of the synchronization type:*

- *The third letter, i.e., "E", requires that if a transition $t \in \tau_y$ uses a label $l \in L_y^\tau$ in one of its poly-role sets to interact with another transition $t' \in \overline{\tau}_y$, through one of the uni-co–role sets of t', then there should not exist any transition other than t that uses l in one of its role sets. This description is formalized in the following relation type,*

$$(\tau_y \times L_y^\tau) \leftarrow \overline{\tau}_y,$$

  which formalizes the above description by requiring that relation $\lambda_3(\tau, y)$ when considered in the inverse form is a total function. Recall that relation $\lambda_3(\tau, y)$ relates a pair (t, l), such that $t \in \tau_y$ and $l \in L_y^\tau$, to a transition $t' \in \overline{\tau}_y$, if t' uses l in one of its co-role sets. The relation type requires t' to associate with at most one (t, l). It requires $\lambda_3(\tau, y)$ to be a function because t' can have only uni-co–role set on $\overline{L}_y^\tau$, and thus it can be related with at most one transition in τ_y. Also, the function must be total because of the way λ_3 is constructed.

- *The fourth letter, i.e., "S", requires that if a transition $t \in \overline{\tau}_y$ uses a label $l \in \overline{L}_y^\tau$ in one of its uni-co–role sets to interact with another transition $t' \in \tau_y$, through one of the poly-role sets of t', then there should not exist any transition other than t' that uses l in one of its role* *sets. This description is formalized in the following relation type,*

$$(\overline{\tau}_y \times \overline{L}^\tau_y) \twoheadrightarrow\!\!\!\!\!\!\!\!\;\;\mapsto \tau_y,$$

which formalizes the above description by requiring that relation $\lambda_4(\tau, y)$ is a partial, surjective function. Recall that relation $\lambda_4(\tau, y)$ relates a pair (t, l), such that $t \in \overline{\tau}_y$ and $l \in \overline{L}^\tau_y$, to a transition $t' \in \tau_y$, if t' uses l in one of its role sets. Thus, (t, l) is required to associate with at most one t'. The relation type requires $\lambda_4(\tau, y)$ to be a function because there must be only one such t' that uses the label l in its role set. The function type is partial because each transition, t, uses one of the labels, l, in one of its uni-co–role sets, and thus, the function is not defined for (t, l'), where $l \neq l'$. The function type is surjective because of the way λ_4 is constructed.

- *The first letter, i.e., "P", specifies that the labels in synlabel(y) can be used by poly-roles. This description is formalized in the following relation type,*

  $$\tau_y \twoheadleftarrow (L^\tau_y \cup \overline{L}^\tau_y),$$

  which formalizes the above description by requiring that relation $\lambda_1(\tau, y)$ when considered in the inverse form to be a total, surjective function. Recall that relation $\lambda_1(\tau, y)$ specifies how the transitions that each has at least one role set using labels in L^τ_y interact through the labels used by either the role sets or the co-role sets of the transitions in τ. The relation type is total function because any label in $L^\tau_y \cup \overline{L}^\tau_y$ must be associated with exactly one transition that uses it in one of its role sets. The function type is surjective because of the way the λ_3 relation, and thus the λ_1 relation, are constructed.

- *The second letter, i.e., "E", requires that the labels in synlabel(y) can be used only by uni-co–roles. This description is formalized in the following relation type,*

  $$\overline{\tau}_y \twoheadrightarrow (L^\tau_y \cup \overline{L}^\tau_y),$$

  which formalizes the above description by requiring that relation $\lambda_2(\tau, y)$ is a total, surjective function. Recall that relation $\lambda_2(\tau, y)$ specifies how the transitions that each has at least one co-role set using labels in $\overline{L}^\tau_y$ interact through the labels used by either the role sets or the co-role sets of the transitions in τ. The relation type is total function because *any transition in $\overline{\tau}_y$ has exactly one uni-co–role set with a label in $L^{\tau}_y \cup \overline{L}^{\tau}_y$. The function type must be surjective because each label in $L^{\tau}_y \cup \overline{L}^{\tau}_y$ must be associated with a co-role set of a transition for the synchronization to make sense.*

The formalization of the relation types of other synchronization types are carried out in a similar manner as in Example 51 for the "PUES" synchronization type.

7.3.2 Integration with the Semantic Definition Schema

Using the characterization of synchronization types in Table 7.3, the definition of the SYN function used in Figure 7.4 can be specified. From Figure 7.4, function SYN was defined as,

$$\begin{aligned} SYN(\mathbb{T}) \;\; = \;\; \Big\{ T - T' | \, (T \in \mathbb{T}) \wedge (T' \subset T) \wedge \bigwedge_{1 \leq i \leq 4} SYN_i(T - T') \wedge \\ \forall X : 2^{T'} \cdot (X \neq \emptyset) \Rightarrow \neg \Big[\bigwedge_{1 \leq i \leq 4} SYN_i((T - T') \cup X) \Big] \Big\}, \end{aligned}$$

where $\mathbb{T}$ is a set of sets of enabled transitions, and $T - T'$ is a maximal subset of synchronizing transitions of $T \in \mathbb{T}$.

The definitions of the SYN_i, $1 \leq i \leq 4$, is then defined as follows,

$$SYN_i(\tau) \equiv \Big[\bigwedge_{y \in \mathbb{Y}} \lambda_i(\tau, y) \in \Lambda_i(\tau, y) \Big] \vee \Big[\bigwedge_{y \in \mathbb{Y}} \lambda_i(\tau, y) = \emptyset \Big],$$

where $\mathbb{Y}$ is the set of all synchronizers of the model.

As such, function SYN ensures that a set of enabled transitions of a model are only considered as a potential small step if the transitions satisfy the requirements of all synchronizers of the model, some of which are vacuously satisfied, when the transitions do not use the labels of a synchronizer. The implication in the second line of the definition of function SYN ensures that a set of synchronizing transitions cannot be extended by additional enabled transitions, while satisfying the synchronization requirements of the synchronizers in $\mathbb{Y}$.

Next, an example of how the above formalism works in the presence of more than one synchronizer is presented.

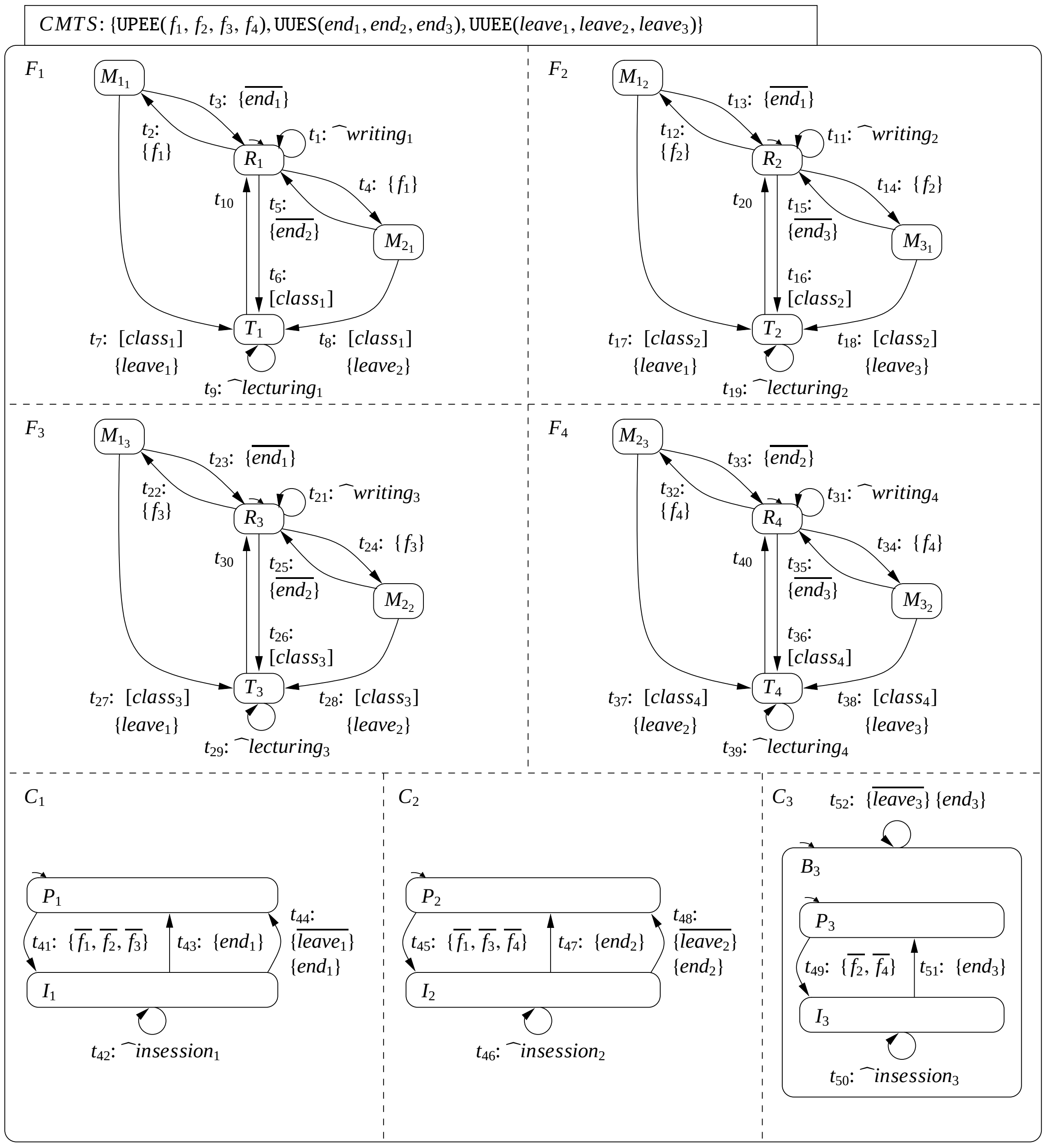


Figure 7.5: Modelling faculty members and their responsibilities, using synchronization (the same model as in Figure 6.1).

Example 52 *Figure 7.5 shows the same model as in Figure 6.1, which is responsible to coordinate the different activities of the faculty members of a department, as described in Example 6.1.*

Let us consider the model when it resides in the set of control states $\{\mathrm{M}_{1_1}, \mathrm{M}_{1_2}, \mathrm{M}_{1_3}, \mathrm{R}_4, \mathrm{I}_1, \mathrm{P}_2, \mathrm{P}_3\}$*. Also, let us consider the variable* class_1 *when it is true, meaning that the first member,* F_1*, of the first committee,* C_1*, needs to leave the meeting to deliver a lecture. The set of transitions* $\tau = \{\mathrm{t}_7, \mathrm{t}_{13}, \mathrm{t}_{23}, \mathrm{t}_{44}\}$ *satisfies the requirements of the three synchronizers in the model,* $\mathrm{UPEE}(\mathrm{f}_1, \mathrm{f}_2, \mathrm{f}_3, \mathrm{f}_4)$*,* $\mathrm{UUES}(\mathrm{end}_1, \mathrm{end}_2, \mathrm{end}_3)$*, and* $\mathrm{UUEE}(\mathrm{leave}_1, \mathrm{leave}_2, \mathrm{leave}_3)$*.*

The set of transitions τ *is synchronizing with respect to* $y_1 = \mathrm{UPEE}(\mathrm{f}_1, \mathrm{f}_2, \mathrm{f}_3, \mathrm{f}_4)$*, just because the transitions in* τ *do not use any of the labels in* $\{\mathrm{f}_1, \mathrm{f}_2, \mathrm{f}_3, \mathrm{f}_4\}$*; therefore, the corresponding* λ_1*,* λ_2*,* λ_3*, and* λ_4 *relations are empty and vacuously satisfy the conditions in SYN function.*

The set of transitions τ *is synchronizing with respect to* $y_2 = \mathrm{UUES}(\mathrm{end}_1, \mathrm{end}_2, \mathrm{end}_3)$ *because,*

i $\lambda_3(\tau, y_2) = \{((\mathrm{t}_{44}, \mathrm{end}_1), \mathrm{t}_{13}), ((\mathrm{t}_{44}, \mathrm{end}_1), \mathrm{t}_{23})\} \in [(\{\mathrm{t}_{44}\} \times \{\mathrm{end}_1\}) \leftarrow \{\mathrm{t}_{13}, \mathrm{t}_{23}\}]$,

ii $\lambda_4(\tau, y_2) = \{((\mathrm{t}_{13}, \mathrm{end}_1), \mathrm{t}_{44}), ((\mathrm{t}_{23}, \mathrm{end}_1), \mathrm{t}_{44})\} \in [(\{\mathrm{t}_{13}, \mathrm{t}_{23}\} \times \{\mathrm{end}_1\}) ⤀ \{\mathrm{t}_{44}\}]$,

iii $\lambda_1(\tau, y_2) = \{(\mathrm{t}_{44}, \mathrm{end}_1)\} \in [\{\mathrm{t}_{44}\} ⤖ \{\mathrm{end}_1\}]$*, and*

iv $\lambda_2(\tau, y_2) = \{(\mathrm{t}_{13}, \mathrm{end}_1), (\mathrm{t}_{23}, \mathrm{end}_1)\} \in [\{\mathrm{t}_{13}, \mathrm{t}_{23}\} \twoheadrightarrow \{\mathrm{end}_1\}]$.

The set of transitions τ *is synchronizing with respect to* $y_3 = \mathrm{UUEE}(\mathrm{leave}_1, \mathrm{leave}_2, \mathrm{leave}_3)$ *because,*

i $\lambda_3(\tau, y_3) = \{((\mathrm{t}_7, \mathrm{leave}_1), \mathrm{t}_{44})\} \in [(\{\mathrm{t}_7\} \times \{\mathrm{leave}_1\}) ⤗ \{\mathrm{t}_{44}\}]$,

ii $\lambda_4(\tau, y_3) = \{((\mathrm{t}_{44}, \mathrm{leave}_1), \mathrm{t}_7)\} \in [(\{\mathrm{t}_{44}\} \times \{\mathrm{leave}_1\}) ⤗ \{\mathrm{t}_7\}]$,

iii $\lambda_1(\tau, y_3) = \{(\mathrm{t}_7, \mathrm{leave}_1)\} \in [\{\mathrm{t}_7\} ⤖ \{\mathrm{leave}_1\}]$*, and*

iv $\lambda_2(\tau, y_3) = \{(\mathrm{t}_{44}, \mathrm{leave}_1)\} \in [\{\mathrm{t}_{44}\} ⤖ \{\mathrm{leave}_1\}]$.

The set of transitions $\tau' = \{\mathrm{t}_7, \mathrm{t}_{23}, \mathrm{t}_{44}\}$*, for example, is also a set of synchronizing transitions with respect to the three synchronizers, but does not satisfy the maximality requirements of the SYN function, and thus is not considered as potential small step.*

7.3.3 Discussion: Non-Hierarchical Computation

The computation of the potential small steps in Figure 7.4 is not entirely hierarchical, because the semantics of synchronization and priority is enforced at the root of the hierarchy tree of a model. This approach is different from the ones for BSMLs, in Figure 4.8 and Figure 4.10, where the hierarchical priority semantics are manifested in the computation of the attributes of all control states, including the intermediary control states in a hierarchy of control states. As mentioned earlier, similar computations as in Figure 4.8 and Figure 4.10 are not possible for SBSMLs because an enabled, high-priority transition may not be included in any potential small step because its synchronization requirements are not satisfied. Similarly, the synchronizing transitions of a potential small step cannot be computed hierarchically and incrementally, because, for example, choosing a pair of low-priority synchronizing transitions according to synchronization type `EE**` could preclude the possibility of replacing one of these transitions with a higher-priority transition that is higher in the hierarchy tree of the model.

The order of the application of the *PRI*, *MAX*, and *SYN* functions in line 1a in Figure 7.4 does matter: If the *PRI* function is applied first, then a potential small step might not be included because a potential small step including a higher-priority transition whose synchronization requirements cannot be satisfied is favoured against another potential small step including a lower-priority transition whose synchronization requirements can be satisfied at this snapshot. The converse, however, is not true: When the *SYN* and *MAX* functions are applied first, whichever potential small step that is favoured according to a priority semantics can be taken as the next small step (its synchronization requirements already hold).

Explicit Priority

The semantic definition mechanism in this section can be adapted to formalize the semantics of the Explicit priority semantics. For a BSML that follows this semantic option, a semantic definition schema can be created that considers the role of the Concurrency and Consistency semantic options in the same way as in the semantic definition schemas in Figure 4.8 and Figure 4.10, but has only a single *PRI* function at the root of the hierarchy tree, similar to the semantic definition schema in Figure 7.4.

Hierarchical Computation for Regular Models

An SBSML model is *regular* if it is only possible for a set of transitions that are pair-wise orthogonal to synchronize by a synchronizer. In a regular SBSML model, only transitions that have the same scope can synchronize via a synchronizer.

For a regular SBSML model, a hierarchical computation similar to the one for BSMLs can be defined. For example, when the Scope-Child priority semantics is chosen, a similar hierarchical computation as in Figure 4.10, on page 107, for BSMLs can be adopted for SBSMLs, except that the merge operator must be defined as below:

$$\begin{array}{rcl} \mathbb{T} \otimes T' & = & \{ (T_1 \cup T'') \mid T_1 \in \mathbb{T} \wedge T'' \subseteq T' \wedge \\ & & \forall X : 2^{(T' - T'')} \cdot X \neq \emptyset \Rightarrow [\neg SYN(T' \cup X) \vee \\ & & (\exists t' \in X \cdot \exists t \in (T_1 \cup T'') \cdot \neg(t \perp t') \wedge \neg((t \lightning t') \vee (t' \lightning t)))] \}. \end{array}$$

The set of synchronizers $\mathbb{Y}$, which is used by function SYN, is also used for the semantic definition of regular models, but here $\mathbb{Y}$ is only the set of synchronizers in the current control state, as opposed to the set of all synchronizers of the model. Set $\mathbb{Y}$ need not keep track of any other synchronizers in the model because the role sets and the co-role sets of a transition lower in the hierarchy tree of the model does not have any effect on the synchronization of the transitions at the higher level, because of the model being regular.

