Semantic Criteria for Choosing a Language for Big-Step Models

Shahram Esmaeilsabzali, Nancy A. Day, Joanne M. Atlee
Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
{sesmaeil,nday,jmatlee}@cs.uwaterloo.ca

Jianwei Niu
Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas, U.S.A 78249
niu@cs.utsa.edu

Abstract

With the popularity of model-driven methodologies, and the abundance of modelling languages, a major question for a requirements engineer is: which language is suitable for modelling a system under study? We address this question from a semantic point-of-view for big-step modelling languages (BSMLs)*. BSMLs are a class of popular behavioural modelling languages in which a model can respond to an input by executing multiple, possibly concurrent, transitions. We deconstruct the operational semantics of a large class of BSMLs into high-level, orthogonal semantic aspects, and analyze the relative advantages and disadvantages of the common semantic options for each of these aspects. Our goal is to empower a requirements engineer to compare and choose an appropriate BSML.*

1 Introduction

With the popularity of model-driven methodologies, and the abundance of modelling languages (and domain-specific languages), a major question for a requirements engineer is: which language is suitable for modelling a system under study (SUS)? We introduce the term *big-step modelling languages (BSMLs)* to characterize a class of popular behavioural modelling languages in which a model can respond to an environmental input by executing a *big-step*, which consists of a sequence of *small-steps*, each of which may contain multiple concurrent transitions. Numerous BSMLs have been introduced (e.g., Statecharts [9] and its variants [31], Synchronous languages [8], and UML Statemachines [25]); many of which have similar syntaxes but subtly different and complicated semantics.

The choice of a BSML for an SUS depends on many factors, including the domain of the SUS, the expertise of the requirements engineer in a class of notations, etc. In this paper, we present the semantic criteria that a requirements engineer should consider when choosing a BSML for modelling a SUS. Our first contribution is a novel deconstruction of the operational semantics of a large class of BSMLs into seven high-level, mostly orthogonal, *semantic aspects*, and an enumeration of the common *semantic options* found in existing BSMLs for each of these aspects. Our second contribution is an analysis of the relative advantages and disadvantages of each semantic option to provide rationale for a requirements engineer to choose one option over another.

Our deconstruction arises from surveying existing BSMLs viewed from the perspective of the big-step as a whole. We separate the operation of a big-step into orthogonal aspects where existing languages have shown variations. We believe these seven aspects capture the essential semantic differences in most existing BSMLs, and thereby empower requirements engineers to compare and choose the most suitable BSML for an SUS. Choosing a set of semantic options involves making trade-offs among considerations such as simplicity, determinism, causality, orderedness, modularity, etc. We envision our work to be used in three ways: (i) as a semantic catalog, to compare the semantics of existing BSMLs and choose an appropriate BSML, (ii) as a semantic scale, to assess the semantic properties of a BSML, and (iii) as a semantic menu, to help design a BSML from scratch.

Our deconstruction is more concise and systematic than previous comparative studies of different subsets of BSMLs (e.g., [17, 8, 31, 4, 29, 30]) because it recognizes a big-step as a whole, rather than only considering its constituent transitions operationally. In our previous work on template semantics [24], we created a formal framework for comparing the semantics of many BSMLs by instantiating a template of 22 parameters that define a small-step. The seven semantic aspects we present here capture cross-cutting dependencies found in the template parameters, creating a deconstruction that defines a big-step directly. This higher level of abstraction isolates the semantic differences between languages more clearly.

The remainder of the paper is organized as follows. In Section 2, we describe the common syntax and common basic semantics that we use throughout the paper. In Section 3, we present our seven semantic aspects for BSMLs,

the semantic options for each, and an analysis of the relative advantages and disadvantages of each semantic option. We use a model of a dialer subsystem of an IP phone device as a running example. In Section 4, we describe the interdependencies between the choices of semantic options. Section 5 compares our work with the related work, including our previous work on template semantics [24]. Finally, in Section 6, we conclude our paper and discuss future work. Our technical report [7] has more detailed discussion, more comprehensive enumerations of the BSMLs that use each semantic option, and more examples.

2 Common Syntax and Basic Semantics

In this section, we present the terminology that we use throughout the paper. We first explain our common syntax, and then describe the common basic semantics, which can be refined by semantic options.

2.1 Syntax

There is a plethora of BSMLs, including those with graphical syntax (e.g., Statecharts variants [31], Argos [22]), and those with textual syntax (e.g., Reactive Modules [1], Esterel [3], SCR [14, 13]). As is usual when studying a class of related notations, we use a syntactic *normal form* that is sufficiently expressive to represent the syntax of other notations [16]. Our normal form syntax is the *composed hierarchical transition system (CHTS)* syntax [24].[1] A *model* is a CHTS and consists of: (i) a *composition tree* whose nodes are distinct *control states*, and (ii) a set of *transitions* between the control states.

