Prescriptive Semantics for Big-Step Modelling Languages

Shahram Esmaeilsabzali and Nancy A. Day

Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1
{sesmaeil,nday}@cs.uwaterloo.ca

Abstract. A big-step modelling language (BSML) is a language in which a model can respond to an environmental input via a sequence of small steps, each of which may consist of the concurrent execution of a set of transitions. BSMLs are a popular class of modelling languages that are regularly reincarnated in different syntactic and semantic variations. In our previous work, we *deconstructed* the semantics of many existing BSMLs into eight high-level, conceptually intuitive *semantic aspects* and their *semantic options*, which together constitute a semantic design space for BSMLs. In this work, we describe a parametric semantic definition schema based on this deconstruction for defining formally the semantics of a wide range of BSMLs. A semantic definition in our framework is *prescriptive* in that the high-level semantic aspects of a BSML are manifested clearly as orthogonal parts of the semantic definition. Our goal is to produce a formal semantic definition that is accessible to various stakeholders of the semantics.

1 Introduction

In this paper, we describe a formal framework to define the semantics of *Big-step Modelling Languages* (BSMLs) [1,2]. BSMLs are a popular, effective class of behavioural modelling languages in which a modeller can specify the reaction of a system to an environmental input as a *big step*, which consists of a sequence of *small steps*, each of which may contain a set of concurrent transitions. There is a plethora of BSMLs, many with graphical syntax (e.g., statecharts variants [3, 4] and Argos [5]), some with textual syntax (e.g., Reactive Modules [6] and Esterel [7]), and some with tabular format (e.g., SCR [8, 9]).

In our previous work, we *deconstructed* the semantics of BSMLs into eight mainly orthogonal *semantic aspects* and their corresponding *semantic options* [1, 2]. The semantic aspects distill the semantic concerns of different BSMLs into high-level semantic concepts. The combinations of the semantic options establish a semantic design space that includes the semantics of many existing BSMLs, as well as new BSML semantics. Our deconstruction is conceptually intuitive because the semantic aspects characterize a big step as a whole, rather than only considering its constituent transitions operationally.

D.S. Rosenblum and G. Taentzer (Eds.): FASE 2010, LNCS 6013, pp. 158–172, 2010.

Our contribution in this paper is the formalization of a wide range of BSML semantics in a manner that follows our deconstruction. Our semantic definition framework consists of a *semantic definition schema*, which has a set of *parameters*. By instantiating the parameters of the semantic definition schema, an operational BSML semantics is derived. The semantic definition schema, its parameters, and the values of the parameters are specified in standard logic and set theory. The semantic aspects of BSMLs correspond to disjoint parameters of the semantic definition schema, and the semantic options of each semantic aspect correspond to the possible values for the parameters. Thus, we achieve a formal semantics without sacrificing the intuitiveness achieved in our deconstruction.

A key insight in our formalization is the recognition, and separation, of the distinct roles in the formal definition for the semantic aspects of *concurrency*, *small-step consistency*, *preemption*, and *priority*. We call these parameters *structural parameters*, because they affect the meaning of the hierarchial structure of a model. Structural parameters are distinct from the more common *dynamic parameters* found in other formalizations, which specify how the state of the model changes from one small step to the next. We believe we are the first to separate these structural parameters disjointly in a manner that matches the factoring into aspects from our high-level deconstruction of the semantics of BSMLs.

In formal semantics, there are two approaches in using formalism: *descriptive* vs. *prescriptive* [10, 11]. Our semantic definition framework is a prescriptive approach to define BSML semantics. A prescriptive semantic definition uses the formalism in an active role, to prescribe a semantics that is already known to the semanticist. A descriptive semantic definition uses the formalism in a passive role to describe, what seems like, the discovery of a semantics in the world of all possible semantics. A prescriptive semantic method can provide insights and guidance about what is a *good* semantics, whereas a descriptive semantics cannot. For example, BNF is a prescriptive method for defining syntax, as opposed to pre-BNF methods, which were descriptive [10]. "In general, the descriptive approach aims for generality even at the expense of simplicity and elegance, while the prescriptive approach aims for simplicity and elegance even at the expense of generality." [11, p.284]. A semantic definition produced by our semantic definition schema is prescriptive in that the high-level semantic aspects of a BSML chosen by its various stakeholders are manifested clearly as orthogonal parts of the semantic definition. A result of our semantic deconstruction and its formalization is a clear scope for the BSML family of modelling languages.

The remainder of the paper is organized as follows. In Section 2, we first describe the common syntax that we use for BSMLs, and then describe our deconstruction of BSML semantics using a feature diagram. In Section 3, we describe our semantic definition schema, by presenting its common elements, its structural and dynamic parameters, and their possible values. We also specify the scope of BSML semantics that our framework covers. In Section 4, we consider the related work, comparing our work with other semantic definition frameworks, including those used in tool-support generator frameworks. In Section 5, we conclude our paper and discuss future work.

2 Background

In Section 2.1, we describe our normal form syntax for BSMLs, and in Section 2.2, we describe our deconstruction of BSML semantics [1,2]. We adopt few syntactic definitions from Pnueli and Shalev's work [12].

2.1 Normal Form Syntax

As is usual when studying a class of related notations, we introduce a *normal form syntax* [13], which is expressive enough for representing the syntax of many BSMLs [1,2]. In our normal form syntax, a model consists of: (i) a *hierarchy tree* of *control states*, and (ii) a set of *transitions* between the control states. Fig. 1 shows the BNF for the hierarchy tree of a model and its transitions, which permits an arbitrary hierarchy of *And* and *Or* control states, with *Basic* control states appearing at the leaves.[1] The symbol "transitions" in the rules is a set of transitions. The highest level control state in a hierarchy tree is called the *root*, which is an *Or* control state with an empty set of transitions.

