Springer Copyright Notice

Published in: ***Requirements Engineering Journal,*** June 2010

"Deconstructing the Semantics of Big-Step Modelling Languages"

Cite as:

Shahram Esmaeilsabzali, Nancy A. Day, Joanne M. Atlee, and Jianwei Niu. 2010. Deconstructing the semantics of big-step modelling languages. Requir. Eng. 15, 2 (June 2010), 235-265.

BibTex:

@article{Esmaeilsabzali:2010:DSB:1821989.1821992,
 author = {Esmaeilsabzali, Shahram and Day, Nancy A. and Atlee, Joanne M. and Niu, Jianwei},
 title = {Deconstructing the Semantics of Big-step Modelling Languages},
 journal = {Requir. Eng.},
 issue_date = {June 2010},
 volume = {15},
 number = {2},
 month = jun,
 year = {2010}
}

DOI: http://dx.doi.org/10.1007/s00766-010-0102-z

Requirements Engineering
Special Issue of RE'09

Deconstructing the Semantics of Big-Step Modelling Languages

Shahram Esmaeilsabzali · Nancy A. Day · Joanne M. Atlee · Jianwei Niu

Received: 19 October 2009 / Accepted: 2 March 2010

Abstract With the popularity of model-driven methodologies, and the abundance of modelling languages, a major question for a requirements engineer is: which language is suitable for modelling a system under study? We address this question from a semantic point-of-view for *big-step modelling languages (BSMLs)*. BSMLs are a class of popular behavioural modelling languages in which a model can respond to an input by executing multiple transitions, possibly concurrently. We deconstruct the operational semantics of a large class of BSMLs into eight high-level, mostly orthogonal semantic aspects, and their common semantic options. We analyze the characteristics of each semantic option. We use feature diagrams to present the design space of BSML semantics that arises from our deconstruction, as well as to taxonomize the syntactic features of BSMLs that exhibit semantic variations. We enumerate the dependencies between syntactic and semantic features. We also discuss the effects of certain combinations of semantic options used together in a BSML semantics. Our goal is to empower a requirements engineer to compare and choose an appropriate BSML from the plethora of existing BSMLs, or to articulate the semantic features of a new desired BSML when such a BSML does not exist.

1 Introduction

With the popularity of model-driven methodologies and the abundance of modelling languages (and domain-specific languages), a major question for a requirements engineer is: which language is suitable for modelling a system under study (SUS)? We introduce the term

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

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

big-step modelling languages (BSMLs) to characterize a class of popular behavioural modelling languages in which a model can respond to an environmental input by executing a *big step*, which consists of a sequence of *small steps*, each of which may contain multiple, possibly concurrent, transitions. Numerous BSMLs have been introduced (e.g., statecharts [17] and its variants [49], synchronous languages [16], and UML StateMachines [38]), many of which have similar syntaxes but subtly different and complicated semantics.

The choice of a BSML for an SUS depends on many factors, including the domain of the SUS, the expertise of the requirements engineer in a class of notations, etc. In this paper, we present the semantic criteria that a requirements engineer should consider when choosing a BSML for modelling an SUS. One can write equivalent behaviours in different semantics by modifying a model (all BSMLs can be reduced to their meaning in primitive modelling languages such as Kripke structures, Büchi automata, labelled transition systems, etc.). However, it can be significantly more convenient (e.g., more succinct, more understandable) to model some behaviours in one semantics than in another. We envision a world where the choice of the features of a language, including its semantic features, are made on a model-by-model basis.

Our first contribution is a novel deconstruction of the operational semantics of a large class of BSMLs into eight high-level, mostly orthogonal, *semantic aspects*, and an enumeration of the common *semantic options* found in existing BSMLs for each of these aspects. While it is impossible to claim that our options are complete, they cover a wide range of existing BSMLs, as well as new semantics that arise through the enumeration of semantic options. Our second contribution is the identification of the characteristics of each semantic option to provide rationale for a requirements engineer to choose one option over another. Our third contribution is a set of carefully constructed examples that succinctly illustrate many of the differences between the semantic options.

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

Our deconstruction is more concise and systematic than previous comparative studies of different subsets of BSMLs (e.g., [7, 16, 26, 46, 47, 49]) because it recognizes a big step as a whole rather than only considering its constituent transitions operationally. In our previous work on template semantics [37], we created a formal framework for comparing the semantics of many BSMLs by instantiating a template of 22 parameters and choosing a set of composition operators that together define a small step. The eight semantic aspects we present here capture cross-cutting dependencies found in the template parameters, creating a deconstruction that defines a big step directly. This higher level of abstraction isolates the semantic differences between languages more clearly.

Compared to our previous work presenting this deconstruction [14], here, (i) we address several additional semantic concerns, namely, external events and variables, interface events and variables, and combo-step maximality; (ii) we present a more systematic treatment of the notion of a combo step; (iii) we provide a taxonomy for the syntactic features of BSMLs that exhibit semantic variations; (iv) we use two feature diagrams to present our semantic

deconstruction and the taxonomy of the syntax of BSMLs; (v) we present the dependencies between the features of the two feature diagrams; (vi) we accompany our presentation of the semantic aspects with more examples; and lastly, (vii) we extend our discussion of the dependencies between semantic options when used together in a BSML.

The remainder of the paper is organized as follows. In Section 2, we describe the common syntax and common basic semantics that we use throughout the paper. In Section 3, we present the deconstruction of the semantics of BSMLs into eight semantic aspects and their options, together with their syntactic requirements. We describe separately each semantic option and its characteristics, accompanied by modelling examples that exhibit the differences between the semantic options. In Section 4, we describe a few subtle side effects that result when certain semantic options are used together in a BSML. Section 5 compares our work with the related work, including our previous work on template semantics [37]. Finally, in Section 6, we conclude our paper and discuss future work.

2 Normal-Form Syntax and Basic Semantics

In this section, we present the terminology that we use throughout the paper. In Section 2.1, we present our normal-form syntax and the possible syntactic features that can be chosen when designing a BSML. In Section 2.2, we describe the common basic semantics, which can be refined by semantic options. In Section 2.3, we describe how the syntax of BSMLs can be represented in our normal-form syntax. We adopt a few syntactic definitions from Pnueli and Shalev's work [42].

2.1 Syntax

There is a plethora of BSMLs, including those with graphical syntax (e.g., statecharts variants [49], Argos [33]), those with textual syntax (e.g., reactive modules [3], Esterel [6]), and those with tabular/equational syntax (e.g., SCR [22,23]). As is usual when studying a class of related notations, we use a syntactic "normal form" that is sufficiently expressive to represent the syntax of other notations [25]. Our normal-form syntax is the *composed hierarchical transition system (CHTS)* syntax [37]. A *model* is a CHTS, and consists of: (i) a *composition tree* whose nodes are distinct *control states*, and (ii) a set of *transitions* between the control states.

Control States: A control state (e.g., $DialDigits$ in Figure 1) is a named artifact that a modeller uses to represent a noteworthy moment in the execution of a model. Such a moment is an abstraction that groups together the past behaviours (consisting of inputs received by the model and the model's past reactions to these inputs) that have a common set of future behaviours. By using a control state, a modeller can describe future behaviour in terms of the current control state and the current environmental inputs.

A control state has a *type*, which is either *Basic*, *Or*, or *And*. A leaf node of a composition tree is a *Basic* control state. An *Or* or an *And* control state is *hierarchical* and has *children*, each of which can be of any type. For example, in Figure 1, control state $Dialing$ is an *And* control state and has two *Or* control states, $Dialer$ and $Redialer$. We use the relations *parent*, *ancestor*, *child*, and *descendant* with their usual meanings. In Figure 1, control state $DialDigits$ is a child of $Dialer$ and a descendant of $Dialing$. Two control states *overlap* if they are the same or one is an ancestor of the other. In Figure 1, control states $Dialer$ and $Redialler$ are not overlapping. The *least common ancestor* of two control states is the

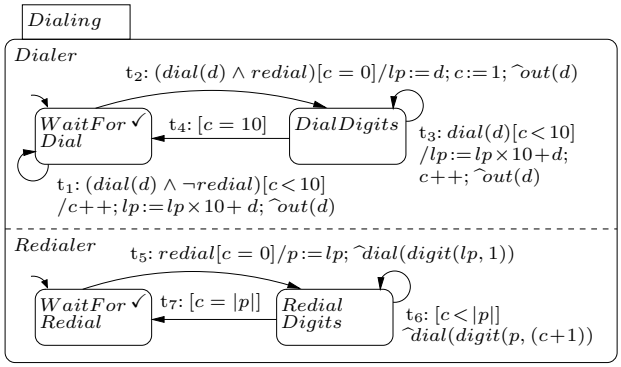

Fig. 1 Dialer/Redialer model.

lowest control state (closest to the leaves of the composition tree) in the hierarchy of the composition tree that is an ancestor of both. In Figure 1, the least common ancestor of $DialDigits$ and $RedialDigits$ is $Dialing$. Two control states are *orthogonal* if neither is an ancestor of the other and their least common ancestor is an *And* control state. In Figure 1, $DialDigits$ and $RedialDigits$ are orthogonal. An *Or* control state has a *default* control state, which is its child and is identified by an incoming arrow that has no source control state. In Figure 1, $WaitForDial$ is the default control state of $Dialer$. The *arena* of a transition t is the lowest *Or* control state in the hierarchy of the composition tree that is the ancestor of both the source and destination control states of the transition. In Figure 1, the arena of transition t_1 is the *Or* control state $Dialer$. A model may have no *And* control states. The root of the composition tree must be an *Or* control state so that the arena of every transition is guaranteed to exist, but otherwise may consist of only *Basic* control states.

Transitions: A transition (e.g., t_1 in Figure 1) has both a *source* and *destination* control state, and consists of four optional parts: (i) an *event trigger*, which is a conjunction of event literals, some of which may be negated (a negated event being prefixed by a "¬"); (ii) a *guard condition (GC)* (enclosed by "[ ]"), which is a boolean expression over the set of variables of the model; (iii) a sequence of *assignments* (prefixed by a "/"); and (iv) a set of *generated events* (prefixed by a "$\widehat{\ }$").

A generated event may have a parameter that can be modelled by associating a variable with it. An assignment consists of a left-hand side variable (LHS), and a right-hand side expression (RHS). The types of variables are not relevant. We assume all variable expressions and assignments of models are well-typed. Variables and events are global; local variables and scoped events can be modelled by a renaming that makes them globally unique.

Figure 2 is a feature diagram [28] that represents the combination of syntactic constructs of BSMLs that are of interest for our semantic aspects. Each feature in the diagram is labelled with the sections that describe its role and semantics. A leaf node of the feature diagram represents a primitive syntactic feature of BSMLs. For example, the **Negated Events** node is the syntactic feature that allows the negation of an internal event to be used in the event trigger of a transition. A non-leaf node represents a syntactic feature that has additional syntactic sub-features in its children nodes. For example, the **Event Triggers** node is the syntactic feature that has syntactic sub-features **Environmental Input Events**, **Interface Events**, and **Negated Events**. In the feature diagram in Figure 2, we use only "and" branches for sub-features of a feature: if a feature is chosen, then all of its child sub-features

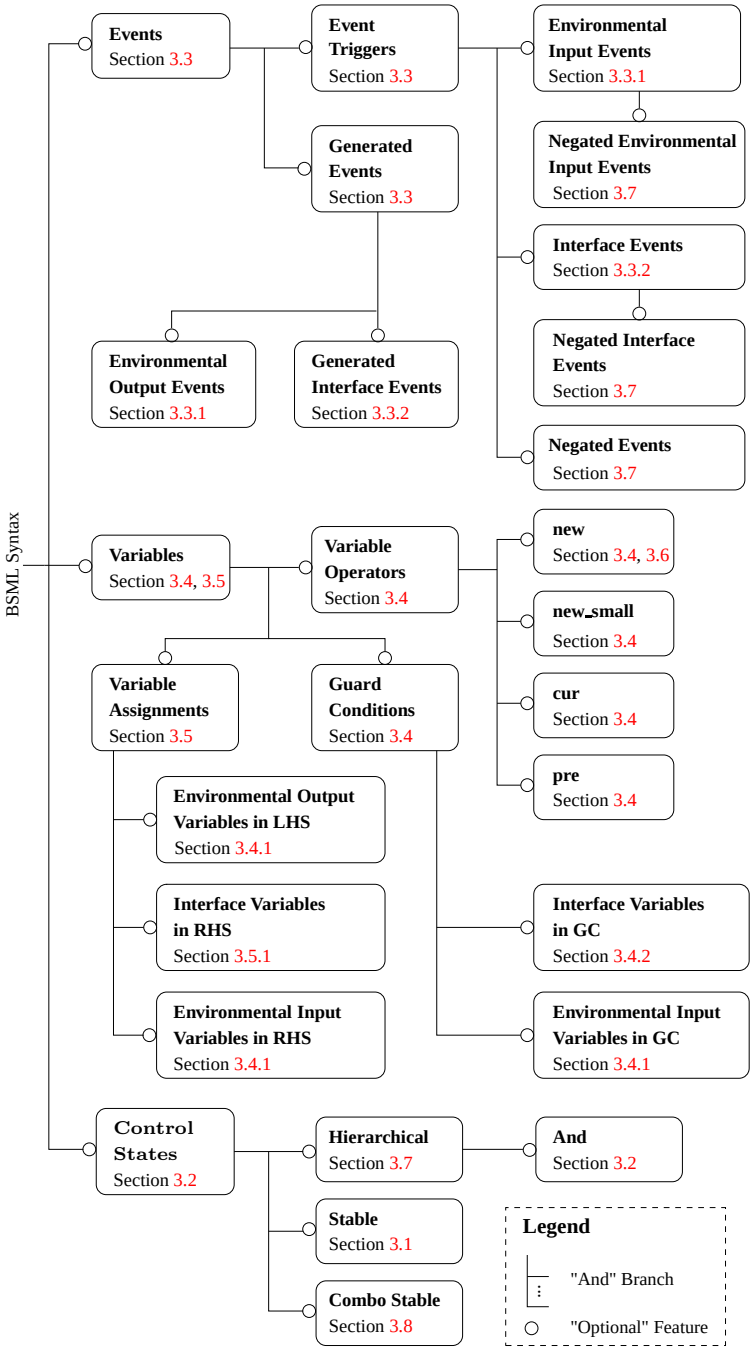

Fig. 2 Feature diagram for the syntactic variation points of interest to our semantic aspects.