The first line in the above definition adds new enabled transitions, whose scopes is the current level of the hierarchy tree, to a set of transitions passed from a lower level of the hierarchy tree; the second line in the above definition ensures that a maximal set of synchronizing transitions of the current level are chosen, unless adding new transitions create an inconsistent set of transitions, as checked in the third line.

Similarly, a hierarchical computation for the Scope-Parent priority semantics in SBSMLs can be defined for regular SBSMLs, based on the computation in Figure 4.8, on page 103.

Interestingly, when the No Priority priority semantics is chosen, however, even if an SBSML is regular, a hierarchical computation of small steps cannot be achieved. Such a computation is inherently non-hierarchical: A set of synchronizing transitions that have a high (low) scope do not have any precedence over a set of transitions in a lower (higher) scope. Thus, all possible combinations of synchronizing transitions must be considered as potential small steps.

7.4 Transformation Schemes and Their Verification

In this section, the transformation schemes for transforming syntactic constructs and semantic options into synchronization mechanisms in SBSMLs are formally described. Furthermore, the correctness of these formal transformations are proved. The informal version of these transformations were presented in Section 6.4.

In this section, the term *original model* refers to the source model in a transformation, which can be either an SBSML model or an extended SBSML model annotated with additional syntactic constructs that are the subject of the transformation scheme. The term *new model* refers to the SBSML model resulting from applying the transformation algorithm to an original model.

At the end of the section, in Section 7.4.6, a discussion of how the synchronization schemes affect the well-formedness of a new model is presented. It is shown that the well-formedness of the resulting new models do not undermine the correctness of the transformation schemes.

7.4.1 Multi-source, Multi-destination Transitions

This section presents a transformation scheme for converting a BSML model that uses multi-source, multi-destination transition syntax to an equivalent BSML model in which a multi-source, multi-destination transition is replaced with regular transitions. First, a few needed definitions are presented, followed by a brief description of the common semantics of SBSMLs when they support multi-source, multi-destination transitions.

Preliminaries

A multi-source, multi-destination, *mt*, as opposed to a regular transition, has a *set* of source control states and a *set* of destination control states. Therefore, in this section, $src(mt)$ and $dest(mt)$ return sets of control states, for both regular and multi-source, multi-destination transitions.

Definition 7.1 *A multi-source, multi-destination transition, mt, is well-formed, if*

i *the set of control states* $src(mt)$ *are pairwise orthogonal;*

ii *the set of control states* $dest(mt)$ *are pairwise orthogonal; and* *iii* $\bigcup_{s \in src(mt)}(children^*(s)) \cap \bigcup_{d \in dest(mt)}(children^*(d)) = \emptyset$.

The rational for well-formedness criteria i and ii is clear: A model cannot reside in more than one of the children of any of its *Or* control states. Criteria iii disallows multi-source, multi-destination transitions that have a kind of loop, similar to a regular self transition. The discussion on page 246 discusses the difficulties of dealing with a special subclass of these transitions.

Henceforth, by a multi-source, multi-destination transition, I mean a well-formed multi-source, multi-destination transition. A model is required to have only well-formed multi-source, multi-destination transitions.

A multi-source, multi-destination transition, mt, is *balanced* if $|src(mt)| = |dest(mt)|$, otherwise it is *imbalanced*. The usual functions and relations used for regular transitions are also used for multi-source, multi-destination transitions; e.g., $gen(mt)$ specifies the set of generated events of mt and $rolesets(mt)$ specifies the set of role sets of mt.

For a multi-source, multi-destination transition, mt, its *highest scope*, denoted by $hs(mt)$, is the highest control state, h, in the hierarchy tree such that there exists $s \in src(mt)$ and $d \in dest(t)$ and $lca(s, d) = h$. Similarly, the *lowest scope* of mt, denoted by $ls(t)$, is the lowest control state, l, in the hierarchy tree such that there exists $s \in src(mt)$ and $d \in dest(t)$ and $lca(s, d) = l$. For a regular transition, t, $hs(t) = ls(t) = lca(src(t), dest(t))$.

The semantics of an SBSML that supports multi-source, multi-destination transitions is the same as the semantics of the SBSML without these transitions except for the Small-Step Consistency and the Preemption semantic sub-aspects, i.e., the definitions of the "$\perp$" and "$\lightning$" relations, and the hierarchical priority semantics, i.e., the SCOPE-PARENT and the SCOPE-CHILD semantic options. Next, a new version of these concepts is presented so that they accommodate for multi-source, multi-destination transitions.

Definition 7.2 *Two transitions, mt and mt', where either could be a multi-source, multi-destination transition, are* orthogonal, *denoted by $mt \perp mt'$, if: (i) for any two control states $s \in src(mt)$ and $s' \in src(mt')$, $s \perp s'$; and (ii) for any two control states $d \in dest(mt)$ and $dest' \in src(mt')$, $d \perp d'$.*

Definition 7.3 *A transition, mt, is an* interrupt *for transition, mt', where either transitions could be a multi-source, multi-destination transition, if:*

i *For any two control states $s \in src(mt)$ and $s' \in src(mt')$, $s \perp s'$; and*

ii *For any two destination control states $d \in dest(mt)$ and $d' \in dest(mt')$, one of the following conditions is true:*

(a) *d' is orthogonal with all control states in $src(t)$ and d is not orthogonal with any transition in $src(t) \cup src(t')$; or*

(b) *Neither d is orthogonal with any control state in $src(t)$ nor d' is orthogonal with any control state in $src(t')$, but $d \in children^{+}(d')$.*

Because all source control states of a multi-source, multi-destination transition are orthogonal, as well as, all its destination control states, it is not possible for a pair of multi-source, multi-destination transitions to be both orthogonal and one being an interrupt for the other.

Definition 7.4 *For a pair of transitions mt and mt', where either transition could be a multi-source, multi-destination transitions, mt has a* higher priority *than mt', i.e., $pri(mt) > pri(mt')$, according to the* Scope-Parent *if $ls(mt)$ is higher than $ls(mt')$ in the hierarchy tree of the model. Similarly, $pri(mt) > pri(mt')$ according to the* Scope-Child *if $hs(mt)$ is lower than $hs(mt')$ in the hierarchy tree of the model.*

Formal Transformation and Correctness

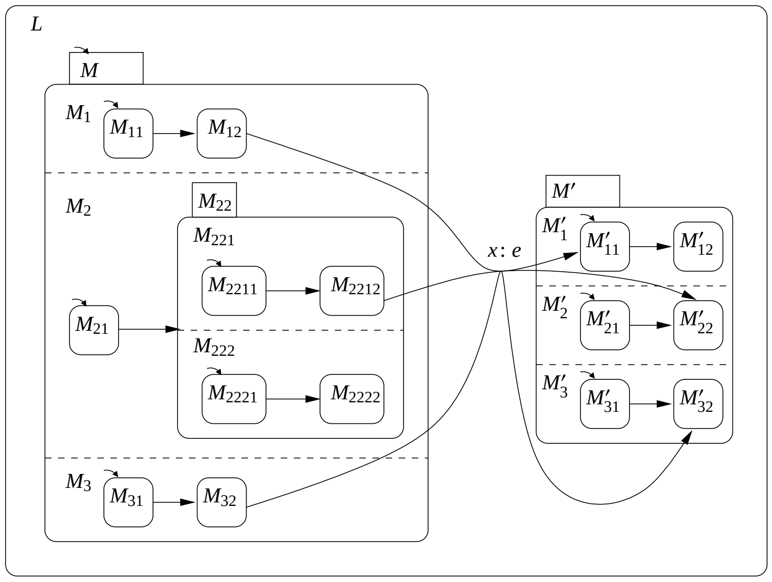
Based on whether a multi-source, multi-destination transition, *mt*, is balanced or imbalanced, a formal transformation scheme to turn it into a set of regular, synchronizing transitions is presented.

Case 1: *mt* is balanced. The transformation of *mt* to a set of regular transitions, denoted by $transtoregular(mt)$, is achieved by Algorithm 1. The algorithm uses variables T and f, which have types: set of transitions and bijective functions between transitions, respectively.

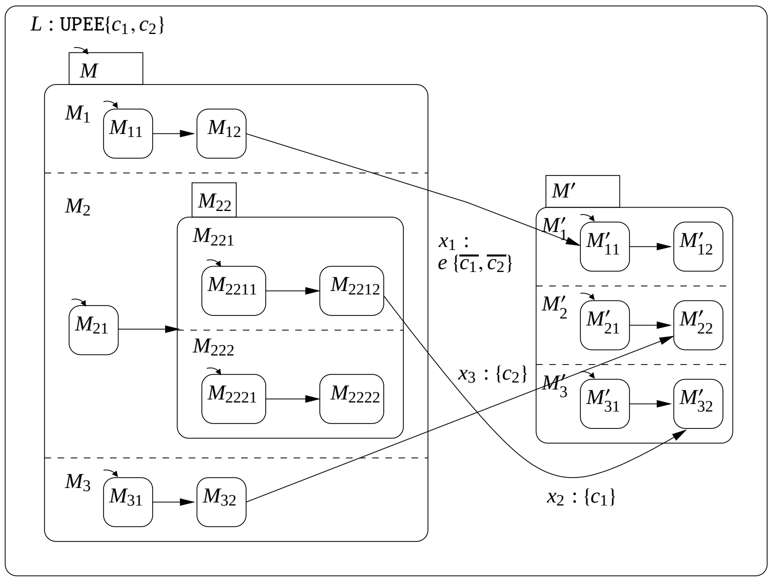
Example 53 *The SBSML model in Figure 7.6 (a) has a balanced multi-source, multi-destination transition* x. *Applying the $transformtoregular(\mathrm{x})$ results in the SBSML model in Figure 7.6 (b), with x_1 being the representative transition, as described in Algorithm 7.5.*

Algorithm 1: *transformtoregular*(*mt*).

```
Input: $mt$
Result: A balanced multi-source, multi-destination transition $mt$ is replaced by $|src(mt)|$
        regular transitions.
if $mt$ is a balanced transition then
    $T := \emptyset$;
    $f := \emptyset$;
    Define any bijective mapping $f : src(mt) \rightarrowtail\!\!\!\twoheadrightarrow dest(mt)$ such that,
    if SCOPE-PARENT semantics chosen then
        There exists $(s_r, s'_r) \in f$ such that $lca(s_r, s'_r) = ls(mt)$;
    end
    else if SCOPE-CHILD semantics chosen then
        There exists $(s_r, s'_r) \in f$ such that $lca(s_r, s'_r) = hs(mt)$;
    end
    foreach $(s_i, s'_i) \in f$ do
        Create a new, regular transition $t_i$ in the model such that $src(t_i) = s_i$ and
        $dest(t_i) = s'_i$;
        $T := T \cup \{t_i\}$;
    end
    Pick a transition $t_r$ from $T$, as a representative transition such that if the SCOPE-PARENT
    or SCOPE-CHILD priority semantics is chosen, $src(t_i) = s_r$ and $dest(t_i) = s'_r$;
    Modify $t_r$ so that $asn(t_r) = asn(t)$, $gen(t_r) = gen(t)$, $trig(t_r) = trig(t)$, $gc(t_r) = gc(t)$,
    $rolesets(t_r) = rolesets(t)$, and $corolesets(t_r) = coroleset(t)$;
    Modify $t_r$ so that it has a new co-role set:
    $corolesets(t_r) = corolesets(t_r) \cup \{\overline{ct_1}, \overline{ct_2}, \cdots, \overline{ct_{n-1}}\}$, where $n = src(t) = dest(t)$, and
    the new co-role set consists of new co-labels not already used in the model;
    foreach $t_j \in T$ $(t_j \neq t_r)$ do
        Modify it so that it has a new, distinct singleton role set $rolesets(t_j) = \{\{ct_j\}\}$;
    end
    Assign the synchronizer UPEE$(ct_1, ct_2, \cdots, ct_{n-1})$ to the control state that, according to
    the well-formedness conditions, can synchronize the transitions in $T$ according to the
    new co-role set and the role sets;
    Add the transitions in $T$ to the model, and Remove $mt$ from the model;
end
```



(a)



(b)

Figure 7.6: Transforming multi-source, multi-destination transitions into regular transitions.

Before proving the soundness of the transformation of *transformtoregular*(mt), the following three lemmas need to be proven.

Lemma 7.1 *For two transitions, mt_1 and mt_2, which each can be either a balanced or a regular transition, they are orthogonal if and only if (iff) all pairs of regular transitions $rt_1 \in transtoregular(mt_1)$ and $rt_2 \in transtoregular(mt_2)$ are orthogonal.*

Proof Idea. If mt_1 and mt_2 are not orthogonal, then there is a pair of control states $s_1 \in src(mt_1)$ and $s_2 \in src(mt_2)$ such that $\neg(s_1 \perp s_2)$, and/or a pair of control states $d_1 \in dest(mt_1)$ and $d_2 \in dest(mt_2)$ such that $\neg(d_1 \perp d_2)$. In any of these cases, because of function f, defined on line 4 in Algorithm 1, being bijective, there would exist a pair of transitions $rt_1 \in transtoregular(mt_1)$ and $rt_2 \in transtoregular(mt_2)$ that are not orthogonal. Conversely, if mt_1 and mt_2 are orthogonal, by definition, all pairs of control states $s_1 \in src(mt_1)$ and $s_2 \in src(mt_2)$ are orthogonal, as well as, all pairs of control states $d_1 \in dest(mt_1)$ and $d_2 \in dest(mt_2)$. Thus, the induced transitions by bijective function f are also all orthogonal because their source control states and their destination control states are pairwise orthogonal. $\square$

Lemma 7.2 *For two transitions, mt_1 and mt_2, which each can be either a balanced or a regular transition, transition mt_1 is an interrupt for mt_2, iff for all pairs of regular transitions $rt_1 \in transtoregular(mt_1)$ and $rt_2 \in transtoregular(mt_2)$, rt_1 is an interrupt for rt_2.*

Proof Idea. If mt_1 is not an interrupt for mt_2, then either condition i or one of the conditions ii(a) or ii(b) in Definition 7.3 does not hold. In any of these cases, a pair of transitions $rt_1 \in transtoregular(mt_1)$ and $rt_2 \in transtoregular(mt_2)$ are created by the bijective function f, on line 4 of Algorithm 1, such that rt_1 is not an interrupt for rt_2, according to the same relationship between control states that announces mt_1 not being an interrupt for mt_2. Conversely, if mt_1 is an interrupt for mt_2, there is no such a pair of transitions $rt_1 \in transtoregular(mt_1)$ and $rt_2 \in transtoregular(mt_2)$ such that rt_1 is not an interrupt for rt_2. $\square$

Lemma 7.3 *At a snapshot of a model, a balanced transition mt has the highest priority iff all regular transitions $rt \in transtoregular(mt)$ could be included together in a potential small step.*

Proof Idea. For the Negation of Triggers priority semantics, the above claim is true because the resulting representative transition in Algorithm 1 has the same trigger as the original multi-source, multi-destination transition. The other regular transitions would also have the highest priority, by virtue of being taken only if the representative transition is enabled. For the Scope-Parent and Scope-Child priority semantics, the lowest scope or highest scope control state of *mt* determines its priority precedence, respectively. Also, since the priority of the representative transition of *mt*, determined in lines 4-9 of Algorithm 1, also similarly depends on the lowest scope or highest scope control state of *mt*, *mt* has a high priority at a snapshot iff its representative transitions and other corresponding regular transitions have a high priority. □

Proposition 7.4 *Transformation in Algorithm 1 is sound.*

Proof Idea. For a balanced transition, *mt*, if all transitions in $T = transtoregular(mt)$ are executed together, their effect would be the same as executing *mt* because: (i) there is one transition in T that adopts $asn(mt)$ and $gen(mt)$; and (ii) the destination control states that *mt* and the transitions in T arrive at are the same. Thus, it remains to prove that if *mt* belongs to a potential small step τ, in the original model, it is also the case that in the new model, there is a similar potential small step $\tau' = \tau - \{t\} \cup T$. This can be proven because *t* is enabled iff the representative transition of T is, and because of lemmas 7.1, 7.2, and 7.3. Therefore, transformation in Algorithm 1 is sound. □

Algorithm 2 repeatedly applies Algorithm 1 to a set of transitions.

```
Algorithm 2: $transformall(MT)$.
  Input: $MT$
  Result: All balanced multi-source, multi-destination transitions in $MT$ are replaced by
          their corresponding regular transitions.
1 foreach $mt \in MT$ do
2   | $transformtoregular(mt)$;
3 end
```

Proposition 7.5 *Replacing all balanced transitions of a model using Algorithm 2 results in a new model with the same behaviour as the original one.*

Proof Idea. By Proposition 7.4, replacing a single multi-source, multi-destination transition is sound. Lemmas 7.1, 7.2, and 7.3 ensure that replacing more than one multi-source, multi-destination transitions does not add to or remove from the behaviour of the original model. Thus, the new model and the old model have the same behaviour. $\square$

Case 2: *mt* is imbalanced. In this case, in order to be able to create similar regular transitions as in Algorithm 1, extra *dummy* control states need to be created. First, some necessary definitions and notation are introduced.

