Semantic Quality Attributes 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 semantic quality attribute of a modelling language is a desired semantic characteristic that is common to all models specified in that language. A modeller can enjoy the luxury of not having to model the invariants of the behaviour that are implicitly enforced by the semantic quality attributes. In this paper, we introduce three semantic quality attributes for the family of big-step modelling languages (BSMLs). In a BSML, a model's reaction to an environmental input is a sequence of small steps, each of which can consist of the execution of a set of transitions from multiple concurrent components. Each of our three semantic quality attributes specifies a desired property about how the sequence of small steps form a big step. We systematically enumerate the range of BSML semantics that satisfy each semantic quality attribute.

1 Introduction

Often when modelling a software system, there are many alternative languages that could be used. To narrow the range of alternatives, a modeller needs to answer the question of why language *A*, and not language *B*, is a more appropriate choice in a certain context. In this paper, we consider this question for the class of Big-Step Modelling Languages (BSMLs) [7, 8] from a semantic point of view.

BSMLs are a class of state-transition modelling languages that are suitable for modeling systems that interact with their environments continuously. In a BSML model, the reaction of a system to an environmental input is modelled as a big step that consists of a sequence of small steps, each of which can be the execution of a set of transitions from multiple concurrent components.[1] Examples of BSMLs are statecharts [10, 18], its variants [2], Software Cost Reduction (SCR) [11, 12], and the un-clocked variants of synchronous languages [9], such as Esterel [4] and Argos [17]. The variety of semantics for events, variables, and control states introduce a range of semantics for BSMLs. Previously, we *deconstructed* the semantics of this family of languages into a set of high-level, orthogonal variation points, which we call semantic aspects, and enumerated the common semantic options of each semantic aspect [7, 8].

A BSML provides a modeller with the convenience of describing the reaction of a system to an environmental input as the execution of a set of transitions, facilitating the

[1] The terms *macro step* and *micro step* are related to our big step/small step terminology. We use our own terminology to avoid connotation with a fixed semantics associated with these terms.

decomposition of a model into concurrent components. However, it also introduces the complexity of dealing with the semantic intricacies related to the *ordering* of these transitions, making it difficult at times to recognize the global properties of these models.

A *semantic quality attribute* of a modelling language is a desired semantic characteristic that is common to all models specified in that language. Thus, a modeller can enjoy the luxury of not having to model the invariants of the behaviour that are enforced by the semantic quality attribute. Our first contribution in this paper is to introduce three semantic quality attributes for BSMLs. Two BSMLs can be compared and distinguished based on their semantic quality attributes. Each semantic quality attribute exempts modellers from worrying about some of the complications of ordering in the sequence of the small steps of a big step. The *priority consistency* attribute guarantees that higher priority transitions are chosen over lower priority transitions. The *non-cancelling* attribute guarantees that if a transition becomes executable during a big step, it remains executable, unless it is executed. The *determinacy* attribute guarantees that all possible orders of small steps in a big step have the same result.

Each of our semantic quality attributes for BSMLs is a cross-cutting concern over our semantic aspects for BSMLs. Our second contribution in this paper is to specify the subsets of BSML semantics that satisfy each of the semantic quality attributes. For each semantic quality attribute, we identify necessary and sufficient constraints over the choices of the semantic options in our deconstruction that result in a BSML semantics that has the semantic quality attribute. In previous work [7, 8], we analyzed the advantages and disadvantages of each of the semantic options individually, to provide rationales to choose one over another. The analysis of the semantic quality attributes in this paper reveals interrelationships among seemingly independent semantic options. It also provides rationales for language design decisions that otherwise would have seemed ad hoc. For example, the specification of non-cancelling BSML semantics highlights the role of concurrency in small-step execution, while the specification of priority-consistent and determinate BSML semantics highlights the role of limiting the number of transitions that each concurrent component of a model can execute in a big step. A language designer or a modeller can either (i) use the semantic quality attributes to narrow the range of semantic options for a language, or (ii) gain insights about a language's attributes after choosing its semantic options.

Compared to related work, we introduce three novel semantic quality attributes for a broader range of modelling languages than considered previously. A notable example of introducing semantic quality attributes is Huizing and Gerth's work on the semantics of BSMLs that support only events. They introduced three semantic quality attributes for the semantics of events, therefore they only needed to deal with one semantic aspect and not cross-cutting concerns. Similar semantic properties to our semantic quality attributes have been considered in primitive formalisms [5, 14, 15, 16, 19], which are models of computations rather than practical languages. These efforts characterize a class of models that satisfy a property whereas we determine the set of BSML semantics that satisfy each semantic quality attribute. For example, our non-cancelling semantic quality attribute is similar to *persistence* for program schemata [15] and for Petri Nets [16].

The remainder of the paper is organized as follows. Section 2 presents an overview of the common syntax and semantics of BSMLs, together with an overview of BSML semantic aspects and semantic options. Section 3 formally presents the semantic quality attributes together with the specification of the subsets of BSML semantics that satisfy each of the semantic quality attributes. Section 4 discusses related work. Section 5 concludes the paper.

2 Background: Big-Step Modelling Languages (BSMLs)

In this section, we present an overview of our deconstruction of the semantics of BSMLs. Section 2.1 presents the common syntax of BSMLs. Section 2.2 presents the common semantics of BSMLs, with Section 2.3 describing its semantic variation points through our deconstruction into semantic aspects and their semantic options. In our framework, an existing BSML is modelled by translating its syntax into the normal form syntax and its semantics into a set of semantic options. Our previous work contains a more comprehensive and a formal treatment of these concepts [6, 7, 8].

2.1 BSML Syntax

We use a normal form syntax to model the syntax of many BSMLs. In our normal form syntax, a BSML model is a graphical, hierarchical, extended finite state machine, consisting of: (i) a hierarchy tree of control states, and (ii) a set of transitions between these control states. In this section, we adopt a few syntactic definitions from Pnueli and Shalev's work [18].