Control States: A control state (e.g., $DialDigits$ in Figure 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.[2]

A control state has a *type*, which is either *Basic*, *Or*, or *Concurrent*. A leaf node of a composition tree is a Basic control state. An Or or a Concurrent control state is *hierarchical*, and has *children*, each of which can be of any type. For example, in Figure 1, control state $Dialing$ is a Concurrent control state and has two Or control states, $Dialer$ and $Redialer$. We use the *parent*, *ancestor*, *child*, and *descendant* relations with their usual meanings. The *least common ancestor* of two control states is the lowest control state (closest to the leaves of the composition tree) in the hierarchy of the composition tree that is an ancestor of both. Two control states are *orthogonal* if neither is an ancestor of the other and their least common ancestor is a Concurrent control state. 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. A *hierarchical transition system* (HTS) is a maximal subtree with no Concurrent control states (e.g., $Dialer$ and $Redialer$).

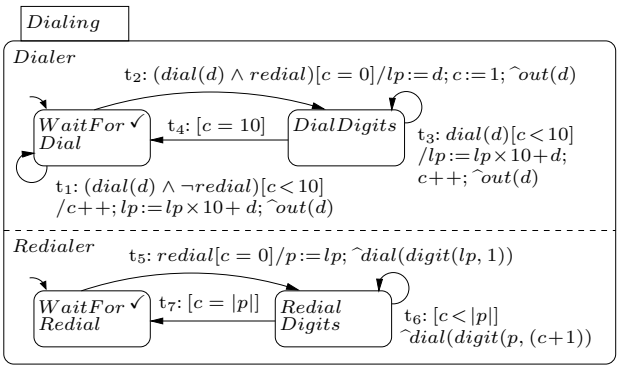

Figure 1. Dialer/Redialer model.

Transitions: A transition (e.g., t_1 in Figure 1) has a *source* and a *destination* control state, and consists of four optional parts: (i) an *event trigger*, which is a conjunction of events and negations of events;[3] (ii) a *variable condition* (enclosed by "[ ]"), which is a boolean expression over the set of variables of the model; (iii) a sequence 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. The types of variables are not relevant, and we assume all models are well-typed. Variables and events are global; local variables and scoped events can be modelled by a renaming that turns them into their global equivalents.

2.2 Common Basic Semantics

Initially, a model resides in the default control states of its Or control states, no events are present, and its variables have their initial values. The operational semantics of a BSML describe 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 illustrates 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.[4] T_i's ($1 \leq i < n$) are small-steps of T, and sp, sp', and sp_i's ($1 \leq i < n$) are its snapshots. Some BSMLs, such as RSML [19], Statemate [11], and Reactive Modules [1], introduce an intermediate grouping of a sequence of small steps, which we call

[1] cf., [24] for the mapping of some BSMLs to the CHTS syntax.

[2] Some BSMLs use the lines of programs to realize control states.

[3] Disjunction can be modelled by splitting a transition into multiple transitions each of which represents one of the disjuncts [28].

[4] Big-steps and small-steps are often called macro-steps and micro-steps, respectively. We adopt new terms to avoid association with the fixed semantics of the languages that use those terms.

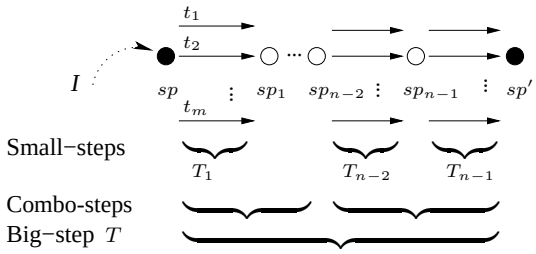

Figure 2. Steps.

a *combo-step*. Sections 3.2 and 3.3 describe when combo-steps are useful.

Snapshots: A *snapshot* is a tuple that 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*. A big-step, small-step, or combo-step has a *source* and a *destination* snapshot (e.g., sp and sp' are source and destination snapshots of T).

Enabledness: In each small step, a set of *enabled* transitions is chosen to be executed. A transition is *enabled* if its event trigger and variable 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.

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 a destination control state (and its descendants). When entering an Or control state, a transition enters its default control state, and when entering a Concurrent control state, it enters all of its children. The semantics of event generation and variable assignment differ between BSMLs. 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 the "SAME" event lifeline option [cf., Section 3.2]). Because of atomicity, a sequence of assignments on a transition can be converted to a set of assignments [18, 20].

When choosing a BSML for modelling an SUS, the domain of the SUS 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: (i) *fast computation* (also known as *synchrony hypothesis* and *zero-time assumption* [3]), which states that the system is fast enough and thus never misses an input; (ii) *helpful environment*, which states that the environment is helpful by not issuing an input when the system is not ready [8]; and (iii) *asynchronous communication*, which states that the system is equipped with a buffering mechanism to store the environmental inputs. The first two assumptions are mutually exclusive with the third one. In this paper, we consider only the BSMLs with the first two assumptions.