Given an SBSML model and its set of transitions, T, $MT_\diamond \subseteq T$ denotes the set of imbalanced transitions in T, and $S_\diamond = \bigcup_{mt \in MT_\diamond}(src(mt) \cup dest(mt))$. For a control state, $s \in S_\diamond$, its *maximum incoming shortage*, denoted by $maxin(s)$, is

$$maxin(s) \quad = \quad \max\{|src(mt)| - |dest(mt)| \mid mt \in MT_\diamond \cdot s \in dest(mt)\}.$$

Similarly, its *maximum outgoing shortage*, denoted by $maxout(s)$, is

$$maxout(s) \quad = \quad \max\{|dest(mt)| - |src(mt)| \mid mt \in MT_\diamond \cdot s \in src(mt)\}.$$

Function $maxinout(s)$ specifies the maximum of $maxin(s)$ and $maxout(s)$. The $maxinout(s)$ is used to determine the number of dummy control states that need to be created for a control state s.

For each imbalanced transition, *mt*, it suffices to create dummy *Basic* control states for one of its source or destination control states, *s*, based on which set has a smaller size. Once these dummy control states are created, the set of source or destination control states of *mt* is adjusted to use them to create balanced transition. Using the value of $maxinout(s)$, all other imbalanced transitions with one of their source or destination control state being *s* can also be adjusted to become balanced simultaneously.

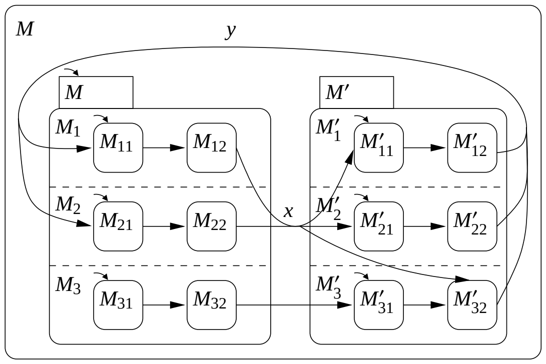
Algorithm 3 uses the above idea to turn all imbalanced transitions of a model into balanced transitions. Its input is the set of imbalanced transitions of a model, $MT_\diamond$, together with the set of control states, $S_\diamond$, which are the control states that could potentially benefit from introducing dummy control states. The regular transitions and balanced transitions remain unaffected by this algorithm, however, the overall hierarchy tree of the model changes. The algorithm uses variables $\mathcal{MT}$ and $\mathcal{S}$ to store the values of $MT_\diamond$ and $S_\diamond$, respectively, before the transformation starts, because $MT_\diamond$ and $S_\diamond$ change as computation proceeds. A few other temporary variables are used in the algorithm, with obvious roles.

```
Algorithm 3: balance($MT_\diamond, S_\diamond$).
   Input: $MT_\diamond, S_\diamond$
   Result: $\mathcal{MT} = \mathcal{MS} = MT_\diamond = S_\diamond = \emptyset$
 1 $\mathcal{MT} := MT_\diamond$;
 2 $\mathcal{S} := S_\diamond$;
 3 while $\mathcal{S} \neq \emptyset$ do
 4     choose any $s$ from $\mathcal{S}$;
 5     Let $MT_s := \{mt \mid mt \in \mathcal{MT} \wedge (s \in src(mt) \vee s \in dest(mt))\}$;
 6     Create a new And control state $s_{new}$ such that $children(s_{new}) = \{s\} \cup new$, where
       $new = \{n_1, \cdots, n_{maxinout(s)}\}$ is a set of Basic control states; if $s$ is a default control state,
       then $default(parent(s)) = s_{new}$ ;
 7     foreach $mt \in MT_s$ do
 8         if ($s \in src(mt)$ and $|src(mt)| < |dest(mt)|$) then
 9             $src(mt) := src(mt) \cup \{n_1, \cdots, n_{|dest(mt)|-|src(mt)|}\}$;
10         end
11         if ($s \in dest(mt)$ and $|src(mt)| > |dest(mt)|$) then
12             $dest(mt) := dest(mt) \cup \{n_1, \cdots, n_{|src(mt)|-|dest(mt)|}\}$;
13         end
14     end
15     $\mathcal{S} := \mathcal{S} - \bigcup_{mt \in MT_s}(src(mt) \cup dest(mt))$;
16     $\mathcal{MT} := \mathcal{MT} - MT_s$;
17 end
```

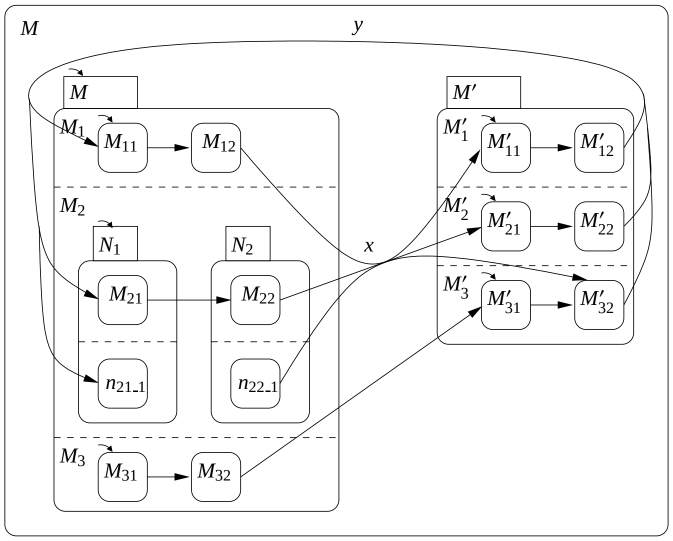
Example 54 *The SBSML model in Figure 7.7(a) is the same as model in Figure 6.3(a), on page 198. In this model, $MT_\diamond = \{\mathrm{x}, \mathrm{y}\}$. The SBSML model in Figure 7.7(b) is the same as the model in Figure 7.7(a) except that the imbalanced transitions in the model (a) are replaced by balanced transitions in model (b), via applying the "balance" algorithm, specified in Algorithm 3, to the set of imbalanced transitions of model (a).*

The model in Figure 6.3(b), on page 198, has a different hierarchy tree than the model in Figure 7.7(b). In that example, for the sake of exposition, an equivalent model is presented that is not based on any particular algorithm.

Next, the proof of the soundness of the transformation in Algorithm 3 is presented.



(a)



(b)

Figure 7.7: Balancing the imbalanced transitions, using the *balance* algorithm.

Lemma 7.6 *Given the set of imbalanced transitions of an SBSML model, $MT_\diamond$, and the set of corresponding control states of $MT_\diamond$, $S_\diamond$, after executing "balance($MT_\diamond, S_\diamond$)", $MT_\diamond = S_\diamond = \emptyset$.*

Proof Idea. Algorithm 3 considers all imbalanced transitions because of the while loop on line 4 and the for loop on line 7. Otherwise, if an imbalanced transition is not considered, it means that its source and destination control states do not belong to $S_\diamond$, before the algorithm is executed, which is not possible by the definition of $S_\diamond$. The next claim is that for each $mt \in MT_\diamond$, the algorithm balances this transition. This can be proven by inspecting the body of the for loop on line 7. If mt is imbalanced because the size of $src(mt)$ being small, it becomes balanced by adjusting $src(mt)$ so that its size is exactly the same as $dest(mt)$, as performed by the if statement on line 8. Otherwise, if mt is imbalanced because the size of $dest(mt)$ being small, it becomes balanced by adjusting $src(mt)$, as performed by the if statement on line 11. Lastly, the Algorithm 3 obviously terminates: Both the while and the for loop iterate over finite sets, whose sizes decrease as computation progresses. As such, after executing the Algorithm 3, $MT_\diamond = S_\diamond = \emptyset$.
□

Proposition 7.7 *Given the set of imbalanced transitions of an SBSML model, $MT_\diamond$, and the set of corresponding control states of $MT_\diamond$, $S_\diamond$, after executing "balance($MT_\diamond, S_\diamond$)", the resulting model has the same behaviour as the original model.*

Proof Idea. For an imbalanced transition, mt, and its corresponding balanced transition, mt', in the new model after applying Algorithm 3, mt is enabled iff mt' is. Also, the effect of both transitions are the same because mt' is only different from mt either in its source or its destination control states. Thus, it remains to show that if two transitions mt_1 and mt_2 could have been taken together in a small step in the old model, their corresponding transitions mt'_1 and mt'_2 could also be taken together in a similar small step. This can be proven by the fact that: (i) the algorithm preserves the comparative priority precedence of the transitions in the old model when they are modified to their corresponding transitions in the new model; (ii) the algorithm preserves the "$\perp$" relation for transitions; i.e., $mt_1 \perp mt_2 \Leftrightarrow mt'_1 \perp mt'_2$; and (iii) the algorithm preserves the "$\lightning$" relation; i.e., $mt_1 \lightning mt_2 \Leftrightarrow mt'_1 \lightning mt'_2$. Claim (i) above is true because the addition of dummy control states and the adjustments of the source and destination control states of imbalanced transitions neither change the trigger of the imbalanced transitions (for the Negation of Triggers priority semantics) nor their relative scopes (for the Parent-Scope and Child-Scope priority semantics).

The relative scope of transitions do not change because the addition of a new *And* control state, on line 6, only adds an additional level in the hierarchy tree without assigning any transition to that level of hierarchy. Claim (ii) above is true because if a pair of control states s_1 and s_2 are orthogonal in the old model, their corresponding control states in the new model, s'_1 and s'_2, are also orthogonal: If s'_1 and/or s'_2 have become the children of new *And* control state(s), through the execution of line 6 of the algorithm, then they would remain orthogonal because an additional *And* control state does not affect their being orthogonal; if s'_1 and/or s'_2 have not been affected by line 6, they would obviously remain orthogonal. Thus, the "$\perp$" relation is preserved. A similar argument is in order for the "$\not\lightning$" relation. Therefore, applying the *balance* algorithm to a model does not change its behaviour. □

Proposition 7.8 *For an SBSML model M_1, if applying the "balance" algorithm (Algorithm 3) to its imbalanced transitions yields model M_2, and applying the "$transformall$" algorithm (Algorithm 2) to the balanced transitions of M_2 yields M_3, the behaviour of M_1, M_2, and M_3 are all the same.*

Proof Idea. By Proposition 7.5 and Proposition 7.7. □

Self Multi-Transitions

A multi-source, multi-destination transition, *mt*, is a *self multi-transition* if

$$\bigcup_{s \in src(mt)}(children^*(s)) \cap \bigcup_{d \in dest(mt)}(children^*(d)) \neq \emptyset.$$

In a self multi-transition, at least one of the source and one of the destination control states are ancestrally related, creating a kind of loop when presenting *mt* graphically.

Algorithm 1, on page 238, which transforms a balanced multi-source, multi-destination transition into a set of orthogonal transitions, can be applied to a self multi-transition to create a balanced multi-source, multi-destination transition. For example, the transformation of transition *mt* in model in Figure 7.8(a) results in the model in Figure 7.8(b).

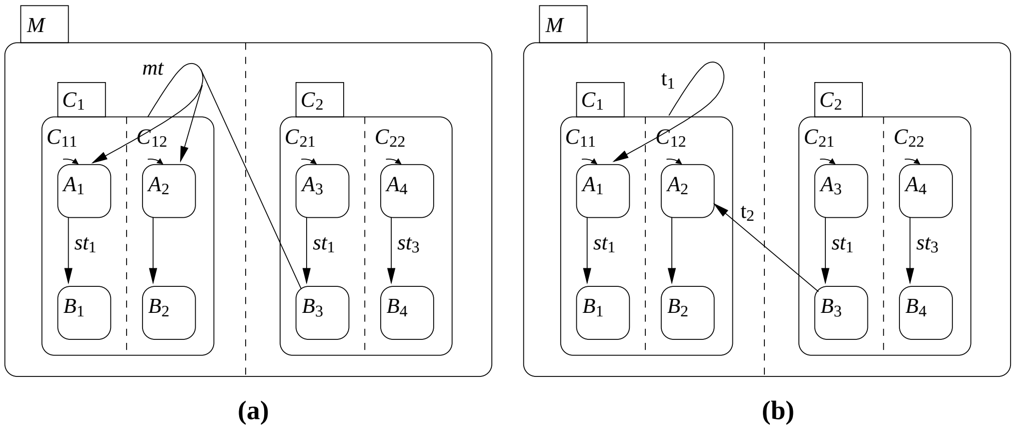


Figure 7.8: Applying algorithm *transformtoregular* to a self multi-transition.

Imbalanced Self Multi-Transitions. If a self multi-transition is imbalanced, however, the *balance* algorithm, in Algorithm 3, cannot be used, when a model is specified in the Scope-Parent or the Scope-Child priority semantics. The example models in Figure 7.9 demonstrate the problem. The model in Figure 7.9(a) has one self multi-transition, namely, x. Applying the *balance* algorithm to this model results in the model in Figure 7.9(b). While in the first model, according to the Scope-Parent priority semantics, $pri(x) = pri(y)$, in the second model $pri(y) > pri(x)$. If the No Priority priority semantics is chosen, however, the *balance* algorithm could be applied soundly to self multi-transitions.

In general, there does not seem to exist any transformation scheme to balance imbalanced self multi-transitions without changing the relative priority of the transitions. This is because, as opposed to non-self multi-transitions, the new *And* control state that is created to adjust the imbalance of a self multi-transition surrounds all source and destination control states of the new balanced transition, making the transitions to have a lower priority (in the case of Scope-Parent semantics) or a higher priority (in the case of Scope-Child semantics). For example, in the model in Figure 7.9(b), the *And* control state P_2 surrounds all control states in the source and destination of x, as opposed to control state P_1 in the model in Figure 7.9(a).

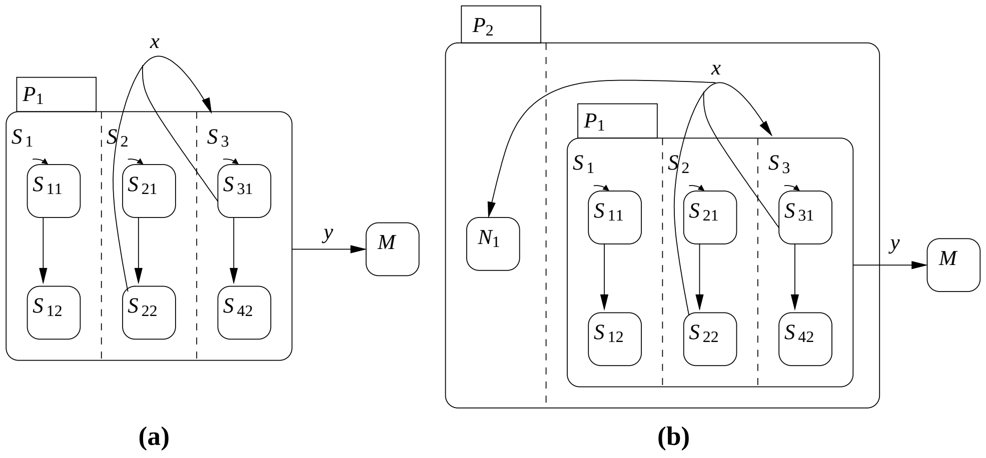


Figure 7.9: Applying the *balance* algorithm to a model that has an in-out self transiton.

7.4.2 BSML Semantic Options

This section introduces three transformation schemes that each represents how a semantic option of one of the Concurrency, Small-Step Consistency, and Preemption semantic aspects can be modelled in SBSMLs using the other semantic option of the aspect. It also presents the proofs of the correctness of these transformation schemes.

Concurrency

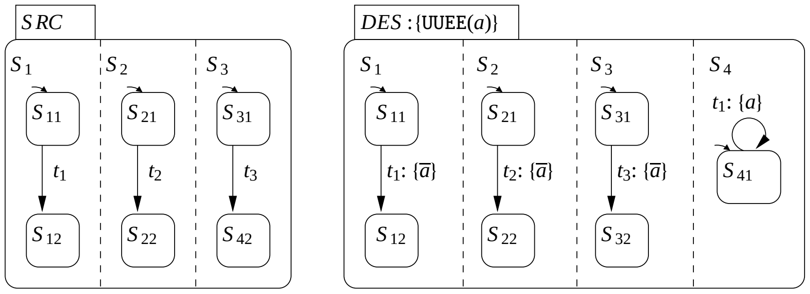
The transformation scheme presented in this section can convert a BSML model that is specified in the Single concurrency semantics to an equivalent SBSML model that is specified in the Many concurrency semantics. The algorithm is designed in a way that it is possible to make parts of an SBSML model to use the Single concurrency semantics while other parts use the Many concurrency semantics.

Algorithm 4 shows how a compound control state, s, of an SBSML, or a BSML, model can be modified so that each small step at most has one transition, t, such that $scope(t) \in children^*(s)$.

For a given set of transitions, T, of an SBSML model, and a control state, s, T_s denotes the set of transitions, $t \in T$, such that $scope(t) \in chidlren^*(s)$.

Example 55 *Figure 7.10 shows two SBSML models that are similar to the models in Figure 6.4, but use different names for control states and transitions. Applying algorithm “manytoone” to*

```
Algorithm 4: manytoone(s).
Input: s
Result: At most one transition whose scope is a child of s, or s itself, can be included in a
        small step.
1 Create a new Basic control state, single;
2 Create a new transition, $t_{single}$, such that $src(t_{single}) = single$, $dest(t_{single}) = single$;
3 Create a new And control state, $s_{new}$, such that $children(s_{new}) = \{s\} \cup \{single\}$;
4 foreach $t \in T_s$ do
5     $rolesets(t) = rolesets(t) \cup \{a\}$, where $a$ is a new label in the model;
6 end
7 Set $corolsets(t_{single}) = \{\{\overline{a}\}\}$;
8 Assign synchronizer UUEE($a$) to $s_{new}$;
```



(a) **(b)**

Figure 7.10: The effect of applying Algorithm 4.

control state SRC *in the model in Figure 7.10(a) results in the SBSML model in Figure 7.10(b) that can execute at most one of the transitions of the original model in each of its small step.*

If the input to the *manytoone* algorithm is the root control state of an SBSML model, then the model behaves as if it was specified in the SINGLE concurrency semantics, instead of the MANY concurrency semantics.