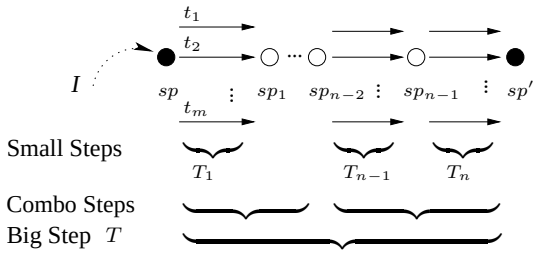

Fig. 3 Steps.

are also chosen, except for the sub-features that are connected to a small circle, which are "optional" sub-features. An optional feature, as opposed to a "mandatory" feature, need not be chosen if its parent feature is chosen. All of the features in the diagram in Figure 2 are optional features. For example, the **Event Triggers** syntactic feature has three sub-features, all of which are optional sub-features.

The syntax of a BSML must have a notion of transition to specify the behaviour of a system, but all other syntactic features in the feature diagram of Figure 2 are optional. In practice, the syntax of most useful BSMLs support at least events or variables.

2.2 Common Basic Semantics

Initially, a model resides in the default control state of each of its *Or* control states, no events are present, and its variables have their initial values. The operational semantics of a BSML describes how a model reacts to an *environmental input* via a *big step*. An environmental input is a set of events and variable assignments that are received from the environment. Figure 3 depicts a big step T, which is a reaction of a model to environmental input I. A big step is an alternating sequence of *small steps* and *snapshots*, where a small step is the execution of a set of transitions (t_i's), and a snapshot is a tuple that stores information.[1] The T_i's ($1 \leq i \leq n$) are small steps of T, and sp, sp', and sp_i's ($1 \leq i < n$) are its snapshots. Throughout the paper, we often represent a big step as the sequence of its small steps; e.g., T is represented as $\langle T_1, T_2, \cdots, T_n \rangle$. Some BSMLs, such as RSML [30] and Statemate [19], introduce an intermediate grouping of a sequence of small steps, which we call a *combo step*. The small steps of a combo step hide some of their effects, e.g., the effect of their assignments, from one another. Sections 3.3, 3.4, and 3.8, describe when combo steps are useful.

Snapshots: A snapshot is a tuple that consists of at least: (i) a *configuration*, which is a set of control states; (ii) a *variable evaluation*, which is a set of ⟨variable name, value⟩ pairs; and (iii) a set of *events*. Each of a big step, a small step, or a combo step has a *source* and *destination* snapshot (e.g., sp and sp' are the source and destination snapshots of T).

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

[1] Big steps and small steps are often called macro steps and micro steps, respectively. We adopt new terms to avoid association with the fixed semantics of the languages that use those terms. The big-step/small-step terminology has been used in the study of the operational semantics of programming languages in a similar spirit as we use them here [40].

control state is in the source configuration of the small step. Different semantic options use different snapshots of a big step to define enabledness.

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

In a few, non-common cases, transition execution can be more involved; e.g., when the least common ancestor of the source and destination control states of a transition is an *And* control state. A discussion of these cases is included in Section 3.2.

The semantics of event generation and variable assignment differ between BSMLs. The execution of a small step is *atomic*: the variable assignments and event generation of one transition cannot be seen by another transition (except for the "PRESENT IN SAME" event lifeline option [cf., Section 3.3]). Because of atomicity, a sequence of assignments on a transition can be converted to a set of assignments [29, 31].

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

- *Fast computation*: This assumption, which is usually referred to as the "synchrony hypothesis" or the "zero-time assumption" [6, 16], postulates that the system is fast enough, and thus never misses an input. The domain of systems that are modelled using this paradigm is called "reactive systems" [6, 16, 20]. A reactive system is usually a mission-critical system that is meant to react to environmental inputs in a timely manner, at the rate produced by the environment; e.g., the controller system of a nuclear reactor. No environmental inputs are missed.
- *Helpful environment*:[2] This assumption postulates that the environment is helpful by issuing an input only when the system is ready [16]. The domain of systems that are modelled using this paradigm is called "interactive systems" [16]. An interactive system is different from a reactive system in that the rate of environmental inputs is dictated by the system, rather than by the environment. An example of an interactive system is an automated banking machine, which interacts with its environment (i.e., a customer) at its own rate when it is ready, rather than at the rate the customer would like to provide inputs for it. An environmental input might be missed by the system when the system is busy processing a previous environmental input.
- *Asynchronous communication*: This assumption postulates that the system has a buffering mechanism to store the environmental inputs, and thus never misses an environmental input.

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

[2] We have adopted the term "helpful environment" from a similar notion in *Interface Automata* [11].

2.3 Representing BSMLs in the Normal-Form Syntax

It is straightforward to represent the syntax of many BSMLs in our normal-form syntax. In our previous work [37], we described the mapping of the syntax of many BSMLs to the CHTS syntax. In this section, we describe a few, less obvious, syntactic representations in our normal-form syntax.

Control states: A BSML may not include the notion of control states. If a model's reaction to an environmental input is always independent of its past behaviours, then the notion of control state is not useful for the model. One way to represent the syntax of a BSML that does not have control states in our normal-form syntax is to create a single control state that serves as the source and destination control states of all transitions. The notion of the hierarchy of control states might still be useful in such a BSML for specifying priority between transitions (cf., Section 3.7 for priority semantics).

A BSML with a textual syntax without explicit control states, such as Esterel [6], realizes a line of a program as a control state. For example, in Esterel [6], an `exit` statement within a parallel command of a model moves the flow of control from within the parallel command to the next command outside the scope of the parallel command. The parallel command and the command after it can be conceptually considered as control states with the parallel command being an *And* control state. The `exit` statement can be considered as a transition that connects the two control states.

SCR [22, 23] is a BSML that uses a tabular format. The notions of "modes" and "transitions between modes" in its syntax can be represented by the notions of control states and transitions between control states, respectively.

Transitions: In our syntax for transitions, we do not include event triggers with disjunctions, because an event trigger that has disjuncts can be split into multiple transitions, each with only one of the disjuncts of the original event trigger and exactly the same other elements as the original; such a transformation yields a model that is semantically the same as the original model [42].

Transitions with multiple-source and/or multiple-destination can be split into multiple single-source, single-destination transitions. However, we would need to extend the semantic options for the concurrency and consistency aspect (in Section 3.2) and the hierarchical-priority aspect (in Section 3.7) to accommodate the execution semantics of a group of transitions that represent a single multiple-source and/or multiple-destination transition. We defer the treatment of the semantics of multiple-source and/or multiple-destination transitions to our future work.

3 Semantic Aspects

We deconstruct the operation of a big step into the stages described in Figure 4. This systematic deconstruction is based on: (i) conceptual sequentiality in the process of creating a small step (partly based on the syntactic elements of the model), (ii) orthogonal concerns in the operation of a big step, and (iii) semantic variation points in existing BSMLs. Each stage of the diagram is associated with one of our *semantic aspects* and is labelled with the corresponding section of the paper that describes it. A semantic aspect may be decomposed into some semantic sub-aspects. A semantic aspect or sub-aspect may have a number of *semantic options*, each of which is a semantic choice for carrying out a stage.

There are eight semantic aspects, as shown by the feature diagram in Figure 5. Semantic aspects are represented by shaded boxes and the Sans Serif font, and semantic options

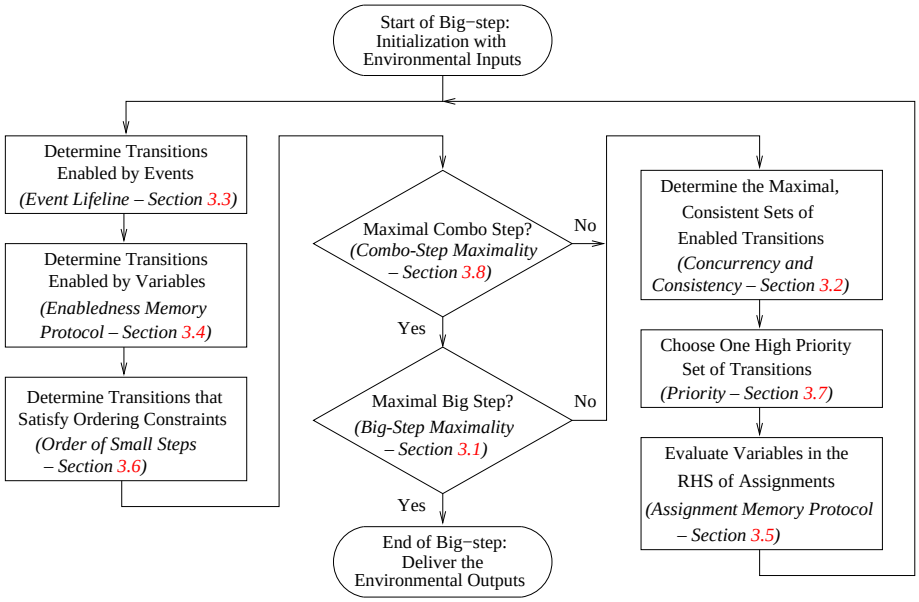

Fig. 4 Operation of a big step.

are represented by clear boxes and the SMALL CAP font. An arced branch in the diagram represents an "exclusive or": if a feature is chosen, then exactly one of its sub-features is chosen. For example, if the Big-Step Maximality semantic aspect is chosen, then exactly one of its options, SYNTACTIC, TAKE ONE, or TAKE MANY should be chosen. For the sake of brevity, we group a set of recurring semantic options for event-related semantic sub-aspects as "Event Options", and reference them via this label in the diagram.

Next, we briefly describe the role of each semantic aspect. The Big-Step Maximality semantic aspect specifies when a big step ends, at which point a new big step starts by sensing new environmental inputs. The Combo-Step Maximality semantic aspect specifies when a combo step ends. The source snapshot at the beginning of a combo step reflects the effects of the execution of the small steps of the previous combo step. The Event Lifeline semantic aspect specifies how far within a big step a generated event can be sensed to trigger a transition. We consider separate sub-aspects for the semantics of *internal events*, which are not meant to be observed by the environment of a model, *external events*, which are used to communicate with the environment, and *interface events*, which are used by a model to specify communications among its different *components*. The Enabledness Memory Protocol semantic aspect specifies the snapshot from which the values of variables are read to enable the guard condition of a transition. The Order of Small Steps semantic aspect describes options for the order of transitions that execute within a big step. From the set of transitions enabled by events, variables, and ordering constraints, the Concurrency and Consistency semantic aspect determines the set of potential small steps: first, it specifies whether more than one transition can be taken in a small step; and second, if more than one transition can be taken, it specifies the consistency criteria for including multiple transitions in a small step. The Priority semantic aspect chooses a small step from the set of potential small steps. The Assignment Memory Protocol semantic aspect specifies the snapshot