⟨root⟩ → ⟨Orstate⟩
⟨Orstate⟩ → **Or** ⟨states⟩ ⟨transitions⟩
⟨Andstate⟩ → **And** ⟨states⟩ ⟨transitions⟩
⟨Basicstate⟩ → **Basic**
⟨states⟩ → ⟨state⟩ | ⟨states⟩ ⟨state⟩
⟨state⟩ → ⟨Orstate⟩ | ⟨Andstate⟩ | ⟨Basicstate⟩

Fig. 1. The BNF for the hierarchy tree of control states and their transitions

An *Or* or an *And* control state, s, has *children*, $children(s)$. For a control state s, $children^*(s)$ is the set of all control states that are children of s either directly or by transitivity. We denote $children^+(s) = children^*(s) \cup s$. Similarly, we use the *parent*, *ancestor*, and *descendant* relations with their usual meanings. An *Or* control state s has a *default* control state, $default(s)$, which is one of its children. A *Basic* control state can be designated as *stable* or *non-stable*.[2]

A transition t has a *source* control state, $src(t)$, and a *destination* control state, $dest(t)$. Additionally, it can have four optional parts: (i) an *event trigger*, $trig(t)$, which is a conjunction of events, $pos\_trig(t)$, and negations of events, $neg\_trig(t)$; (ii) a *guard condition*, $cond(t)$, which is a boolean expression over a set of variables; (iii) a sequence of *assignments*, $asn(t)$; and (iv) a set of *generated events*, $gen(t)$.

The *least common ancestor* of two control states s and s', $lca(s, s')$, is the lowest control state (closest to the leaves of the hierarchy tree) in the hierarchy tree such that: $s, s' \in children^*(lca(s, s'))$. Given a transition t, we assume a

[1] Normally, a control state has a name, but we do not need it in our formalization.
[2] For example, a stable control state can be used to model non-pseudo control states of *compound transitions* in UML StateMachines [14] or `pause` commands in Esterel [7].

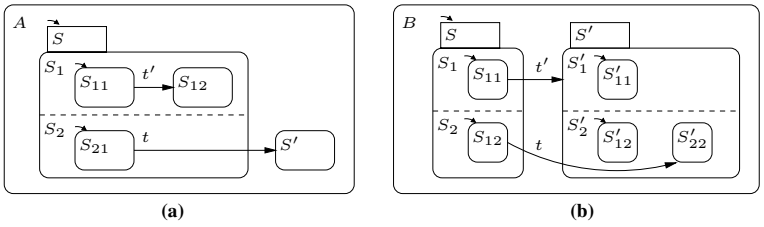

Fig. 2. Interrupting transitions

parsing mechanism that associates t with control state $lca(src(t), dest(t))$. Two control states s and s' are *orthogonal*, $s \perp s'$, if neither is an ancestor of the other and $lca(s, s')$ is an *And* control state. We call two transitions t and t' orthogonal, $t \perp t'$, if $src(t) \perp src(t')$ and $dest(t) \perp dest(t')$. The *arena* of a transition t, $arena(t)$, is the lowest *Or* control state in the hierarchy tree such that: $src(t), dest(t) \in children^*(arena(t))$. The *source scope* of a transition t, $ss(t)$, specifies the highest control state that t exits upon execution. If $src(t) \in children^+(dest(t))$, then $ss(t)$ is $dest(t)$, and if $dest(t) \in children^+(src(t))$, then $ss(t)$ is $src(t)$. Otherwise, $ss(t)$ is the highest control state such that: $src(t) \in children^+(ss(t))$ and $dest(t) \notin children^+(ss(t))$. Similarly, the *destination scope* of a transition t, $ds(t)$, is defined.

A transition t is an *interrupt for* transition t', $t \lightning t'$, if 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 (Fig. 2(a)); or (ii) the destination of neither transition is orthogonal with the sources of the two transitions, but the destination of t is a descendant of the destination of t' (Fig. 2(b)). (In Fig. 2, a dashed line separates the children of an *And* control state, and an arrow without a source signifies the default control state of an *Or* control state.) We use the *interrupt for* relation to model the notion of *preemption* [7,5].[3] For a set of transitions τ, its set of *interrupted transitions*, $interr(\tau) \subset \tau$, consists of transitions t' such that for each $t' \in interr(\tau)$ there is a $t \in \tau$ and $t \lightning t'$.

2.2 BSML Semantics

Initially, a model resides in the default control state of each of its *Or* control states, its variables have their initial values, and its events are not present. Fig. 3 depicts the structure of a big step, operationally. A big step is the reaction of a model to an *environmental input*, which consists of an alternating sequence of small steps and *snapshots*. An environmental input, e.g., I in Fig. 3, consists of a set of environmental input events, $I.events$, and a set of variable assignments, $I.asns$. In some BSMLs, a sequence of small steps are grouped together

[3] For example, in Esterel [7], the concurrent execution of a statement and an `exit` statement is modelled by the *interrupt for* relation according to condition (i) above, and the concurrent execution of two `exits` is modelled by condition (ii) above.

into a *combo step*, which hides some of the effects of its small steps from one another (e.g., [15, 16]). A snapshot is a collection of *snapshot elements*, each of which is a collection of related information about the execution of a model. For example, there is a snapshot element, S_c, which maintains the set of *current* control states that a model resides in. We follow the convention of using *sp* itself, or *sp* with a superscript, as the name of a snapshot; e.g., *sp* and *sp*′. Also, we follow the convention of always using a subscript to name a snapshot element. To access a snapshot element in a snapshot, we annotate the snapshot element name with the superscript of the snapshot; e.g., S_c and S'_c access the snapshot element S_c in snapshots *sp* and *sp*′, respectively.