A possible undesired side effect of the *manytoone* algorithm is that it disables the role of some, or all, of the synchronizers in the original model, depending on which control state it is applied to. This is because, within the scope of the control state that the algorithm is applied to, at most one transition of the original model can be executed in a small step, precluding the possibility of synchronization.

Proposition 7.9 *Given an SBSML model, its set of transitions, T, and one of its compound control states, s, applying the "manytoone" algorithm, i.e., Algorithm 4, to s results in a new model whose behaviour is different from the original model in that, for each potential small step τ in the original model, there is a corresponding potential small step τ' in the new model such that one of the following statements is true:*

i $\tau = (\tau' \cap T)$*, and there does not exist any* $t \in \tau$ *such that* $t \in T_s$*; or*

ii $(\tau' \cap T) \subset \tau$*, and for each* $t \in (\tau - \tau') \cap T_s$*, there exists exactly one transition* $t' \in (\tau' \cap T_s)$*, and there does not exists a transition* $t'' \in (\tau \cap T_s)$ *such that* $pri(t'') > pri(t')$*.*

Proof Idea. In the case i above, if the set of transitions, τ, is a potential small step in the original model, the same set of corresponding transitions, τ', in the new model is a potential small step, because: (a) clearly, if a transition $t \in \tau$ is enabled, it is also enabled in the new model; and (b) if two enabled transitions $t_1, t_2 \in \tau$ satisfy the structural semantics of the SBSML semantics that the original model is specified in, they also satisfy the structural semantics in the new model, because, first, t_1 and t_2 do not belong to T_s, unless $|\tau| = 1$; and second, their small-step consistency, preemption, priority, and synchronization interrelationships are not affected by the *manytoone* algorithm. Lastly, τ' cannot have an extra transition that τ does not have. If there exists a $t \in (\tau' \cap T) - \tau$, it should be the case that $t \notin T_s$. Also, it should be the case that τ is not a maximal set of transitions that can be taken in the original model as a potential small step, which cannot be true by the definition of SBSML semantics.

In the case ii above, it is effectively required that $(\tau' \cap T) = \tau - T_s \cup \{t'\}$, such that $t' \in T_s$ cannot be replaced with a higher-priority transition $t'' \in T_s$. Using a similar argument as for the case i above, it can be shown that all transitions in τ that do not belong to T_s also belong to a potential small step τ' in the new model. It remains to show that exactly one of the transitions in $\tau \cap T_s$ can be included in τ', which is clearly the case because of the transformation in algorithm *manytoone*: If $|\tau' \cap T_s| > 1$, it means that the synchronizer $\texttt{UUEE}(a)$ is synchronizing according to the semantics of $\texttt{UUSE}$, which cannot be the case. Lastly, t' cannot be replaced with a higher-priority transition t'', because of the *PRI* function in the SBSML semantic definition schema in Figure 7.4, described on page 220. □

Small-Step Consistency

The transformation scheme presented in this section can convert a BSML model that is specified in the ARENA ORTHOGONAL small-step consistency semantics to an equivalent SBSML model that is specified in the SOURCE/DESTINATION ORTHOGONAL small-step consistency semantics. The algorithm is designed in a way that it is possible to make parts of an SBSML model to use the ARENA ORTHOGONAL small-step consistency semantics while other parts use the SOURCE/DESTINATION ORTHOGONAL small-step consistency semantics. First, some notation are presented, before presenting the transformation algorithm.

For a set of transitions, T, and one of its transitions, $t \in T$, the set of *arena conflicting* transitions with t, denoted by function $ac(t)$, is the set of all transitions in T, such that,

$$t' \in ac(t) \Leftrightarrow (t' \in T) \wedge (t \perp t') \wedge \neg(arena(t) \perp arena(t')).$$

For a set of transitions T, $allac(T)$ denotes $\bigcup_{t \in T} ac(t)$.

Given a compound control state, s, of an SBSML, or a BSML, model, Algorithm 5 specifies how transitions whose scopes are within s, i.e., the transitions in T_s, can be changed so that they follow the ARENA ORTHOGONAL semantic option, instead of the SOURCE/DESTINATION ORTHOGONAL semantic option.

If the input to the *srcdesttoarena* algorithm is the root control state of an SBSML model, then, the model effectively behaves as if it was specified in an ARENA ORTHOGONAL small-step semantics, instead of a SOURCE/DESTINATION ORTHOGONAL semantics.

Example 56 *Figure 7.11 shows two SBSML models that are similar to the ones in the transformation on 200, except that the unnecessary renaming in Figure 7.11(b) are avoided here. Applying algorithm "srcdesttoarena" to control state* SRC *in the model in Figure 7.11(a), results in the SBSML model in Figure 7.11(b). In the SBSML model in Figure 7.11(b),*

$$\begin{gathered} ac(\mathrm{t}_2) = \{\mathrm{t}_4, \mathrm{t}_5\}, \\ ac(\mathrm{t}_4) = \{\mathrm{t}_2, \mathrm{t}_5\}, \\ ac(\mathrm{t}_5) = \{\mathrm{t}_2, \mathrm{t}_4\}, \textit{ and} \\ allac(\{\mathrm{t}_1, \mathrm{t}_2, \mathrm{t}_3, \mathrm{t}_4, \mathrm{t}_5\}) = \{\mathrm{t}_2, \mathrm{t}_4, \mathrm{t}_5\}. \end{gathered}$$

The SBSML model in Figure 7.11(b) is different from the SBSML model in Figure 7.11(a)

```
Algorithm 5: srcdesttoarena($s$).
  Input: $s$
  Result: A pair of transitions whose scopes belong to $children^*(s)$ can be included in a
          small step if their arenas are orthogonal.
1 Based on the size of $T_c = allac(T_s)$, create a set of new labels $A = \{a_1, \cdots, a_n\}$, where
  $n = |T_c|$;
2 Define any bijective mapping $f : T_c \rightarrowtail\!\!\!\!\twoheadrightarrow A$;
3 Create a set of new Basic control state, $B = \{B_1, \cdots, B_n\}$;
4 Create a set of new self transitions $T_B = \{t_{B_1}, \cdots, t_{B_n}\}$, such that $src(t_{B_i}) = dest(t_{B_i}) = B_i$ ;
5 foreach $t_{B_i} \in T_B$ do
6     $corolesets(t_{B_i}) = \{\{\overline{a_i}\}\}$;
7 end
8 foreach $t \in T_c$ do
9     $rolesets(t) = rolesets(t) \cup [\bigcup_{h \in ac(t)}(f(h))]$ ;
10 end
11 Create a new And control state, $s_{new}$, such that $children(s_{new}) = \{s\} \cup B$;
12 Assign synchronizer PUEE($A$) to $s_{new}$
```

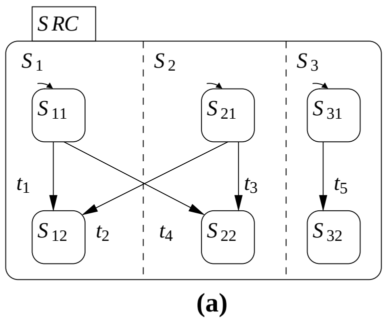
in that no pair of transitions of the original model that have overlapping arenas can be taken together in the same small step.

Proposition 7.10 *Given an SBSML model, its set of transitions, T, and one of its compound control states, s, applying the "srcdesttoarena" algorithm, i.e., Algorithm 5, to s results in a new model whose behaviour is different from the original model in that, for each potential small step τ in the original model, there is a corresponding potential small step τ' in the new model such that one of the following statements is true:*

i $\tau = (\tau' \cap T)$, *and there does not exist any* $t_1, t_2 \in (\tau \cap T_s)$ *such that* $t_1 \perp t_2$ *and* $arena(t_1) \not\perp arena(t_2)$*; or*

ii $(\tau' \cap T) \subset \tau$, *and for each* $t_1 \in (\tau - (\tau' \cap T))$, *there exists a transition* $t_2 \in (\tau \cap T_s)$ *such that* $t_1 \perp t_2$ *and* $arena(t_1) \not\perp arena(t_2)$*; furthermore,* τ' *is maximal, i.e., it cannot be extended with additional transitions in* $\tau - \tau'$*, and* τ' *has the highest priority, i.e., none of its transitions cannot be replaced with a higher-priority transition in* $\tau - \tau'$*.*

Proof Idea. In the case i above, since no two transitions in T_s that have overlapping arenas are included in τ, all transitions of τ, including the ones in $\tau \cap T_s$, can be included in a potential



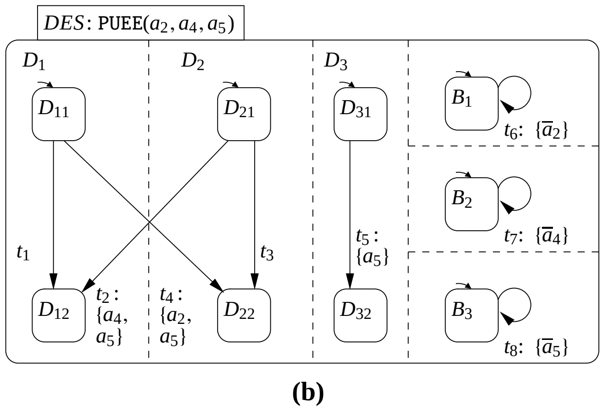


Figure 7.11: The effect of applying Algorithm 5.

small step, τ', of the new model. For a transition, $t \in \tau$, if $t \in (\tau \cap T_s)$, it can synchronize with its corresponding transition in T_B, created on line 4 of the algorithm, and thus can be included in τ'; this synchronization is possible because each of the transitions in T_B is orthogonal with all transitions in T_s. Otherwise, if $t \in (\tau - T_s)$, it need not synchronize with any transition in T_B, and thus it can be included in τ' because it can be included in τ.

In the case ii above, if $t_1 \in (\tau - (\tau' \cap T))$, then it means that t_1 is not included in τ' because it could not have synchronized with a transition in T_B according to the synchronizer introduced in line 4 of Algorithm 5, otherwise t_1 could not have belonged to τ either. But if t_1 cannot synchronize with a transition in T_B, it means that there is another transition, $t_2 \in \tau'$, that is synchronizing on the same label that t_1 needs to synchronize. But because of the way role sets of transitions are constructed on line 9 of Algorithm 5, that is only possible if $t_2 \in ac(t_1)$, which means $t_1 \perp t_2$ and $arena(t_1) \not\perp arena(t_2)$. Lastly, τ' is maximal and high priority because of the semantic definition schema in Figure 7.4 and the definition of PRI and SYN functions in the schema. $\square$

Preemption

The transformation that disallows two transitions that one is an interrupt for another to be included in the same small step is similar to the transformation presented for disallowing a pair of orthogonal transitions whose arenas are not orthogonal in the same small step. The idea of transformation is the same in that for each transition, t, first, its set of *interrupting conflict* transitions, $ic(t)$, similar to the set of arena conflicting, $ac(t)$, described above, should be defined. Using this syntactic information, a similar algorithm as Algorithm 5 can be designed that creates new dummy control states that have self transitions whose missions are to disallow a pair of transitions that one is an interrupt for another to be executed together, by exclusively synchronizing with one or the other, but not both. To avoid duplication, this algorithm and its proof of correctness, which are very similar to the ones for small-step consistency transformation, are not presented.

7.4.3 The Present In Same Event Lifeline Semantics

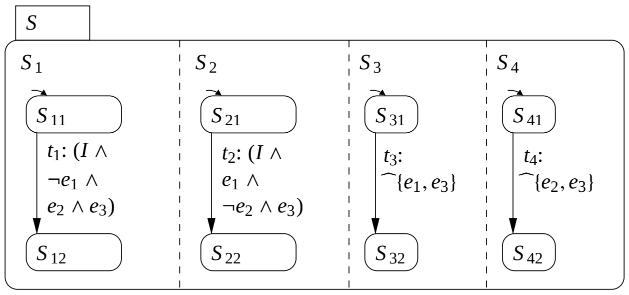
The transformation scheme presented in this section shows how the Present In Same event lifeline semantics can be modelled using the synchronization capability of SBSMLs. Algorithm 6 receives a BSML model specified in the Present In Same event lifeline semantics and replace its internal events, which follow the semantics of the Present In Same event lifeline semantics, with necessary synchronization instrumentation that model the semantics of these events. The input to the algorithm consists of the set of transitions of a model, T, its set of internal events, $\{e_1, \cdots, e_n\}$, which are called signals in some BSMLs that support the Present In Same event lifeline semantics, and its root control state.

Intuitively, Algorithm *tosignals*, in Figure 6, works as follows: For each signal, e_i, a pair of labels are created, namely, x_i and x_i'. For each signal, e_i, a new control state and two self transitions on it, namely, t_{x_i} and $t_{x_i'}$, are defined so that it is not possible for both e_i and $\neg e_i$ to trigger transitions in the same small step. Label x_i is used for synchronization of transitions that generate e_i, while label x_i' is used for synchronization of transitions that are triggered with the negation of e_i. Lastly, for each signal, e_i, a third label, l_i, is defined so that the generated events of one transition is related to the trigger of another via a synchronization mechanism. Both l_i and x_i are necessary so that a small step cannot have two disjoint subsets, one including transitions that generate and are triggered with e_i, and another including transitions that are triggered with the negation of e_i.

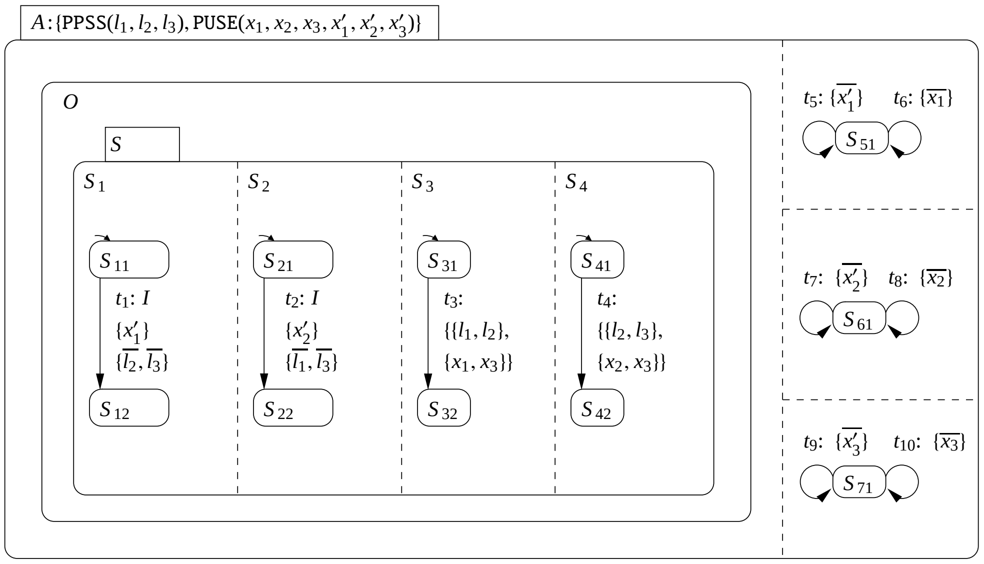
In Algorithm *tosignals*, in Figure 6, it is assumed that: (i) there is no transition in the model such that it generates an event and is triggered with the negation of the event; and (ii) there is no transition in the model such that it is both triggered with an event and generates it.

Algorithm 6: *tosignals*$(T, \{e_1, \cdots, e_n\}, root)$.

```
Input: $T, \{e_1, \cdots, e_n\}, root$
Result: Events/signals in the PRESENT IN SAME event lifeline semantics are replaced by
        synchronization instrumentation.
Create a set of new Basic control state, $B = \{B_1, \cdots, B_n\}$;
Create a set of new labels $X = \{x_1, \cdots, x_n\}$ ;
Create a set of new labels $X' = \{x'_1, \cdots, x'_n\}$ ;
Create $n$ new self transitions, $\{t_{x_1}, \cdots, t_{x_n}\}$, such that $src(t_{x_i}) = B_i$, $dest(t_{x_i}) = B_i$, and
  $corolesets(t_{x_i}) = \{\{\overline{x_i}\}\}$, for $1 \leq i \leq n$ ;
Create $n$ new self transitions, $\{t_{x'_1}, \cdots, t_{x'_n}\}$, such that $src(t_{x'_i}) = B_i$, $dest(t_{x'_i}) = B_i$, and
  $corolesets(t_{x'_i}) = \{\{\overline{x'_i}\}\}$, for $1 \leq i \leq n$ ;
Create a set of new labels $L = \{l_1, \cdots, l_n\}$;
foreach $t \in T$ do
    foreach $e_i \in \{e_1, \cdots, e_n\}$ do
        if $e_i \in pos\_trig(t)$ then
            $corolesets(t) = corolesets(t) \cup \{\overline{l_i}\}$ ;
            $pos\_trig(t) = pos\_trig(t) - \{e_i\}$;
        end
        else if $e_i \in neg\_trig(t)$ then
            $rolesets(t) = rolesets(t) \cup \{x'_i\}$ ;
            $neg\_trig(t) = neg\_trig(t) - \{e_i\}$;
        end
        else if $e_i \in gen(t)$ then
            $rolesets(t) = rolesets(t) \cup \{l_i\} \cup \{x_i\}$ ;
            $gen(t) = gen(t) - \{e_i\}$;
        end
    end
end
Create a new Or control state, $M$, and a new And control state, $s_{new}$, such that
  $children(s_{new}) = root \cup B$ and $parent(s_{new}) = M$, where $M$ is the new root control state;
Assign synchronizer PPSS($L$) and PUSE($X \cup X'$) to $s_{new}$ ;
```


S
S_1
S_2
S_3
S_4
S_{11}
t_1: $(I \wedge \neg e_1 \wedge e_2 \wedge e_3)$
S_{12}
S_{21}
t_2: $(I \wedge e_1 \wedge \neg e_2 \wedge e_3)$
S_{22}
S_{31}
t_3: $\frown\{e_1, e_3\}$
S_{32}
S_{41}
t_4: $\frown\{e_2, e_3\}$
S_{42}

(a)


A: {PPSS(l_1, l_2, l_3), PUSE$(x_1, x_2, x_3, x'_1, x'_2, x'_3)$}
O
S
S_1
S_2
S_3
S_4
S_{11}
t_1: I $\{x'_1\}$ $\{\overline{l_2}, \overline{l_3}\}$
S_{12}
S_{21}
t_2: I $\{x'_2\}$ $\{\overline{l_1}, \overline{l_3}\}$
S_{22}
S_{31}
t_3: $\{\{l_1, l_2\}, \{x_1, x_3\}\}$
S_{32}
S_{41}
t_4: $\{\{l_2, l_3\}, \{x_2, x_3\}\}$
S_{42}
t_5: $\{\overline{x'_1}\}$
t_6: $\{\overline{x_1}\}$
S_{51}
t_7: $\{\overline{x'_2}\}$
t_8: $\{\overline{x_2}\}$
S_{61}
t_9: $\{\overline{x'_3}\}$
t_{10}: $\{\overline{x_3}\}$
S_{71}

(b)

Figure 7.12: The effect of applying Algorithm 6.