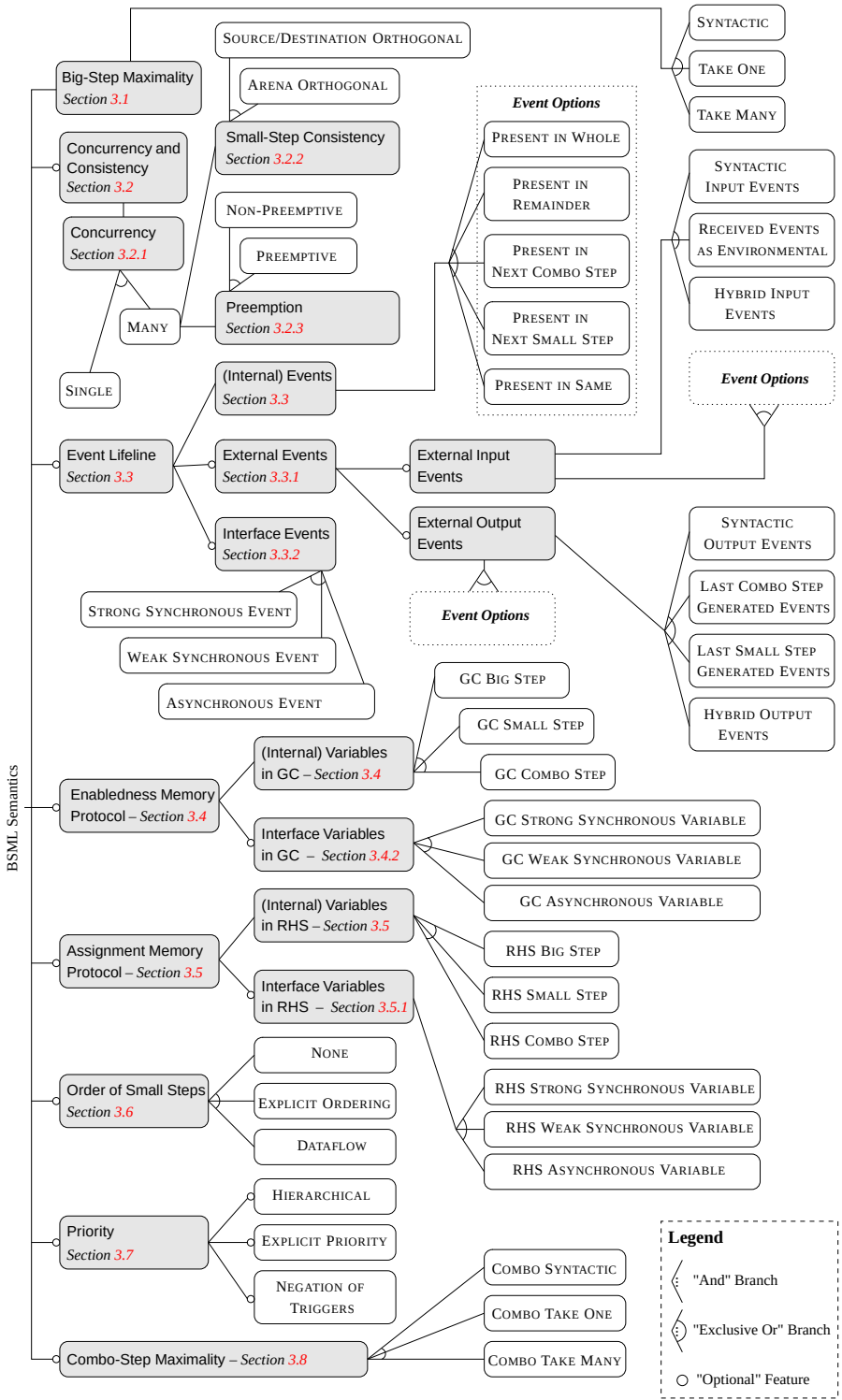

Fig. 5 A feature diagram for BSML semantics.

1. **Events** ⇔ Event Lifeline
2. (**Interface Events** ⇔ **Generated Interface Events**) ∧
   ((**Interface Events** ∧ **Generated Interface Events**) ⇔ Interface Events)
3. **Environmental Input Events** ⇔ SYNTACTIC INPUT EVENTS
4. **Environmental Output Events** ⇔ SYNTACTIC OUTPUT EVENTS
5. (**Negated Events** ∨ **Negated Interface Events** ∨ **Negated External Events**) ⇔
   NEGATION OF TRIGGERS
6. **Variable Conditions** ⇔ Enabledness Memory Protocol
7. **Variable Assignments** ⇔ Assignment Memory Protocol
8. **Interface Variables in GC** ⇔ Interface Variables in GC
9. **new** ⇔ DATAFLOW
10. **new** ⇒ (GC BIG STEP ∨ GC SMALL STEP ∨ RHS BIG STEP ∨ RHS SMALL STEP)
11. **new_small** ⇒ (GC SMALL STEP ∨ RHS SMALL STEP)
12. **cur** ⇒ (GC BIG STEP ∨ RHS BIG STEP)
13. **pre** ⇒ (GC SMALL STEP ∨ RHS SMALL STEP)
14. **Interface Variables in RHS** ⇔ Interface Variables in RHS
15. HIERARCHICAL ⇒ **Hierarchical**
16. **And** ⇔ Concurrency and Consistency
17. **Stable** ⇔ SYNTACTIC
18. **Combo Stable** ⇔ COMBO SYNTACTIC
19. Combo-Step Maximality ⇔
    (PRESENT IN NEXT COMBO STEP ∨ GC COMBO STEP ∨ RHS COMBO STEP)
20. COMBO TAKE MANY ⇒ (SYNTACTIC ∨ TAKE MANY)
21. PRESENT IN SAME ⇒ MANY

Fig. 6 Dependencies between syntactic and semantic features. (**Bold:** syntactic features, Sans Sarif: semantic aspects, and SMALL CAP: semantic options.)

from which the value of a variable in the right-hand side of an assignment is read. Similar to events, we distinguish between the semantics of *internal variables*, *external variables*, and *interface variables*.

A BSML semantics must subscribe to a Big-step Maximality semantics, as shown by the corresponding mandatory feature in the diagram in Figure 5. The other aspects are optional and depend on the syntactic features included in the BSML. A BSML semantics might have more than one priority semantic option, which together constitute its priority semantics (cf., Section 3.7).

Dependencies between Features: A semantic aspect or a semantic option might be relevant for the semantics of a BSML only if a certain syntactic construct is allowed in the BSML. Figure 6 enumerates the dependencies between the syntactic and semantic features. To describe these dependencies, we use the names of syntactic features in Figure 2 and the names of semantic aspects and options in Figure 5 as propositions, which indicate the choice of the feature in its corresponding feature diagram. We use standard logical operators to describe the dependencies. The "$p \Rightarrow q$" operator is logical implication: if p is true then q must be true. The "$p \Leftrightarrow q$" operator is logical equivalence: either p and q are both true, or both are false. The "$p \vee q$" operator is logical or: either p, q, or both are true. The "$p \wedge q$" operator is logical and: both p and q are true.

The last three dependencies in Figure 6 are between semantic features, as opposed to between syntactic and semantic features. These dependencies will be explained in the sections on the semantic aspects.

In the feature diagram in Figure 5, a semantic (sub-)aspect, or its parent, is labelled with the section in which it is described. We have chosen to order these sections in a manner that minimizes the required forward referencing to other semantics aspects (although some for-

ward referencing cannot be avoided). In each section, we summarize the semantic options for each aspect in a table that includes: a brief description of each semantic option, a list of its characteristics, and a list of representative BSMLs for each option. We identify each characteristic as a relative advantage or disadvantage, signified by a "+" or "-", respectively, based on our understanding of the conventional wisdom on this characteristic. Such wisdom may not always be appropriate for a model depending on the domain of the SUS, the preference of the modeller, etc. These options cover the variations found in most existing BSMLs. As in Figure 5, we use the SMALL CAP font to express the names of semantic options. Throughout the section, we present many examples that are meant to demonstrate the differences between semantic options (but not to endorse one over another). The model snippets in our examples are not complete. Finally in Section 3.9, we present a table summarizing the semantic options chosen by a number of BSMLs.

3.1 Big-Step Maximality

The big-step maximality semantics of a BSML specifies when the sequence of small steps of a big step concludes. Table 1 lists the three possible semantic options. In the SYNTACTIC option, a BSML allows a modeller to designate syntactically a basic control state of a model as a *stable* control state. During a big step, once a transition t that enters a stable control state is executed, no other transition whose arena overlaps with the arena of t can be executed. In the TAKE ONE option, once a transition t is executed during a big step, no other transition whose arena overlaps with the arena of t can be executed. As such, each *Or* control state can contribute a maximum of one transition to a big step. Lastly, the TAKE MANY option allows a sequence of small steps to continue until there are no more enabled transitions to be executed.

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

Sequential transitions vs. non-terminating big steps: In the SYNTACTIC and TAKE MANY options, it is possible to specify a computation as a big step that consists of multiple sequential transitions within an *Or* control state. But, in these two semantics, it is also possible for a big step to never terminate because the execution of the big step never reaches a snapshot in which there are no more transitions to be executed. In the SYNTACTIC maximality semantics, additionally, a big step may never terminate because the model never reaches a syntactically designated stable control state. Some BSMLs with the SYNTACTIC semantics require the non-stable control states of a model to have "else" transitions so that a big step can always reach a stable configuration (e.g., [18, 38]). In the TAKE ONE semantics, a sequence of transitions in an *Or* control state cannot be included in a big step, but a big step always terminates.

Stable control states can be used to model the semantics of the `pause` command in Esterel [6, 47]. During a big step, once all non-overlapping control states of the model's configuration have executed the `pause` command, the big step ends. As such, if the `pause` command is executed outside of a parallel command, then the big step terminates. But if the `pause` command is executed inside a branch of a parallel command, then the big step terminates when every branch of the parallel command has executed the `pause` command. Stable control states can also be used to model the semantics of "compound transitions" in Rhapsody [18] and UML StateMachines [38]: the "pseudo states" of a model are modelled

Table 1 Big-step maximality semantic options.

Options	Definition	Characteristics	Examples
SYNTACTIC	No two transitions with overlapping arenas that enter designated "stable" control states can be taken in the same big step.	(+) Syntactical scope for big steps (+) Sequential *Or* transitions (-) Non-terminating big steps	Esterel [6] (`pause` command), Rhapsody [18] and UML StateMachines [38] "run to completion"
TAKE ONE	No two transitions with overlapping arenas can be taken in the same big step.	(+) Terminating big steps (-) Unclear, non-syntactical scope for big steps	statecharts [17,21,42], reactive modules [3], and Argos [33]
TAKE MANY	Small steps continue until there are no more enabled transitions.	(+) Sequential *Or* transitions (-) Unclear, non-syntactical scope for big steps (-) Non-terminating big steps	Statemate [19] and RSML [30]

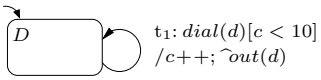

Fig. 7 Dialer system.

as non-stable control states, and "states" are modelled as stable control states. Some of the BSMLs that support the TAKE ONE semantics, such as reactive modules [3] and Argos [33], are influenced by the principles of synchronous hardware, which assumes that, during a big step, a non-concurrent part of a model can take only one transition (equivalently, each hardware component reacts once during a clock tick). The TAKE MANY semantic option is usually used by the BSMLs that support the notion of combo step (e.g., Statemate [19] and RSML [30]). The Statemate tool suite can be configured to use either the TAKE ONE semantic option, whose big steps are referred to as "steps", or the TAKE MANY semantic option together with combo steps, whose big steps are referred to as "super steps" [19].

Example 1 *The model in Figure 7 collects a dialed digit of a phone device (environmental input event* $\mathrm{dial(d)}$*) and transmits the dialed digit* d *to the IP network via generated event* $\mathrm{out(d)}$*. Variable* c *allows a maximum of 10 digits to be collected, at which point the central IP system would connect the caller to the dialed callee (we do not model the connection functionality of the system). The "*$++$*" operator denotes increment by one.*

Consider a semantics that if event $\mathrm{dial(d)}$ *is received from the environment, it persists until the end of the big step. Also, consider the snapshot where* c *is zero and* $\mathrm{dial(d)}$ *is received from the environment. If the* TAKE MANY *big-step semantics is chosen, then transition* $\mathrm{t_1}$ *is executed 10 times in succession, sending the same digit 10 times. If the* TAKE

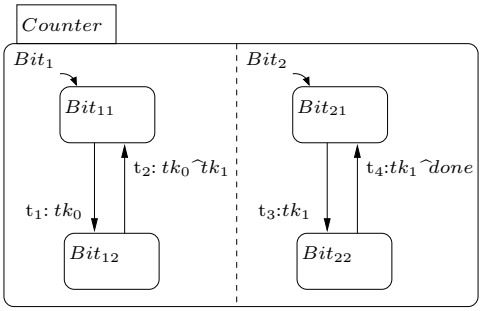

Fig. 8 A two-bit counter.

ONE *big-step maximality semantics is chosen, or the* SYNTACTIC *semantics is chosen and control state* D *is designated as stable, then the model behaves correctly.*

Example 2 *The model in Figure 8 is for a two-bit counter.*[3] *Control states* Bit_1 *and* Bit_2 *model the least and most significant bits of the counter, respectively. Each time the environmental input event* tk_0*, which represents a clock tick, is received, the counter increments by one. Consider a semantics where environmental inputs persist throughout the big step. After an even number of ticks,* Bit_1 *sends event* tk_1*, thereby instructs* Bit_2 *to toggle its status. After counting four clock ticks, the* Counter *generates the* done *event. Consider the snapshot where the model resides in control states* Bit_{11} *and* Bit_{21} *and a semantics where each small step comprises the execution of exactly one transition. If the* TAKE ONE *big-step semantics is chosen, then the model behaves correctly. The first* tk_0 *input event produces the big step* $\langle\{\mathrm{t}_1\}\rangle$*, the second* tk_0 *input event produces the big step* $\langle\{\mathrm{t}_2\}, \{\mathrm{t}_3\}\rangle$*, the third* tk_0 *input event again produces the big step* $\langle\{\mathrm{t}_1\}\rangle$*, and lastly, the fourth* tk_0 *input event produces the big step* $\langle\{\mathrm{t}_2\}, \{\mathrm{t}_4\}\rangle$*, which generates event* done*. If the* TAKE MANY *big-step semantics is chosen, then the model behaves incorrectly by creating non-terminating big steps; for example, upon receiving the first* tk_0 *input event, the model can engage in the following non-terminating big step:* $\langle\{\mathrm{t}_1\}, \{\mathrm{t}_2\}, \{\mathrm{t}_1\}, \{\mathrm{t}_2\}, \cdots\rangle$.