The execution of a transition in a small step includes *exiting* a set of control states and *entering* a set of control states. The *exited control states* of transition t, $exited(t) = children^+(ss(t))$, specify the control states that could be exited upon execution of t, based on the current control states, S_c, which the model resides in. The *entered control states* of transition t, $entered(t)$, specifies exactly the control states that are entered upon execution of t. A control state s belongs to $entered(t)$ if $s \in children^+(ds(t))$ and one of the following three conditions holds: (i) $dest(t) \in children^+(s)$; (ii) there exists a control state $s' \in entered(t)$ such that, (a) either s' is an *And* control state and $s \in children(s')$, or s' is an *Or* control state and $s = default(s')$, and (b) $lca(s, dest(t))$ is not an *Or* control state; or (iii) there exists a control state $s' \in entered(t)$ such that, (a) either s' is an *And* control state and $s \in children(s')$, or s' is an *Or* control state and $s = default(s')$, and (b) $s' \in children^+(dest(t))$. For example, in Fig. 2(b), $dest(t) = S'_{22}$, $ds(t) = S'$, and $entered(t) = \{S', S'_1, S'_2, S'_{11}, S'_{22}\}$. Therefore, the execution of a small step τ removes the set of control states $\bigcup_{t \in \tau} exited(t)$ from S_c, and adds the set of control states $\bigcup_{t \in (\tau - interr(\tau))} entered(t)$ to it.

Semantic Deconstruction. Fig. 4 shows our deconstruction of BSML semantics into eight semantics aspects, and their semantic options [1, 2], as a feature diagram [17]. The Sans Serif and SMALL CAPS fonts represent the semantic aspects and the semantic options, respectively. The feature diagram in Fig. 4 enumerates the BSML semantics that arise from our deconstruction.[4] A simple subtree in the feature diagram is an *and* choice: if the parent is chosen, all of its children, except for the *optional* children that are distinguished by a small circle attaching to them must be chosen. An arced subtree in the feature diagram is an *exclusive or* choice: if the parent is chosen, exactly one of its children can be chosen, which might be an optional feature. For example, a BSML semantics must subscribe to exactly one semantic option of the Big-Step Maximality.

Semantic Aspects. The Big-Step Maximality semantic aspect specifies when a big step concludes, meaning the model can sense the next environmental input. Similarly, the Combo-Step Maximality semantic aspect specifies when a combo step concludes, and the effect of the execution of the small steps of the combo

[4] There are few semantic dependencies between the choices of semantic options [1, 2], but we do not consider them here, and assume that they are satisfied.

