“Semantic Criteria for Choosing a Language for Big-Step Models”

Cite as:


BibTex:

@inproceedings{Esmaeilsabzali:2009:SCC:1683309.1684864,  
author = {Esmaeilsabzali, Shahram and Day, Nancy A. and Atlee, Joanne M. and Niu, Jianwei},  
title = {Semantic Criteria for Choosing a Language for Big-Step Models},  
booktitle = {Proceedings of the 2009 17th IEEE International Requirements Engineering Conference, RE},  
series = {RE '09},  
year = {2009},  
pages = {181--190}  
}

DOI: https://doi.org/10.1109/RE.2009.29
Semantic Criteria for Choosing a Language for Big-Step Models

Shahram Esmaeilsabzali, Nancy A. Day, Joanne M. Atlee
David R. 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 address this question from a semantic point-of-view for a large class of BSMLs. 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 crosscutting 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 covers the related work. 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]. 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.

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 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 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

\[ \text{Dialing} \]

\[ t_2: (\text{dial}(d) \land \text{redial})(e = 0) | lp := d; c := 1; \neg \text{out}(d) \]

Figure 1. Dialer/Redialer Model.

Concurrent 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 state. A hierarchical transition system (HTS) is a maximal subtree with no Concurrent control states (e.g., Dialer and Redialer).

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 negation of events; (ii) a variable condition (enclosed by \( [\cdot] \)), which is a boolean expression over the set of variables of the model; (iii) a sequence of assignments (prefixed by a \( \sim \)); and (iv) a set of generated events (prefixed by a \( \sim \)). 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 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, and a snapshot is a tuple that stores information.

\[ T_i \text{'s (}1 \leq i < n\text{)} \]

are small-steps of \( T \), and \( sp, sp' \), and \( sp_1 \text{'s (}1 \leq i < n\text{)} \)

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

\[ \text{Dialer} \]

\[ t_2: (\text{dial}(d) \land \text{redial})(e = 0) | lp := d; c := 1; \neg \text{out}(d) \]

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

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.
There are three types of assumptions: (i) BSML regarding its ability to take multiple transitions in domain of the SUS must satisfy the assumptions of the section can be converted to a set of assignments \[18, 20\].

Concurrent control state, it enters all of its children. When entering an \(\Delta\) 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 is chosen to be executed. A transition is

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 descendents), and enters a destination control state (and its descendents). When entering an \(\Delta\) 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 its 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.

3 Semantic Aspects

We deconstruct the operation of a big-step into 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 the environment is sensed for new environmental inputs, and a new big-step begins. 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:

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

Table 1. Maximality Semantic Options.

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]).

**Combos-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 dial(d)) and transmits the dialed digit d to the IP network via generated event out(d). Variable c allows maximum 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.

Assume dial(d) persists during a big-step (i.e., if received, it is always present), and c is zero. If the TAKE MANY option is chosen, then upon receiving the first digit, the model sends the same digit 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) 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.
Advantages

Causality and orderedness
Argos [22] and Esterel [3]

Examples

Unorderedness and
Modularity and global consistency

Disadvantages

Non-causality
Statecharts [6]

Multiple-instance events
Statecharts [12, 28]

Table 2. Lifeline Semantics.

<table>
<thead>
<tr>
<th>Options</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHOLE</td>
<td>Modularity and global consistency</td>
<td>Non-causality</td>
<td>Argos [22] and Esterel [3]</td>
</tr>
<tr>
<td>REMAINDER</td>
<td>Causality</td>
<td>Unorderedness and global inconsistency</td>
<td>Statecharts [11, 28]</td>
</tr>
<tr>
<td>NEXT_COMBO-STEP</td>
<td>Causality and some level of orderedness</td>
<td>Unclear scope of combo-steps and multiple-instance events</td>
<td>Statemate [11] and RSML [19]</td>
</tr>
<tr>
<td>NEXT_SMALL-STEP</td>
<td>Causality and orderedness</td>
<td>Multiple-instance events</td>
<td>Statecharts [6]</td>
</tr>
<tr>
<td>SAME</td>
<td>Instantaneous communication</td>
<td>Non-causality and multiple-instance events</td>
<td>Used in [24]</td>
</tr>
</tbody>
</table>

Figure 5. Global Consistency vs. Causality.