3.2 Concurrency and Consistency

BSMLs vary in how the enabled transitions of a model execute together in a small step. Table 2 lists the three concurrency and consistency semantic sub-aspects that specify: (i) concurrency: whether more than one transition can be taken in a small step, and if so, (ii) small-step consistency: which transitions can be taken together, considering the composition tree of a model, and (iii) preemption: whether the execution of one transition in a small step can *preempt* the execution of another transition or not.

3.2.1 Concurrency

[3] This example is adopted from [33], where a more elaborate version of it is used as the running example of the paper.

Table 2 Concurrency and consistency semantic options.

Options	Definition	Characteristics	Examples
Concurrency			
SINGLE	A small step consists of the execution of exactly one transition.	(+) Simplicity (-) Non-determinism	statecharts [17, 21, 42], Stateflow [9], and reactive modules [3]
MANY	A small step may consist of the execution of more than one transition.	(+) Low chance for non-determinism (-) Race conditions	Argos [33] and Esterel [6]
Small-Step Consistency			
ARENA ORTHOGONAL	The arenas of two distinct transitions of a small step are orthogonal.	(+) Simplicity (-) High chance for non-determinism	Argos [33] and Esterel [6]
SOURCE/DESTINATION ORTHOGONAL	The source control states and destination control states of two distinct transitions of a small step are pairwise orthogonal.	(+) Low chance for non-determinism (-) Complex	N/A
Preemption			
NON-PREEMPTIVE	Two transitions that one is an "interrupt for" another can be taken in a small step.	(+) Support for "last wishes" (-) Counterintuitive flow of control	Argos [33], and semantics of `exit` and `trap` statements in Esterel [6]
PREEMPTIVE	Two transitions that one is an "interrupt for" another cannot be taken in a small step.	(+) Simple flow of control (-) No support for "last wishes"	N/A

There is a dichotomy in hardware and software about how to model the execution of a system: *single-transition* vs. *many-transition* [35, 43, 45, 48]. Similarly, in BSMLs, there are two options: (i) a small step can execute only one transition in a small step (the SINGLE option), and (ii) all enabled transitions that can be taken together are taken in a small step (the MANY option). The SINGLE option is simple because it does not have to deal with the complexities of executing multiple transitions (e.g., race conditions), but it can cause undesired non-determinism because two enabled transitions can execute in different orders.

Race conditions: A model has a *race condition* when more than one transition in a small step assign values to a variable. Typically, one of the assignments is chosen non-deterministically [37], but there are other options [13].

Example 3 *Figure 9 shows the model for describing the behaviour of a simple traffic light system at an intersection.*[4] *The model consists of And control state* TrafficLight*, which itself consists of two Or control states: the* NS *control state controls the traffic in the north-south direction and the* EW *control state controls the traffic in the east-west direc-*

[4] This example is adopted from [27].

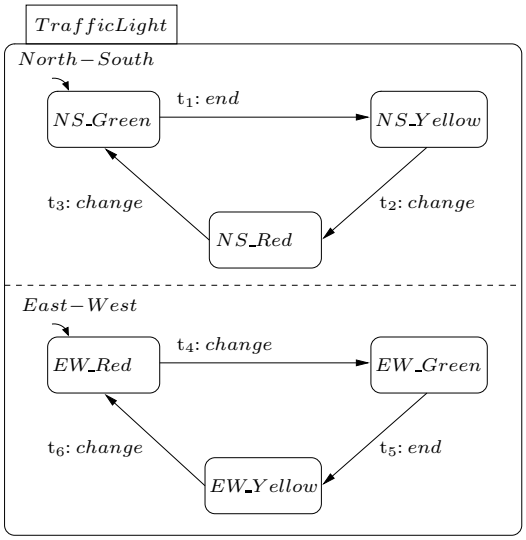

Fig. 9 Traffic light system.

tion. We assume that the environment provides the sequence of environmental input events: $\mathrm{end}, \mathrm{change}, \mathrm{end}, \mathrm{change}, \cdots$*, in a timely manner according to the schedule of the traffic light. Environmental input event* end *designates the end of green light for a direction by changing its green lights to yellow. Environmental input event* change *changes the direction of traffic by switching the red lights to green lights, and the yellow lights to red lights. The system is initialized so that the lights for north-south direction are green, and the lights for east-west direction are red. Consider the snapshot where the model resides in control states* EW_Red *and* NS_Yellow*, and environmental input event* change *is received. If we choose the* TAKE ONE *big-step maximality semantics and the* SINGLE *concurrency semantics, then the model can choose to execute the big step consisting of the sequence of transitions* $\langle\{\mathrm{t}_2\}, \{\mathrm{t}_4\}\rangle$*, or the sequence of transitions* $\langle\{\mathrm{t}_4\}, \{\mathrm{t}_2\}\rangle$*, non-deterministically. However, executing the latter sequence of transitions permits the model to arrive at snapshot* EW_Green *and* NS_Yellow*, which is not a desirable behaviour. If the* MANY *concurrency semantics is chosen, then model executes big step* $\langle\{\mathrm{t}_2, \mathrm{t}_4\}\rangle$*, arriving at control states* EW_Green *and* NS_Red.

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

3.2.2 Small-Step Consistency

In the SOURCE/DESTINATION ORTHOGONAL semantic option, transitions whose source control states and destination control states are pairwise orthogonal can be taken together in a small step. The ARENA ORTHOGONAL option is more restrictive in that two transitions can be included in the same small step only if their arenas are orthogonal (where the arena

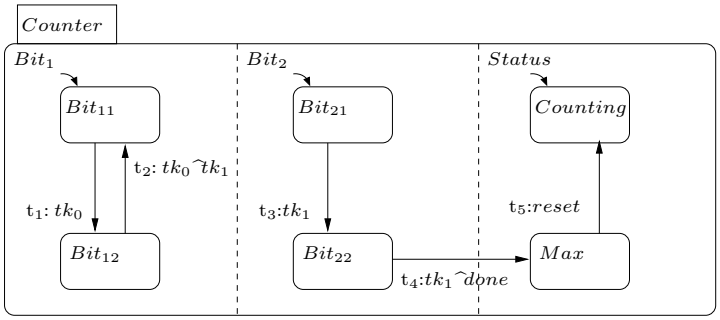

Fig. 10 The revised two-bit counter.

of a transition is the lowest *Or* control state in the hierarchy of the composition tree that is the ancestor of the source and destination control states of the transition). In comparison, the ARENA ORTHOGONAL option is simpler than the SOURCE/DESTINATION ORTHOGONAL option, but it can introduce undesired non-determinism by not taking all of the enabled transitions that the SOURCE/DESTINATION ORTHOGONAL option takes. The ARENA ORTHOGONAL semantic option and the TAKE ONE big-step maximality semantics are conceptually analogous: the former semantic option disallows two transitions whose arenas are the same or ancestrally related to be included in a small step, while the latter disallows the two transitions to be included in a big step.

Example 4 *The model in Figure 10 is similar to the model in Example 2, but has an extra Or control state that specifies whether the counter is in the process of counting, or it has already counted four ticks and should be reset. Consider the snapshot where the model resides in control states,* Bit_{12}, Bit_{22}, *and* $\mathrm{Counting}$, *and the fourth* tk_0 *event is received. We choose the* MANY *concurrency semantics. Also, we choose the* PRESENT IN SAME *event communication mechanism (explained in Section 3.3), in which a generated event can enable a transition in the same small step. If we choose the* ARENA ORTHOGONAL *semantics, then only* $\{\mathrm{t}_2\}$ *can be taken, but not* $\{\mathrm{t}_4\}$, *because the arena of* t_4 *is a parent of the arena of* t_2. *If we choose the* SOURCE/DESTINATION ORTHOGONAL *semantics, then* $\langle\{\mathrm{t}_2, \mathrm{t}_4\}\rangle$ *can be taken, and the model behaves correctly. (The execution of* t_4 *involves exiting the Or control state* Bit_2 *and reentering its default control state* Bit_{21}. *The destination configuration of the small step is* Bit_{11}, Bit_{21}, *and* Max.*)*

3.2.3 Preemption

The notion of *preemption* [5] is relevant for a pair of transitions when one is an *interrupt for* the other. A transition t is an interrupt for transition t' when the sources of the transitions are orthogonal and one of the following conditions holds: (i) the destination of t' is orthogonal with the source of t, and the destination of t is not orthogonal with the sources of either transitions (Figure 11(a)); or (ii) the destination of neither transition is orthogonal with the sources of the two transitions, but the destination of t is a descendant of the destination of t' (Figure 11(b)). The NON-PREEMPTIVE option allows such a t and t' to be executed together in the same small step, whereas the PREEMPTIVE option does not. In the NON-PREEMPTIVE option, the effect of executing such a small step $\{t, t'\}$ includes the variable assignments and event generations of both transitions, but the destination configuration of

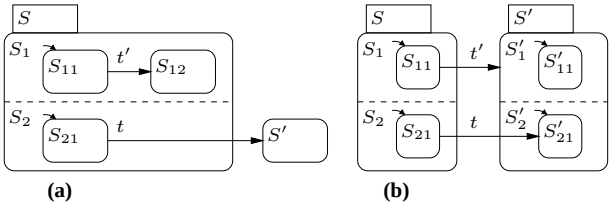

Fig. 11 Interrupting transitions.

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

The NON-PREEMPTIVE semantics can be used to model the "weak preemption" semantics of `exit` and `trap` statements in Esterel [6, 16]. The concurrent execution of an `exit` command with a non-`exit` command complies with the condition (i) above of the interrupt for relation. The concurrent execution of two `exit` commands complies with the condition (ii) above of the interrupt for relation. In Argos [33], a different notion of hierarchical control state than ours is used. A transition with a source of a non-*Basic* control state S is an interrupt for a transition whose arena is S or a descendent of S. We can translate this notion of control state and interrupt to our framework by turning S into an *And* control state with two children: one representing S without the interrupt transition, and another having only one transition that models the interrupt transition. In Esterel [6, 16], in addition to the NON-PREEMPTIVE semantics, there is a syntax to specify PREEMPTIVE behaviour through the "strong preemption" semantics of `watching` statements. In a "`do <statements> watching(e)`" statement, the execution of "`<statements>`" is immediately aborted when event e occurs, without satisfying the "last wish" of "`<statements>`". Such a `watching` statement can be translated into our normal-form syntax by creating a transition whose source is an *And* or *Or* control state that represents the "`<statements>`", and it is triggered with event e. The additional transition in the aforementioned translation is not an interrupt for any transition.

Example 5 *The model in Figure 12 is an extension of the model in Figure 1. A control state that is labelled with a "✓" represents a stable control state. This model is a model of a dialer system that receives the dialed digits of a phone, through event* $\mathrm{dial(d)}$*, and transmits these digits via output events* $\mathrm{out(d)}$*, to establish the connection with a destination phone number. Compared to the model in Figure 1, the model in Figure 12 additionally controls the total number of calls that can be established at each point of time. If the maximum number of concurrent calls is reached, which is determined by the boolean environmental input variable* limit*, the dialing process is aborted via transition* t*. Consider the snapshot where environmental input variable* limit *is true, the model resides in control states* $\mathrm{WaitforDial}$ *and* $\mathrm{WaitforRedial}$*, the value of variable* c*, which is the number of dialed digits so far, is nine, and the environmental input* $\mathrm{dial(d)}$ *is received, i.e., the caller dials the last digit of a phone number. We choose the* SYNTACTIC *concurrency and the* MANY *concurrency semantics. If we choose the* PREEMPTIVE *option, the system may abort the dialing process by executing* $\langle\{\mathrm{t}\}\rangle$*, and not* $\langle\{\mathrm{t}_1\}\rangle$*. But if we choose the* NON-PREEMPTIVE *option, then the call would go through by executing* $\langle\{\mathrm{t}_1, \mathrm{t}\}\rangle$*. (The execution of small step* $\langle\{\mathrm{t}_1, \mathrm{t}\}\rangle$ *involves*

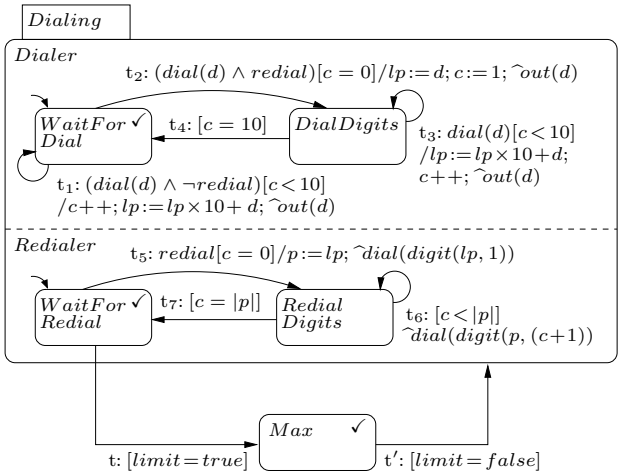

Fig. 12 Interrupting transitions.

exiting the And control state Dialing *and reentering the default control state of its children* Dialer *and* Redialer. *The destination configuration of the small step is* Max.*)*

3.3 Event Lifeline

A generated event of a transition is broadcast to all parts of a model. An event's *status*, which is either *present* or *absent*, can be sensed by the event trigger of a transition. The *event lifeline* semantics of a BSML specifies the snapshots of a big step in which a generated event can be sensed as present. Table 3 shows the five event lifeline semantics: (i) in the PRESENT IN WHOLE option, a generated event is present throughout its big step, from the beginning of its big step; (ii) in the PRESENT IN REMAINDER option, a generated event is present in the snapshot after it is generated and persists until the end of its big step; (iii) in the PRESENT IN NEXT COMBO STEP option, a generated event is present only during the next combo step; (iv) in the PRESENT IN NEXT SMALL STEP option, a generated event is present only in the next snapshot; and (v) in the PRESENT IN SAME option, a generated event is present only during the small step in which it is generated (instantaneous communication). Figure 13 depicts the event lifeline of the event e generated in small step T_2, according to the different event lifeline semantics. Each name of an event lifeline semantics is followed by a line that depicts the extent of the big step in which e is present, according to that semantics.