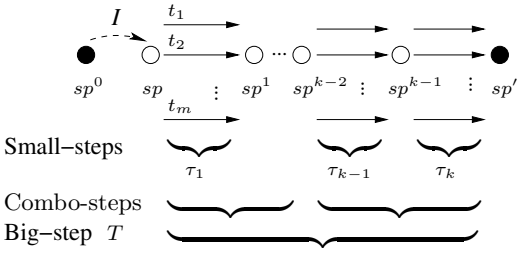

Fig. 3. Steps

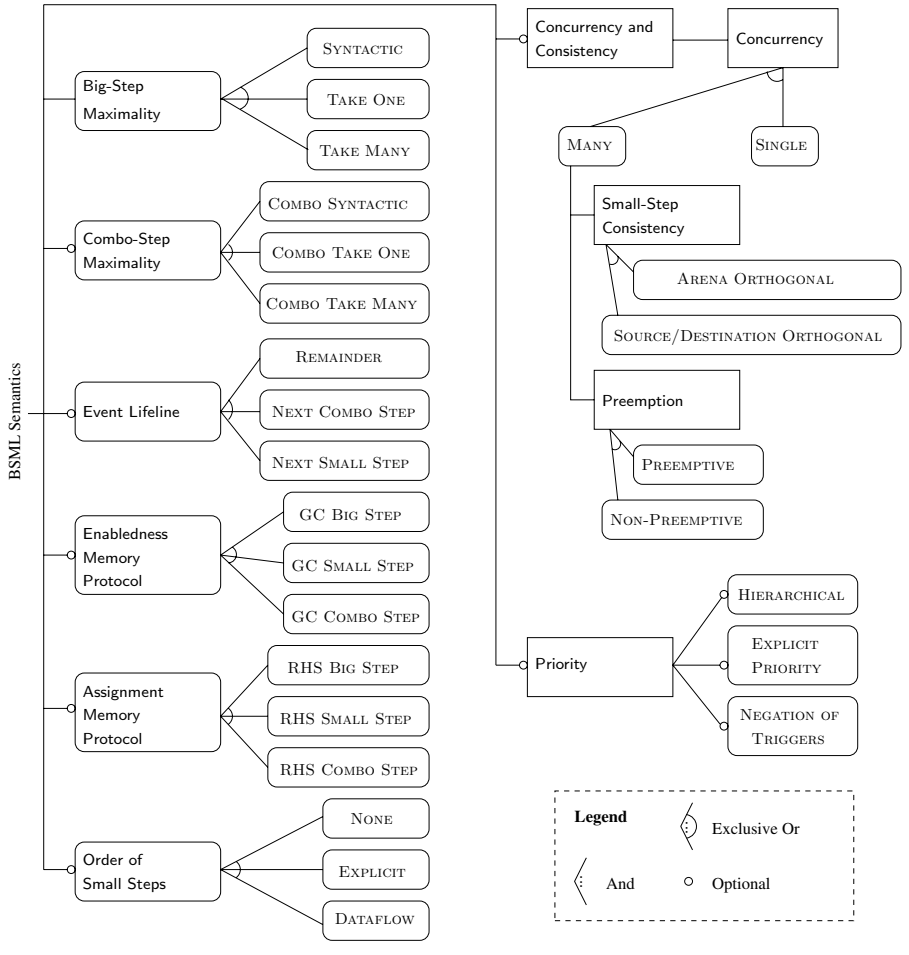

Fig. 4. Semantic aspects and their semantic options

step that has been hidden during the combo step becomes available. The Event Lifeline semantic aspect specifies the sequence of snapshots of a big step in which a generated event can be sensed as present by a transition.[5] The Enabledness Memory Protocol semantic aspect specifies the snapshots of a big step from which the values of the variables of the guard condition (GC) of a transition are obtained. Similarly, the Assignment Memory Protocol semantic aspect specifies the snapshots of a big step from which the values of the variables in the right-hand side (RHS) of the assignments of a transition are obtained. The Order of Small Steps semantic aspect determines how potential small steps at a snapshot are ordered. The Concurrency and Consistency semantic aspect consists of three sub-aspects. The Concurrency sub-aspect specifies whether one or more than one transition can be taken in a small step. If more than one transition can be taken in a small step, the Small-Step Consistency sub-aspect specifies the criteria for including a set of transitions in a small step. The Preemption sub-aspect specifies whether two transitions where one is an *interrupt for* the other can be included in a small step, or not. Lastly, the Priority semantic aspect assigns relative priority to a pair of transitions that can replace one another in a small step. As an example, the semantics of Statemate [16] is characterized by the Take Many, Combo Take One, Next Combo Step, GC Combo Step, RHS Combo Step, None, Single, and Hierarchical semantic options.

3 Semantic Definition Schema

In this section, we present our method for the formalization of BSML semantics. A BSML semantic definition is an instantiation of our *semantic definition schema*. The semantic definition schema is a set of parametric definitions in standard logic and set theory. Fig. 5 shows the structure of the semantic definition schema, whose elements we describe throughout this section. An arrow between two nodes in the figure specifies that the element in the source of the arrow uses the element in the destination of the arrow. The highest level predicate is N_{Big}, which is a relation between two snapshots sp^0 and sp, and an input I. Predicate N_{Big} characterizes the big steps of the model.

The leaf nodes in Fig. 5 are the *parameters* of the schema. We distinguish between a *structural parameter* and a *dynamic parameter*. A structural parameter deals with the structure of the hierarchy tree of a model, as opposed to a dynamic parameter. The corresponding semantic aspect of a structural parameter is a *structural* semantic aspect, and the corresponding semantic aspect of a dynamic parameter is a *dynamic* semantic aspect. In Fig. 4, we use rectangles and rounded rectangles to distinguish between structural and dynamic aspects, respectively. Similarly, In Fig. 5, we use rectangles and rounded rectangles to distinguish between structural and dynamic parameters, respectively.

A value for a structural parameter is a predicate that specifies how enabled transitions together can form a small step. A value for a dynamic parameter is a set of snapshot elements, which is usually a singleton. A snapshot element x_1

[5] Our deconstruction does not include asynchronous events, which use buffers.

is characterized by its type, and three predicates, which are each parameters: (i) `reset_`x_1, which specifies how x_1 changes at the beginning of a big step when an environmental input is received; (ii) `next_`x_1, which specifies how the value of x_1 is changed when a small step is executed; and (ii) `en_`x_1, which specifies the role of x_1 in determining a transition as enabled. We denote the set of all snapshot elements that are used by a BSML semantics as $SpEl = \{x_1, x_2, \cdots, x_n\}$.

The remainder of this section is organized as follows. Section 3.1 describes the common, non-parametric elements of the semantic definition schema. Sections 3.2 and 3.3 present the possible values for the structural and dynamic parameters, respectively. Section 3.4 describes the scope of our formalization.

3.1 Common Elements

Fig. 6 shows the semantic definition schema. Definitions in lines 1-5 specify how a big step is formed from small steps. Line 1 specifies predicate N_{Big}, which creates a big step by sensing the environmental input I at snapshot sp^0 (via predicate *reset*), taking k small steps (via predicate N), and concluding the big step at snapshot sp', when there are no further small steps to be taken, i.e., when $enabled(root, sp') = \emptyset$. The *reset* predicate in line 2, is the conjunction of the "`reset`" predicates of the snapshot elements of a BSML semantics; it specifies the effect of receiving the environmental input I. Line 5 specifies the operation of a small step through the N_{small} predicate, which itself is the conjunction of the "`next`" predicates of the snapshot elements of a BSML semantics. The effect of executing a small step is captured in the destination snapshot of the small step. The N_{small} predicates are chained together via the N relation to create a sequence of small steps, as shown in lines 3 and 4.

Definitions in lines 6-9 determine the set of potential small steps at a snapshot by walking over the hierarchy tree of a model, according to the BNF in Fig. 1, in a bottom-up way. At a snapshot *sp*, $enabled(root, sp)$ provides a set of sets of transitions, one of which is non-deterministically chosen as the next small step. For readability, we have used a pair of brackets "[ ]" to separate the syntactic parameter of the predicates from the snapshot parameter *sp*. Parameter Tr is the set of transitions of an *And* or *Or* control state. Line 9 specifies how transitions of the children of an *And* control state can be executed in a small step: the static parameter $\|$, which is associated with the Concurrency semantic aspect, specifies whether one or more than one transition can be executed in a small step.