Control states. A **control state** is a named artifact that a modeller uses to represent a noteworthy moment in the execution of a model. A control state has a **type**, which is one of *And*, *Or*, or *Basic*. The set of control states of a model form a hierarchy tree. The leaves of the **hierarchy tree** of a model, and only they, are *Basic* control states. In a hierarchy tree, an *And* or an *Or* control state has a set of **child** control states. A control state is a **descendant** of another control state if it is its child through transitivity. Similarly, the **parent** and **ancestor** relations are defined. Two control states **overlap** if they are the same or one is an ancestor of the other. The children of an *And* control state are separated by dashed lines graphically. One of the children of an *Or* control state is its **default** control state, which is signified by an arrow without a source. Fig. 1 is an example BSML model that we use for illustration. The model specifies the behaviour of a system that controls the safety of an entrance to an industrial area. The system is either in the *Automatic* or in the *Manual* control state. The children of control state *Automatic* are *Or* control states *Temp*, *Lock*, and *Fan*. The default control state of *Temp* is *Low*, which is a *Basic* control state. Control state *Off* is one of the descendants of the *Automatic* control state. The **least common ancestor** of a set of control states is the lowest (closest to the leaves) control state in the hierarchy tree such that each of the control states is its descendant; e.g., the least common ancestor of *Locked* and *Unlocked* is *Lock*. Two control states are **orthogonal** if neither is an ancestor of the other and their least common ancestor is an *And* control state; e.g., *Locked* and *Off* are orthogonal.

Transitions. Each transition, t, has a **source control state**, $src(t)$, and a **destination control state**, $dest(t)$, together with the following four optional elements: (i) a **guard condition** (GC), $gc(t)$, which is a boolean expression over a set of variables, enclosed by a "[ ]"; (ii) a **triggering condition**, $trig(t)$, which is the conjunction of a set of events

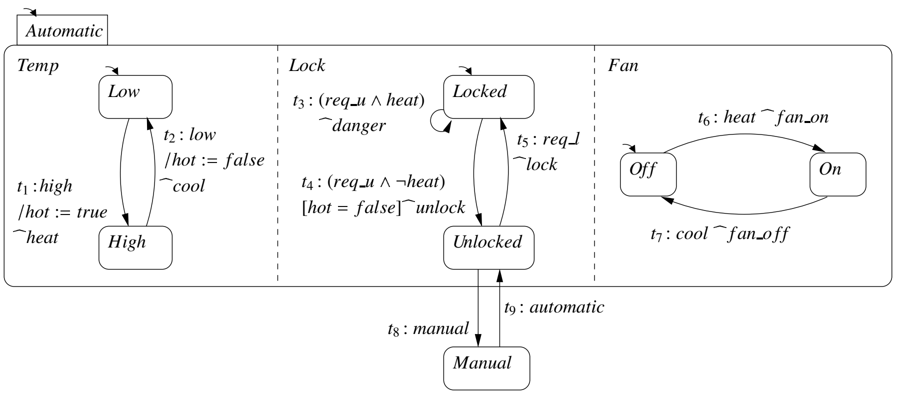
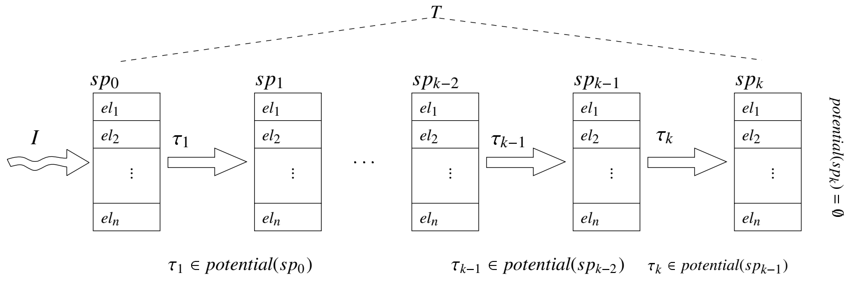


Fig. 1. Example: Industrial entrance control system

and negation of events; (iii) a set of **variable assignments**, $asn(t)$, which is prefixed by a "/", with at most one assignment to each variable; and (iv) a set of **generated events**, $gen(t)$, which is prefixed by a "⁀". For example, in the model in Fig. 1, t_4 is a transition, with $src(t_4) = Locked$, $dest(t_4) = Unlocked$, $gc(t_4) = (hot = false)$, $asn(t_4) = \emptyset$, $trig(t_4) = (req\_u \wedge \neg heat)$, and $gen(t_4) = \{unlock\}$. When a set is a singleton, we drop its curly brackets; e.g., $asn(t_1) = hot := true$. The **scope** of a transition is the least common ancestor of its source and destination control states. The **arena** of a transition is the lowest *Or* control state that is an ancestor of its source and destination control states. The **root** of the hierarchy tree of a model, which is not always explicitly shown, is an *Or* control state so that the arena of each transition is defined. For example, in the model in Fig. 1, the scope of t_4 is *Lock* and the arena of t_8 is the root *Or* control state not shown in the figure. Two transitions are **orthogonal** if their source control states are orthogonal, as well as, their destination control states; e.g., t_4 and t_6. A transition, t, is an **interrupt for** another transition, 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; 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'. For example, in the model in Fig. 1, t_8 is an interrupt for t_6, according to condition (i).

2.2 Common Semantics of BSMLs

A BSML model describes the continuous interaction of a system with its environment. A BSML model reacts to an environmental input through a **big step**, which consists of an alternating sequence of snapshots and **small steps**, with each small step consisting of the execution of a set of transitions. An **environmental input** consists of a set of events and assignments from the environment that are used throughout a big step. In the model in Fig. 1, events *high*, *low*, *req_u*, *req_l*, *manual*, and *automatic* are environmental input events. A **snapshot** is a collection of **snapshot elements**, each of which maintains information about an aspect of computing a big step. There is a snapshot element that maintains the set of current control states of a model: If a model resides in an *And* control state, it resides in all of its children; if a model resides in an *Or* control state, it resides in one of its children, by default in its default control state. The execution of a small step causes a set of control states in the snapshot element to be **exited** and a set of control states to be **entered**. These sets are computed based on the scopes of the transitions in a small step. There are other snapshot elements for variables, events, etc.