Example 57 *Figure 7.12 shows the effect of applying the "tosignals" algorithm to the SBSML model in Figure 7.12(a). The result is the SBSML model in Figure 7.13(b) whose events are replaced by synchronization instrumentation, and has the same behaviour as the original model.*

The models in Figure 7.12 are similar to the ones in Figure 6.7, on page 202, except that: (i) the name of events and labels are changed here to match the transformation algorithm; and (ii) here the model in Figure 7.12(b) is created by exactly following the steps in Algorithm 6, as opposed to the model in Figure 202 that is manually created, with a slightly different set of control states.

Proposition 7.11 *Given an SBSML model that uses internal events according to the* PRESENT

In Same *event lifeline semantics, applying tosignals*$(T, \{e_1, \cdots, e_n\}, root)$*, where* T *is the set of transitions of the model,* $E = \{e_1, \cdots, e_n\}$ *is its set of events, and root is its root control state, results in an SBSML model that has the same behaviour as the original model.*

Proof Idea. To prove the above claim, it suffices to show that for each small step, τ, in the original model, there is a small step, τ', in the new model that includes the corresponding transitions of τ, and vice versa.

For each such a τ, there exists a corresponding τ' because: (i) if a transition, t, in τ is not triggered by an internal event, its corresponding transition, t', can be included in τ', because *tosignals* does not instrument that transition; (ii) if a pair of transitions, t_1 and t_2, are included in τ because $pos\_trig(t_1) \cap gen(t_2) \neq \emptyset$, their corresponding transitions, t'_1 and t'_2, can also be included in τ' because of their synchronization instrumentation on lines 4, 10, and 18 ; and (iii) if a transition, t, in τ is triggered by the negation of an internal event that is not generated by any transition in τ, its corresponding transition, t', can also be included in τ', because the corresponding transitions of a pair of transitions that one generates an event and the other is triggered by its negation cannot be included in a small step of the new model, because of the instrumentations on lines 5 and 14 and the fact that the self transitions in X and X', defined on lines 2 and 3, respectively, pairwise share the same control state; and (iv) lastly, Algorithm 6 does not affect the way a set of maximal, high-priority transitions are grouped together in the new model compared to the original model, because: (a) it does not change the relative precedence of the transitions, according to any of the priority semantics; and (b) the synchronizers introduced on line 24 do not put any restrictions on the maximality of a small step.

Using the above lines of arguments conversely, it can also be proven that for each τ', there exists a corresponding τ. Thus, the original and the new model have the same behaviour. □

7.4.4 Composition Operators

In this section, the formal transformation of some of the common composition operators introduced in template semantics [75, 74] to their equivalent SBSMLs are considered.

In the following transformation schemes, a *component* or an *operand* of a template semantic composition operator corresponds to a compound control state of an SBSML. Thus, the input to a transformation algorithm for a composition operator consists of a set of compound control states, plus any extra syntactic elements, such as synchronization events in the case of rendezvous and environmental synchronization composition operators. As such, a hierarchy of composition operators of a model in template semantics can be transformed into a hierarchy of control states of an SBSML model. A model in template semantics that does not have any composition operator behaves the same as its equivalent BSML model. In template semantics, there is no notion of synchronizer.

In template semantics, originally, the composition operators are considered as binary operators, but here they are considered as n-ary operators.

Interleaving

"In interleaving composition, only one component can execute transitions in a *step* [emphasis mine]" [75], where step has the same meaning as small step in this dissertation.

Algorithm 7 specifies how an interleaving composition operator can be transformed into an *And* control state that has the same behaviour as the composition operator. The input to the algorithm *tointerleaving* is a set of control states $\{s_1, \cdots, s_n\}$. As before, for the set of transitions of a model, T, T_s is the set of all transitions in T such that $lca(src(t), dest(t)) \in children^* s$.

Algorithm 7: *tointerleaving*$(\{s_1, \cdots, s_n\})$.

```
   Input: $\{s_1, \cdots, s_n\}$
   Result: Each small step includes the transitions of at most one of the $T_{s_i}$'s, where
           $s_i \in \{s_1, \cdots, s_n\}$.
1  Create a set of new labels $A = \{a_1, \cdots, a_n\}$;
2  Create a new Basic control state, $int$;
3  Create $n$ new self transition, $X = \{x_1, \cdots, x_n\}$, such that $src(x_i) = int$, $dest(x_i) = int$, and
   $corolesets(x_i) = \{\{\overline{a_i}\}\}$, for $1 \leq i \leq n$ ;
4  Create a new And control state, $s_{new}$, such that $children(s_{new}) = \{s_1, \cdots, s_n\} \cup \{int\}$;
5  foreach $s_i \in \{s_1, \cdots, s_n\}$ do
6      foreach $t \in T_{s_i}$ do
7          $rolesets(t) = rolesets(t) \cup \{a_i\}$;
8      end
9  end
10 Assign synchronizer $\texttt{UUSE}(A)$ to $s_{new}$;
```

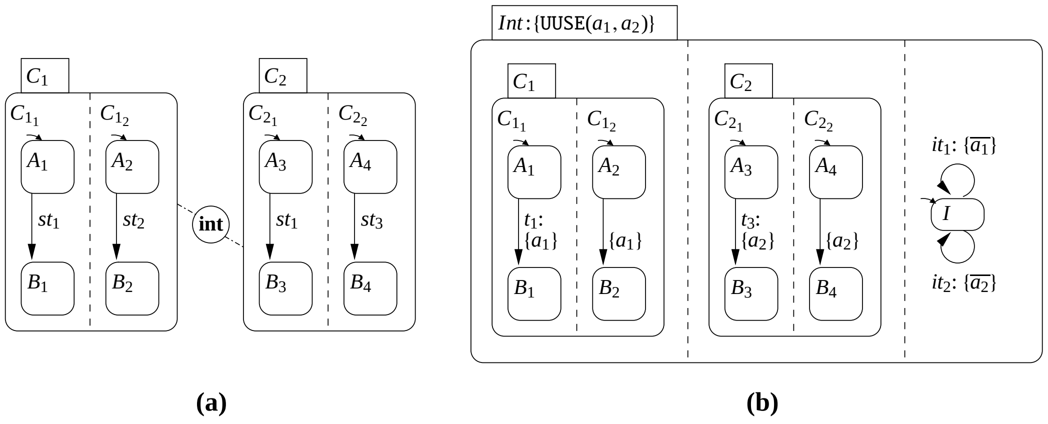


Figure 7.13: The effect of applying Algorithm 7.

Example 58 *Figure 7.13 shows that how the effect of applying the "tointerleaving" algorithm to a model that uses the interleaving composition operator results in an SBSML model with the same behaviour. The SBSML model in Figure 7.13(b) is the result of "tointerleaving(C_1, C_2)". The models in Figure 6.8, on page 203 are similar to the ones in Figure 7.13, except that the model in Figure 6.8(b) is not obtained through applying algorithm "tointerleaving", in order to obtain a simpler model.*

Proposition 7.12 *Given a model in template semantics with one interleaving composition operator "$\texttt{int}(s_1, \cdots, s_n)$", replacing the composition operator with "tointerleaving($\{s_1, \cdots, s_n\}$)" yields an SBSML model that has the same behaviour as the original model.*

Proof Idea. To prove the above claim, it suffices to show that for each small step, τ, in the original model, there is a small step, τ', in the new model that includes the corresponding transitions of τ, and vice versa. As such, it should be proven that "only one component" of the original model "can execute transitions in a" small step of the new model. This is true because at each small step only one of the transitions in X, created on line 3 of the algorithm, can be executed, which in turn can synchronize exclusively with the transitions of one of the control states, which each represents a component of the composition operator in the original model. Thus, the transformation in Algorithm 7 is sound. $\square$

Rendezvous

The rendezvous composition operator requires that, “exactly one transition in the sending component generates a *synchronization event* [emphasis mine] that triggers exactly one transition in the receiving component” [75], where synchronization events of a composition operator are unique and syntactically specified. In Section 6.4.4, a simple semantics for rendezvous composition operator was considered that assumed models such as the processes in CCS, whose semantics can be modelled using synchronizers of type `UUEE`. However, in the general case an operand of a rendezvous composition operator itself can be an arbitrary control state, possibly an *And* control state that can execute concurrent transitions. In such a general case, the semantics of rendezvous composition operator in template semantics additionally requires that: “Transitions that are enabled by non-synchronization events or that generate non-synchronization events can execute only in an interleaved manner.” [74] In the general case, the only syntactic assumption made about a model is that each of its transitions can synchronize according to one rendezvous composition operator and over only one of its synchronization events.

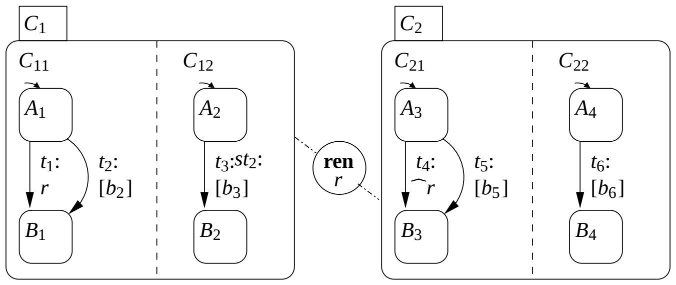
Algorithm 8 specifies a transformation scheme for the rendezvous composition operator for the general case. The input to the algorithm is a set of control states, $\{s_1, \cdots, s_n\}$, $n > 2$, each of which is an operand of the rendezvous composition operator, and a set of synchronization events, $\{e_1, \cdots, e_m\}$, $m > 1$. The set of labels in A in the algorithm are used to ensure that a set of transitions in a small step that do not synchronize over synchronization events belong to at most one of the components. The set of labels in L are used to model the synchronization events of the rendezvous composition operator. Label set O is used to ensure that a small step does not include both transitions that synchronize over a synchronization event and the transitions that do not.

Example 59 *The model in Figure 7.14(a) shows a model that uses the rendezvous composition operator over two components* C_1 *and* C_2*. Applying “torendezvous*$(\{\mathrm{C}_1, \mathrm{C}_2\}, \{\mathrm{e}\})$*” results in the SBSML model in Figure 7.14(b) that is equivalent to the original model.*

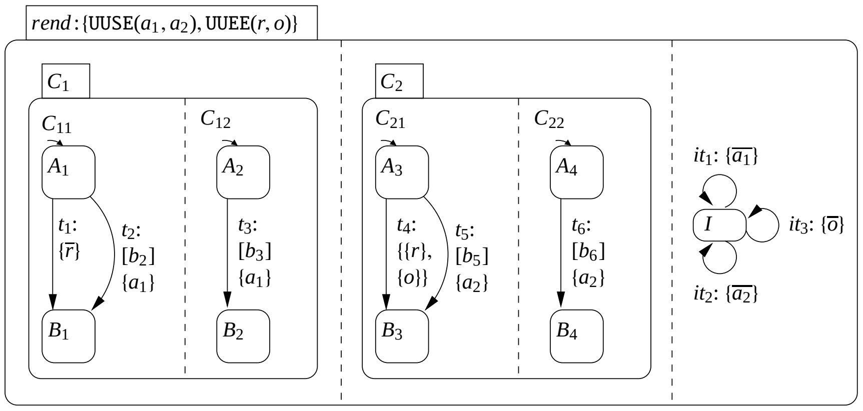
Proposition 7.13 *Given a model in template semantics with a rendezvous composition operator “*$\mathtt{ren}(\{s_1, \cdots, s_n\}, \{e_1, \cdots, e_m\})$*”, replacing the composition operator with “torendezvous*$(\{s_1, \cdots, s_n\}, \{e_1, \cdots, e_m\})$*” yields an SBSML model that has the same behaviour as the original model.*

Algorithm 8: *torendezvous*($\{s_1, \cdots, s_n\}, \{e_1, \cdots, e_m\}$).

```
Input: $\{s_1, \cdots, s_n\}, \{e_1, \cdots, e_m\}$
Result: Either a transition from $t \in T_{s_i}$ and a transition from $t' \in T_{s_j}$, $1 \leq i \leq n$, such that
        $e_j \in pos\_trig(t)$ and $e_j \in gen(t')$, $1 \leq j \leq m$, are included in a small step, or the
        small step includes non-synchronizing transitions from at most one of the $T_{s_i}$'s,
        $1 \leq i \leq n$.
1  Create a set of new labels $A = \{a_1, \cdots, a_n\}$;
2  Create a set consisting of a new label $O = \{o\}$;
3  Create a set of new labels $L = \{l_1, \cdots, l_m\}$;
4  Create a new Basic control state, $int$;
5  Create $n$ new self transitions, $X = \{x_1, \cdots, x_n\}$, such that $src(x_i) = int$, $dest(x_i) = int$, and
   $corolesets(x_i) = \{\{\overline{a_i}\}\}$, for $1 \leq i \leq n$ ;
6  Create one last self transitions, $t_o$, such that $src(t_o) = int$, $dest(t_o) = int$, and
   $corolesets(t_o) = \{\{\overline{o}\}\}$ ;
7  Create a new And control state, $s_{new}$, such that $children(s_{new}) = \{s_1, \cdots, s_n\} \cup \{int\}$;
8  foreach $s_i \in \{s_1, \cdots, s_n\}$ do
9      foreach $t \in T_{s_i}$ do
10         if $gen(t) \cap \{e_1, \cdots, e_m\} = e_j$ then
11             $rolesets(t) = rolesets(t) \cup \{l_j\} \cup \{o\}$;
12             $gen(t) = \emptyset$;
13         else if $pos\_trig(t) \cap \{e_1, \cdots, e_m\} = e_j$ then
14             $corolesets(t) = corolesets(t) \cup \{\overline{l_j}\}$;
15             $pos\_trig(t) = \emptyset$;
16         end
17         else
18             $rolesets(t) = rolesets(t) \cup \{a_i\}$;
19         end
20     end
21 end
22 end
23 Assign synchronizer UUSE($A$) to $s_{new}$;
24 Assign synchronizer UUEE($L \cup O$) to $s_{new}$;
```



(a)



(b)

Figure 7.14: The effect of applying Algorithm 8.

Proof Idea. To prove the above claim, it should be shown that for each small step, τ, in the original model, there is a small step, τ', in the new model that includes the corresponding transitions of τ, and vice versa. If such a τ or τ' does not exist when the other exists, it means that the transformation algorithm *torendezvous* is not sound. To prove the soundness of Algorithm 8, it should be shown that in the new model either a pair of transitions that synchronize over a synchronizing event are included a small step or the transitions that do not synchronize over synchronizing events and belong to one component are included in the small step, but not both kinds of transitions. Two transitions that synchronize over a synchronization event can be included in a small step of the new model only exclusively, because such two transitions synchronize according to a synchronizer of type `UUEE`, and furthermore, no additional such pair of transitions can be included in the same small step because one of the transitions in the first pair of transitions also exclusively synchronizes with transition t_o, created on line 6 of the algorithm, via label o. If such a pair of synchronizing transitions is not included in a small step, the transitions of only one of the components can be included in the small step, because only one of the transitions in X, created on line 5 of the algorithm, can be executed in each small step. Thus, algorithm *torendezvous* is sound because it mimics the behaviour of the *rendezvous* composition operator that it translates. □

Environmental Synchronization

The environmental synchronization operator requires that, "both components execute in the same *microstep* [emphasis mine] if the executing transitions all have the same trigger event, e, which is a designated *synchronization event* [emphasis mine] (line 1), and if all components that can react to this event participate in the step ...." [75], where the term "microstep" corresponds to the term "small step" in this dissertation and a synchronization event can be received only from the environment. Similar to the case in the rendezvous composition operator, when synchronizing transitions cannot be taken, "in the 'unsync' case, none of the executing transitions is triggered by a synchronization event, so one or the other component takes a step in isolation (interleaving)." [75] In template semantics, the following well-formedness condition is assumed for environmental synchronization composition: If a transition is triggered by a synchronization event or generates a synchronization event, neither it is triggered with any other events, synchronization or otherwise.