Line 14 defines the *merge* operator, used in lines 6 and 8, which depends on static parameters C, P, and Π. At each level of the hierarchy tree of a model, the merge operator decides how to choose from the set of sets of enabled transitions at the lower level of the tree (parameter $\mathbb{T}$), and the transitions at the current level of the tree (parameter T'). Parameter $\mathbb{T}$ is in a special font because its type is set of sets of transitions, as opposed to a set of transitions such as T'. The result of a merge operation is a new set of sets of transitions. A transition belonging to a set of transitions $T_1 \in \mathbb{T}$ is included in the corresponding new set of transitions for T_1, unless it is replaced with a transition in the current level. Similarly, a transition $t \in T'$ in the current level is included in a new set

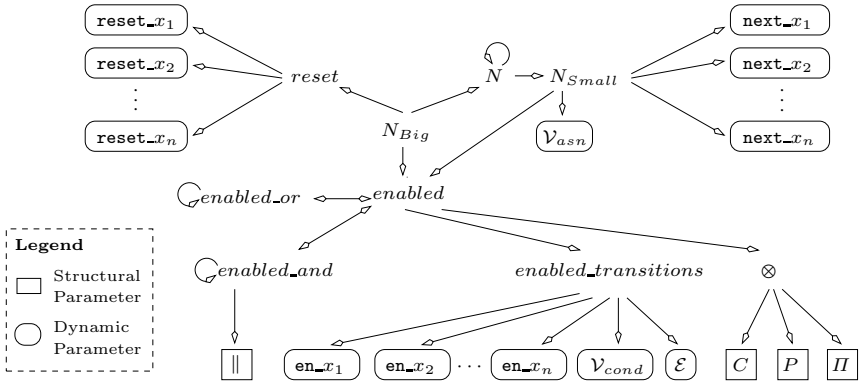

Fig. 5. The structure of the semantic definition schema

1. $N_{Big}(sp^0, I, sp') \equiv reset(sp^0, I, sp) \wedge (\exists k \geq 0 \cdot N^k(sp, sp')) \wedge enabled(root, sp') = \emptyset$
2. $reset(sp^0, I, sp) \equiv \bigwedge_{1 \leq i \leq n} \texttt{reset\_}x_i(sp^0, I, sp)$
3. $N^0(sp, sp') \equiv sp = sp'$
4. $N^{k+1}(sp, sp') \equiv \exists \tau, sp'' \cdot N_{Small}(sp, \tau, sp'') \wedge N^k(sp'', sp')$
5. $N_{Small}(sp, \tau, sp') \equiv \bigwedge_{1 \leq i \leq n} \texttt{next\_}x_i(sp, \tau, sp') \wedge \tau \in enabled(root, sp)$

6. $enabled([\textbf{Or}, \langle s_1, s_2, \cdots, s_m \rangle, Tr], sp) = enabled\_or([\langle s_1, s_2, \cdots, s_m \rangle]) \otimes enabled\_transitions([Tr], sp)$
7. $enabled\_or([\langle s_1, s_2, \cdots, s_m \rangle], sp) = enabled([s_1], sp) \cup enabled\_or([\langle s_2, \cdots, s_m \rangle], sp)$
8. $enabled([\textbf{And}, \langle s_1, s_2, \cdots, s_m \rangle, Tr], sp) = enabled\_and([\langle s_1, s_2, \cdots, s_m \rangle], sp) \otimes enabled\_transitions([Tr], sp)$
9. $enabled\_and([\langle s_1, s_2, \cdots, s_m \rangle], sp) = enabled([s_1], sp) \parallel enabled\_and([\langle s_2, \cdots, s_m \rangle], sp)$
10. $enabled\_transitions([Tr], sp) = \{t : Tr \mid evaluate([cond(t)], \mathcal{V}_{cond}) \wedge ([src(t)] \in S_c) \wedge ([neg\_trig(t)] \cap \mathcal{E} = \emptyset) ([pos\_trig(t)] \subseteq \mathcal{E}) \bigwedge_{1 \leq i \leq n} \texttt{en\_}x_i([t], sp)\}$
11. $enabled([\textbf{Basic}], sp) = \emptyset$
12. $enabled\_or(\langle\rangle, sp) = \emptyset$
13. $enabled\_and(\langle\rangle, sp) = \emptyset$

Where *evaluate* evaluates a variable expression with respect to a snapshot element, and operators $\parallel$ and $\otimes$ are concurrency and merger operators, respectively.

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

Fig. 6. Semantic definition schema

of transitions unless it is replaced by a transition at the current or lower level. A new set of transitions is maximal in that no more transitions can be added to it without violating the Small-Step Consistency, Preemption, or Priority semantics, represented by parameters C, P, and Π, respectively.

Line 10 specifies if a transition t is enabled, by checking whether: $cond(t)$ is true with respect to the values of the variables, $src(t)$ is in the set of current control states, which the model resides in, $trig(t)$ is satisfied with respect to the statuses of events, and the "en" conditions of all snapshot elements are satisfied. Lines 11-13 are the base cases of the recursive definitions.