T
sp_0
sp_1
sp_{k-2}
sp_{k-1}
sp_k
el_1
el_2
el_n
I
τ_1
τ_{k-1}
τ_k
$potential(sp_k) = \emptyset$
$\tau_1 \in potential(sp_0)$
$\tau_{k-1} \in potential(sp_{k-2})$
$\tau_k \in potential(sp_{k-1})$

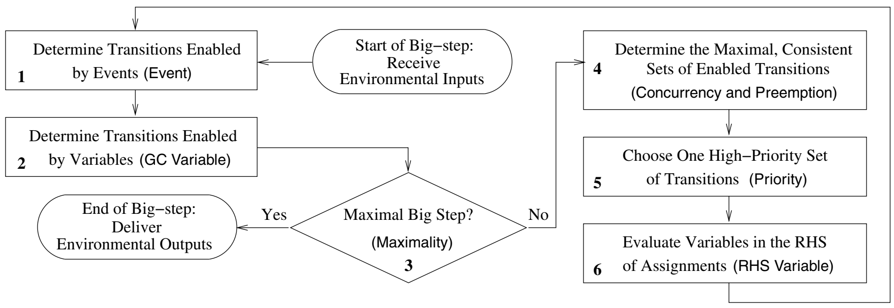
Fig. 2. Big step $T = \langle I, sp_0, \tau_1, sp_1, \cdots, \tau_k, sp_k \rangle$

Fig. 2 depicts the structure of a big step, which relates a source snapshot, sp_0, and an environmental input, I, with a destination snapshot, sp_k. The effect of receiving I, which is captured in snapshot sp_0, generates a sequence of small steps until there are no more small steps to be executed. Function *potential* specifies the set of all **potential small steps** at a snapshot, one of which is non-deterministically chosen as the next small step. Each big step or small step has a **source snapshot** and a **destination snapshot**. Formally, we represent a big step, T, as a tuple, $T = \langle I, sp_0, \tau_1, sp_1, \cdots, \tau_k, sp_k \rangle$. We use the accessor functions, *length*, *smallsteps*, and *trans*, to access the number of small steps of a big step, the set of sets of transitions representing the small steps of the big step, and the set of all transitions executed by the small steps of the big step, respectively. We use "." to access an element of a tuple. As an example, for a big step, T, $T.sp_i$, where $1 \leq i \leq length(T)$, is the destination snapshot of its i th small step. For a BSML model M, we denote **all its possible big steps** as $bigsteps(M)$. This set includes all possible big steps in response to all environmental inputs at all possible snapshots. In our examples, we refer to a big step via its constituent sequence of sets of transitions.

2.3 BSML Semantic Variations

Fig. 3a, adapted from [7], describes our deconstruction of the semantics of BSMLs into six stages that each corresponds to a semantic variation point that affects how a big step is created. We call these variation points **semantic aspects**, and the possible variations of each its **semantic options**. Table 3b, which we will refer to throughout the rest of the paper, lists our semantic aspects and their semantic options, together with a brief description of each. We use the Sans Serif and the Small Cap fonts to refer to the name of a semantic aspect and a semantic option, respectively.

A big step is created by iterating through the stages of the flowchart in Fig. 3a. One iteration of the flowchart corresponds to one small step. For each transition, t, stage 1 determines whether $trig(t)$ is true with respect to the statuses of events, according


Start of Big−step: Receive Environmental Inputs
1 Determine Transitions Enabled by Events (Event)
2 Determine Transitions Enabled by Variables (GC Variable)
Maximal Big Step? (Maximality) 3
Yes
No
End of Big−step: Deliver Environmental Outputs
4 Determine the Maximal, Consistent Sets of Enabled Transitions (Concurrency and Preemption)
5 Choose One High−Priority Set of Transitions (Priority)
6 Evaluate Variables in the RHS of Assignments (RHS Variable)

(a) Semantic variation points in the operation of a big step

Aspect/Option	Description
Event	*Specifies the extent that a generated event is present in a big step:*
REMAINDER	In all snapshots of the big step after it is generated.
NEXT SMALL STEP	In the destination snapshot of the small step that it is generated in.
NEXT BIG STEP	In all snapshots of the next big step after the big step it is generated in.
GC Variable	*Specifies the snapshot from which values of variables for checking the guard condition of a transition are obtained:*
GC BIG STEP	Obtains the value of a variable from the beginning of the big step.
GC SMALL STEP	Obtains the value of a variable from the source snapshot of the small step.
Maximality	*Specifies when a sequence of small steps concludes as a big step:*
TAKE ONE	Once a transition, t, is executed during a big step, no other transition whose arena overlaps with the arena of t can be executed.
TAKE MANY	No constraints on the sequence of small steps.
Concurrency	*Specifies the number of transitions that can be taken in a small step:*
SINGLE	Exactly one transition per small step.
MANY	More than one transition possible per small step.
Preemption	*Specifies whether interrupting transitions can execute in a small step:*
PREEMPTIVE	Two transitions, one an interrupt for the other, cannot be taken together.
NON-PREEMPTIVE	Two transitions, one an interrupt for the other, can be taken together.
Priority	*Specifies if a transition has a higher priority than another:*
NEGATION	For a pair of transitions, t and t', t is assigned a higher priority than t' by conjoining $trig(t')$ with the negation of a positive event in $trig(t)$.
SCOPE-CHILD	The lower the scope of a transition, the higher its priority is.
SCOPE-PARENT	The higher the scope of a transition, the higher its priority is.
NO PRIORITY	Neither SCOPE CHILD nor SCOPE PARENT is chosen.
RHS Variable	*Specifies the snapshot from which values of variables for evaluating the right-hand side (RHS) of an assignment are obtained:*
RHS BIG STEP	Obtains the value of a variable from the beginning of the big step.
RHS SMALL STEP	Obtains the value of a variable from the source snapshot of the small step.