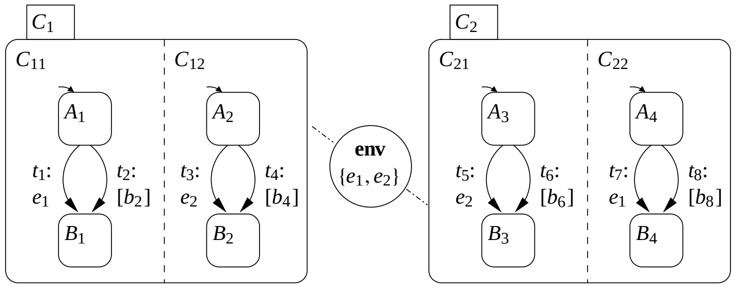
Algorithm 9 specifies a transformation scheme for the environmental synchronization operator. The input to the algorithm is a set of control states, $\{s_1, \cdots, s_n\}$, $n > 2$, each of which is an operand of the composition operator, and a set of synchronization events, $\{e_1, \cdots, e_m\}$, $m > 1$. The set of labels in A are used to ensure that a set of transitions in a small step that do not synchronize over any synchronization events belong to at most one of the components. The set of labels in L are used to model the synchronization events of the composition operator.

Algorithm 9: *toenvironmental*$(\{s_1, \cdots, s_n\}, \{e_1, \cdots, e_m\})$.

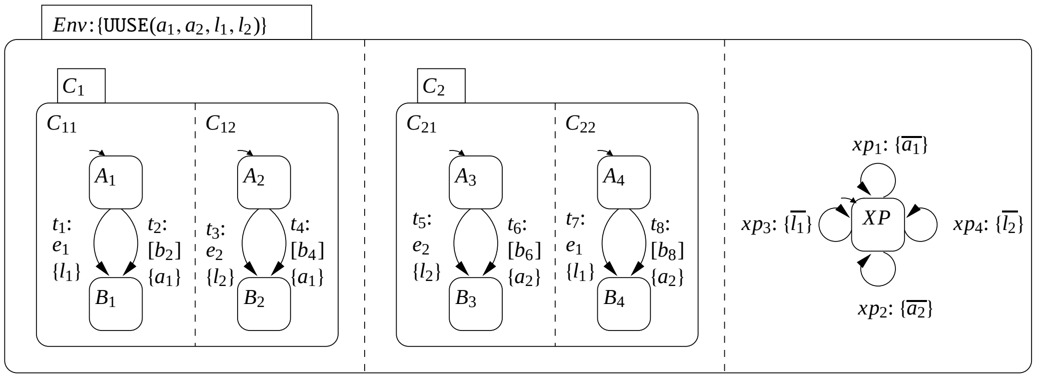
```
   Input: $\{s_1, \cdots, s_n\}$, $\{e_1, \cdots, e_m\}$
   Result: In a small step, either a maximal set of transitions that synchronize over the same
           environmental input synchronization event can be included, or the
           non-synchronizing transitions from at most one of the $T_{s_i}$'s, $1 \leq i \leq n$, are
           included.
1  Create a set of new labels $A = \{a_1, \cdots, a_n\}$;
2  Create a set of new labels $L = \{l_1, \cdots, l_m\}$;
3  Create a new Basic control state, $int$;
4  Create $n$ new self transitions, $X = \{x_1, \cdots, x_n\}$, such that $src(x_i) = int$, $dest(x_i) = int$, and
   $corolesets(x_i) = \{\{\overline{a_i}\}\}$, for $1 \leq i \leq n$ ;
5  Create $m$ new self transitions, $P = \{p_1, \cdots, p_m\}$, such that $src(p_i) = int$, $dest(p_i) = int$, and
   $corolesets(p_i) = \{\{\overline{l_i}\}\}$, for $1 \leq i \leq m$ ;
6  Create a new And control state, $s_{new}$, such that $children(s_{new}) = \{s_1, \cdots, s_n\} \cup \{int\}$;
7  foreach $s_i \in \{s_1, \cdots, s_n\}$ do
8      foreach $t \in T_{s_i}$ do
9          if $trig(t) \cap \{e_1, \cdots, e_m\} = e_j$ then
10             $rolesets(t) = rolesets(t) \cup \{l_j\}$;
11         end
12         else
13             $rolesets(t) = rolesets(t) \cup \{a_i\}$;
14         end
15     end
16 end
17 Assign synchronizer $\mathtt{UUSS}(A \cup L)$ to $s_{new}$;
```

Example 60 *The model in Figure 7.15(a) shows a model that uses the environmental synchronization operator over two components* C_1 *and* C_2. *Applying "toenvironmental*$(\{\mathrm{C}_1, \mathrm{C}_2\}, \{\mathrm{e}_1, \mathrm{e}_2\})$*" results in the SBSML model in Figure 7.15(b) that is equivalent to the original model.*

The models in Figure 6.9(b), on page 204, shows an SBSML model that has the same behaviour as the model in 6.9(a). The source model does not include any non-synchronizing tran-



(a)



(b)

Figure 7.15: The effect of applying Algorithm 9.

sition. The transformation used in that figure, in order to demonstrate the role of poly-roles, first identifies the maximum number of synchronization transitions that can be taken together according to an environmental input event. If some of the transitions in such a maximal set cannot be taken in a small step, the added dummy synchronizing transitions need to replace them. Here, a more general approach has been adopted that also considers non-synchronizing transitions.

Proposition 7.14 *Given a model in template semantics with one environmental synchronization composition operator "$\mathtt{env}(\{s_1, \cdots, s_n\}, \{e_1, \cdots, e_m\})$", replacing the composition operator with "toenvrionmental$(\{s_1, \cdots, s_n\}, \{e_1, \cdots, e_m\})$" yields an SBSML model that has the same behaviour as the original model.*

Proof Idea. Similar to the proof for the soundness of the other composition operators, it suffices to show that for each small step, τ, in the original model, there is a small step, τ', in the new model that includes the corresponding transitions of τ, and vice versa. As such, it should be proven that a small step either includes a maximal set of transitions that synchronize over the same synchronization event or it includes the transitions that do not synchronize over synchronizing events and belong to one component are included in the small step, but not both. It is not possible for the non-synchronizing transitions of different components to be included in the same small step, because of the self transitions created by line 4 in the algorithm, which are on the same control state. Similarly, it is not possible for the synchronizing transitions that synchronize over different synchronization events to be included in the same small step because of the self transitions that are created by line 5 in the algorithm. Lastly, non-synchronizing and synchronizing transitions cannot be included in the same small step because their corresponding self transitions, created on lines 4 and 5 of the algorithm, cannot be executed together in the same small step, because their corresponding self transitions are over the same control state. Thus, algorithm *toenvironmental* is sound. □

7.4.5 Workflow Patterns

This section considers the transformation of the sequence workflow pattern, whose name is the same as the name of a similar composition operator in template semantics. The formalization of the transformation schemes of the other workflow patterns, some of which were described informally in Section 6.4.5, are not considered in this dissertation because: (i) these workflow patterns are not originally defined in the context of BSMLs or SBSMLs; and (ii) there is no conclusive formal semantics for these patterns in the literature. However, as shown informally in Section 6.4.5, the transformation schemes for the workflow patterns are mainly in the same style as other syntactic constructs, semantic variations, and composition operators.

Sequence

When two components are connected through a sequence workflow pattern/composition operator, "the first component executes in isolation until it terminates (i.e., reaches its *final basic states*[emphasis mine]) and then the second component executes in isolation. If component one is a composite component, then all of its basic components must reach final basic states before the second component can start," [74] where a final basic state is either syntactically designated in a language or is a control state of a model that has no outgoing transitions.

In Section 6.4.5 the two components of a sequence composition operator are connected via a multi-source, multi-destination transition, which implements the semantics of the sequence workflow pattern, but introduces an extra idle small step between the execution of the first and the second component. As mentioned in Section 6.4.5, an alternative interpretation of the semantics of the sequence workflow pattern, such as the one in template semantics [75], disallows the extra idle small step. This section presents a transformation algorithm that neither introduces any extra idle small steps nor uses any synchronizers. First, some notation need to be introduced.

Given a compound control state, s, $final(s)$ denotes the set of final *Basic* control states of s. Each *Or* control state, s, has at most one final control state, such that $final(s) \neq default(s)$.[3] There is no need to allow more than one final control state in an *Or* control state because two final control states can be merged by directing the incoming transitions of one to the other. The final control states of an *And* control states is the union of the final control states of its children.

Given a control state, s, $incoming(s)$ is the set of all transitions, such that for each of these transitions, t, $lca(src(t), dest(t)) \notin children^{+}(s)$ and $dest(t) \in children^{*}(s)$.

Algorithm 10 specifies a transformation scheme for the sequence composition operator. The input to the algorithm is two compound control states, s_1, s_2, each of which is an operand of the composition operator. The set of variables $\{v_1, \cdots, v_n\}$ is used to determine when the final states

[3]If $final(s) = default(s)$, perhaps $|children(s)| = 1$, which means the *Or* control state is itself virtually a *Basic* control state.

of the first component are all arrived in, upon which, in the same small step, transition t_{last} is executed to move the control of the model to the control states of the second component. When control state s_1 is reentered, all variables in $\{v_1, \cdots, v_n\}$ are reset.

Algorithm 10: *tosequence*(s_1, s_2).

```
   Input: $s_1, s_2$
   Result: First, transitions of $s_1$ are executed, followed by the ones of $s_2$.
1  Create a set of new boolean variables $\{v_1, \cdots, v_n\}$ that corresponds to the set of control
   states $final(s_1) = \{f_1, \cdots, f_n\}$;
2  foreach $f_i \in \{f_1, \cdots, f_n\}$ do
3      forall the $t \in incoming(f_i)$ do
4          $asn(t) := asn(t) \cup \{``v_i := true''\}$;
5      end
6  end
7  Create a new Basic control state, $last$;
8  Create a new transition, $t_{last}$, such that $src(t_{last}) = last$, $dest(t_{last}) = s_2$, and
   $gc(t_{last}) = (\texttt{new\_small}(v_1) \wedge \cdots \wedge \texttt{new\_small}(v_n))$;
9  Create a new And control state, $s_{new}$, such that $children(s_{new}) = \{s_1, s_2\} \cup \{last\}$;
10 foreach $t \in incoming(s_1)$ do
11     forall the $v_i \in \{v_1, \cdots, v_n\}$ do
12         $asn(t) := asn(t) \cup \{``v_i := false''\}$;
13     end
14 end
```

Example 61 *The model in Figure 7.16(a) shows a model that uses the sequence operator over two components* M *and* Q. *The result of "*$tosequence(\mathrm{M}, \mathrm{Q})$*" is the SBSML model in Figure 7.16(b), which is equivalent to the original model.*

Proposition 7.15 *Given a model in template semantics with one sequence composition operator "*$\texttt{seq}(s_1, s_2)$*", replacing the composition operator with "*$toseq(s_1, s_2)$*" yields an SBSML model that has the same behaviour as the original model.*

Proof Idea. It suffices to show that for each small step, τ, in the original model, there is a small step, τ', in the new model that includes the corresponding transitions of τ, and vice versa. As such, it should be proven that initially each small step includes transitions from s_1, and once all final states of s_1 are entered, the control is passed to s_2, after which only small steps including

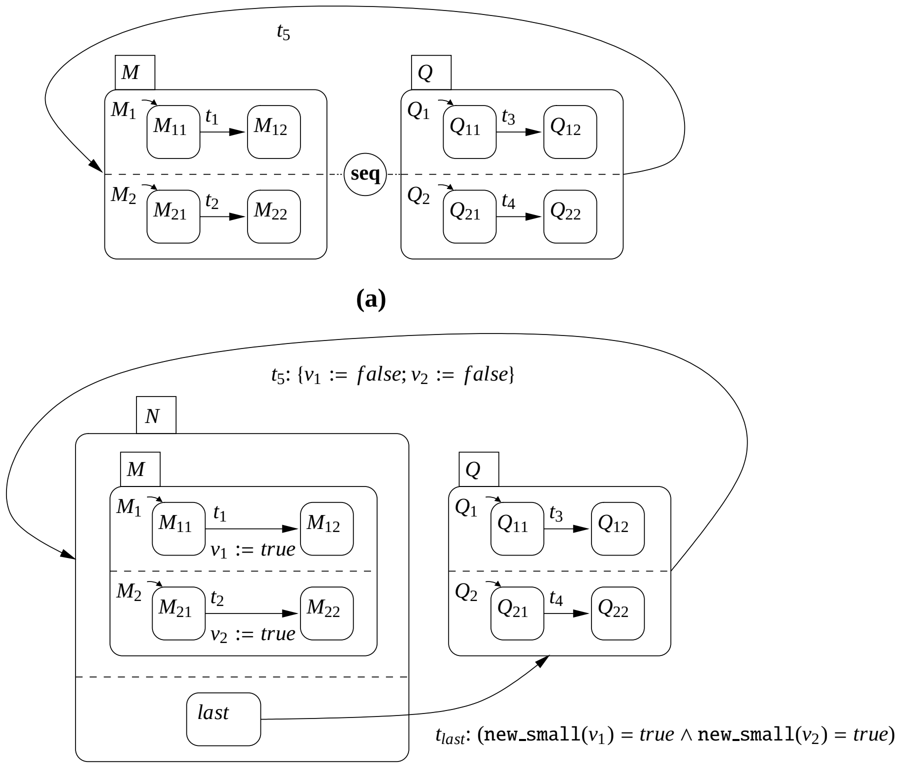


Figure 7.16: The effect of applying Algorithm 10.

transitions of s_2 are executed. The transformation in Algorithm 10 clearly does not allow small steps that include transitions from both s_1 and s_2. Furthermore, variables $\{v_1, \cdots, v_n\}$ are all true at the end of a small step if and only if an incoming transition to each final control states of s_1 has been executed. But in this last small step, t_{small} is executed, because its guard, which checks the values of $\{v_1, \cdots, v_n\}$ variables at the end of a small step, is true, and furthermore, t_{last} is an interrupt for the last incoming transitions of the final control states of s_1. Thus, once all final control states of s_1 are entered, the control is passed to s_2, as desired. Lastly, upon reentrance to s_1, the $\{v_1, \cdots, v_n\}$ variables are all reset to false afresh, through the way the incoming transitions to s_1 are modified on line 12 of the algorithm. □

A Transition-Aware Semantics. It should be noted that the above transformation uses the `new_small` keyword whose semantics is transition aware, and out of the scope of the semantic formalization in Chapter 4. It seems that this is the price that needs to be paid to avoid the extra idle small steps between the execution of the transitions of the operands of a sequence operator. Instead of `new_small` keyword, synchronization can be used so that two components that are connected by a sequence operator synchronize when the first component and all its subcomponents finish their execution, after which the second component starts its execution. However, such an approach again creates extra unnecessary small steps due to the synchronization necessary to recognize when all subcomponents have finished their execution. Such an approach is described by Milner [72, p.172–174,190–192], where he explains how CSP sequence operator, ";", [48, p. 171] can be translated into CCS. But this approach introduces extra CCS silent actions "τ".

7.4.6 Effect of Transformation Schemes on Well-Formedness

To achieve a clear exposition of the transformation schemes and their proofs of correctness in the preceding sections, it was assumed that the changes in the role sets, co-role sets, and the synchronizers of a model do not violate the well-formedness conditions of the model, as described in Section 7.1.2. However, a transformation algorithm can create a non-well–formed SBSML model. However, such a model can be *corrected* in a straightforward manner, while preserving the intended behaviour of the model. Furthermore, it is shown that the proofs of correctness presented for transformation algorithms remain valid, because of the types of well-formedness violations and the proposed corrections.

The only well-formedness criterion that could be violated by a new model produced by a transformation algorithm is the criterion iv of the well-formedness of SBSMLs, which states: Two labels that are associated with the same synchronization type do not belong to two different role sets or two different co-role sets of the same transition. The following pattern of scenarios can cause such a violation. A transformation scheme may introduce a new singleton role set, say $\{x\}$, for a transition t that already has a singleton role set, say $\{y\}$, where both $\{x\}$ and $\{y\}$ are meant to be synchronized by synchronizers that have the same type that belongs to "`U***`". According to criterion v, $\{x\}$ and $\{y\}$ need to be merged, but $\{x, y\}$ does not match "`U***`", violating criterion ii. Thus the corresponding synchronizers of $\{x\}$ and $\{y\}$ also need to be merged to a same synchronizer whose synchronization type belongs to "`P***`". The merge of the synchronizer can in turn trigger another merge with an already existing synchronizer that has the same synchronization type as the newly-created synchronizer. A similar pattern of well-formed violation can happen for synchronizers of synchronization type "`*U**`". A similar correction can be applied to this second pattern of violation scenarios by merging the synchronizer to a new synchronizer.

The above corrective steps to transform a model to a well-formed model, however, preserve the original behaviour. First, for example, a transition, t, with role sets $\{\{x\}, \{y\}\}$ that synchronize with two different synchronizer of the same synchronization type "`U***`", by definition, behaves exactly the same as t having a single role set $\{x, y\}$ that synchronizes with a single synchronizer of synchronization type "`P***`": In both models t is required to have synchronization over x and y, according to the semantics of the third and fourth letters of the corresponding synchronizers, which are the same in both cases. The above statement, however, is true if the semantics of such non-well–formed SBSML models is defined according to the semantic definition schema on page 220, which is the case; i.e., even if a model violates the well-formedness criterion iv, its behaviour is defined according to the formal semantics in Section 7.2.[4] Similarly, merging the co-role sets of a transition that synchronize according to synchronizers that have the same synchronization type that belongs to "`*U**`" does not change the behaviour of the model. Lastly, merging two synchronizers that have the same synchronization type belonging to "`P***`" or "`*P**`" does not change the behaviour of the model. Thus, the corrective steps outlined above do not change the behaviour of a model, and therefore, the reasoning presented in the proofs presented in this

[4]As an example of an SBSML model with an undefined semantics, an SBSML model that allows a transition to have a role set $\{x, y\}$ that synchronizes according to a synchronizer of synchronization type "`U***`" has a nonsensical meaning. However, none of the transformation algorithms create such nonsensical models.

section remain sound in the presence of the aforementioned types of non-well–formed SBSML models.