Table 2. Lifeline Semantics.

<table>
<thead>
<tr>
<th>Options</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHOLE</td>
<td>Modularity and global consistency</td>
<td>Non-causality</td>
<td>Argos [22] and Esterel [3]</td>
</tr>
<tr>
<td>REMAINDER</td>
<td>Causality</td>
<td>Unorderedness and global inconsistency</td>
<td>Statecharts [11, 28]</td>
</tr>
<tr>
<td>NEXT_COMBO-STEP</td>
<td>Causality and some level of orderedness</td>
<td>Unclear scope of combo-steps and multiple-instance events</td>
<td>Statemate [11] and RSML [19]</td>
</tr>
<tr>
<td>NEXT_SMALL-STEP</td>
<td>Causality and orderedness</td>
<td>Multiple-instance events</td>
<td>Statecharts [6]</td>
</tr>
<tr>
<td>SAME</td>
<td>Instantaneous communication</td>
<td>Non-causality and multiple-instance events</td>
<td>Used in [24]</td>
</tr>
</tbody>
</table>

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 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(c) [14, 13] (generated when the value of variable c 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 e1 is generated earlier than event e2, then transitions that are triggered by e1 do not necessarily execute earlier than the ones triggered by e2. 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

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 be also 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 be also 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 “redial” functionality. Variable lp stores the last dialed phone number. Upon receiving

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

8In [17], modularity is defined only for events, but, in the same spirit, we extend it to other parts of syntax.
the redial environmental input, Redialer instructs Dialer, by generating the corresponding dial events, to dial the digits of |p|. (We denote the size of an integer x as |x| and its nth digit as digit(x, n).) Variable \( p \) is necessary because once redialling starts |p| is overwritten. Consider the snapshot where the environmental input event redial is received, \( c \) is zero, and |p| is 10. Let us assume that redial event, when received, persists throughout the big-step. Also, let us assume the semantics that follows the SYNTACTIC maximality semantics (annotating a designated control state with a “✓”), a memory protocol that uses the up-to-date values of variables (cf., Section 3.3 and 3.7), a concurrency and consistency semantics where all enabled transitions that can be possibly taken together in a small-step are taken (cf., Section 3.5), and the NEXT SMALL-STEP event lifeline semantics. Such a semantics executes big-step \( t_5 \), \( \{t_2, t_6\} \), \( \{t_3, t_6\} \), \( \cdots \), \( \{t_4, t_7\} \), which transmits the first digit twice and does not transmit the last digit. If we assume the SAME event lifeline semantics, then the model behaves correctly: \( \{t_5, t_2\} \), \( \{t_3, t_6\} \), \( \cdots \), \( \{t_4, t_7\} \).\(^9\)

### 3.3 Enabledness Memory Protocol

The enabledness memory protocol of a BSML determines the value 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.

---

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

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

Table 3. Enabledness Memory Protocols.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Obtains Value From</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p ) ( e ) (e.g., [19])</td>
<td>big-step source snapshot</td>
<td>✓</td>
</tr>
<tr>
<td>( c ) ( u ) (e.g., [12])</td>
<td>small-step source snapshot</td>
<td>✓</td>
</tr>
<tr>
<td>( n ) ( w ) (e.g., [11])</td>
<td>small-step source snapshot</td>
<td>✗</td>
</tr>
<tr>
<td>new small ( n ) ( w ) ( (e.g., [27])</td>
<td>small-step destination snapshot</td>
<td>✗</td>
</tr>
</tbody>
</table>

Table 4. Variable Operators.

**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 \( n \) \( w \) is different from \( c \) \( u \) 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 assigned to its operand in the current small-step. By definition, operator \( p \) \( e \) is not relevant for the BIG-STEP memory protocol and operator \( c \) \( u \) is not relevant for the SMALL-STEP memory protocol.

**Example 3** Consider the semantic options in Example 2 that lead to an incorrect behaviour. If we modify the VC of \( t_6 \) to “\( \{\text{new small}(c) < |p|\} \) and its event generation to “dial(digit(new small(c) + 1, p))”, then the model behaves correctly: \( t_5 \), \( \{t_2, t_6\} \), \( \{t_3, t_6\} \), \( \cdots \), \( t_3 \), \( \{t_4, t_7\} \).

### 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 [5] 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 by using 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 avoid undesired non-determinism in a model 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 might 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.