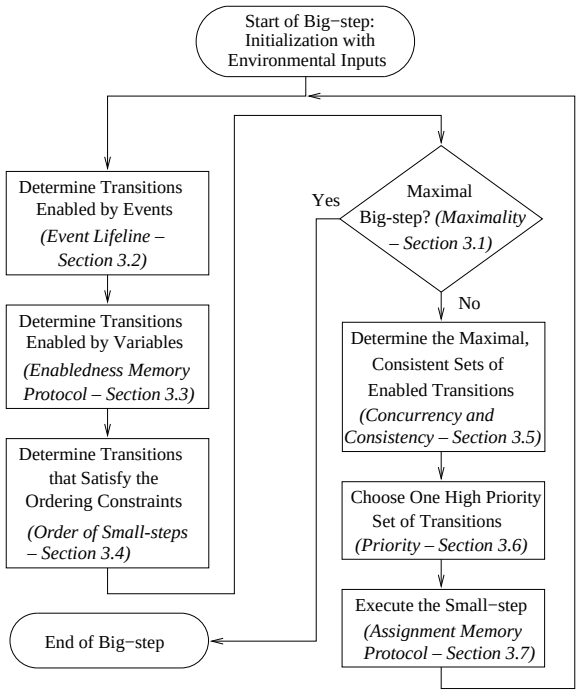

Figure 3. Operation of a big-step.

3 Semantic Aspects

We deconstruct the operation of a big-step into the seven stages described in Figure 3. 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. Each stage of the diagram represents one of our *semantic aspects*. A semantic aspect includes a collection of *semantic options*, each of which is a semantic choice for carrying out a stage.[5]

There are seven semantic aspects: *maximality*, *event lifeline*, *enabledness memory protocol*, *order of small-steps*, *concurrency and consistency*, *priority*, and *assignment memory protocol*. The maximality aspect specifies when a big-step ends, at which point a new big-step starts by sensing the new environmental inputs. The event lifeline aspect specifies how far within a big-step a generated event can be sensed to trigger a transition. The enabledness memory protocol aspect specifies the snapshot from which the values of variables are read to enable the variable condition of a transition. The order of small-steps aspect describes options for the order of transitions within a big-step. From the transitions enabled by events, variables, and ordering constraints, the concurrency and consistency aspect chooses the potential small-steps: first, it specifies whether more than one transition can be taken in a small-step, and

[5]The model in Figure 3 can be extended to include combo-steps by creating an inner loop that controls the maximality of combo-steps.

Table 1. Maximality semantic options.

Options	Advantages	Disadvantages	Examples
SYNTACTIC	Traceable big-steps	Non-terminating big-steps	Esterel [3] (`pause` command), and Rhapsody [10] and UML Statemachines [25] "run-to-completion"
TAKE ONE	Terminating big-steps	Unclear destination	Statecharts [9, 12, 28], Reactive Modules [1], and Argos [22]
TAKE MANY	Can specify sequential computation	Unclear destination and non-terminating big-steps	Statemate [11] and RSML [19]

second, if more than one transition can be taken, it specifies the consistency criteria for including multiple transitions in a small-step. The priority aspect chooses a small-step from the set of potential small-steps. The assignment memory protocol aspect specifies the snapshot from which the value of a variable in the righthand side of an assignment is read.

In the following subsections, we describe each semantic aspect. We summarize the semantic options for each aspect and their relative advantages and disadvantages in a table, which also includes representative BSMLs for each option. These options cover the variations found in most existing BSMLs. We use the SMALL CAP font for the name of semantic options. Throughout the section, we present examples that are meant to demonstrate the differences between semantic options (but not to endorse one over another). The model snippets in our examples are not complete. Finally, Section 3.8 mentions an additional aspect regarding distinguishing environmental and controlled variables/events.

3.1 Maximality

The maximality semantics of a BSML specifies when the sequence of small-steps of a big-step concludes (i.e., when the model becomes *stable*). Table 1 lists the possible semantic options. There are three ways to specify the end of a big-step: (i) the SYNTACTIC option, where a modeller can designate the entrance to a control state or the execution of a transition as the end of a big-step; (ii) the TAKE ONE option, which allows each HTS to take at most one transition in a big-step;[6] and (iii) the TAKE MANY option, which allows a sequence of small-steps to continue executing until there are no more enabled transitions to be executed. The TAKE MANY option is useful for modelling a computation that consists of multiple sequential steps within a HTS.

Scope of a big-step: In the TAKE ONE and the TAKE MANY options, the end of a big-step is not obvious, which can be confusing for a modeller. In the SYNTACTIC option, the end of a big-step can be traced syntactically, but a big-step might have the option to conclude or continue. In the SYNTACTIC and TAKE MANY options, it is possible for a big-step to never terminate. 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., [25, 10]).

[6]The execution of a transition whose source is a Concurrent control state is counted towards all of its descendant HTSs.

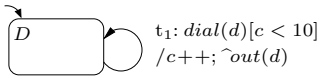

Figure 4. Dialer system.

Combo-step maximality: The same semantic options can be used for the maximality of combo-steps (but usually the TAKE ONE and TAKE MANY options are chosen for combo-step and big-step maximality, respectively [11, 19]).

Example 1 *The model in Figure 4 collects a dialed digit of a phone device (environmental input event* $\mathrm{dial(d)}$*) and transmits the dialed digit* d *to the IP network via generated event* $\mathrm{out(d)}$*. 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 (we do not model the connection functionality of the system). The "*$++$*" operator denotes increment by one.*

In a semantics where event $\mathrm{dial(d)}$ *persists during a big-step (i.e., if received, it is always present), and* c *is zero, then if the* TAKE MANY *option is chosen, upon receiving the first digit, the same digit is sent 10 times. If the* SYNTACTIC *option is chosen and entering* D *ends a big-step, or the* TAKE ONE *option is chosen, the model behaves correctly.*

3.2 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. The maximum lifeline of an event is the big-step in which it is generated. There are five lifeline semantics (shown in Table 2): (i) in the WHOLE option, a generated event is present throughout its big-step, from the beginning of its big-step; (ii) in the 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 NEXT COMBO-STEP option, a generated event is present during the next combo-step; (iv) in the NEXT SMALL-STEP option, a generated event is present in the next snapshot; and (v) in the SAME option, a generated event is communicated instantaneously only during

Table 2. Lifeline semantics.