The PRESENT IN WHOLE semantic option supports the "perfect synchrony hypothesis" [4,33]. If we consider a big step as the reaction of a synchronous circuit during a "tick" of the clock, the semantics of the perfect synchrony hypothesis is similar to the signal rules of synchronous hardware. In synchronous hardware, a signal is either present or absent during a tick of a clock, but not both. The PRESENT IN SAME semantic option is different from the other semantic options in that the generated events of a small step cannot affect the enabledness of another small step, making the small steps of a big step independent of one another. The PRESENT IN SAME semantic option is inspired by the semantics of synchronization and rendezvous in process algebras [15,24,36].

Table 3 Event lifeline semantics.

Options	Definition	Characteristics	Examples
PRESENT IN WHOLE	A generated event in a big step is assumed to be present throughout the big step.	(+) Modularity (+) Global consistency (-) Non-causality (-) Counterintuitive behaviour	Argos [33] and Esterel [6]
PRESENT IN REMAINDER	A generated event in a big step is sensed as present after it is generated.	(+) Causality (-) Unorderedness (-) Global inconsistency	statecharts [21, 42]
PRESENT IN NEXT COMBO STEP	A generated event can be sensed as present only in the next combo step after it is generated.	(+) Causality (+) Partial orderedness (-) Multiple-instance events	Statemate [19] and RSML [30]
PRESENT IN NEXT SMALL STEP	A generated event can be sensed as present only in the next small step after it is generated.	(+) Causality (+) Orderedness (-) Multiple-instance events	statecharts[10]
PRESENT IN SAME	A generated event can be sensed as present only in the same small step it is generated in.	(+) Instantaneous communication (-) Non-causality (-) Multiple-instance events	Used in [37]

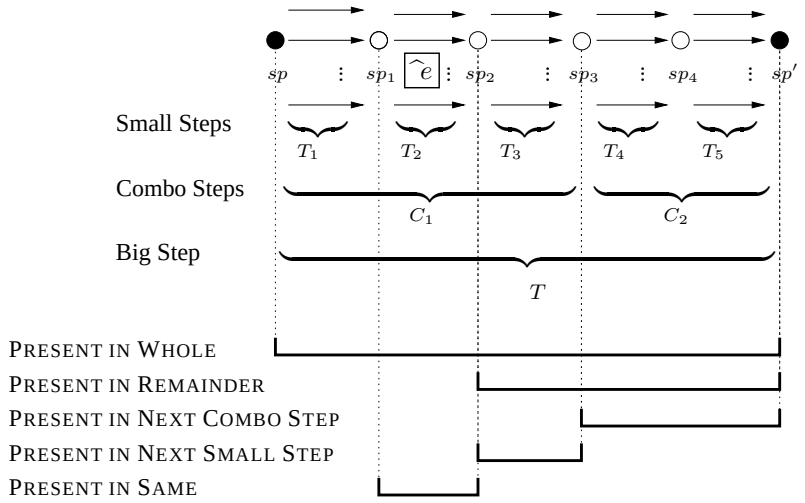

Fig. 13 The event lifeline of the generated event e according to different event lifeline semantic options.

Implicit events: Some BSMLs use *implicit events* in their syntax, which represent events that are generated in response to a certain property of the computation of a model. For example, the implicit event `entered(`s`)` [41] is generated when control state s is entered, and implicit event `@T(`$cond$`)` [22,23] is generated when the value of boolean expression $cond$ changes from false to true. Implicit events may or may not have the same semantics as the event lifeline semantics of named events.

Causality: A big step is *causal* if its small steps can be sequenced as: $T_1, T_2, \cdots, T_n$, such that any event that triggers a transition in small step T_i ($1 \leq i \leq n$) must be generated by some earlier small step in $T_1, T_2, \cdots, T_{i-1}$. To a modeller, the transitions of a non-causal big step may seem counterintuitive, and execute out of the blue. The PRESENT IN WHOLE and the PRESENT IN SAME semantic options can create non-causal big steps. To avoid non-causal big steps, some BSMLs that use the WHOLE event lifeline semantics introduce a notion of a "correct" model, which never creates a non-causal big step [6,7,47]. Analysis tools can be used to detect "incorrect" models, conservatively, and reject them at compile time [7,16]. But if a BSML supports variables, the detection of incorrect models is undecidable [16].

Orderedness: The PRESENT IN REMAINDER semantics lacks a "rigorous causal ordering" [30]: if event e_1 is generated earlier than event e_2, it need not be the case that transitions triggered by e_1 are executed earlier than transitions triggered by e_2. The PRESENT IN NEXT COMBO STEP semantics was devised to alleviate this problem by having a "rigorous causal ordering" between combo steps, while being insensitive to the order of event generation within a combo step [19,30]. A disadvantage of the PRESENT IN NEXT COMBO STEP semantics is that a modeller needs to keep track of the scope of a combo step in order to consider its generated events all at once in the next combo step. The PRESENT IN NEXT SMALL STEP semantics is ordered: a transition triggered by an internal event e can be executed only if e is generated by a transition in the previous small step.

Modularity: The PRESENT IN WHOLE option is "modular" [26] with respect to events: an event generated during a big step can be conceptually considered the same as an environmental input event because it is present from the beginning of the big step. All other event lifeline semantics are non-modular. In a non-modular event lifeline semantics, concurrent parts of a model cannot play the role of the environment for each other, because extensions of the model may change the behaviour in different ways than the environment does. As a result, a model cannot be constructed incrementally.

Multiple-instance events: An *instance* of an event in a big step is a contiguous segment of the snapshots of a big step where the event is present. In the PRESENT IN NEXT COMBO STEP, PRESENT IN NEXT SMALL STEP, and PRESENT IN SAME event lifeline semantics, multiple instances of the same event, generated by different small steps, may exist in the same big step. Thus, the status of an event can change multiple times in a big step, making it complicated for a modeller to determine whether an event is present in a certain snapshot of a big step, or not.

Global inconsistency: When negated events are included in the BSML syntax, the PRESENT IN REMAINDER semantic option can produce "globally inconsistent" big steps [41, 42]. A big step is globally inconsistent if it includes a transition that generates an event and a transition triggered by the absence of that event. A globally inconsistent big step is undesired because an event is sensed both as absent and present in the same big step. The PRESENT IN REMAINDER semantic option can achieve a variation of the original global consistency semantics [41,42], by not taking a transition that generates an event that was sensed as absent earlier in the big step [32]. The global inconsistency problem is not relevant for other semantic options because the PRESENT IN REMAINDER semantic option is the only seman-

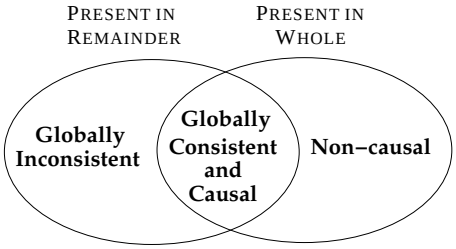

Fig. 14 Global consistency vs. causality.

tic option that allows maximum one instance of an event in a big step and yet permits the aforementioned inconsistency. The other lifeline semantics that permit multiple instances of an event in the same big step are globally inconsistent, but by design.

Global consistency vs. causality: Figure 14 shows the relationship between the big steps of the PRESENT IN REMAINDER semantics and the PRESENT IN WHOLE semantics. A big step T that is included according to a globally consistent PRESENT IN REMAINDER semantics can also be included by a PRESENT IN WHOLE semantics because T's generated events, by the definition of global consistency, can be assumed to be present from the beginning of the big step. Conversely, a big step T' that is included by a causal PRESENT IN WHOLE semantics can also be included by a PRESENT IN REMAINDER semantics because, by the definition of causality, an event is sensed as present by a transition of T' only if it is already generated in the big step. Therefore, if global consistency is guaranteed syntactically (e.g., there are no negated event triggers), then the set of big steps in the PRESENT IN REMAINDER semantics is a subset of the big steps of the PRESENT IN WHOLE semantics.

Events with parameters: An event can have a value parameter, as in Esterel [6].[5] For an event with a value parameter, the value of its parameter is determined per instance of the event. When an event instance is generated by more than one transition, the value of its parameter is determined by a "combine function" [6]. A combine function is a commutative, associative function, such as addition, that "combines" the different values of the parameter of an event that are generated by a set of transitions. In the PRESENT IN REMAINDER, PRESENT IN NEXT COMBO STEP, PRESENT IN NEXT SMALL STEP, and PRESENT IN SAME semantics, a combine function combines the values of the parameter of an event generated by transitions in the previous and current small steps, previous combo step, previous small step, and current small step, respectively. In the PRESENT IN WHOLE option, the value of the parameter of an event instance is fixed during a big step, and is determined by combining all of the values of the parameter of the event generated during the big step.

Example 6 *In Example 2, when considering the* TAKE ONE *big-step maximality semantics, the semantics that subscribes to the* PRESENT IN WHOLE, PRESENT IN REMAINDER, *or* PRESENT IN NEXT SMALL STEP *event lifeline semantics all yield the expected behaviour. If the* TAKE ONE *big-step maximality semantics, the* MANY *concurrency semantics, the* ARENA ORTHOGONAL *small-step consistency semantics, the* PREEMPTIVE *preemption semantics (or the* NON-PREEMPTIVE *preemption semantics) are chosen, then the* PRESENT IN SAME *semantics also yields the expected behaviour.*

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

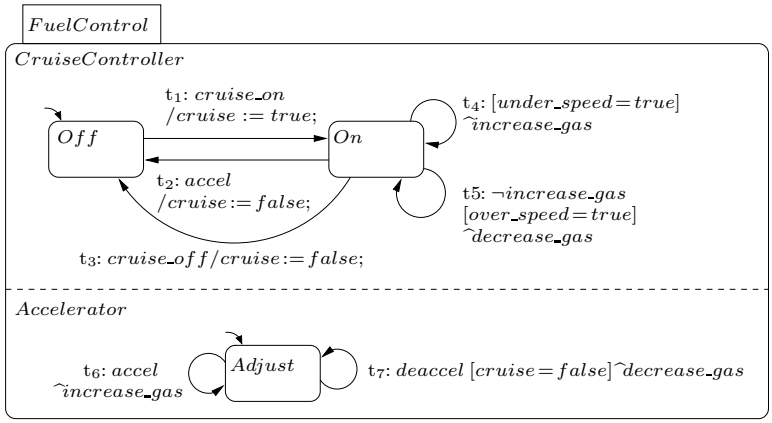

Fig. 15 Speed control system for a car.