(b) Semantic aspects (in Sans Serif font) and their options (in SMALL CAP font)

Fig. 3. Semantic variations in the operation of a big step

to the Event semantic aspect. The three semantic options of this semantic aspect each determines a different extent of a big step (the current big step or the next one) that a generated event is considered as present. Stage 2 determines whether $gc(t)$ is true with respect to the values of variables, according to the GC Variable semantic aspect. The first semantic option for this semantic aspect uses the values of variables from the beginning of the big step, whereas the second option uses the up-to–date values of variables from the source snapshot of the small step. Stages 1 and 2, together with the current set of control states of a model, determine the enabledness of individual transitions. Stage 3 checks whether the current big step is **maximal**, according to the Maximality semantic aspect, in which case, the big step ends. The first semantic option for this semantic aspect requires that if a transition, t, has been executed in a big step, no other transition whose arena overlaps with t's can be executed. As an example, according to the first semantic option, in the model in Fig. 1, if t_9 is executed, t_5 cannot be executed because the arena of t_5 overlaps with the arena of t_9. The second semantic option places no constraint over the maximality of a big step: A big step can continue as long as there are enabled transitions that can be executed.

If the big step is not maximal, stage 4 determines the sets of transitions that can be taken together. Each such set is **complete** in that no transition can be added to it without violating the two related semantic sub-aspects of the Concurrency and Preemption semantic aspect. The Concurrency sub-aspect specifies the number of transitions possible in a small step, with two possible semantic options: one vs. many. When the latter semantic option is chosen, two transitions can be included in the same small step if they are orthogonal. The Preemption sub-aspect determines whether a pair of transitions where one is an interrupt for another can be taken together or not. Among the sets of transitions produced by stage 4, the sets that have the highest priority, according to the Priority semantic aspect, are chosen by stage 5; the result is the set of potential small steps at the current snapshot. The options for priority semantics include using the negation of events to assign a transition, t, a higher priority than a transition, t', by including one of the positive events in $trig(t)$ as a negated event in $trig(t')$. For example, in the model in Fig 1, t_3 has a higher priority than t_4, denoted by $pri(t_3) > pri(t_4)$, because $trig(t_3)$ includes *heat* while $trig(t_4)$ includes $\neg heat$. Additionally, if the No Priority semantic option is not chosen, one of the two *hierarchical* priority semantic options is used to compare the priority of two transitions based on their scopes. Stage 6 evaluates the right-hand side (RHS) of an assignment, according to the RHS Variable semantic aspect, when executing one of the potential small steps. Similar semantic options as the ones for the GC Variable semantic aspect are possible for the RHS Variable semantic aspect, but are used to evaluate the RHS of assignments instead of their GCs.

We use the following notation to describe the semantic quality attributes. The **set of enabled transitions** of a model at a snapshot, sp, denoted by $enabled(sp)$, is the set of all transitions such that, for each transition, its source control state is in the current set of control states and it passes stages 1 and 2 of flowchart in Fig. 3a. Similarly, the **set of executable transitions** at snapshot sp, denoted by $executable(sp)$, is the set of all transitions that belong to a potential small step at sp. Formally, $\forall t \in executable(sp) \Leftrightarrow \exists \tau \in potential(sp) \cdot t \in \tau$. By definition, $t \in executable(sp) \Rightarrow t \in enabled(sp)$, but not vice versa, because of the Maximality and Priority semantic aspects.

In this paper, we consider BSML semantics in which the executability of each small step relies only on its source snapshot [6]. A few of the semantic variations that are not considered here lend themselves to a different set of semantic quality attributes, as will be briefly discussed in Section 4.

3 Semantic Quality Attributes for BSMLs

In this section, we formally describe three semantic quality attributes for BSMLs. For each semantic quality attribute, we enumerate all combinations of the semantic options in Table 3b that each results in a semantics that satisfies the semantic quality attribute.

3.1 Priority Consistency

In a *priority-consistent* BSML semantics, higher priority transitions are chosen to execute over lower priority transitions. A model cannot have two big steps, T_1 and T_2, where T_1 includes transitions that are all of lower or incomparable priority than T_2's. A priority semantic option enforces a priority semantics only in the individual small steps of a big step, but not in the whole big step. A priority-consistent BSML semantics is useful since it exempts a modeller from worrying about a high-priority transition being executed in some big steps but not others. We call a semantics that is not priority consistent, *priority inconsistent.*

Example 1. The two models in Fig. 4 are used to demonstrate examples of priority-inconsistent behaviour. Both models are considered when they reside in their default control states, environmental input event i is present, and, for the first model, when $x = 0$. In the model in Fig. 4(a), we consider a BSML semantics that subscribes to the Many concurrency semantics, the Take Many maximality semantics, the Scope-Parent priority semantics, and the GC Small Step GC variable semantics. Two big steps are possible: $T_1 = \langle\{t_1, t_4\}, t_2, t_6\rangle$ and $T_2 = \langle\{t_1, t_4\}, t_3, t_5\rangle$. This behaviour is priority inconsistent since $pri(t_6) > pri(t_5)$: the scope of t_6 is the parent of the scope of t_5, and T_1 executes t_6 but T_2 executes t_5. For the model in Fig. 4(b), the same semantic options as for the model in Fig. 4(a) are considered, plus the Remainder event semantics. Two big steps are possible: $T'_1 = \langle\{t_1, t_2\}, t_5\rangle$ and $T'_2 = \langle\{t_1, t_3\}, t_4\rangle$. This behaviour is priority inconsistent since $pri(t_5) > pri(t_4)$ according to the Negation semantics: $trig(t_5)$ includes b while $trig(t_4)$ includes $\neg b$, and T'_1 executes t_5 but T'_2 executes t_4. If the Next Big Step event semantics had been chosen, which requires a generated event to be present only in the next big step, two priority-consistent big steps would have been possible: $T'_1 = \langle\{t_1, t_2\}\rangle$ and $T'_2 = \langle\{t_1, t_3\}\rangle$, followed by big steps $\langle t_5\rangle$ and $\langle t_4\rangle$, respectively.