Options	Advantages	Disadvantages	Examples
WHOLE	Modularity and global consistency	Non-causality	Argos [22] and Esterel [3]
REMAINDER	Causality	Unorderedness and global inconsistency	Statecharts [12, 28]
NEXT COMBO-STEP	Causality and some level of orderedness	Unclear scope of combo-steps and multiple-instance events	Statemate [11] and RSML [19]
NEXT SMALL-STEP	Causality and orderedness	Multiple-instance events	Statecharts[6]
SAME	Instantaneous communication	Non-causality and multiple-instance events	Used in [24]

the small-step in which it is generated.[7] *Implicit events*, such as `entered(`s`)` [27] (generated when control state s is entered) or `@T(`$cond$`)` [14, 13] (generated when the value of condition $cond$ changes from false to true), may have the same semantics as the event lifeline semantics of named events, or not.

Multiple-instance events: The last three lifeline semantics allow multiple instances of the same event, generated by different small-steps, to exist in the same big-step. These semantics are complex in that the status of an event can change multiple times in a big-step.

Casuality: A big-step is *causal* if its small-steps can be sequenced such that the small-steps prior to taking a transition generate any events that are sensed as present by the transition. To a modeller, the transitions of a non-causal big-step may seem to execute out of the blue.

Orderedness: The REMAINDER semantics lacks a "rigorous causal ordering" [19]: if event e_1 is generated earlier than event e_2, then transitions that are triggered by e_1 do not necessarily execute earlier than the ones triggered by e_2. The 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 [11, 19]. A disadvantage 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.

Modularity: The WHOLE option is *modular* [17] with respect to events: a generated event 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. In a non-modular lifeline semantics, a part of a model cannot play the role of the environment for another part, which means that a model cannot be constructed incrementally. Extensions of the model may change the behaviour in different ways than the environment does. Therefore, all parts of a model should be created together.[8]

[7] The SAME semantics is usually used in process algebras (e.g., CCS [23] and CSP [15]), but can also be adapted for BSMLs [24].

[8] In [17], modularity is defined only for events, but, in the same spirit, we extend it to other parts of syntax in other sections.

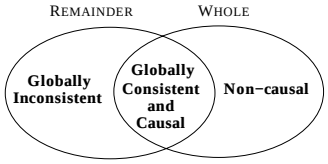

Figure 5. Global consistency vs. causality.

Global inconsistency: The REMAINDER option can produce *globally inconsistent* big-steps [27, 28]. A big-step is globally inconsistent if it includes a transition that generates an event and a transition triggered by the negation 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. It is possible to avoid a globally inconsistent big-step by not taking a transition that generates an event that was sensed as absent earlier [21].

Global consistency vs. causality: Figure 5 shows the relationship between the big-steps of the REMAINDER semantics and the WHOLE semantics. A big-step T that is produced by a globally consistent REMAINDER semantics can also be produced by a WHOLE semantics because T's generated events, by the definition of global consistency, can be assumed to be present from the beginning of the big-step. Conversely, a big-step T' that is produced by a causal WHOLE semantics can also be produced by a REMAINDER semantics because, by the definition of causality, an event is 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 REMAINDER semantics generates a subset of the big-steps of the WHOLE semantics.

Example 2 *The model in Figure 1 is an extension of the model in Figure 4 to support a "redial" functionality. Variable* lp *stores the last dialed phone number. Upon receiving the* redial *environmental input,* $\mathrm{Redialer}$ *instructs* Dialer*, by generating the corresponding* dial *events, to dial the digits of* lp*. (We denote the size of an integer* x *as* $|\mathrm{x}|$ *and its* n*th digit as* $\mathrm{digit}(\mathrm{x},\mathrm{n})$*.) Variable* p *is necessary because once redialling starts* lp *is overwritten. Consider the snapshot where the environmental input event* redial *is received,*

Table 3. Enabledness memory protocols.

Options	Advantages	Disadvantages	Examples
BIG-STEP	Non-interference and modularity	Non-sequentiality	Statecharts [12], SCR [14, 13], and Reactive Modules [1]
SMALL-STEP	Sequentiality	Interference	Esterel [3] and SCR [14, 13]
COMBO-STEP	Non-interference and sequentiality	Unclear scope of combo-steps	Statemate [11]

c *is zero, and* $|\mathrm{lp}|$ *is 10. The environmental input event* redial*, when received, persists throughout the big-step. A semantics that follows the* SYNTACTIC *maximality semantics (annotating a designated control state with a "✓"), the* NEXT SMALL-STEP *event lifeline semantics, and uses the up-to-date values of variables, can produce the big-step* t_5, $\{\mathrm{t}_2, \mathrm{t}_6\}, \{\mathrm{t}_3, \mathrm{t}_6\}, \cdots, \{\mathrm{t}_3, \mathrm{t}_6\}, \{\mathrm{t}_4, \mathrm{t}_7\}$*, which transmits the first digit twice and does not transmit the last digit. The* SAME *event lifeline semantics produces the correct big-step* $\{\mathrm{t}_5, \mathrm{t}_2\}$, $\{\mathrm{t}_3, \mathrm{t}_6\}$, $\cdots$, $\{\mathrm{t}_4, \mathrm{t}_7\}$.[9] *Other semantic aspects that contribute to the behaviour of this model are described in the following subsections.*