7.5 Relevance of Semantic Quality Attributes for SBSMLs

This section considers the relevance of each of the three semantic quality attributes introduced for BSMLs, described in Chapter 5, for SBSMLs.

7.5.1 Non-Cancelling SBSML Semantics

Recall that in a non-cancelling BSML semantics, once a transition of a model becomes executable in a big step, it remains executable during the big step. However, the non-cancelling semantic quality attribute is not relevant for SBSMLs, because, as discussed in Section 7.3.3, an enabled, high-priority transition may not be executable at a snapshot because its synchronization requirements cannot be satisfied. As such, a notion of an executable transition in SBSMLs can be only defined with respect to the executability of other transitions, which is not consistent with the notion of non-cancelling BSML semantics.

7.5.2 Priority-Consistent SBSML Semantics

Recall that in a priority-consistent BSML semantics, the higher-priority transitions are chosen to execute over lower-priority transitions during a big step. The priority consistency semantic quality attribute is also relevant for SBSMLs. Exactly the same semantic characterization as the one for the priority-consistent BSML semantics holds for the SBSML semantics. The following proposition restates the necessary and sufficient conditions for an SBSML semantics to be priority consistent, which is similar to the Proposition 5.6 for BSMLs, on page 5.6.

Proposition 7.16 *An SBSML semantics that subscribes to a hierarchical priority semantics together with the* NEGATION OF TRIGGERS *priority semantics is priority consistent if and only if it* *satisfies* $\mathbf{P} \equiv \mathbf{P}_{\textbf{Hierarchical}} \wedge \mathbf{P}'_{\textbf{Negation}}$, *where*

$$\begin{array}{lcl} \mathbf{P}_{\textbf{Hierarchical}} & \equiv & \text{TAKE ONE}, \\ \mathbf{P}'_{\textbf{Negation}} & \equiv & \mathbf{P}_{\textbf{Negation}} \wedge PXEvent \wedge PIEvent, \\ \mathbf{P}_{\textbf{Negation}} & \equiv & \neg\textbf{\textit{Negated Events}} \\ PXEvent & \equiv & \text{X.P.I. REMAINDER} \vee \neg\textbf{Negated External Events},\ \textit{and} \\ PIEvent & \equiv & \text{ASYNCHRONOUS EVENT} \vee \neg\textbf{Negated Interface Events}. \end{array}$$

Proof Idea. A similar proof of correctness as the one for Proposition 5.6, on page 169, can be developed for this proposition. An outline of this proof is presented below.

Predicate $\mathbf{P}_{\textbf{Hierarchical}}$ guarantees that the execution of an SBSML model cannot proceed in two different ways: One arriving at a configuration where a high-priority transition according to a hierarchical priority semantics can be taken, and one arriving at a configuration where a low-priority transition according to a hierarchical priority transition can be take. (The correctness proof of Proposition 5.4, on page 164, can be conferred for more detail.) Predicate $\mathbf{P}'_{\textbf{Negation}}$ guarantees that a priority-inconsistent behaviour according to the NEGATION OF TRIGGERS semantics does not arise: Internal events are not supported, because of $\mathbf{P}_{\textbf{Negation}}$, so it is not possible for a big step to include a high-priority transition, whose event trigger has just been generated, while another big step includes a low priority transition, because the event trigger of the high-priority transition is not generated yet. Predicates *PXEvent* and *PIEvent* require the external environmental input events and the interface events, respectively, to be either present or absent throughout a big step, so that a lower-priority transition can be only taken if the triggering event of a higher-priority transition is not present at the source snapshot of the current big step. (The correctness proof of Proposition 5.5, on page 168, can be conferred for more detail.)

Conversely, if an SBSML semantics is priority consistent it should satisfy predicate **P**, otherwise counter example models similar to the ones in Example 36, Example 37, and Example 38 can be created that exhibit a priority-inconsistent behaviour. (The aforementioned examples are relevant for SBSMLs too, since a BSML model is an SBSML model without any synchronizers.)

Thus, an SBSML semantics is priority consistent iff it satisfies **P**. □

7.5.3 Determinate SBSML Semantics

Recall that in a determinate BSML semantics, if two big steps of a BSML model execute the same (multi) set of transitions in different orders, their destination snapshots are equivalent. The determinacy semantic quality attribute is also relevant for SBSMLs. The same semantic characterization as the one for BSMLs, in Proposition 5.10, on page 176 holds for SBSMLs.

Proposition 7.17 *An SBSML semantics is determinate with respect to variables and events if and only if its constituent semantic options satisfies the predicate* $\mathbf{D} \equiv \mathbf{D}'_{\mathbf{Variables}} \wedge \mathbf{D}'_{\mathbf{Events}}$*, where*

$$\begin{array}{lcl}
\mathbf{D}'_{\mathbf{Variables}} & \equiv & \mathbf{D}_{\mathbf{Variables}} \wedge \mathit{DIAssign}, \\
\mathbf{D}_{\mathbf{Variables}} & \equiv & [\neg \textbf{\textit{Variable Assignments}} \vee \text{RHS B{\scriptsize IG} S{\scriptsize TEP}}] \vee \\
 & & [(\text{RHS S{\scriptsize MALL} S{\scriptsize TEP}} \vee \text{RHS C{\scriptsize OMBO} S{\scriptsize TEP}}) \Rightarrow (\text{T{\scriptsize AKE} O{\scriptsize NE}} \wedge \text{M{\scriptsize ANY}})], \\
\mathit{DIAssign} & \equiv & [\neg \textbf{\textit{Interface Variables in RHS}} \vee \text{RHS A{\scriptsize SYNCHRONOUS} V{\scriptsize ARIABLE}}] \vee \\
 & & [\text{RHS W{\scriptsize EAK} S{\scriptsize YNCHRONOUS} V{\scriptsize ARIABLE}} \Rightarrow (\text{T{\scriptsize AKE} O{\scriptsize NE}} \wedge \text{M{\scriptsize ANY}})], \\
\mathbf{D}'_{\mathbf{Events}} & \equiv & \mathbf{D}_{\mathbf{Events}} \wedge \mathit{DOEvent}, \\
\mathbf{D}_{\mathbf{Events}} & \equiv & [\neg \textbf{\textit{Generated Events}} \vee \text{P.I. R{\scriptsize EMAINDER}}] \vee \\
 & & [(\text{P.I. N{\scriptsize EXT} S{\scriptsize MALL} S{\scriptsize TEP}} \vee \text{P.I. N{\scriptsize EXT} C{\scriptsize OMBO} S{\scriptsize TEP}}) \Rightarrow (\text{T{\scriptsize AKE} O{\scriptsize NE}} \wedge \text{M{\scriptsize ANY}})], \ \textit{and} \\
\mathit{DOEvent} & \equiv & [\neg \textbf{\textit{External Output Events}} \vee \text{O.P.I. R{\scriptsize EMAINDER}}] \vee \\
 & & [(\text{O.P.I. N{\scriptsize EXT} S{\scriptsize MALL} S{\scriptsize TEP}} \vee \text{O.P.I. N{\scriptsize EXT} C{\scriptsize OMBO} S{\scriptsize TEP}}) \Rightarrow (\text{T{\scriptsize AKE} O{\scriptsize NE}} \wedge \text{M{\scriptsize ANY}})].
\end{array}$$

Proof Idea. A similar proof of correctness as the one for Proposition 5.10, on page 176, can be developed for this proposition. A sketch of this proof is presented below.

First, a similar lemma to Lemma 5.7, on page 172, can be stated for SBSMLs: If two big steps of an SBSML model that is specified in an SBSML semantics that follows the TAKE ONE big-step maximality semantics and the MANY concurrency semantics have the same set of transitions, they are the same. Predicate $\mathbf{D}'_{\mathbf{Variables}}$ guarantees determinacy with respect to variables because it ensures that either the values of variables in the assignments are obtained from the beginning of a big step, which guarantees determinacy, or the SBSML semantics subscribes to both the TAKE ONE and the MANY semantic options, thus if two big steps have the same set of transitions they are the same. Similarly, the $\mathbf{D}'_{\mathbf{Events}}$ guarantees determinacy with respect to events because either the events are required to accumulate during a big step or the SBSML semantics subscribes to both the TAKE ONE and the MANY semantic options, thus if two big steps have the same set of transitions they are the same.

Conversely, if an SBSML semantics is determinate it should satisfy predicate **D**, otherwise counter example models simile to the ones mentioned in the proof of Proposition 5.10 can be constructed that exhibit non-determinate behaviours.

Thus, an SBSML semantics is determinate iff it satisfies **D**. □

7.6 Summary

This chapter presented a formal semantic definition method for SBSMLs. It also presented transformation schemes, in forms of algorithms, that showed how the semantics of various modelling constructs, as well as, some structural semantic options, can be modelled in SBSMLs. For each transformation scheme, the proof of its correctness was presented. Lastly, the relevance of the semantic quality attributes of BSMLs for SBSMLs was discussed.

Chapter 8

Conclusion and Future Work

"The other point of view sees mathematics as playing primarily an active role. According to this point of view, machines, languages, and systems are (or should be) the computer scientists' own creations, so that they can freely choose to create them to conform to mathematically simple principles. The mathematics is directed toward design rather than study, and mathematics is used not so much to describe existing objects as to plan new ones. This we call the *prescriptive* approach." [5, p.283–284]

Edward Ashcroft and William Wadge

This section presents a brief summary of the dissertation, and then presents a summary of the contributions followed by plans for future work.

This dissertation presents a semantic deconstruction for a wide range of modelling languages that have in common that the reaction of a model specified in them is a big step consisting of a sequence of small steps, each of which is the execution of a set of transitions. The thesis uses the term big-step modelling languages (BSMLs) to refer to this family of modelling languages. The semantic deconstruction distinguishes between these languages based upon eight semantic aspects, each of which is a semantic variation point that has a set of semantic options. The dissertation provides an analysis of the relative advantages and disadvantages of the semantic options of each semantic aspect to enable modellers and language designers to compare two BSMLs and choose one over another, based on the properties of their constituent semantic options.

The dissertation introduces a prescriptive semantic definition framework for formalizing the semantics of BSMLs. A semantics produced in this framework is prescriptive in that the constituent semantic options of the semantics of a BSML are manifested clearly as mainly separate parts of its semantic definition. My goal has been to produce semantic definitions that are understandable and accessible to various stakeholders of a semantics, by the virtue of being partitioned clearly into intuitively meaningful parts.

The dissertation introduces three semantic quality attributes, which represent useful patterns for big steps of a model. These semantic quality attributes are cross-cutting concerns over the semantic aspects of BSMLs. To characterize the BSMLs that satisfy each of these semantic quality attributes, the dissertation presents necessary and sufficient conditions over the choices of the semantic options of the BSMLs that guarantee that semantic quality attribute. The dissertation presents also the outlines of the proofs of the correctness of these characterizations.

Lastly, the dissertation presents a synchronization capability for BSMLs, introducing the class of synchronizing big-step modelling languages (SBSMLs). It presents a semantic definition framework for SBSMLs that is similar to the one for BSMLs, but does not need to consider the role of the concurrency and consistency semantic sub-aspects because one of the two semantic options of each of these sub-aspects can be used to model the other semantic option. The dissertation also shows how SBSMLs can be used to model the semantics of many useful modelling constructs, such as multi-source, multi-destination transitions, some of the template semantics composition operators, and some of the workflow patterns. Algorithms are presented that each is a transformation scheme for modelling one of the aforementioned concurrency and consistency semantic options or modelling constructs. For each of the transformation schemes, the outline of the proof of its correctness is presented.

8.1 Summary of Contributions

The contributions of this dissertation can be summarized by the following five statements.

- The dissertation presents a high-level semantic framework that unifies the semantics of a large family of seemingly different modelling languages, namely, the family of BSMLs. This high-level big-step semantic deconstruction enables one to understand the semantics of a BSML through its constituent semantic options and in comparison to the constituent semantic options of other BSMLs. The big-step semantic deconstruction is accompanied by criteria to differentiate between two semantic options so that one can choose one semantic option out of the several that are possible.

- To provide understandability when formalizing the big-step semantic deconstruction, the dissertation presents a semantic definition framework that prescriptively maps each constituent semantic option of a BSML into a separate part of the semantic definition. This formalization provides a detailed account of the BSML semantics in an accessible way, so that one can trace the formalization of a semantic option to a particular part of a semantic definition.

- The dissertation presents three semantic quality attributes that each distinguishes between two BSMLs based on whether they provide a certain kind of semantic facility for dealing with the ordering of the small steps of a big step or not. The characterization of these semantic quality attributes reveal interrelationships among seemingly independent semantic options in a BSML. They also provide rationales for language design decisions that otherwise would have seemed ad hoc. For example, the specification of non-cancelling BSML semantics highlights the role of concurrency in small-step execution, while the specifications of priority consistent and determinate BSML semantics highlight the role of limiting the number of transitions that each concurrent component of a model can execute in a big step.

- To provide uniformity in dealing with various semantic concepts and various modelling constructs that all use a form of synchronization, the dissertation introduces an explicit synchronization capability to BSMLs, resulting the new family of SBSMLs. The dissertation presents also a formal semantics for SBSMLs in a prescriptive manner, using a novel, declarative way to characterize the semantics of different synchronization types.

- Lastly, the dissertation introduces transformation schemes that each translates a common syntactic construct that is not supported in the normal-form syntax of BSMLs and SBSMLs into a form of synchronization in SBSMLs. These transformation schemes provide the means for a systematic way to design new composition operators, workflow patters, and other syntactic constructs whose semantics can be described using synchronization. Similarly, the dissertation presents transformation schemes that each shows how a certain semantic option can be modelled using an alternative semantic option together with a synchronization mechanism in SBSMLs. These transformation schemes deem some of the semantic aspects of the big-step semantic deconstruction unnecessary when considered for SBSMLs.

8.2 Future Work

I am interested in continuing the research reported in this dissertation in the following five directions.

8.2.1 Including More Languages

I plan to extend the BSML semantic deconstruction framework to include the modelling languages that support asynchronous communication, as taxonomized in Section 2.2.4, on page 25. A modelling language such as UML StateMachines [78] can be considered a BSML, except that events generated in a UML StateMachine model is communicated through asynchronous channels. First, I plan to identify the new semantic aspects and/or semantic options that are required to include these languages in the semantic deconstruction. These semantic aspects and/or semantic options should then be integrated into the existing semantic formalizations of BSMLs and SBSMLs in a prescriptive manner. Also, the semantic quality attributes for BSMLs should be redefined and re-characterized to account for these new languages.

8.2.2 Identifying More Semantic and Syntactic Criteria

The dissertation has introduced semantic criteria to compare two BSMLs to choose one over another when modelling a system under study. These criteria are in the form of advantages and disadvantages of individual semantic options, as well as, in the form of semantic quality attributes, which consider the collective effect of a set of semantic options. Section 5.4.1 described examples of how the syntax and semantics of a BSML can be considered together to achieve a semantic quality attribute in the BSML, instead of considering only the semantic options. I plan to explore more of these hybrid, syntactic and semantic characterizations of semantic quality attributes. Furthermore, using such a hybrid approach, I expect to identify more semantic quality attributes. For example, determinism can be an interesting hybrid, syntactic and semantic quality attribute to be considered for BSMLs.

For SBSMLs, I plan to adapt the complexity results of Joung and Smolka [55] and extend them for the synchronization types introduced in my dissertation. It would be then possible to provide a complexity criteria when choosing to include a synchronization type in a language or when choosing to include a modelling construct whose semantics is based on certain synchronization type(s) in a language.

8.2.3 Identifying Non-Technical Criteria

The semantic criteria presented in this dissertation compare two BSMLs mainly from a technical points of view. They do not consider criteria such as usability of a language in modelling a system under study. A useful research direction is to identify qualitative criteria for the semantics of BSMLs, to differentiate a semantic option from another, or to differentiate two BSML semantics, based on the collective effects of their constituent semantic options, from one another. The identification and the evaluation of each of these criteria in a BSML, however, require designing careful empirical experiments. In particular, these experiments should consider the role of the domain that a certain BSML is being used in. My long-term goal is to create a catalogue of BSMLs and domains with the technical and qualitative criteria that distinguish a language from one another.

8.2.4 A Unifying Framework for the Enabledness Semantic Aspects

The dissertation has shown how the structural semantics aspects, which determine how a set of transitions can be taken together to form a small step, can be uniformly described using synchronization. The enabledness semantic aspects, however, do not enjoy such a unifying semantic definition method. Currently, I am working on a semantic definition language that succinctly and uniformly describes the enabledness semantic aspects. I plan to integrate this language into the semantic definition schema of BSMLs and SBSMLs.

8.2.5 Tool Support

Lastly, the big-step semantic deconstruction can benefit from tool support in two ways.

First, analysis tools for model checking and simulation of BSMLs and SBSMLs can be developed systematically: The operational, prescriptive semantics introduced in this dissertation lends itself to an implementation that can be decomposed into components that each corresponds to a semantic aspect or a semantic option of the big-step semantic deconstruction. When developing such tools, one of my major design goals will be to provide for an effective validation of the implementation through a rigorous method of inspection. I plan to use the experience and effort in similar tool suites that are developed by my colleagues for providing parametric tool support generator frameworks [65, 87].