Definition 1. *A BSML semantics is* priority consistent *if for all BSML models, M,*

$$\forall T_1, T_2 \in bigsteps(M) \cdot (T_1.I = T_2.I) \wedge (T_1.sp_0 = T_2.sp_0) \Rightarrow trans(T_1) \doteq trans(T_2),$$

where

$$\tau_1 \doteq \tau_2 \equiv \neg(\tau_1 \gtrdot \tau_2) \wedge \neg(\tau_2 \gtrdot \tau_1), \text{ and}$$
$$\tau_1 \gtrdot \tau_2 \equiv (\exists t_1 \in \tau_1 \cdot \exists t_2 \in \tau_2 \cdot pri(t_1) > pri(t_2)) \wedge \neg(\exists t_2 \in \tau_2 \cdot \exists t_1 \in \tau_1 \cdot pri(t_2) > pri(t_1)).$$

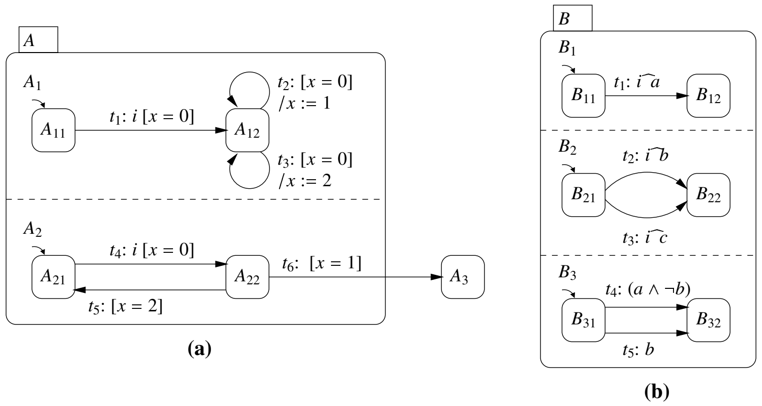


Fig. 4. Examples of priority-inconsistent behaviour

Intuitively, $\tau_1 \doteq \tau_2$ *if it is not the case that* τ_1 *has a transition that has a higher priority than a transition in* τ_2 *without* τ_2 *having such a transition, and also not vice versa.*

Priority-Consistent Semantics. Proposition 1 specifies the BSML semantics that are priority consistent. We use the name of a semantic option as a proposition to specify all BSML semantics that subscribe to it.

Proposition 1. *A BSML semantics is priority-consistent if and only if its constituent semantic options satisfy predicate* **P** ≡ **P1** ∧ **P2***, where*

$$\mathbf{P1} \equiv \neg\text{No Priority} \Rightarrow \text{Take One}, \text{ and } \mathbf{P2} \equiv \text{Next Big Step}.$$

Proof Idea. Predicate **P1** is a necessary condition for a BSML semantics to be priority-consistent. This can be proven by contradiction. If **P1** is not true in a priority-consistent BSML semantics, a *counter example model* with a priority-inconsistent behaviour can be constructed. For example, for any BSML that subscribes to the Scope-Parent priority semantics and the Take Many maximality semantics, the model in Fig. 5 is such a counter example model: when the model resides in its default control states, two big steps, $T_1 = \langle t_1, t_3 \rangle$ and $T_2 = \langle t_2 \rangle$, are possible, but $trans(T_1) \succ trans(T_2)$ because $pri(t_3) > pri(t_2)$. Predicate **P2** is a necessary condition for a priority-consistent BSML semantics, because for a BSML semantics that subscribes to an event semantics other than the Next Big Step semantics, regardless of its other semantic options, there is a counter example model with a priority-inconsistent behaviour. For example, the model in Fig. 4(b), in Example 1, would have a priority-inconsistent behaviour even if the Single concurrency and/or the Next Small Step event semantic options are chosen. Predicates **P1** and **P2** are also sufficient conditions for priority-consistent BSML semantics, according to a hierarchical and the Negation priority semantics, respectively. This can be proven by showing that if **P** is true in a BSML semantics, a high-priority transition is either executable and

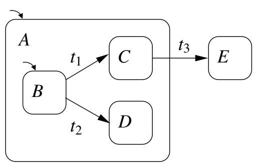


Fig. 5. A counter example

taken during a big step, or it cannot become executable during that big step. If a BSML semantics subscribes to the SCOPE-PARENT (SCOPE-CHILD) priority semantics, a transition that has a higher (lower) scope than an already executed transition cannot become executable because of the TAKE ONE maximality semantics. Similarly, for the NEGATION priority semantics, if a high-priority transition is not enabled due to the absence of an event, it cannot possibly become enabled in that big step, because the statuses of events do not change. As such, a BSML semantics is priority consistent if and only if **P**. □

3.2 Non-cancelling

In a *non-cancelling* BSML semantics, once a transition of a model becomes executable in a big step, it remains executable during the big step, unless, it is taken by the next small step, it cannot be taken any more because of the maximality constraints, or its scope is the same as or a descendant of the scope of a transition executed in the next small step. A non-cancelling BSML semantics is useful since it exempts a modeller from worrying about an enabled transition of interest mistakenly becoming disabled. We call a BSML semantics that is not non-cancelling, *cancelling*.

Example 2. We consider the model in Fig. 1 when it is in its initial control states and when events *req_u* and *high* are received from the environment. We choose the TAKE MANY maximality semantics. Initially, t_4 is executable. If t_1 is executed in the first small step, making *heat* present and $hot = true$, t_4 would not be enabled any more, although the big step is not maximal and the scope of t_4 does not overlap with t_1's. This is a cancelling behaviour, which can be averted by executing t_1 and t_4 together.