The dynamic parameters $\mathcal{V}_{asn}$, $\mathcal{V}_{cond}$, $\mathcal{E}$ are the names of snapshot elements in *SpEl* that determine: the values of variables for evaluating the RHS of assignments, the values of variables for evaluating the GC of transitions, the statuses of events for evaluating the triggers of transitions, respectively. The value of each of these parameters is determined by the choice of a semantic option for its corresponding semantic aspect. Parameter $\mathcal{V}_{asn}$ is used in the formalization of the Assignment Memory Protocol, which, for the sake of brevity, we do not consider its formalization in this paper.

3.2 Structural Parameters

A value for a structural parameter is a predicate that specifies an aspect of how a small step is formed. In this section, we describe the possible values for parameter $\|$, and parameters C, P, and Π of the merge operator $\otimes$.

Table 1 specifies the values for the *concurrency* operator $\|$, based on the semantic options of the Concurrency semantic sub-aspect. The SINGLE semantics allows exactly one transition per small step, where as the MANY semantics allows all of the children of an *And* control state to have transitions in a small step.

Table 2 specifies the values for parameters C and P of the merge operator, which are determined by the Small-Step Consistency and Preemption semantic sub-aspects, respectively. These two semantic sub-aspects are relevant only for the MANY concurrency semantics. When the SINGLE concurrency semantics is chosen, both C and P are *false*. For two distinct t and t' to be included in a small step, the ARENA ORTHOGONAL semantics requires their arenas to be orthogonal, and the SOURCE/ DESTINATION ORTHOGONAL semantics requires the transitions themselves to be orthogonal. For two transitions t and t' where one is an *interrupt for* another, the NON-PREEMPTIVE semantics allows them to be taken in the same small step, but the PREEMPTIVE semantics does not.

Table 1. Concurrency semantics

Semantic Option	Parameter Value
SINGLE	$\mathbb{T} \parallel \mathbb{T}' = \mathbb{T} \cup \mathbb{T}'$
MANY	$\mathbb{T} \parallel \mathbb{T}' = \{T_1 \cup T_1' \mid T_1 \in \mathbb{T} \wedge T_1' \in \mathbb{T}'\}$

Table 2. Small-Step Consistency and Preemption semantics

Semantic Option	Parameter Value
Small-Step Consistency	
ARENA ORTHOGONAL	$C(t,t') \equiv arena(t) \perp arena(t')$
SOURCE/DESTINATION ORTHOGONAL	$C(t,t') \equiv t \perp t'$
Preemption	
NON-PREEMPTIVE	$P(t,t') \equiv (t \not\downarrow t') \vee (t' \not\downarrow t)$
PREEMPTIVE	$P(t,t') \equiv false$

Table 3. Priority semantics

Semantic Option	Parameter Value
NO PRIORITY	$\Pi(\mathbb{T}, T', T_1, T_1', T'') \equiv true$
CHILD ARENA	$\Pi(\mathbb{T}, T', T_1, T_1', T'') \equiv T_1' = \emptyset$

Table 3 specifies two possible values for parameter Π. For the sake of brevity, we consider only two HIERARCHICAL semantic options, but there are more Priority semantic options [1,2]. The CHILD ARENA semantics assigns a higher priority to a transition whose arena is lower in the hierarchy tree.

As an example, for the semantics that follow the MANY, ARENA ORTHOGONAL, PREEMPTIVE, and NO PRIORITY semantic options, the merge operator can be simplified to $\mathbb{T} \otimes T' = \mathbb{T} \cup \{\{t\} |\ t \in T'\}$.

3.3 Dynamic Parameters

A value for a dynamic parameter is a set of snapshot elements, possibly a singleton, which accomplish the semantics of a corresponding semantic option. For a snapshot element $x_1 \in SpEl$, the $\texttt{reset\_}x_1(sp^0, I, sp)$ predicate specifies the effect of receiving environmental input I in snapshot sp^0 on snapshot element x_1, which is captured in snapshot sp. The $\texttt{en\_}x_1(sp, t)$ predicate specifies if transition t is enabled with respect to snapshot element x_1. If $\texttt{en\_}x_1$ is not specified, then it is $true$. The $\texttt{next\_}x_1(sp, \tau, sp')$ predicate specifies the effect of executing small step τ in snapshot sp, which is captured in snapshot sp'. We annotate the name of the snapshot element that represents a semantic option with a subscript that is the same as the name of the semantic option.

In this section, for the sake of brevity, we consider the values of the dynamic parameters for the Event Lifeline semantic aspect only. The corresponding snapshot elements for the semantic options of other semantic aspects are defined similarly and independent of each other, except for the combo-step semantic options (cf., the formalization of the NEXT COMBO STEP semantics below).

Event Lifeline. There are three semantic options for the Event Lifeline semantic aspect. The REMAINDER, NEXT COMBO STEP, and NEXT SMALL STEP