Example 4 Consider the semantic options in Example 2 that lead to an incorrect behaviour. Yet another way to fix the problem is to modify the model by moving the "p := lp" assignment from $t_5$ to $t_2$, changing the VC of $t_6$ to "c < |new(p) − 1|", and its event generation to "dia(digit(new small(c) + 1, p))". Such a model then behaves correctly: $t_5$, $t_2$, $t_6$, \{t_3, t_6\}, \cdots, t_3, \{t_4, t_7\}.

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 in the small-step.

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 also other options [7].

Table 5. Order of Small-steps Semantic Options.

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

Example 5 In Example 2, if we assume the SINGLE option, instead of the MANY option, then the model in Figure 1 can create a correct big-step or 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\}, \cdots, $t_4$, $t_7$.

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 OZ control state in the hierarchy of the composition tree that is the 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.\footnote{The same semantic options can be defined at the big-step scope, without requiring the MANY semantics (e.g., ARENA ORTHOGONAL in [28]).}

3.5.3 Preemption

The notion of preemption [2] is relevant for transitions that are not pairwise orthogonal. A transition $t$ is an interrupt for transition $t'$ when the sources of 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 descendent of the destination of $t'$ (Figure 6(b)). The NON-PREEMPTIVE option allows such a $t$ and $t'$ to be executed together in the
Table 6. Concurrency and Consistency Semantic Options.

<table>
<thead>
<tr>
<th>Options</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Concurrency</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SINGLE</td>
<td>Simplicity</td>
<td>Non-determinism</td>
<td>Statecharts [9, 12, 28],</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Stateflow [5], and Reactive Modules [1]</td>
</tr>
<tr>
<td>MANY</td>
<td>Greater determinism</td>
<td>Race conditions</td>
<td>Argos [22] and Esterel [3]</td>
</tr>
<tr>
<td><strong>Small-step consistency</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARENA ORTHOGONAL</td>
<td>Simplicity</td>
<td>Non-determinism</td>
<td>Statecharts [9, 12]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Complex</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Preemption</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NON-PREEMPTIVE</td>
<td>Supports “last wish”</td>
<td>Complicated flow of control</td>
<td>Argos [22] and Esterel [3]</td>
</tr>
<tr>
<td>PREEMPTIVE</td>
<td>Simple flow of control</td>
<td>No support for “last wish”</td>
<td>Statecharts [28]</td>
</tr>
</tbody>
</table>

Figure 6. Interrupting Transitions.

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.

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 Redialer, its destination is a new control state $S$, which is not ancestrally related to Dialing, and its variable condition is “[limit = 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 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 priority [24]). The 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 conjointing the original trigger of $t_1$, dial(d), with the negation of the event trigger of $t_2$, dial(d) \ redial, resulting in $t_1$ having the trigger dial(d) \ ¬redial. The effect is that $t_2$ will be chosen when the redial event occurs instead of $t_1$.

3.7 Assignment Memory Protocol

The assignment memory protocol of a BSML determines the value of variables that a transition reads for the right-hand side (RHS) 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 two memory protocols of a BSML need not be the same (e.g., SCR [14, 13]).
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 between the elements (i.e., events and/or variables) that the environment provides (environmental input elements) and the elements that the model can control, syntactically. 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 report [7].

4 Interdependencies

The hierarchical structure of our aspects captures most of the interdependencies among 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.

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.

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.

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.

<table>
<thead>
<tr>
<th>Options</th>
<th>Advantages</th>
<th>Disadvantages</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLICIT</td>
<td>Greater control</td>
<td>Tedious to use</td>
<td>Used in [24]</td>
</tr>
<tr>
<td>NEGATION OF TRIGGERS</td>
<td>Greater control, no</td>
<td>Tedious to use</td>
<td>Statecharts [28], Esterel [3],</td>
</tr>
<tr>
<td></td>
<td>additional syntax</td>
<td></td>
<td>and Argos [22]</td>
</tr>
</tbody>
</table>

**Table 7. Priority Semantic Options.**

*Example 8* Consider the semantic options in Example 2 that lead to a correct behaviour. If we assume 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 for all other reads that are meant to obtain the current value of a variable, they should be prefixed by the `cas` operator.
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 tradeoffs between choosing different features. 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-prioritised 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