3.3 Enabledness Memory Protocol

The *enabledness memory protocol* of a BSML determines the values of variables that a transition *reads* for its variable condition (VC). There are three possible memory protocols (shown in Table 3): (i) in the BIG-STEP option, a read of a variable returns its value at the beginning of the big-step; (ii) in the SMALL-STEP option, a read of a variable returns its value from the end of the last small-step; and (iii) in the COMBO-STEP option, a read of a variable returns its value at the beginning of the current combo-step.

Non-interference vs. sequentiality: The 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. The SMALL-STEP option, which is an "interfering" semantics, is useful for specifying a sequence of computation where each small-step reads the values from the previous small-step. The COMBO-STEP option enjoys non-interference inside a combo-step and sequentiality when a sequence of combo-steps is considered. In the COMBO-STEP option, it is not straightforward to determine the scope of each combo-step.

Syntactic keywords: 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 4 lists some common operators and specifies whether they are *total* or not. A non-total operator may *block* until it can be evaluated. 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 (cf., Section 3.4). Operator `new_small` returns the value of its operand at the end of the current small-step. By definition, operator `pre` is not relevant for the BIG-STEP memory protocol and operator `cur` is not relevant for the SMALL-STEP memory protocol.

[9] In both cases, if the size of the redialled number is less than 10, the model cannot stabilize, and remains in $DialDigits$ control state.

Table 4. Variable operators.

Operator	Obtains Value From	Total
`pre` (e.g., [19])	big-step source snapshot	✓
`cur` (e.g., [12])	small-step source snapshot	✓
`new` (e.g., [1])	small-step source snapshot	✗
`new_small` (e.g., [27])	small-step destination snapshot	✓

Example 3 *In Example 2, we used the* SMALL-STEP *memory protocol. If we use the semantic options that lead to an incorrect behaviour, but modify the VC of* t_6 *to "*$[\mathrm{new\_small}(\mathrm{c}) < |\mathrm{p}|]$*" and its event generation to "*$\mathrm{dial}(\mathrm{digit}(\mathrm{new\_small}(\mathrm{c}) + 1, \mathrm{p}))$*", then the model behaves correctly:* t_5, $\{\mathrm{t}_2, \mathrm{t}_6\}$, $\{\mathrm{t}_3, \mathrm{t}_6\}$, $\cdots$, t_3, $\{\mathrm{t}_4, \mathrm{t}_7\}$.

3.4 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 variable conditions and event triggers, some BSMLs non-deterministically execute one (the NONE option), while others order their executions either by syntactic means (the EXPLICIT option) or by *dataflow* orders (the DATAFLOW option). Stateflow is an example of the EXPLICIT option because the transitions of a model are executed according to the graphical, clockwise order of its HTSs [5]. A dataflow order allows only those orders of small-step executions in which a transition that writes to a variable is executed before transitions that read the variable. The dataflow order of a model can be specified by an explicit partial order between its variables (e.g., SCR [14, 13]), or via variable operator `new`, as described in Section 3.3, to determine data dependencies (e.g., Reactive Modules [1]). In the latter case, each big-step of a model might have a different dataflow order. The EXPLICIT and DATAFLOW options can be used to avert undesired non-determinism by disallowing the execution of the small-steps that do not satisfy the ordering constraints. The EXPLICIT 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.

Table 5. Order of small-steps semantic options.

Options	Advantages	Disadvantages	Examples
NONE	Simplicity	Non-determinism	Statecharts [9, 12, 28]
EXPLICIT	Control over ordering and greater determinism	Possible unintended orders	Stateflow [5]
DATAFLOW	Natural for some domains and greater determinism	Possible cyclic orders	SCR [14, 13] and Reactive Modules [1]

Table 6. Concurrency and consistency semantic options.

Options	Advantages	Disadvantages	Examples
Concurrency			
SINGLE	Simplicity	Non-determinism	Statecharts [9, 12, 28], Stateflow [5], and Reactive Modules [1]
MANY	Greater determinism	Race conditions	Argos [22] and Esterel [3]
Small-step consistency			
ARENA ORTHOGONAL	Simplicity	Non-determinism	Statecharts [9, 12]
SOURCE/DESTINATION ORTHOGONAL	Greater determinism	Complex	Statemate [11] and UML Statemachines [25]
Preemption			
NON-PREEMPTIVE	Supports "last wish"	Complicated flow of control	Argos [22] and Esterel [3]
PREEMPTIVE	Simple flow of control	No support for "last wish"	Statecharts [28]