Second, the big-step semantic deconstruction, its normal-form syntax, its syntactic features, the semantic aspects, and their semantic options can be all formalized in logic, using methods similar to the ones previously developed for formalizing the semantics of modelling languages [24, 25, 75, 74]. Once a logical formalization of my semantic definition framework has been obtained, it is possible to analyze and prove various properties of a BSML, including its semantic quality attributes, formally. My goal will be to develop a formalization framework that can not only be extended with new syntactic and semantic features but also strives for reuse of proofs for languages that have syntactic and/or semantic features in common. Similarly, the syntax and the semantics of SBSMLs can be formalized and analyzed. The transformation schemes, presented in Chapter 7, can also be formalized so that one can prove the correctness of these transformation schemes systematically using theorem provers.

References

[1] The Esterel V7 reference manual version V7.30, initial IEEE standardization proposal. 2005.[1] 50, 57

[2] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of message sequence charts. *IEEE Transactions on Software Engineering (TSE)*, 29(7):623–633, 2003. 78, 174

[3] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999. 2, 16, 36, 37, 40, 60, 62, 64, 66, 70, 71, 81, 83

[4] Edward A. Ashcroft and William W. Wadge. Generality considered harmful: A critique of descriptive semantics. Technical Report CS-79-01, University of Waterloo, Cheriton School of Computer Science, 1979. 3, 9, 86

[5] Edward E. Ashcroft and William W. Wadge. Ŗ for semantics. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(2):283–294, 1982. 3, 5, 9, 86, 276

[6] Luciano Baresi and Mauro Pezzè. Formal interpreters for diagram notations. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 14(1):42–84, 2005. 4, 131

[7] Albert Benveniste, Paul Le Guernic, and Christian Jacquemot. Synchronous programming with events and relations: The SIGNAL language and its semantics. *Science of Computer and Programming*, 16:103–149, 1991. 26

[8] Jan A. Bergstra and Jan Willem Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985. 209

[1]Each bibliographic entry is followed by the list of the pages in the dissertation that reference that entry.

[9] Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors. *The Handbook of Process Algebra*. Elsevier, 2001. 8, 84, 209

[10] Gérard Berry. A hardware implementation of pure ESTEREL. *Sadhana, Academy Proceedings in Engineering Sciences, Indian Academy of Sciences*, 17(1):95–130, 1992. 26, 48, 65

[11] Gérard Berry. Preemption in concurrent systems. In *Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *LNCS*, pages 72–93. Springer, 1993. 43

[12] Gérard Berry. Hardware and software synthesis, optimization, and verification from esterel programs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1217 of *LNCS*, pages 1–3. Springer-Verlag, 1997. 26

[13] Gérard Berry. The constructive semantics of pure esterel draft version 3, 1999. http://www-sop.inria.fr/esterel.org/. 186

[14] Gérard Berry and Georges Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992. 2, 16, 20, 25, 26, 27, 36, 40, 44, 47, 48, 50, 55, 60, 65, 73, 83, 185, 210, 211

[15] Simon Bliudze and Joseph Sifakis. A notion of glue expressiveness for component-based systems. In *19th International Conference Conference on Concurrency Theory CONCUR'08*, volume 5201 of *LNCS*, pages 508–522. Springer, 2008. 210

[16] Frederic Boussinot. Sugarcubes implementation of causality. Technical Report RR-3487, INRIA, Institut National de Recherche en Informatique et en Automatique, 1998. 48, 84

[17] Janusz A. Brzozowski and H. Zhang. Delay-insensitivity and semi-modularity. *Formal Methods in System Design*, 16(2):191–218, 2000. 186

[18] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988. 188, 189

[19] Daniel M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, October 1984. 57

[20] Michelle L. Crane and Juergen Dingel. On the semantics of UML state machines: Categorization and comparison. Technical Report 501, Queens Univerity, 2005. 4

[21] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston, 2000. 30

[22] James Dabney and Thomas L. Harman. *Mastering Simulink*. Pearson Prentice Hall, 2004. 40, 70, 71, 127

[23] Nancy Day. A model checker for statecharts: Linking CASE tools with formal methods. Master's thesis, University of British Columbia, 1993. 47

[24] Nancy A. Day. *A Framework for Multi-Notation, Model-Oriented Requirements Analysis*. PhD thesis, University of British Columbia, 1998. 131, 281

[25] Nancy A. Day and Jeffrey J. Joyce. Symbolic functional evaluation. In *12th International Conference on Theorem Proving in Higher Order Logics TPHOL'99*, volume 1690 of *LNCS*, pages 341–358, 1999. 4, 131, 281

[26] Luca de Alfaro and Thomas A. Henzinger. Interface Automata. In *Proceedings of the Joint 8th European Software Engeneering Conference and 9th ACM SIGSOFT Symposium on the Foundation of Software Engeneering (ESEC/FSE-01)*, volume 26, 5 of *Software Engineering Notes*, pages 109–120. ACM Press, 2001. 26

[27] Jacques Derrida. *Points...: Interviews, 1974-1994*. Stanford University Press, 1 edition, 2 1995. 29

[28] Laura K. Dillon and Kurt Stirewalt. Inference graphs: A computational structure supporting generation of customizable and correct analysis components. *IEEE Transactions on Software Engineering (TSE)*, 29(2):133–150, 2003. 4, 131, 132

[29] Stephen A. Edwards. Compiling Esterel into sequential code. In *37th Conference on Design Automation (DAC)*, pages 322–327, 2000. 26

[30] Rik Eshuis. Reconciling statechart semantics. *Science of Computer Programming*, 74(3):65–99, 2009. 186

[31] Shahram Esmaeilsabzali and Nancy A. Day. Prescriptive semantics for big-step modelling languages. In *13th International Conference on Fundamental Approaches to Software Engineering (FASE'10)*, volume 6013 of *LNCS*, pages 158–172. Springer Verlag, 2010. 14

[32] Shahram Esmaeilsabzali and Nancy A. Day. Semantic quality attributes for big-step modelling languages. In *14th International Conference Fundamental Approaches to Software Engineering (FASE'11)*, volume 6603 of *LNCS*, pages 65–80. Springer Verlag, 2011. 14

[33] Shahram Esmaeilsabzali, Nancy A. Day, and Joanne M. Atlee. A common framework for synchronization in requirements modelling languages. In *13th International Conference on Model Driven Engineering Languages and Systems (MoDELS'10), Part II*, volume 6395 of *LNCS*, pages 198–212. Springer Verlag, 2010. 15

[34] Shahram Esmaeilsabzali, Nancy A. Day, Joanne M. Atlee, and Jianwei Niu. Big-step semantics. Technical Report CS-2009-05, University of Waterloo, Cheriton School of Computer Science, 2009. 14, 41

[35] Shahram Esmaeilsabzali, Nancy A. Day, Joanne M. Atlee, and Jianwei Niu. Semantic criteria for choosing a language for big-step models. In *17th IEEE International Requirements Engineering Conference (RE'09)*, pages 181–190, 2009. 14

[36] Shahram Esmaeilsabzali, Nancy A. Day, Joanne M. Atlee, and Jianwei Niu. Deconstructing the semantics of big-step modelling languages. *Requirements Engineering*, 15(2):235–265, 2010. 14, 174

[37] Colin Fidge. A comparative introduction to CSP, CCS and LOTOS. Technical Report 93-24, The university of Queensland, Department of Computer Science, 1994. 209

[38] Jimin Gao, Mats Per Erik Heimdahl, and Eric Van Wyk. Flexible and extensible notations for modeling languages. In *FASE'07*, volume 4422 of *LNCS*, pages 102–116, 2007. 4, 131, 132

[39] Nicholas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous data flow programming language Lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991. 26

[40] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer, 1993. 25, 26, 44, 48, 84, 185, 210, 211

[41] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987. 2, 11, 13, 19, 36, 40, 71, 73, 197

[42] David Harel and Hillel Kugler. The RHAPSODY Semantics of statecharts (or, On the Executable Core of the UML). In *Integration of Software Specification Techniques forApplications in Engineering*, volume 3147 of *LNCS*, pages 325–354. Springer-Verlag, 2004. 36, 37, 73

[43] David Harel and Amnon Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):293–333, 1996. 24, 36, 37, 47, 48, 60, 73, 76, 83, 186

[44] David Harel and Amir Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*. Springer-Verlag, 1985. 25

[45] David Harel, Amir Pnueli, J. P. Schmidt, and R. Sherman. On the formal semantics of statecharts. In *Proc. of the Second IEEE Symp. on Logic in Computation*, pages 54–64, 1987. 36, 40, 47, 60, 62, 71, 83

[46] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, 1996. 2, 16, 27, 50, 54, 58, 60, 64, 67, 70, 71, 81, 83, 186

[47] K. L. Heninger, J. Kallander, David Lorge Parnas, and J. E. Shore. Software requirements for the A-7E aircraft. Technical Report 3876, United States Naval Research Laboratory, 1978. 2, 16, 27, 50, 54, 58, 60, 64, 67, 70, 71, 81, 186

[48] Tony Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985. 4, 203, 209, 213, 270

[49] Tony Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice Hall, 1998. 5, 16, 69, 133

[50] Cornelis Huizing and Rob Gerth. Semantics of reactive systems in abstract time. In *REX Workshop*, volume 600 of *LNCS*, pages 291–314, 1992. 5, 48, 57, 61, 84, 85, 133, 184

[51] i Logix Inc. *Statemate 4.0 Analyzer User and Reference Manual*, 1991. 41

[52] ISO. LOTOS: a formal description technique based on the temporal ordering of observational behaviour. Technical Report 8807, International Standards Organisation, 1989. 4

[53] Roman Jakobson. The translation studies reader. In Lawrence Venuti, editor, *On linguistic aspects of translation*. Routledge, 2 edition, 8 2004. 135

[54] Ryszard Janicki and Maciej Koutny. Structure of concurrency. *Theoretical Computer Science*, 112(1):5–52, 26 April 1993. 186

[55] Yuh-Jzer Joung and Scott A. Smolka. A comprehensive study of the complexity of multiparty interaction. *Journal of the ACM*, 43(1):75–115, 1996. 209, 210, 280

[56] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, SEI, Carnegie Mellon University, 1990. 21

[57] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, May 1969. 186

[58] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993. 62

[59] Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968. 102

[60] Leslie Lamport. Computer science and state machines. In *Concurrency, Compositionality, and Correctness, Essays in Honor of Willem-Paul de Roever*, volume 5930 of *LNCS*, pages 60–65. Springer, 2010. 16

[61] Leslie Lamport and Fred B. Schneider. Pretending atomicity. Technical Report 44, Digital Equipment Corporation, 1989. 25

[62] Lawrence H. Landweber and Edward L. Robertson. Properties of conflict-free and persistent Petri nets. *Journal of the ACM*, 25(3):352–364, 1978. 186

[63] Nancy G. Leveson, Mats P. E. Heimdahl, Holly Hildreth, and Jon D. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering (TSE)*, 20(9):684–707, 1994. 24, 36, 37, 47, 48, 55, 57, 62, 66, 76, 83

[64] Robert J. Lipton. Reduction: A method of proving properties of parallel programs. *Communications of the ACM*, 18:717–721, 1975. 25

[65] Yun Lu, Joanne M. Atlee, Nancy A. Day, and Jianwei Niu. Mapping template semantics to SMV. In *19th International Conference on Automated Software Engineering (ASE'04)*, pages 320–325, 2004. 4, 131, 132, 281

[66] Andrea Maggiolo-Schettini, Adriano Peron, and Simone Tini. Equivalences of statecharts. In *7th International Conference Conference on Concurrency Theory CONCUR'96*, volume 1119 of *LNCS*, pages 687–702, 1996. 49, 52, 84, 114

[67] Andrea Maggiolo-Schettini, Adriano Peron, and Simone Tini. A comparison of statecharts step semantics. *Theoretical Computer Science*, 290(1):465–498, 2003. 5, 84, 85

[68] Florence Maraninchi and Yann Rémond. Argos: an automaton-based synchronous language. *Computer Languages*, 27(1/3):61–92, 2001. 2, 16, 26, 36, 37, 38, 40, 44, 47, 48, 73, 80, 83, 185, 186, 210, 211

[69] John McCarthy. Towards a mathematical science of computation. In *IFIP '62*. 1962. 100

[70] George J. Milne. Circal and the representation of communication, concurrency, and time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(2):270–298, 1985. 209

[71] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3):267–310, July 1983. Fundamental study. 39, 199, 210

[72] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989. 4, 188, 194, 199, 203, 209, 270

[73] Peter D. Mosses. *Action Semantics*, volume 26 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1992. 133

[74] Jianwei Niu. *Template Semantics: A Parameterized Approach to Semantics-Based Model Compilation*. PhD thesis, University of Waterloo, 2005. 4, 10, 11, 13, 27, 75, 85, 130, 257, 260, 267, 281

[75] Jianwei Niu, Joanne M. Atlee, and Nancy A. Day. Template semantics for model-based notations. *IEEE Transactions of Software Engineering (TSE)*, 29(10):866–882, 2003. 4, 10, 11, 13, 27, 41, 47, 72, 73, 75, 85, 90, 130, 197, 202, 257, 258, 260, 263, 267, 281

[76] Jianwei Niu, Joanne M. Atlee, and Nancy A. Day. Understanding and comparing model-based specification notations. In *11th IEEE International Requirements Engineering Conference (RE'03)*, pages 188–199. IEEE Computer Society, 2003. 85

[77] OASIS-Standard. Web services business process execution language version 2.0. 2007. 205

[78] OMG. OMG Unified Modeling Language (OMG UML), Superstructure, v2.1.2. 2007. Formal/2007-11-01. 8, 36, 37, 186, 211, 279

[79] David L. Parnas and Jan Madey. Functional documents for computer systems. *Science of Computer Programming*, 25(1):19–23, 1995. 53, 64

[80] Doron Peled. On projective and separable properties. *Theoretical Computer Science*, 186(1-2):135–156, 1997. 186

[81] Mauro Pezzè and Michal Young. Constructing multi-formalism state-space analysis tools: Using rules to specify dynamic semantics of models. In *19th International Conference on Software Engineering (ICSE'97)*, pages 239–249, 1997. 4, 131, 132

[82] Jan Philips and Peter Scholz. Compositional specification of embedded systems with statecharts. In Michel Bidoit and Max Dauchet, editors, *Theory and Practice of Software Development (TAPSOFT'97)*, volume 1214 of *LNCS*, pages 637–651. Springer-Verlag, 1997. 47, 84, 201

[83] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002. 23

[84] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977. 186

[85] Amir Pnueli and M. Shalev. What is in a step? In *J.W. De Bakker, Liber Amicorum*, pages 373–400. CWI, 1989. 49, 50, 51, 62, 92

[86] Amir Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In *International Conference on Theoretical Aspects of Computer Software (TACS'91)*, volume 526 of *LNCS*, pages 244–264. Springer, 1991. 11, 13, 20, 28, 36, 40, 47, 49, 55, 60, 70, 71, 73, 83, 84, 92, 116, 185, 186, 197, 211

[87] Adam Prout, Joanne M. Atlee, Nancy A. Day, and Pourya Shaker. Semantically configurable code generation. In *11th International Conference on Model Driven Engineering Languages and Systems (MoDELS'08)*, volume 5301 of *LNCS*, pages 705–720, 2008. 4, 281

[88] Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Models for concurrency: Towards a classification. *Theoretical Computer Science*, 170(1–2):297–348, 15 December 1996. 39

[89] Sandeep K. Shukla and Michael Theobald. Special issue on formal methods for globally asynchronous and locally synchronous (GALS) systems. *Formal Methods in System Design*, 28(2):91–92, 2006. 57

[90] Signe J. Silver and Janusz A. Brzozowski. True concurrency in models of asynchronous circuit behavior. *Formal Methods in System Design*, 22(3):183–203, 2003. 39, 186

[91] J. Spivey. *The Z Notation. A Reference Manual*. Prentice-Hall, 2 edition, 1992. 225

[92] Ali Taleghani and Joanne M. Atlee. Semantic variations among UML StateMachines. In *9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06)*, volume 4199 of *LNCS*, pages 245–259. Springer, 2006. 84

[93] Olivier Tardieu. A deterministic logical semantics for pure Esterel. *ACM TOPLAS*, 29(2):8:1–8:26, 2007. 36, 48, 84, 186, 211

[94] Robert D. Tennent. The denotational semantics of programming languages. *Communications of the ACM*, 19:437–453, 1976. 1

[95] Andrew C. Uselton and Scott A. Smolka. A compositional semantics for statecharts using labeled transition systems. In *5th International Conference Conference on Concurrency Theory CONCUR'94*, volume 836 of *LNCS*, pages 2–17. Springer, 1994. 84

[96] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003. 11, 13, 197, 202, 205

[97] Robert van Glabbeek. The linear time – branching time spectrum. In J. C. M. Baeten and J. W. Klop, editors, *Proceedings of CONCUR'90*, LNCS 458, pages 278–297. Springer-Verlag, 1990. 39

[98] Dániel Varró. A formal semantics of UML statecharts by model transition systems. In *International Conference on Graph Transformations*, volume 2505 of *LNCS*, pages 378–392. Springer, 2002. 132

[99] Michael von der Beeck. A comparison of statecharts variants. In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, volume 863 of *LNCS*, pages 128–148. Springer, 1994. 2, 4, 16, 84

[100] Ludwig Wittgenstein. Major works: Selected philosophical writings. In *The Blue Book*. Harper Perennial Modern Classics, 1 edition, 3 2009. 86

[101] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *Transactions on Software Engineering and Methodology (TOSEM)*, 6(1):1–30, 1997. 53, 64