$\texttt{reset\_}E_{\text{REMAINDER}}(sp^0, I, sp)$	$\equiv E_{\text{REMAINDER}} = I.events$
$\texttt{next\_}E_{\text{REMAINDER}}(sp, \tau, sp')$	$\equiv E'_{\text{REMAINDER}} = E_{\text{REMAINDER}} \cup gen(\tau)$
$\texttt{reset\_}E_{\text{NEXT COMBO STEP}}(sp^0, I, sp)$	$\equiv E_{\text{NEXT COMBO STEP}} = I.events$
$\texttt{next\_}E_{\text{NEXT COMBO STEP}}(sp, \tau, sp')$	$\equiv E'_{\text{NEXT COMBO STEP}} = \text{if } EndC \text{ then } E_{Collect} \cup gen(\tau) \text{ else } E_{\text{NEXT COMBO STEP}}$
$\texttt{reset\_}E_{Collect}(sp^0, I, sp)$	$\equiv E_{collect} = \emptyset$
$\texttt{next\_}E_{Collect}(sp, \tau, sp')$	$\equiv E'_{collect} = \text{if } EndC \text{ then } \emptyset \text{ else } E_{Collect} \cup gen(\tau)$
$EndC$	$\equiv (\nexists \tau' \cdot \tau' \in enabled(root, sp' \oplus Cs))$
$\texttt{reset\_}E_{\text{NEXT SMALL STEP}}(sp^0, I, sp)$	$\equiv E_{\text{NEXT SMALL STEP}} = I.events$
$\texttt{next\_}E_{\text{NEXT SMALL STEP}}(sp, \tau, sp')$	$\equiv E'_{\text{NEXT SMALL STEP}} = gen(\tau)$

Fig. 7. Snapshot elements for Event Lifeline semantics

semantics require a generated event to be present after it is generated in the remainder of the big step, in the next combo step, and in the next small step, respectively. Fig. 7 presents the snapshot elements for formalizing each semantic option. The value of parameter $\mathcal{E}$, in Fig. 6, is $E_{\text{REMAINDEER}}$, $E_{\text{NEXT COMBO STEP}}$, or $E_{\text{NEXT SMALL STEP}}$, based on the chosen Event Lifeline semantics.

In the NEXT COMBO STEP semantics, the last small step of a combo step must be identified so that the statuses of events are adjusted at the end of the combo step. The $EndC$ predicate in Fig. 7 identifies the last small step of a combo step. Its definition relies on the set of snapshot elements, Cs, which is the set of snapshot elements that specify the notion of combo step in a semantics. For example, if the NEXT COMBO STEP semantic option alone is chosen, then $Cs = \{E_{\text{NEXT COMBO STEP}}, E_{Collect}\}$. If the GC COMBO STEP semantic option is also chosen, then Cs also includes its corresponding snapshot elements. The *override* operator, $\oplus$, replaces the corresponding snapshot elements of its first parameter, which is a snapshot, with the snapshot elements in the second parameter. Thus, $sp' \oplus Cs$ is a new snapshot, replacing the corresponding snapshot elements of sp' with the ones in Cs from snapshot sp. Without Cs, the definitions of the snapshot elements of two combo-step semantic options become cyclic.

3.4 Scope of Formalization

We call a BSML semantics a *forward-referencing* semantics if the enabledness of a transition depends on the execution of other transitions in the current or future small steps. For example, in the Event Lifeline semantics, another semantic option is that an event generated by a transition in a future small step is sensed as present by the current small step [1,2]. Our current semantic definition schema does not cover the forward-referencing semantics because they convolute the role of structural and dynamic parameters.

4 Related Work

Our work is related to *tool-support generator frameworks* (TGFs) that take the definition of a notation, including its semantics, as input, and generate tool support, such as model checking and simulation capability, as output [18, 19, 20, 21, 22, 23, 24, 25]. TGFs differ in the *semantic input formats* (SIF) they use, and the procedure by which they obtain tool support for a notation. By being able to specify the semantics of various notations, an SIF is comparable with our semantic definition schema. An SIF can be an existing formalism, such as higher-order logic [19], structural operational semantic format [20], or a new formalism, such as template semantics [21]. While TGFs and their SIFs strive for flexibility and extensibility, to accommodate for new notations, we have strived for creating a semantic definition framework that produces semantic definitions whose elements can be identified as high-level semantic concepts. We believe that the former approach is in the spirit of *descriptive* semantics, where as ours is in the spirit of *prescriptive* semantics. This is because we observe that TGFs often aim for almost open-ended extensibility and flexibility, and aim to support an unclear, broad range of semantics, which leads to a general, descriptive style of semantic definition. If an SIF is used in a prescriptive way, then there should exist a clear scope of notations in mind so that the formalism can be used in an active way to define a semantics prescriptively. But such a clear scope contradicts the open-ended extensibility and flexibility goals of a TGF. We suggest that for the SIF of a TGF to produce prescriptive semantics a task similar to what we undertook for BSMLs in our previous work [1, 2] should be carried out first. This task also serves to define a suitable SIF itself, as well as the scope of the flexibility and extensibility of the TGF.

The mechanism by which we define our semantics is influenced by that of template semantics [21]. In template semantics, a semantics is defined by: (i) instantiating values for the *template parameters* of its snapshot elements, and (ii) choosing, or defining, a set of *composition operators*. In particular, (i) we have adopted lines 1-5 in Fig. 6 from the definition of *macro step* in template semantics; and (ii) we have adapted the notion of snapshot elements in template semantics to model the dynamic parameters of a BSML semantics. We do not need the notion of a composition operator because the characteristics of a composition operator are manifested in our structural parameters. Compared to the template parameters of template semantics, the parameters of our semantic definition schema correspond to higher level semantic concerns, while having fewer dependencies among them.

Our semantic definition schema can be compared with general semantic definition methods that organize a semantic definition into a hierarchy of concepts, such as *modules* and *sub-modules*, in *Action Semantics* [26], or *theories* that are related by *linkings*, in *Unifying Theories of Programming* [13]. These methods promote modularity and orthogonality in semantic definition. In comparison, we have developed a specialized framework for BSML semantics whose concepts are parameters that correspond to our semantic aspects.

Lastly, our work is comparable to that of Huizing and Gerth [27]. Huizing and Gerth taxonomize and specify the semantics of events in BSMLs, covering the semantic options of our Event Lifeline semantic aspect in this paper. In comparison, we consider additional semantic aspects.