Example 4 *Consider the semantic options in Example 2 that lead to an incorrect behaviour. Another way to fix the incorrect behaviour is to modify the model by moving the "*$\mathrm{p} := \mathrm{lp}$*" assignment from* t_5 *to* t_2*, changing the VC of* t_6 *to "*$\mathrm{c} < |\mathrm{new}(\mathrm{p}) - 1|$*", and its event generation to "*$\mathrm{dial}(\mathrm{digit}(\mathrm{new\_small}(\mathrm{c}) + 1, \mathrm{p}))$*". Such a model then behaves correctly:* t_5, t_2, t_6, $\{\mathrm{t}_3, \mathrm{t}_6\}$, $\cdots$, t_3, $\{\mathrm{t}_4, \mathrm{t}_7\}$*, because the dataflow order does not allow* t_2 *and* t_6 *to be executed together.*

3.5 Concurrency and Consistency

BSMLs vary in how the enabled transitions of a model execute together. Table 6 lists the three concurrency and consistency semantic sub-aspects that specify: (i) whether more than one transition can be taken in a small-step, and if so, (ii) which transitions can be taken together, considering the composition tree of a model, and (iii) whether the execution of one transition in a small-step can *preempt* the execution of another transition or not.

3.5.1 Concurrency

There are two options: (i) a small-step can execute one transition at a time (the SINGLE option), and (ii) all enabled transitions that can be taken together are taken in a small-step (the MANY option). 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 assigns values to a variable. Typically, one of the assignments is chosen non-deterministically [24], but there are other options [7].

Example 5 *In Example 2, we used the* MANY *concurrency semantics. If we use the* SINGLE *option, instead of the* MANY *option, then the model in Figure 1 can create both a correct big-step and an incorrect big-step (e.g.,* t_5, t_2, t_3, $\cdots$, t_7, t_4*), non-deterministically. However, if we use the* EXPLICIT *order of small-step semantics according to the graphical, clockwise order of the HTSs, then the model always behaves correctly:* t_5, t_2, t_6,t_3, t_6, t_3, $\cdots$, t_7, t_4.

Next, we consider two semantic sub-aspects that are relevant when the MANY semantics is chosen, and specify the set of transitions that can be taken together in a small-step. The *small-step consistency* sub-aspect deals with transitions that do not *interrupt* each other and the *preemption* sub-aspect deals with transitions that do *interrupt* each other.

3.5.2 Small-step Consistency

In the SOURCE/DESTINATION ORTHOGONAL option, transitions whose sources and destinations 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 the ancestor of the source and destination control states

Table 7. Priority semantic options.

Options	Advantages	Disadvantages	Examples
HIERARCHICAL	Simplicity	Incomplete prioritization	‘‘Arena-parent” in Statemate [11]
EXPLICIT	Greater control	Tedious to use	Used in [24]
NEGATION OF TRIGGERS	Greater control, no additional syntax	Tedious to use	Statecharts [28], Esterel [3], and Argos [22]

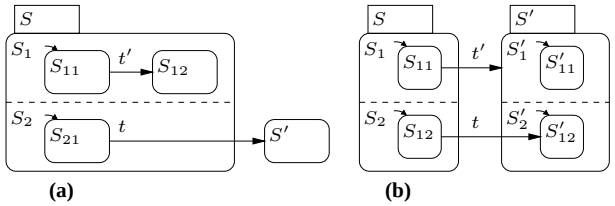

Figure 6. Interrupting transitions.

of the transition. In comparison, the ARENA ORTHOGONAL option is simpler than the SOURCE/DESTINATION ORTHOGONAL option, but it can introduce undesired nondeterminism by not taking all of the enabled transitions that the SOURCE/DESTINATION ORTHOGONAL option takes.[10]

3.5.3 Preemption

The notion of *preemption* [2] is relevant for a pair of transitions if they cannot be taken together according to the small-step consistency semantics. 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 6(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 6(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. When the NON-PREEMPTIVE semantics is considered, the destination configuration of a small-step that includes such a t and t' is determined by t's destination (i.e., the destination of t' is not relevant). While complex, the NON-PREEMPTIVE option satisfies the "last wishes" of the children of a Concurrent control state that is interrupted.[11]

Example 6 *We extend the model in Figure 1 to disallow a call to be initiated if the "limit" number of concurrent calls is reached (determined by environmental boolean input variable* limit*). Consider a new, high priority transition* t' *whose source is* $\mathrm{Redialer}$*, its destination is a new control state* S*, which is not ancestrally related to* $\mathrm{Dialing}$*, and its variable condition is "*$[\mathrm{limit} = \mathrm{true}]$*". If* limit *is true and a caller dials the last digit of a phone number, then the* PREEMPTIVE *option aborts the dialing, but the* NON-PREEMPTIVE *option allows the call to go through.*

3.6 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 7 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.

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. For example, "arena-parent" is a priority semantics that gives a higher priority to a transition whose arena is the highest in the hierarchy of a composition tree. The EXPLICIT 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 priorityThe NEGATION OF TRIGGERS option is not an independent way to assign priority, but uses the notion of "negation" to assign priorities: t_1 can be assigned a higher priority than t_2 by conjuncting the negation of the event trigger and variable condition of t_2 with the ones of t_1, respectively.