Definition 2. *A BSML semantics is* non-cancelling *if for all BSML models, M,*

$$\forall T \in bigsteps(M) \cdot \forall i\ (1 \leq i \leq length(T)) \cdot \forall t \in executable(sp_{i-1}) \Rightarrow \\ (t \in \tau_i) \vee (t \in executable(sp_i)) \vee (\exists t' \in \tau_i . tookone(t', t) \vee dominated(t', t)),$$

where tookone(t', t) is true, if by executing t', t cannot be taken because of the TAKE ONE *maximality semantics of the BSML; and dominated(t', t) is true, if the scope of t is the same as or a descendent of the scope of t'; e.g., $src(t) = dest(t')$.*

The rationale for including the third disjunct in Definition 2 is twofold; when predicate *tookone*(t', t) is true, t is not a transition of interest any more; and when predicate *dominated*(t', t) is true, the execution of t' has entered and/or exited the scope of t. As such, whichever predicate is true, it is natural to consider the enabledness of t afresh.

Achieving a non-cancelling BSML semantics not only relies on enabledness and concurrency semantics but also on hierarchical semantics, as the next example shows.

Example 3. We consider a modified version of the model in Fig. 4(a) in which $gc(t_2) = gc(t_3) = true$. We consider this new model when it resides in control states A_{12} and A_{22}, and $x = 2$. If a BSML semantics that subscribes to the SINGLE concurrency, the SCOPE-PARENT priority, and the GC SMALL STEP GC variable semantic options is chosen, initially both t_5 and t_2 are executable, but if t_2 is executed, t_5 becomes unexecutable because t_6 becomes enabled and $pri(t_6) > pri(t_5)$. If the MANY concurrency semantics had been chosen, instead of the SINGLE concurrency semantics, the above cancelling behaviour would have been averted because t_2 and t_5 would have been executed together.

Non-Cancelling Semantics. A non-cancelling semantics can be achieved by one of the following two approaches: (i) once a transition becomes executable, the execution of other transitions does not affect its executability; or (ii) once a transition becomes executable, it is immediately executed, precluding the possibility of becoming unexecutable. For example, by choosing the NEXT BIG STEP event semantics together with the NO PRIORITY semantics, a non-cancelling semantics via approach (i) can be achieved. By choosing the NEXT SMALL STEP semantics together with a concurrency semantics that ensures that an executable transition is executed immediately, a non-cancelling semantics via approach (ii) can be achieved. Formally,

Proposition 2. *A BSML semantics is non-cancelling if and only if its constituent semantic options satisfy predicate* **N** ≡ **N1** ∧ **N2** ∧ **N3***, where*

$$\begin{aligned}
\mathbf{N1} &\equiv (\text{TAKE MANY} \wedge \neg\text{NO PRIORITY}) \Rightarrow \mathit{Maximizer},\\
\mathbf{N2} &\equiv \text{NEXT SMALL STEP} \Rightarrow \mathit{Maximizer},\\
\mathbf{N3} &\equiv \text{GC SMALL STEP} \Rightarrow \mathit{Maximizer}, \text{ and}\\
\mathit{Maximizer} &\equiv \text{MANY} \wedge [(\text{TAKE MANY} \wedge \text{NO PRIORITY}) \Rightarrow \text{NON-PREEMPTIVE}].
\end{aligned}$$

Proof Idea. Predicate **N** is a sufficient condition for a non-cancelling BSML semantics because it characterizes only non-cancelling BSML semantics. When **N1**, **N2**, and **N3** are all satisfied through their antecedents being false, a non-cancelling semantics according to approach (i) above is achieved, otherwise a non-cancelling semantics according to approach (ii) can be achieved. When the antecedents of **N1**, **N2**, and **N3** are all false, it can be proven that if an executable transition that is not executed in the immediate small step becomes unexecutable, it is because of the maximality semantics, and not because it is not enabled or does not have a high priority any more. When the antecedents of at least one of **N1**, **N2**, or **N3** is true, predicate *Maximizer* requires the MANY concurrency semantics to ensure that as many as possible enabled transitions are executed immediately. The second conjunct of the *Maximizer* predicate ensures that when the TAKE MANY semantics is chosen, a pair of executable transitions that have the same priority and one is an interrupt for the other are taken together in the same small step, according to the NON-PREEMPTIVE semantics; otherwise, the execution of the lower-scope transition can disable the higher-scope transition. To prove that predicate **N** is also a necessary condition for any non-cancelling BSML semantics, it can be shown that **N** characterizes all possible non-cancelling BSML semantics. This can be proven, via proofs by contradiction, to show that: (a) approaches (i) and (ii) above are the only ways to achieve a non-cancelling semantics; and (b) predicate **N** does not over-constrain the choices of semantic options. □

3.3 Determinacy

In a *determinate* BSML semantics, if two big steps of a model in response to the same environmental input execute the same (multi) set of transitions in different orders, their destination snapshots are *equivalent*. An equivalence relation, denoted by "≡", can be defined with respect to any subset of the snapshot elements. We consider determinacy for both events and variables. A determinate BSML semantics is useful since it exempts a modeller from worrying about the effect of an out-of–order execution of the transitions in a big step. We call a BSML semantics that is not determinate, *non-determinate*.

Definition 3. *A BSML semantics is* determinate *if for all BSML models, M,*

$$\forall T_1, T_2 \in bigsteps(M) \cdot [(T_1.I = T_2.I) \wedge (T_1.sp_0 = T_2.sp_0) \wedge$$
$$(\biguplus_{\tau_1 \in smallsteps(T_1)} \tau_1 = \biguplus_{\tau_2 \in smallsteps(T_2)} \tau_2)] \Rightarrow T_1.sp_{length(T_1)} \equiv T_2.sp_{length(T_2)},$$

where "$\uplus$" is the sum operator for multisets.

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

Definition 4. *A big step, T, is* single assignment *if there are no two transitions in the big step that assign values to the same variable. A BSML model, M, is* single assignment *if all big steps $T \in bigsteps(M)$ are single assignment.*

A crude way to achieve single assignment models is to require the TAKE ONE maximality semantics and that at most one transition of a model assigns a value to each variable.