Example 7 *The model in Figure 1 is an extension of the model in Figure 7 to support a "redial" functionality. Variable* lp *stores the last dialed phone number. Upon receiving the* redial *environmental input event,* Redialer *instructs* Dialer*, by generating the corresponding* dial *events, to dial the digits of* lp*. (We denote the size of an integer* x *as* |x| *and its* n*th digit as* digit(x, n)*.) Variable* p *is necessary because once redialling starts* lp *is overwritten. Consider the snapshot where the environmental input event* redial *is received,* c *is zero, and* |lp| *is 10. The environmental input event* redial *persists throughout the big step. A semantics that follows the* SYNTACTIC *big-step maximality semantics (annotating a stable control state with a "✓"), the* MANY *concurrency semantics, the* ARENA ORTHOGONAL *small-step consistency semantics, the* PREEMPTIVE *preemption semantics, the* PRESENT IN NEXT SMALL STEP *event lifeline semantics, and uses the up-to-date values of variables, can produce the big step* $\langle \{t_5\}, \{t_2, t_6\}, \{t_3, t_6\}, \cdots, \{t_3, t_6\}, \{t_4, t_7\}\rangle$*, which transmits the first digit twice and does not transmit the last digit. If we choose the* PRESENT IN SAME *event lifeline semantics, the model produces the correct big step* $\langle \{t_5, t_2\}, \{t_3, t_6\}, \cdots, \{t_4, t_7\}\rangle$*. In both cases, if the size of the redialled number is less than 10, the model cannot stabilize, and remains in* DialDigits *control state.*

Example 8 *The model in Figure 15 is a simple model of a cruise control system of a car. The system regulates the amount of power transmitted to the wheels of the car by adjusting the amount of gas that is provided to the engine, in order to maintain the speed specified by the cruise control system. If the cruise control system is on, de-acceleration does not have any effect on the amount of gas that is provided to the engine. But if the cruise control system is on and the acceleration event is received, then the cruise control system is turned off, and acceleration is processed as usual. The two Or control states of the And control state* FuelControl *process the cruise control and acceleration/de-acceleration functionalities, respectively. The environmental input events* cruise_on *and* cruise_off *turn the cruise control system on and off, respectively. The environmental input events* accel *and* deaccel *specify whether the accelerator is being pressed or de-pressed, respectively. The boolean environmental input variables* over_speed *and* under_speed *specify whether the vehicle is moving faster or slower, respectively, than the target speed set by the cruise control system. Events* increase_gas *and* decrease_gas *slightly increase and decrease the amount of fuel into the engine, respectively.*

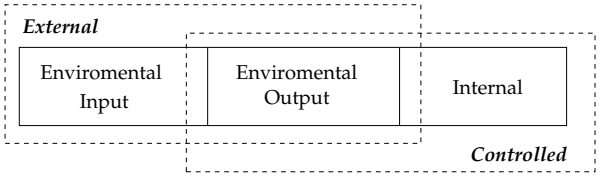

Fig. 16 A taxonomy for events.

Consider the moment when the cruise control system is on, the system is slightly over speed, and the accelerator is pressed; i.e., when the system resides in control state On*,* $\mathrm{over\_speed} = \mathrm{true}$*, and* accel *is received from the environment. We choose the* TAKE ONE *big-step maximality semantics and the* SINGLE *concurrency semantics. If we choose the* PRESENT IN WHOLE *semantic option, then the only possible big step consists of* $\{\mathrm{t}_6\}$ *and* $\{\mathrm{t}_2\}$*, which results in the desired behaviour for the system. If we choose the* PRESENT IN REMAINDER *semantic option, then additionally* $\langle\{\mathrm{t}_5\}, \{\mathrm{t}_6\}\rangle$ *is a valid big step, which both decreases and increases the amount of gas to the engine. The latter big step is globally inconsistent, because* $\mathrm{increase\_gas}$ *is sensed as absent by* t_5 *and is generated by* t_6*. If the variation of global consistency semantics in [32] is chosen, then* $\langle\{\mathrm{t}_5\}\rangle$ *is a valid big step;* t_6 *cannot be taken during the big step since it generates* $\mathrm{increase\_gas}$*.*

3.3.1 External Events

The model in Figure 1 uses event $dial$ in two different ways: (i) as an environmental input event initiated by a human caller, and (ii) as an internal event generated by the $Redialer$. To avoid modelling flaws, many have advocated that the interface of a system with its environment should be clearly and explicitly specified [39,50]. A celebrated way to achieve this interface, as shown in Figure 16, is to distinguish between the events that the environment can control, *environmental input events*, and the events that are generated by the model, *controlled events*. A controlled event may be observable by the environment (i.e., an *environmental output event*), or not (i.e., an *internal event*). The environmental input and output events of a model together constitute the *external events* of the model.

A BSML may choose distinct event lifeline options for environmental input events, environmental output events, and internal events, as shown in the feature diagram of Figure 5. Often, the event lifeline semantics of the environmental input events is the PRESENT IN WHOLE semantics, and the event lifeline semantics of the environmental output events is the same as the event lifeline semantics of the internal events.

A BSML may syntactically distinguish environmental input events and environmental output events from each other, and from internal events. Alternatively, we call a BSML *non-distinguishing* if it does not distinguish syntactically between the external events and the internal events of a model. In these BSMLs, it is still possible to consider inputs received at the beginning of the big step as environmental inputs, and outputs generated in the last small step or last combo step of a big step as environmental outputs, each with distinct event lifeline choices. Table 4 lists the possible semantic options for differentiating environmental input events and internal events. In the SYNTACTIC INPUT EVENTS option, an environmental input event is syntactically distinguished. Thus a BSML that subscribes to this option is a "distinguishing" BSML. In the RECEIVED EVENTS AS ENVIRONMENTAL option, an event that is received at the beginning of a big step is considered an environmental input

Table 4 Differentiating environmental input events from internal events.

Options	Definition	Characteristics	Examples
SYNTACTIC INPUT EVENTS	Only syntactically distinguished events are treated as environmental inputs.	(+) Separates system from environment (-) Usually different semantics for different event types	Esterel [6]
RECEIVED EVENTS AS ENVIRONMENTAL	Any event that is received from the environment at the beginning of a big step is treated as an environmental input.	(+) Treats input and internal events uniformly (-) No boundary between system and environment	statecharts [42] and RSML [30]
HYBRID INPUT EVENTS	Only genuine inputs that are received from the environment at the beginning of a big step are treated as environmental inputs.	(+) Distinguishes between internal and genuine input events (-) Complex	N/A

event. In the HYBRID INPUT EVENTS option, an event that is received at the beginning of a big step is considered an environmental input event only if it is a *genuine input* of a model, meaning it is not generated by any transitions in the model. As shown in Figure 5, an event lifeline semantics for the environmental input events can be chosen, regardless of the choice of the semantic option for distinguishing the input events. For example, if the semantics for environmental inputs is the RECEIVED EVENTS AS ENVIRONMENTAL semantic option together with the PRESENT IN NEXT SMALL STEP semantic option, then an input event that is received at the beginning of a big step persists only for the first small step of the big step. Environmental output events have similar options; events generated in either the last small step or last combo step of a big step could be considered as environmental output events.

Example 9 *In Example 7, we assumed the non-distinguishing semantics for the model in Figure 1 because event* dial *can be both received from the environment and generated, possibly in the same big step. Event* redial *is a genuine input. Both the* RECEIVED EVENTS AS ENVIRONMENTAL *and* HYBRID INPUT EVENTS *semantic options, together with the* PRESENT IN WHOLE *event lifeline semantics, yield a behaviour that matches the behaviour specified in Example 7.*

If we use the single-input assumption [22,23], which requires that dial *and* redial *are not both received from the environment in the same big step, then* dial *cannot be received from the environment at the beginning of a big step and generated in the same big step.*

3.3.2 Interface Events

Some BSMLs structure a model as a set of *components*, each of which is a CHTS. The components of a model communicate with each other through their *interface events* according to an *inter-component communication mechanism*. Figure 17 refines the taxonomy of Figure 16 by including interface events as a subset of the controlled events of a model. We require an interface event to be generated by one component, which we call its *sending component*. A component that accesses an interface event is its *receiving component*. As such,

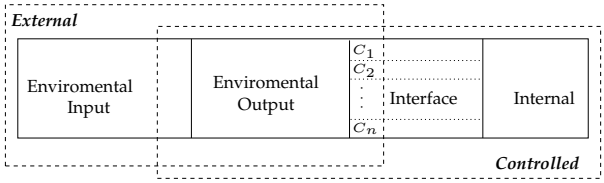

Fig. 17 A taxonomy of events for inter-component communication.

the interface events of a model are partitioned into sets, shown by dashed lines in Figure 17, each of which is generated by one component.

Table 5 lists the three possible inter-component communication semantic options for interface events. In the STRONG SYNCHRONOUS EVENT option, a generated interface event is sensed as present throughout the big step in which it is generated, from the beginning of the big step (similar to the PRESENT IN WHOLE semantic option for internal events). In the WEAK SYNCHRONOUS EVENT option, a generated interface event is present in the big step in which it is generated, but only after it is generated (similar to the PRESENT IN REMAINDER semantic option for internal events). In the ASYNCHRONOUS EVENT option, a generated interface event is present in the next big step, from the beginning of the big step. The STRONG SYNCHRONOUS EVENT and the WEAK SYNCHRONOUS EVENT semantic options have similar advantages and disadvantages as the PRESENT IN WHOLE and PRESENT IN REMAINDER semantic options, respectively. The ASYNCHRONOUS EVENT semantic option is unique in that a generated event in a big step can influence the behaviour of the model in the next big step. This semantics for interface events can potentially be a source of complication for a modeller because it is at odds with the semantics of other kinds of events in a semantics, i.e., internal events and environmental input/output events, whose statuses cannot persist beyond a current big step. In the ASYNCHRONOUS EVENT semantics, a generated interface event in a big step acts similar to an environmental input event in the next big step. As such, the ASYNCHRONOUS EVENT semantics is modular with respect to interface events, because an interface event, similar to an environmental input event, is either present from the beginning of a big step or is not present at all.

There are several BSMLs that support the notion of inter-component event communication. The "hybrid semantics" of Huizing and Gerth [26], which distinguishes between "local" and "global" events, treats the "global" events of a model according to the STRONG SYNCHRONOUS EVENT semantic option. The semantics of "output" events in RSML [30] follows the ASYNCHRONOUS EVENT semantics; an "output" event is generated by a component via a "`SEND`" command, and can be received by a component via a "`RECEIVE`" event in the next big step. Similarly, the semantics of "registered" events in Esterel [1] follows the ASYNCHRONOUS EVENT semantics. In "globally asynchronous locally synchronous (GALS)" languages [8,44], the communication of events within "local" components of a system follows the semantics of the PRESENT IN WHOLE option, and the "global" communication of events between components follows the semantics of the ASYNCHRONOUS EVENT option.

Example 10 *The model in Figure 18 shows a door controller system, which is responsible for unlocking the door to an industrial area only if the temperature inside the area is not above 40°C. The system has two components,* Lock *and* Thermometer*, separated by the thick dashed line. The two components communicate via two interface events,* check_temp *and* heat*. There are three environmental input events,* lock*,* open*, and* reset*. Event* unlock

Table 5 Semantic options for interface events.

Options	Definition	Characteristics	Examples
STRONG SYNCHRONOUS EVENT	A generated interface event of a big step is sensed as present from the beginning of the big step.	(+) Modularity (+) Unique status for an interface event during a big step (-) Non-causality	"Hybrid Semantics" [26]
WEAK SYNCHRONOUS EVENT	A generated interface event of a big step is sensed as present in the snapshot after it is generated.	(+) Causality (-) Unclear status of an interface event during a big step	N/A
ASYNCHRONOUS EVENT	A generated interface event of a big step is sensed as present in the next big step after it is generated.	(+) Modularity (-) Previous big step affects current big step	"Output" events in RSML [30] and "GALS" [44]

is the environmental output event of the model. Consider the snapshot in which the model resides in its Idle *and* Measure *control states,* $\mathrm{temp} = 99$*, and event* open *is received from the environment. If we choose the* TAKE MANY *big-step maximality semantics, the* SINGLE *concurrency semantics, and the* STRONG SYNCHRONOUS EVENT *semantic option, then the big step* $\langle\{t_1\}, \{t_6\}, \{t_3\}\rangle$ *is the only possible big step, which, correctly, does not open the door. If we choose the* WEAK SYNCHRONOUS EVENT *semantic option, then additionally,* $\langle\{t_1\}, \{t_2\}, \{t_6\}\rangle$ *is a valid big step, which opens the door although the temperature is 99°C. If we choose the* ASYNCHRONOUS EVENT *semantic option, the only possible big step is* $\langle\{t_1\}, \{t_2\}, \{t_6\}\rangle$*, in which event* heat *is sensed in the next big step, after the door has already been opened.*

3.4 Enabledness Memory Protocol

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

Traceability: In the GC BIG STEP semantics, the value of a variable at a snapshot in a big step is obtained from the beginning of the big step, but the assignments to the variable

[6] As shown in Table 6, in SCR [22, 23], both the GC BIG STEP and GC SMALL STEP memory protocols are used, but in different syntactic constructs of the language, namely in the "event tables" and "condition tables", respectively.

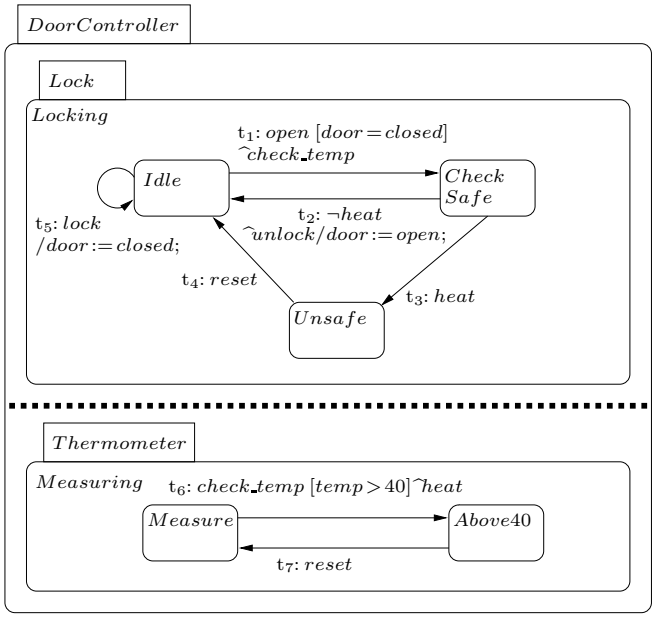

Fig. 18 Door controller system: using interface events $heat$ and $check\_temp$.