Exhaustiveness vs. simplicity: The HIERARCHY option can be easily understood by a modeller, but may render many transitions as priority-incomparable. The EXPLICIT option provides a great control over specifying the relative priority of a set of transitions, but can be tedious to use (e.g., a wrong relative priority for a pair of transitions can be deduced transitively). In the NEGATION OF TRIGGERS and EXPLICIT 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.

Example 7 *In the model in Figure 1,* t_2 *is assigned a higher priority than* t_1 *by conjoining the original event trigger of* t_1*,* $\mathrm{dial(d)}$*, with the negation of the event trigger of* t_2*,* $\mathrm{dial(d)} \wedge \mathrm{redial}$*, resulting in* t_1 *having the event trigger* $\mathrm{dial(d)} \wedge \neg\mathrm{redial}$*. The effect is that* t_2 *will be chosen when the* redial *event occurs instead of* t_1*.*

[10] The same semantic options can be defined at the big-step scope, without requiring the MANY semantics (e.g., ARENA ORTHOGONAL in [28]).

[11] The NON-PREEMPTIVE semantics can be used to model the "weak preemption" semantics of `exit` and `trap` statements in Esterel [3, 8].

3.7 Assignment Memory Protocol

The *assignment memory protocol* of a BSML determines the values of variables that a transition reads for the right-hand side of its assignments. Exactly the same semantic options as those of the enabledness memory protocol, as described in Section 3.3, are possible for the assignment memory protocol. The enabledness and assignment memory protocols of a BSML need not be the same (e.g., SCR [14, 13]).

Example 8 *Consider the semantic options in Example 2 that lead to a correct behaviour. If we use the* BIG-STEP *memory protocol for both enabledness and assignment, we do not need variable* p *because a read of* lp *returns its value at the beginning of the big-step. But then to read the current value of a variable, we must prefix it by the* `cur` *operator.*

3.8 External Communication

The model in Figure 1 uses event $dial$ in two different ways: (i) as an environmental input event initiated by a caller, and (ii) as an internal event generated by the $Redialer$. To avoid modelling flaws, many have advocated that the interface of a model with its environment should be explicitly specified [26, 32]. A celebrated way to achieve this interface is to distinguish syntactically between the elements (i.e., events and/or variables) that the environment provides and the elements that the model can control. The controlled elements can be further partitioned into elements that are observable by the environment, observable by some *components* of a model, and local elements. The semantics of these types of elements vary and are described in our technical report [7].

4 Interdependencies

The hierarchical structure of our aspects captures most of the interdependencies between semantic aspects and options. In this section, we describe five cross-cutting interdependencies.

A sufficient (but not necessary) condition for an unambiguous DATAFLOW semantics is to require the TAKE ONE maximality semantics with each variable assigned value by at most one HTS, as is done in SCR [14, 13] and Reactive Modules [1]. A DATAFLOW semantics is ambiguous if a variable is assigned a value more than once in a big-step. Similarly, for the EXPLICIT semantics when the order of HTSs determine the order of small-steps, the TAKE ONE maximality semantics is usually required.

Frequently, the SINGLE concurrency semantics is chosen with the EXPLICIT order of small-step semantics when the EXPLICIT ordering permits 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.

The use of the SAME event lifeline semantics only makes sense with the use of the MANY concurrency semantics so that the transition that generates the event and the transition that senses the event execute concurrently.

Using the NEXT SMALL-STEP event lifeline semantics and the SINGLE concurrency semantics together has the effect that an enabled transition may not have the chance to execute because it only remains enabled for one small-step. This effect can be a source of undesired non-determinism. A similar effect exists if the SMALL-STEP enabledness memory protocol is chosen with the SINGLE option.

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 has the effect of giving interrupt transition t in Figure 6(b) a higher priority than t'. Similarly, the "arena-parent" priority semantics gives transition t in Figure 6(a) a higher priority than transition t'.

5 Related Work

We cover a more comprehensive class of BSMLs and range of BSML semantics than found in related work. Relative to previous comparative studies of different subsets of BSMLs (e.g., Statecharts variants [31, 17], Synchronous languages [8], Esterel variants [4, 30], and UML Statemachines [29]), we isolate the essential semantic aspects in a language-independent manner and in terms of the big-step as a whole. Huizing and Gerth [17] compare simple BSMLs that have only events, covering most of the event lifeline semantic options and the observability of events among components. In our deconstruction, we are able to describe these options more concisely and place them in the context of other semantics aspects for BSMLs.

By considering a big-step as a whole, we have raised the level of abstraction of the semantic variations compared to our previous work on template semantics [24]. The composition operators of template semantics are modelled via the concurrency and consistency, and event lifeline semantic aspects. For example, the *interleaving* and *parallel* composition operators correspond to the SINGLE and MANY semantic options, respectively; and the *rendezvous* composition operator is represented via the SAME event lifeline semantics and the MANY concurrency semantics. The *interrupt* composition operator is modelled via the small-step consistency and preemption semantic options. By relating parts of the behaviour of composition operators to the step semantic aspects, we provide a foundation for understanding the range of possible composition operators.