5 Conclusion and Future Work

We presented a prescriptive, parametric semantic definition framework for defining the semantics of big-step modelling languages (BSMLs). We showed how the semantics of a wide range of BSMLs, as identified in our previous work [1, 2], can be formally specified by our semantic definition schema. Because of the high level semantic aspects and the orthogonality of their corresponding semantic parameters in the semantic definition schema, we believe that our framework is a prescriptive way to define a BSML semantics, which produces an understandable semantic definition for the various stakeholders of the semantics. A key contribution of our formalization is a parameterization that matches the structural parameters into Concurrency and Consistency and Priority semantic aspects.

In our previous work [1,2], we analyzed the *advantages* and *disadvantages* of each semantic option in isolation, but not as a whole when considered together. We plan to use our formal semantic definition schema to identify useful BSML *semantic properties*, and the class of BSML semantics that satisfy each of them. We are also interested in developing a tool-support generator framework that respects the structure of our semantic definitions, so that we can inspect, or verify, the correctness of the implementation with respect to a semantic definition.

References

1. Esmaeilsabzali, S., Day, N.A., Atlee, J.M., Niu, J.: Semantic criteria for choosing a language for big-step models. In: RE 2009, pp. 181–190 (2009)
2. Esmaeilsabzali, S., Day, N.A., Atlee, J.M., Niu, J.: Deconstructing the semantics of big-step modelling languages. Submitted to Requirements Engineering Special Issue of RE 2009 (October 2009)
3. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming 8(3), 231–274 (1987)
4. von der Beeck, M.: A comparison of statecharts variants. In: Langmaack, H., de Roever, W.-P., Vytopil, J. (eds.) FTRTFT 1994 and ProCoS 1994. LNCS, vol. 863, pp. 128–148. Springer, Heidelberg (1994)
5. Maraninchi, F., Rémond, Y.: Argos: an automaton-based synchronous language. Computer Languages 27(1/3), 61–92 (2001)
6. Alur, R., Henzinger, T.A.: Reactive modules. Formal Methods in System Design 15(1), 7–48 (1999)
7. Berry, G., Gonthier, G.: The Esterel synchronous programming language: Design, semantics, implementation. Science of Computer Programming 19(2), 87–152 (1992)
8. Heninger, K.L., Kallander, J., Parnas, D.L., Shore, J.E.: Software requirements for the A-7E aircraft. Technical Report 3876, United States Naval Research Laboratory (1978)

9. Heitmeyer, C., Jeffords, R., Labaw, B.: Automated consistency checking of requirements specifications. ACM TOSEM 5(3), 231–261 (1996)
10. Ashcroft, E.A., Wadge, W.W.: Generality considered harmful: A critique of descriptive semantics. Technical Report CS-79-01, University of Waterloo, Cheriton School of Computer Science (1979)
11. Ashcroft, E.A., Wadge, W.W.: R/ for semantics. ACM TOPLAS 4(2), 283–294 (1982)
12. Pnueli, A., Shalev, M.: What is in a step: On the semantics of statecharts. In: Ito, T., Meyer, A.R. (eds.) TACS 1991. LNCS, vol. 526, pp. 244–264. Springer, Heidelberg (1991)
13. Hoare, T., Jifeng, H.: Unifying Theories of Programming. Prentice Hall, Englewood Cliffs (1998)
14. OMG: OMG Unified Modeling Language (OMG UML), Superstructure, v2.1.2, Formal/2007-11-01 (2007)
15. Leveson, N.G., Heimdahl, M.P.E., Hildreth, H., Reese, J.D.: Requirements specification for process-control systems. TSE 20(9), 684–707 (1994)
16. Harel, D., Naamad, A.: The Statemate semantics of statecharts. ACM TOSEM 5(4), 293–333 (1996)
17. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, SEI, Carnegie Mellon University (1990)
18. Pezzè, M., Young, M.: Constructing multi-formalism state-space analysis tools: Using rules to specify dynamic semantics of models. In: ICSE 1997, pp. 239–249 (1997)
19. Day, N.A., Joyce, J.J.: Symbolic functional evaluation. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) TPHOLs 1999. LNCS, vol. 1690, pp. 341–358. Springer, Heidelberg (1999)
20. Dillon, L.K., Stirewalt, K.: Inference graphs: A computational structure supporting generation of customizable and correct analysis components. IEEE TSE 29(2), 133–150 (2003)
21. Niu, J., Atlee, J.M., Day, N.A.: Template semantics for model-based notations. IEEE TSE 29(10), 866–882 (2003)
22. Lu, Y., Atlee, J.M., Day, N.A., Niu, J.: Mapping template semantics to SMV. In: ASE 2004, pp. 320–325 (2004)
23. Baresi, L., Pezzè, M.: Formal interpreters for diagram notations. ACM TOSEM 14(1), 42–84 (2005)
24. Gao, J., Heimdahl, M.P.E., Wyk, E.V.: Flexible and extensible notations for modeling languages. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 102–116. Springer, Heidelberg (2007)
25. Prout, A., Atlee, J.M., Day, N.A., Shaker, P.: Semantically configurable code generation. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 705–720. Springer, Heidelberg (2008)
26. Mosses, P.D.: Action Semantics. Cambridge Tracts in Theoretical Computer Science, vol. 26. Cambridge University Press, Cambridge (1992)
27. Huizing, C., Gerth, R.: Semantics of reactive systems in abstract time. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 291–314. Springer, Heidelberg (1992)