Table 6 Enabledness memory protocols.

Options	Definition	Characteristics	Examples
GC BIG STEP	The value of a variable during a big step is obtained from the beginning of the big step.	(+) Non-interference (+) Modularity (-) Non-sequentiality in small steps	statecharts [21, 42], SCR [22, 23], and reactive modules [3]
GC SMALL STEP	The value of a variable is its up-to-date value, obtained from the beginning of the small step.	(+) Sequentiality in small steps (+) Straightforward traceability (-) Interference	Esterel [6] and SCR [22, 23]
GC COMBO STEP	The value of a variable during a combo step is obtained from the beginning of the combo step.	(+) Some non-interference (+) Some sequentiality in small steps (-) Complicated traceability	Statemate [19]

need to be traced so that its value is updated for the next big step. In the GC SMALL STEP semantics, the value of a variable at a snapshot in a big step is determined by tracing all of the assignments to the variable since the beginning of the big step. In the GC COMBO STEP semantics, the value of a variable at a snapshot in a big step is determined by tracing all of the assignments from the beginning of the current combo step. But a big step may

Table 7 Variable operators.

Operator	Obtains Value From	Memory Protocols	Total
`pre` (e.g., [30])	big-step source snapshot	GC SMALL STEP	✓
`cur` (e.g., [21])	small-step source snapshot	GC BIG STEP	✓
`new` (e.g., [3])	small-step source snapshot	GC BIG STEP and GC SMALL STEP	✗
`new_small` (e.g., [41])	small-step destination snapshot	GC SMALL STEP	✓

have several combo steps, which, compared to the other memory protocols, could make the tracing of the value of a variable complicated.

Modularity with respect to variables: In general, a semantics is "modular" if it treats the behaviour of a new concurrent part of the model the same as the behaviour of the environment [26]. Originally, "modularity" was defined with respect to events [26], but, in the same spirit, we extend it for variables. The GC BIG STEP is modular with respect to variables because even if a new concurrent part of a model assigns new values to variables, the new values are visible only at the beginning of the next big step, just like new environmental values. The other semantic options are not modular because the behaviour of an addition to an existing model, unlike the environment, affects the intermediate snapshots of a big step.

Non-interference vs. sequentiality in small steps: The GC BIG STEP option is *non-interfering*: an earlier small step of a big step does not affect the read value of a later small step. The GC SMALL STEP option, which is an "interfering" semantics, is useful for specifying a sequence of computations where each small step reads the values from the previous small step. The GC COMBO STEP option enjoys non-interference inside a combo step and sequentiality of combo steps. In the GC COMBO STEP option, a big step could consist of multiple combo steps, which a modeller needs to keep track of each of their scopes.

Variable operators: A BSML may provide a *variable operator* that obtains a value of a variable that is different from its value according to its memory protocol. Table 7 lists some common operators and specifies whether they are *total* or not. As specified in the table, each variable operator is relevant for certain enabledness memory protocols. A non-total operator may *block* until it can be evaluated.

Operator `new` is different from `cur` in that it can be evaluated only if its operand has already been assigned a value during the big step, which means it requires a "dataflow" order for the execution of small steps within a big step (cf., Section 3.6).

Operator `new_small` returns the value of its operand at the end of the current small step. A variable in the GC of a transition that is prefixed with the `new_small` operator requires an *evaluation order* between the transitions of the small step, in order to obtain the newly assigned value of the variable at the end of the small step. If a variable is not assigned a value during a small step, then its value when prefixed with the `new_small` operator returns the value of the variable at the source snapshot of the small step.[7]

Two transitions can create *cyclic evaluation order* by using the `new_small` operator over variables that are assigned values by one another.

[7] It is possible to define a non-total `new_small` operator that returns a value for a variable, only if it is assigned a value in the current small step. Such an operator would be in the spirit of the "next" operator in SMV language [34], which is an input language for a family of model checkers with the same name. However in the semantics of SMV, unlike in BSMLs, even if a variable is not assigned a value during a small step, it is assigned a non-deterministic value, which, in effect, makes the "next" operator a total operator.

Example 11 *In Example 7, we used the* GC SMALL STEP *enabledness memory protocol. If we use the semantic options that led to an incorrect behaviour in that example, but modify the guard condition of* t_6 *to "*$[\mathrm{new\_small}(\mathrm{c}) < |\mathrm{p}|]$*" and its event generation to "*$\mathrm{dial}(\mathrm{digit}(\mathrm{new\_small}(\mathrm{c}) + 1, \mathrm{p}))$*", then the model behaves correctly:* $\langle\{\mathrm{t}_5\}, \{\mathrm{t}_2, \mathrm{t}_6\}, \{\mathrm{t}_3, \mathrm{t}_6\}, \cdots, \{\mathrm{t}_3\}, \{\mathrm{t}_4, \mathrm{t}_7\}\rangle$.

The operators in Table 7 are not relevant for the GC COMBO STEP memory protocol, but they can be extended to be used in the context of GC COMBO STEP memory protocol. For example, a version of `cur` operator for the GC COMBO STEP semantic option would return the current value of a variable considering all of the assignments to the variable since the beginning of the current combo step. Similarly, a `new_small` operator can be defined for the GC BIG STEP memory protocol.

3.4.1 External Variables

As with events, it is useful to distinguish syntactically between the variables of the model that can be modified by the environment and the variables of the model that can be modified by the system [39,50]. Figure 16, which depicts a taxonomy of events, also represents the taxonomy for distinguishing environmental variables. The *environmental output variables* and *environmental input variables* of a model are the sets of the variables of the model that can be read from and written to by the environment, respectively. The *internal variables* of a model are those variables that are not communicated with environment.[8] The union of the set of environmental input variables and the set of environmental output variables of a model is its set of *external variables*. The union of the set of environmental output variables and the set of internal variables of a model is its set of *controlled variables*, which is the set of variables that can be written to by the system. Many modelling languages, including some BSMLs, provide syntax to distinguish between different types of variables [3,22,23, 39]. Unlike for events, the notion of "non-distinguishing BSMLs" (cf., Section 3.3.1) is not relevant with respect to variables, because most BSMLs either syntactically distinguish between environmental input variables and controlled variables, or they do not support the notion of environmental input variables at all (i.e., variables are not assigned values by the environment).

When external variables are distinct from the internal variables, the memory protocol semantic aspects described in Sections 3.4 and 3.5 specify the semantics of internal variables. The notion of memory protocol for environmental input variables is not relevant because they are never assigned a value by a transition; they keep the same value throughout the big-step. Normally, an output variable is not read by the model, therefore we have not included options for it in our feature diagram. If it is, the semantics of environmental output variables can be any of the memory protocols, but it would not likely be the BIG STEP semantics.

3.4.2 Interface Variables in GC

Some BSMLs allow a component of a model, which is usually a physically distinct part of the model, to communicate with another component of the model via *interface variables*. Figure 17, which depicts the taxonomy of events including interface events, also illustrates the taxonomy of variables including interface variables. As for interface events, we require

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

the well-formedness constraint that an interface variable can be written to by only one component (the *sending component*), but can be ready by multiple components (the *receiving components*). The semantics of interface variables, similar to memory protocols for internal variables, specifies when a change to an interface-variable value becomes the value returned by a read of that variable.

Table 8 lists the possible inter-component communication semantic options. In the GC STRONG SYNCHRONOUS VARIABLE option, a write to an interface variable during a big step can be read by the GC of a transition right from the beginning of the same big step; i.e., if an interface variable is assigned a value, only this new value is read during the big step. In the GC WEAK SYNCHRONOUS VARIABLE option, a write to an interface variable can be read after the variable is written to, but the variable can also be read before it is written to, in which case it returns its value from the previous big step (similar to the GC SMALL STEP semantic option). In the GC ASYNCHRONOUS VARIABLE option, a write to an interface variable can be read by the GC of any transition in the next big step (similar to the GC BIG STEP semantic option).

Blocking read vs. communication delay: The GC STRONG SYNCHRONOUS VARIABLE semantics is compatible with the "zero-time computation" principle of the synchrony hypothesis [4,6]: that is, the value of an interface variable is exchanged between two components in "zero-time". However, there should exist a "dataflow order" (cf., Section 3.6) between the small steps of a big step so that the value of an interface variable is read only after it has been assigned. A component that is waiting for the new value of an interface variable is said to be *blocking*. It is possible for two transitions to block cyclically on each other. In the GC WEAK SYNCHRONOUS VARIABLE semantic option, a read operation on a variable never blocks, but it may return a *stale value* of the variable from the previous big step or a newly assigned value from the current big step. In the GC ASYNCHRONOUS VARIABLE semantic option, a read operation on a variable never blocks, but there is a delay of one big step between writing a new value to a variable and reading the new value.

Modularity with respect to interface variables: The GC STRONG SYNCHRONOUS VARIABLE and GC ASYNCHRONOUS VARIABLE semantic options are modular with respect to interface variables because the value of an interface variable in these semantic is the same throughout the big step, similar to an environmental input variable. In these two semantics, the behaviour of a component that is added to an existing model is perceived as that of environment, when it comes to the interface variables in the GC of transitions of the existing model. The GC WEAK SYNCHRONOUS VARIABLE semantic option is not modular with respect to interface variables because the value of an interface variable may change during a big step, unlike the value of an environmental input variable.

Example 12 *The model in Figure 19 is similar to the model in Example 10, but has been modified: (i) to use the interface variable* heat*, instead of interface event* heat*; and (ii) the functionality of* Locking *the door is separated from the functionalities of the* Controller *of the lock and the* Thermometer*, to allow for the lock to work with different controllers.*

Consider the snapshot where the model resides in its Idle*,* Ready*, and* Measure *control states, the door is closed,* $\mathrm{temp} = 99$*,* $\mathrm{heat} = \mathrm{false}$*, and event* open *is received from the environment. We choose the* SYNTACTIC *big-step maximality semantics, the* SINGLE *concurrency semantics, the* PRESENT IN REMAINDER *event lifeline semantics, the* GC *(and* RHS*)* SMALL STEP *enabledness (assignment) memory protocols, and the* GC STRONG SYNCHRONOUS EVENT *interface event semantics. If we choose the* GC STRONG SYNCHRONOUS VARIABLE *semantic option, then the big step* $\langle\{\mathrm{t}_1\}, \{\mathrm{t}_6\}, \{\mathrm{t}_9\}, \{\mathrm{t}_8\}, \{\mathrm{t}_3\}\rangle$ *is the only possible big step, which correctly does not open the door. If we choose the* GC

Table 8 Semantic options for interface variables.

Options	Definition	Characteristics	Examples
GC STRONG SYNCHRONOUS VARIABLE	Either an interface variable is not written to during a big step, or all of its reads happen after it has been written to and it returns the newly assigned value.	(+) Modularity (-) Blocking read and cyclic dataflow order	Composition in reactive modules [3]
GC WEAK SYNCHRONOUS VARIABLE	An interface variable can be read before or after it is written to; in the latter case it returns the newly assigned value.	(+) Non-blocking read (-) Stale values for interface variables	N/A
GC ASYNCHRONOUS VARIABLE	The value written to an interface variable during a big step can be read in the next big step.	(+) Non-blocking read (+) Modularity (-) Delayed read	"Output" variables in RSML [30]

WEAK SYNCHRONOUS VARIABLE *semantic option, then the big step* $\langle\{\mathrm{t}_1\}, \{\mathrm{t}_6\}, \{\mathrm{t}_7\}, \{\mathrm{t}_9\}, \{\mathrm{t}_2\}\rangle$ *is also possible, which opens the door although the temperature is 99°C. Reversing the order of* $\{\mathrm{t}_9\}$ *and* $\{\mathrm{t}_2\}$ *yields another big step that opens the door. If we choose the* GC ASYNCHRONOUS VARIABLE *semantic option, then the* true *value of* heat *is only sensed in the next big step, and thus the door is opened.*

3.5 Assignment Memory Protocol

The *assignment memory protocol* of a BSML determines the values of variables that a transition reads when evaluating the righthand side (RHS) of its assignment expressions. Exactly the same semantic options as those of the enabledness memory protocol are identified: RHS BIG STEP, RHS SMALL STEP, and RHS COMBO STEP. (Their names are prefixed with "RHS" instead of "GC".) The enabledness and assignment memory protocols of a BSML need not be the same (e.g., SCR [22,23]).[9] The same advantages and disadvantages as the semantic options of the "enabledness memory protocol", in Table 6, apply to the corresponding semantic options of the "assignment memory protocol" semantic aspect.