6 Conclusion and Future Work

We have presented a novel deconstruction of the semantics of big-step modelling languages into seven high-level, mostly orthogonal semantic aspects. We analyzed the relative advantages and disadvantages of the common semantic options of each aspect. The design/choice of a language involves making tradeoffs between different options. Our framework empowers requirements engineers and language designers to make such tradeoffs in an informed way. For example, if averting non-determinism is desirable, semantics that permit race condition, unordered execution of small-steps, SINGLE concurrency, non-prioritized transitions, etc. are less suitable choices. SCR [14, 13] is an example of a BSML with simpler semantics than many others because its lack of hierarchical control states means it does not require the semantic aspects of small-step consistency, preemption, and priority. Using our aspects and options, we can create languages that do not currently exist. For example, the semantics in Example 5, which avoids the undesired non-determinism of the SINGLE concurrency semantics, is not found in an existing BSML.

We are creating a formal language to describe our semantic aspects concisely. In the future, we plan to map these options to formal parameters that implement the big-step at the small-step level, such as those found in template semantics, to implement tool suites to support BSMLs. We believe our work will be useful in understanding how semantic choices affect the simplicity and performance of analysis tools.

Acknowledgments

We thank the reviewers for their insightful comments, which have improved our paper.

References

[1] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[2] G. Berry. Preemption in concurrent systems. In *Proc. of FSTTCS*, volume 761 of *LNCS*, pages 72–93. Springer, 1993.

[3] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Sci. of Comp. Prog.*, 19(2):87–152, 1992.

[4] F. Boussinot. Sugarcubes implementation of causality. Technical Report RR-3487, Inria, Institut National de Recherche en Informatique et en Automatique, 1998.

[5] J. Dabney and T. L. Harman. *Mastering Simulink*. Pearson Prentice Hall, 2004.

[6] N. Day. A model checker for Statecharts: Linking CASE tools with formal methods. Master's thesis, University of British Columbia, 1993.

[7] S. Esmaeilsabzali, N. A. Day, J. M. Atlee, and J. Niu. Big-step semantics. Technical Report CS-2009-05, University of Waterloo, D.R. Cheriton School of Computer Science, 2009.

[8] N. Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers; Boston, 1993.

[9] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. of Comp. Prog.*, 8(3):231–274, 1987.

[10] D. Harel and H. Kugler. The RHAPSODY Semantics of Statecharts (or, On the Executable Core of the UML). In *Integration of Soft. Spec. Techniques for Appl. in Eng.*, volume 3147 of *LNCS*, pages 325–354. Springer Verlag, 2004.

[11] D. Harel and A. Naamad. The STATEMATE semantics of Statecharts. *ACM TOSEM*, 5(4):293–333, 1996.

[12] D. Harel, A. 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.

[13] C. Heitmeyer, R. Jeffords, and B. Labaw. Automated consistency checking of requirements specifications. *ACM TOSEM*, 5(3):231–261, 1996.

[14] K. L. Heninger, J. Kallander, D. L. Parnas, and J. E. Shore. Software requirements for the A-7E aircraft. Technical Report 3876, United States Naval Research Laboratory, 1978.

[15] T. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[16] T. Hoare and H. Jifeng. *Unifying Theories of Programming*. Prentice Hall, 1998.

[17] C. Huizing and R. Gerth. Semantics of reactive systems in abstract time. In *Proc. of the Real-Time: Theory in Practice, REX Workshop*, pages 291–314. Springer Verlag, 1992.

[18] L. Lamport and F. B. Schneider. Pretending atomicity. Technical Report 44, Digital Equipment Corporation, 1989.

[19] N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, and J. D. Reese. Requirements specification for process-control systems. *IEEE TSE*, 20(9):684–707, 1994.

[20] R. J. Lipton. Reduction: A method of proving properties of parallel programs. *Communications of the ACM*, 18:717–721, 1975.

[21] A. Maggiolo-Schettini, A. Peron, and S. Tini. Equivalences of Statecharts. In *Proc. CONCUR*, pages 687–702, 1996.

[22] F. Maraninchi and Y. Rémond. Argos: an automaton-based synchronous language. *Comp. Lang.*, 27(1/3):61–92, 2001.

[23] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[24] J. Niu, J. M. Atlee, and N. A. Day. Template semantics for model-based notations. *IEEE TSE*, 29(10):866–882, 2003.

[25] OMG. OMG Unified Modeling Language (OMG UML), Superstructure, v2.1.2. 2007. Formal/2007-11-01.

[26] D. L. Parnas and J. Madey. Functional documents for computer systems. *Sci. of Comp. Prog.*, 25(1):19–23, 1995.

[27] A. Pnueli and M. Shalev. What is in a step? In *J.W. De Bakker, Liber Amicorum*, pages 373–400. CWI, 1989.

[28] A. Pnueli and M. Shalev. What is in a step: On the semantics of Statecharts. In *TACS*, pages 244–264, 1991.

[29] A. Taleghani and J. M. Atlee. Semantic variations among UML StateMachines. In *MoDELS*, volume 4199 of *LNCS*, pages 245–259. Springer, 2006.

[30] O. Tardieu. A deterministic logical semantics for pure Esterel. *ACM Trans. on Prog. Lang. and Systems*, 29(2):8:1–8:26, 2007.

[31] M. von der Beeck. A comparison of Statecharts variants. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *LNCS*, pages 128–148. Springer, 1994.

[32] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM TOSEM*, 6(1):1–30, 1997.