Example 4. The model in Fig. 6 controls the operation of a chemical plant.[2] The environmental input events *inc_one* and *inc_two* indicate that the amount of a chemical substance in the plant needs to be incremented by one or two units, respectively. We consider the model when: it resides in its default control states, $inc = inc_1 = inc_2 = 0$, and the environmental input events *inc_one* and *inc_two* are received together. The model is single-assignment only if environmental input event *reset* is received neither with *inc_one* nor with *inc_two*. If a BSML semantics subscribes to the MANY, the TAKE MANY, the REMAINDER, the GC SMALL STEP, and the RHS SMALL STEP semantic options, two big steps are possible: $T_1 = \langle t_1, t_5, \{t_2, t_7\}, t_3, t_4\rangle$ and $T_2 = \langle t_3, t_5, \{t_4, t_7\}, t_1, t_2\rangle$. T_1 assigns the value one to *inc* while T_2 assigns value two, which is a non-determinate behaviour because T_1 and T_2 execute the same set of transitions. If the RHS BIG STEP semantic option had been chosen, instead of the RHS SMALL STEP semantic option, the assignment to *inc* by t_5 would have read the values of inc_1 and inc_2 from the beginning of the big step, and thus a determinate behaviour would have been achieved.

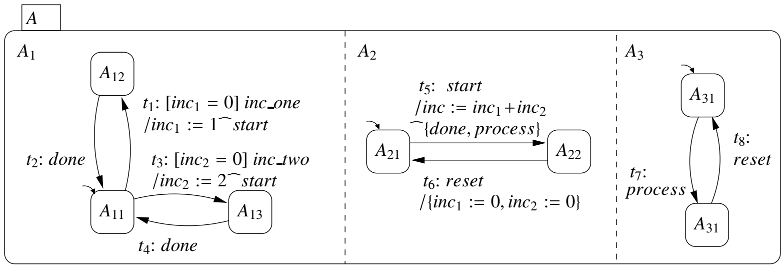


Fig. 6. An example of a non-determinate behaviour

Determinate Semantics. First, we present a lemma that explains why the first semantics in Example 4 is not determinate, but would have been so if the TAKE ONE semantic option had been chosen. We then specify the class of all determinate BSML semantics.

[2] This model is adapted from the sequence-chart model in the motivating example in [1].

Lemma 1. *In a BSML semantics that subscribes to the* Take One *and* Many *semantic options, starting from the same snapshot, if two big steps, T_1 and T_2, of a single-assignment model consist of the same sets of transitions, then they are the same.*

Proof Sketch. This claim can be proven inductively by traversing over the sequence of small steps of T_1 and T_2. Starting from snapshots $T_1.sp_0$ and $T_2.sp_0$, their first small steps, $T_1.\tau_1$ and $T_2.\tau_1$, should be the same. If not, let us assume that there exists a transition, t, such that $t \in (T_1.\tau_1 - T_2.\tau_1)$. However, such a t cannot exist: Transition t can only be not taken by $T_2.\tau_1$ if it is replaced by a $t' \in T_2.\tau_1$ such that t and t' cannot be taken together according to the Concurrency and Preemption semantics. But if that is true, T_2 can never execute t because the Take One maximality semantics precludes the possibility of such a t being taken after t' had been taken. Thus, it should be the case that $T_1.\tau_1 = T_2.\tau_1$. Similarly, it should be the case that all $T_1.\tau_i$'s and $T_2.\tau_i$'s, $1 < i \leq length(T_1)$, are the same. Therefore, T_1 and T_2 are the same. □

Proposition 3. *A BSML semantics is determinate if and only if its constituent semantic options satisfy predicate* $\mathbf{D} \equiv \mathbf{D1} \wedge \mathbf{D2}$*, where*

$\mathbf{D1} \equiv$ RHS Big Step $\vee$ (RHS Small Step $\Rightarrow$ (Take One $\wedge$ Many)), and
$\mathbf{D2} \equiv$ (Remainder $\vee$ Next Big Step) $\vee$ (Next Small Step $\Rightarrow$ (Take One $\wedge$ Many)).

Proof Idea. Predicate **D** is a sufficient condition for a BSML semantics to be determinate. Predicate **D1** deals with determinacy for variables, while predicate **D2** deals with determinacy for events. The first disjunct of **D1** achieves determinacy for variables because of the single-assignment assumption and the fact that early assignments do not affect the later ones. Similarly, the first disjunct of **D2** achieves determinacy for events because the Remainder and Next Big Step semantics both accumulate all of the generated events of a big step. The second disjuncts of **D1** and **D2** are valid characterization of determinate BSML semantics because of Lemma 1. Predicate **D** is also a necessary condition for a determinate BSML semantics. If **D1** does not hold for a determinate BSML semantics, then it means that it subscribes to the RHS Small Step assignment semantics but not to both the Take One maximality semantics and the Many concurrency semantics. If the BSML semantics does not subscribe to the Take One maximality semantics, then the model in Example 4 shows a non-determinate behaviour for such a BSML semantics, which is a contradiction. If the BSML semantics does not subscribe to the Many concurrency semantics, but subscribes to the Take One, a counter example model can be constructed, as follows. We consider the following three orthogonal enabled transitions in a model, $t_1 : /a := 1$, $t_2 : /b := 1$, and $t_3 : /c := a+b$, when $a = b = 0$. Based on the order of the execution of t_1, t_2, and t_3, the value of c is either zero, one, or two, which is a non-determinate behaviour. Similarly, it can be proven that **D2** is a necessary condition for a determinate BSML semantics. Thus, **D** is necessary and sufficient condition for a BSML semantics to be determinate. □

4 Related Work

Huizing and Gerth identified the three semantic criteria of *responsiveness*, *modularity*, and *causality* for the Single concurrency semantics, the Take One maximality semantics, and events semantics [13] only. Their modularity criterion requires that a generated event by a model is treated the same as an event received from the environment. The two semantics in their framework that are modular, namely, semantics A and D, can be shown to be also non-cancelling. Semantics A corresponds to the NEXT BIG STEP event lifeline semantics; semantics D corresponds to the WHOLE event lifeline semantics. In the WHOLE event lifeline semantics, which we informally considered in our deconstruction [7, 8], a generated event is present throughout the big step in which it is generated.

Pnueli and Shalev introduced a *globally consistent* event semantics [18], which is the same as the REMAINDER semantics except that if the absence of an event has made a transition enabled in a small step, the event is not generated later. Global consistency and priority consistency with respect to the NEGATION priority semantics are comparable. The difference is that the former is defined at the level of individual big steps whereas the latter is defined at the level of all big steps that have the same source snapshot.

Synchronous languages are used to model/program reactive systems that are meant to behave deterministically [9]. We have categorized the un-clocked variations of synchronous languages, such as Esterel [4] and Argos [17], as BSMLs that support the WHOLE event lifeline semantics [7, 8]. A model is deterministic if its reaction to an environmental input as a big step always results in a unique destination snapshot. Determinism is related to determinacy: A deterministic semantics is by definition determinate, but not vice versa. A determinate semantics does not preclude the possibility of a model reacting to a single environmental input via two big steps with different sets of transitions. In the presence of variables, determinism can be considered only as a property of a model but not of a semantics, because, as opposed to events, variables can have infinite ranges, making it impossible to handle determinism at the level of the description of a semantics. In the absence of variables, however, deterministic semantics for Esterel [3, 20] and Argos [17] have been developed.

Similar concepts as our semantic quality attributes have been considered in different models of computation, but at the level of models instead of semantics. For example, in Petri nets, the notion of *persistence* [16], which requires a transition to remain enabled until it is taken, is similar to our non-cancelling semantic quality attribute. In asynchronous circuits, the notions of *semi-modularity* and *quasi semi-modularity* are similar to our non-cancelling semantic quality attribute, and the notion of *speed independence* is analogous to our determinacy semantic quality attribute [5, 19]. Janicki and Koutny introduce the notion of *disabling* in the context of a relational model of concurrency [14], which is similar to our priority consistency semantic quality attribute. Lastly, the notions of *persistence* and *determinacy*[3] for *program schemata* [15] are analogous to our non-cancelling and determinacy semantic quality attributes, respectively. In general, compared to the aforementioned concepts, (i) our semantic quality attributes are defined for semantics, rather than individual models; and (ii) they are aimed at practical requirements modelling languages, instead of models of computation.

5 Conclusion and Future Work

In this paper, we introduced three semantic quality attributes, namely, non-cancelling, priority consistency, and determinacy, for the family of big-step modelling languages

[3] We have adopted the name of our semantic quality attribute from this work.

(BSMLs). When a BSML supports a semantic quality attribute, modellers are exempt from worrying about certain complications of the ordering of transitions in a model. We formally specified the subsets of BSML semantics that satisfy each semantic quality attribute. Next, we plan to identify combinations of meaningful syntactic well-formedness constraints and semantic options that achieve a semantic quality attribute. For example, if the syntax of a BSML is restricted to *simple* transitions, where a transition, t, is simple if $parent(src(t)) = parent(dest(t))$, then predicate **N1** in Proposition 2 can be dropped.

References

1. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. IEEE Transactions on Software Engineering 29(7), 623–633 (2003)
2. 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)
3. Berry, G.: The constructive semantics of pure Esterel draft version 3 (1999), `http://www-sop.inria.fr/esterel.org/`
4. Berry, G., Gonthier, G.: The Esterel synchronous programming language: Design, semantics, implementation. Science of Computer Programming 19(2), 87–152 (1992)
5. Brzozowski, J.A., Zhang, H.: Delay-insensitivity and semi-modularity. Formal Methods in System Design 16(2), 191–218 (2000)
6. Esmaeilsabzali, S., Day, N.A.: Prescriptive semantics for big-step modelling languages. In: Rosenblum, D.S., Taentzer, G. (eds.) FASE 2010. LNCS, vol. 6013, pp. 158–172. Springer, Heidelberg (2010)
7. Esmaeilsabzali, S., Day, N.A., Atlee, J.M., Niu, J.: Semantic criteria for choosing a language for big-step models. In: RE 2010, pp. 181–190. IEEE Computer Society Press, Los Alamitos (2009)
8. Esmaeilsabzali, S., Day, N.A., Atlee, J.M., Niu, J.: Deconstructing the semantics of big-step modelling languages. Requirements Engineering 15(2), 235–265 (2010)
9. Halbwachs, N.: Synchronous Programming of Reactive Systems. Kluwer, Dordrecht (1993)
10. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming 8(3), 231–274 (1987)
11. Heitmeyer, C., Jeffords, R., Labaw, B.: Automated consistency checking of requirements specifications. ACM Transactions on Software Engineering and Methodology 5(3), 231–261 (1996)
12. Heninger, K.L., Kallander, J., Parnas, D.L., Shore, J.E.: Software requirements for the A-7E aircraft. Tech. Rep. 3876, United States Naval Research Laboratory (1978)
13. 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)
14. Janicki, R., Koutny, M.: Structure of concurrency. Theoretical Computer Science 112(1), 5–52 (1993)
15. Karp, R.M., Miller, R.E.: Parallel program schemata. Journal of Computer and System Sciences 3(2), 147–195 (1969)
16. Landweber, L.H., Robertson, E.L.: Properties of conflict-free and persistent Petri nets. Journal of the ACM 25(3), 352–364 (1978)

17. Maraninchi, F., Rémond, Y.: Argos: an automaton-based synchronous language. Computer Languages 27(1/3), 61–92 (2001)
18. 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)
19. Silver, S.J., Brzozowski, J.A.: True concurrency in models of asynchronous circuit behavior. Formal Methods in System Design 22(3), 183–203 (2003)
20. Tardieu, O.: A deterministic logical semantics for pure Esterel. ACM Transactions on Programming Languages and Systems 29(2), 1–26 (2007)