Variable operators: The same four variable operators listed in Table 7 can be used in the RHS of assignments. However, when using the `new_small` operator in an assignment expression, it may be impossible to find an "evaluation order". For example, for two assignments, $a := \texttt{new\_small}(b) - 1$ and $b := \texttt{new\_small}(a) + 2$, which have a cyclic evaluation order, the value of a and b cannot be evaluated.

Example 13 *The model in Figure 20, which is adopted from an example in [25], is meant to specify a computation that maintains the invariant of* $\mathrm{a} - \mathrm{b}$ *remaining the same before and after the execution of a big step. Consider the snapshot where the model resides in its control states* S_1 *and* S_4*,* $a = 7$*, and* $b = 2$*. We choose the* SINGLE *concurrency semantics. If we choose the* TAKE MANY *big-step maximality semantics together with the* RHS BIG

[9] In SCR [22,23], the RHS SMALL STEP assignment memory protocol is used together with a combination of the GC BIG STEP and GC SMALL STEP enabledness memory protocols.

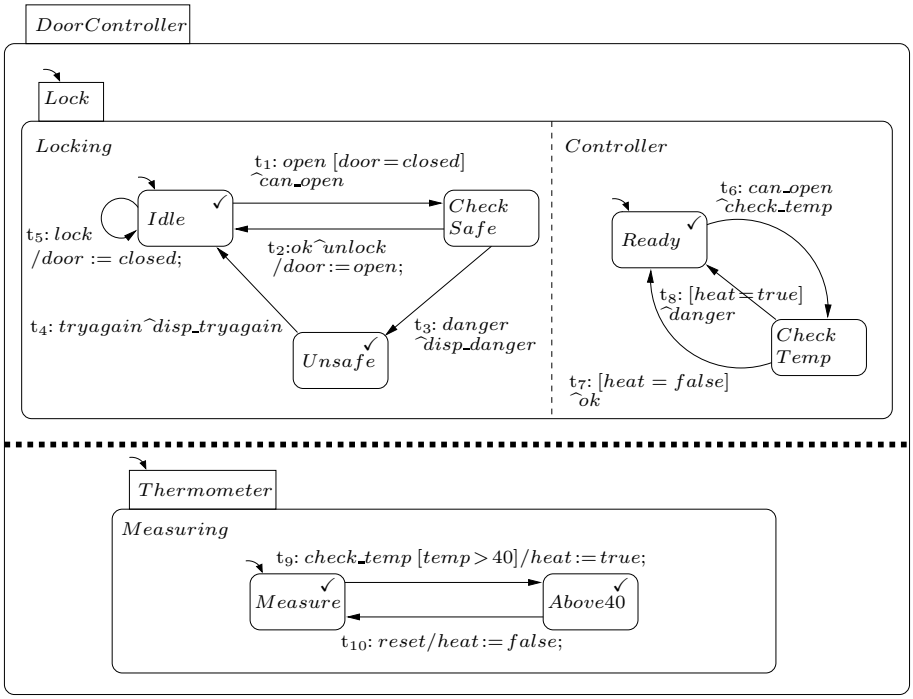

Fig. 19 Door controller system: using interface variable $heat$ and interface event $check\_temp$.

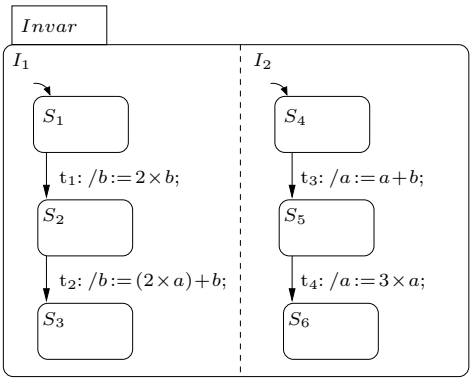

Fig. 20 A model for maintaining an invariant between a and b.

STEP *assignment memory protocol, then the end result would be* $a = 21$ *and* $b = 16$*, which maintains the value of* $\mathrm{a}-\mathrm{b}$ *the same before and after the big step. If we choose the* RHS SMALL STEP *semantic option, then the model can create a big step that does not maintain the invariant; for example, the execution of the big step* $\langle\{\mathrm{t}_1\}, \{\mathrm{t}_2\}, \{\mathrm{t}_3\}, \{\mathrm{t}_4\}\rangle$ *results in* $a = 75$ *and* $b = 18$*.*

3.5.1 Interface Variables in RHS

Similar to using interface variables in the GC of transitions, as described in Section 3.4.2, interface variables can be used in the RHS of assignments of the transitions of the different components of a system. Exactly the same semantic options as those for interface variables in GC of transitions can be used for the semantics of interface variables in the RHS of assignments, but their names prefixed with "RHS" instead of "GC": RHS STRONG SYNCHRONOUS VARIABLE, RHS WEAK SYNCHRONOUS VARIABLE, and RHS ASYNCHRONOUS VARIABLE. The interface variables in GC semantics of a BSML need not be the same as its interface variables in RHS semantics. Similar to the GC STRONG SYNCHRONOUS VARIABLE option, a cyclic dataflow order might arise when the RHS STRONG SYNCHRONOUS VARIABLE semantic option is chosen. The same advantages and disadvantages as the ones for the semantic options of the inter-component variable communication, in Table 8, are relevant for the corresponding semantic options of the interface variables in RHS semantic aspect.

3.6 Order of Small Steps

At a snapshot, when it is possible to execute more than one small step based on the enabledness of transitions with respect to guard conditions and event triggers, some BSMLs non-deterministically execute one (the NONE option), while others order their executions either by syntactic means (the EXPLICIT ORDERING option) or by *dataflow* orders (the DATAFLOW option), as shown in Table 9. Stateflow is an example of the EXPLICIT ORDERING option because the transitions of a model are executed according to the graphical, clockwise order of their arenas [9]. A dataflow order allows only those sequences of the execution of small-steps where a transition that writes to a variable is executed before transitions that read the variable. The dataflow order of a model can be specified by an explicit partial order between its variables (e.g., SCR [22, 23]), or via variable operator `new`, as described in Section 3.4, to determine data dependencies (e.g., reactive modules [3]). In the statecharts semantics of Pnueli and Shalev [42], the boolean operator `assigned` is used in the event trigger of a transition to determine whether a variable is assigned a value during a big step or not, which in effect induces dataflow order between small steps of the big step.[10] The EXPLICIT ORDERING and DATAFLOW options can be used to avert undesired non-determinism by disallowing the execution of the small steps that do not satisfy the ordering constraints. In the DATAFLOW semantic option, each big step of a model might have a different dataflow order. The EXPLICIT ORDERING option can be difficult to use because a modeller may introduce an unintended order of transitions. The DATAFLOW semantics can be difficult to use because a modeller might create a cyclic dataflow order, either directly or by transitivity. The DATAFLOW semantics is compatible with the domain of some synchronous hardware systems where there is an inherent distinction between the value of a variable at the beginning of a big step, i.e., when the clock ticks, and during a big step when a value might be assigned to a variable.

Example 14 *Consider the semantic options in Example 7 that lead to an incorrect behaviour. One way to fix the incorrect behaviour is to modify the model by moving the*

[10] The GC STRONG SYNCHRONOUS VARIABLE and RHS STRONG SYNCHRONOUS VARIABLE semantic options for interface variables, described in Section 3.4.2 and Section 3.5.1, respectively, can also introduce dataflow orders.

Table 9 Order of small steps semantic options.

Options	Definition	Characteristics	Examples
NONE	Small steps are not ordered.	(+) Simplicity (-) Non-determinism	statecharts [17, 21]
EXPLICIT ORDERING	Execution of small steps is ordered syntactically.	(+) Control over ordering (+) Control over non-determinism (-) Possible unintended ordering	Stateflow [9]
DATAFLOW	Small steps are ordered so that an assignment to a variable happens before it is being read.	(+) Natural for some domains (-) Control over non-determinism (-) Possible cyclic orders	SCR [22, 23], reactive modules [3], and statecharts [42]

"$\mathrm{p} := \mathrm{lp}$" assignment from t_5 to t_2, changing the GC of t_6 to "$\mathrm{c} < |\mathrm{new}(\mathrm{p})| - 1$", and its event generation to "$\mathrm{dial}(\mathrm{digit}(\mathrm{new\_small}(\mathrm{c}) + 1, \mathrm{p}))$". Such a model then behaves correctly: $\langle\{\mathrm{t}_5\}, \{\mathrm{t}_2\}, \{\mathrm{t}_6\}, \{\mathrm{t}_3, \mathrm{t}_6\}, \cdots, \{\mathrm{t}_3\}, \{\mathrm{t}_4, \mathrm{t}_7\}\rangle$, because the dataflow order does not allow t_2 and t_6 to be executed together.

Example 15 *In Example 7, we chose the* MANY *concurrency semantics and the* PRESENT IN NEXT SMALL STEP *event lifeline semantics, which lead to an incorrect behaviour. If we choose the* SINGLE *concurrency semantics, then the model can create both a correct big step, and an incorrect, non-terminating big step (e.g., $\langle\{\mathrm{t}_5\}, \{\mathrm{t}_2\}, \{\mathrm{t}_6\}, \{\mathrm{t}_6\}, \cdots\rangle$), non-deterministically. However, if we use the* EXPLICIT ORDERING *order of small-steps semantics according to the graphical, clockwise order of the arena of transitions, then the model always behaves correctly: $\langle\{\mathrm{t}_5\}, \{\mathrm{t}_2\}, \{\mathrm{t}_6\}, \{\mathrm{t}_3\}, \{\mathrm{t}_6\}, \{\mathrm{t}_3\}, \cdots, \{\mathrm{t}_7\}, \{\mathrm{t}_4\}\rangle$.*

3.7 Priority

At a snapshot of a model, there could exist multiple sets of transitions that can be chosen non-deterministically to be executed as its small step. Table 10 shows three common ways for assigning a priority to a transition to avert non-determinism. A set of transitions T_1 has a higher priority than a set of transitions T_2, if for each pair of transitions $t_1 \in T_1$ and $t_2 \in T_2$, either t_1 has a higher priority than t_2 or they are not comparable priority wise.

The HIERARCHICAL option is a set of priority semantics that use the hierarchical structure of the control states of a model to compare the relative priority of two enabled transitions. A HIERARCHICAL priority semantics is defined by its *basis*, which is one of the three values, SOURCE, DESTINATION, ARENA, and its *scheme*, which is either PARENT or CHILD. For example, ARENA-PARENT is a priority semantics that gives a higher priority to a transition whose arena is the highest in the hierarchy of a composition tree. The EXPLICIT PRIORITY priority option explicitly assigns priority to the transitions of a model (e.g., by assigning numbers to transitions and giving a greater number a higher priority [37]). The NEGATION OF TRIGGERS option is not an independent way to assign priority, but uses the notion of "negation" to assign priorities: t_1 can be assigned a higher priority than t_2 by conjoining the negation of the event trigger and guard condition of t_2 with the ones of t_1.

Exhaustiveness vs. simplicity: The HIERARCHICAL option can be easily understood by a modeller, but may render many transitions as priority incomparable. The EXPLICIT

Table 10 Priority semantic options.

Options	Definition	Characteristics	Examples
HIERARCHICAL	The source and destination control states of transitions determine priority.	(+) Simplicity (-) Incomplete prioritization	ARENA-PARENT in Statemate [19] and SOURCE-CHILD in Rhapsody [18]
EXPLICIT PRIORITY	Each transition is given an explicit, relative priority.	(+) Exhaustive prioritization (-) Tedious to use	Used in [37]
NEGATION OF TRIGGERS	A transition is given higher priority than another by strengthening the event trigger and GC of the second transition such that is not enabled when the first transition is enabled.	(+) Exhaustive prioritization (+) No additional syntax (-) Tedious to use	statecharts [42], Esterel [6], and Argos [33]

PRIORITY option provides great control over specifying the relative priority of a set of transitions, but can be tedious to use (e.g., a wrong relative priority for a pair of transitions can be deduced transitively). In the NEGATION OF TRIGGERS and EXPLICIT PRIORITY options, it can be difficult to identify the pair of transitions where it is necessary to assign a relative priority because whether two transitions are both enabled or not in a small step depends on the source snapshot. But in principle, it is possible to specify a priority scheme for a model exhaustively.

Combination of priority semantics: It is possible to use more than one priority semantics in the semantics of a BSML, as shown in the feature diagram in Figure 5. In such a BSML, if a pair of transitions are not comparable according to the first priority semantics, then they are compared according to the second semantics, and so on. By the definition of enabledness, if the NEGATION OF TRIGGERS is used in a BSML, its semantics overrides the other priority semantics.

Example 16 *In Example 5, if we choose the* SINGLE *concurrency and the* ARENA-CHILD *priority semantics, then the model always executes* $\langle\{\mathrm{t}_1\}\rangle$ *as its big step, allowing the call to go through.*

Example 17 *In the model in Figure 1,* t_2 *is assigned a higher priority than* t_1 *by conjoining the original event trigger of* t_1, $\mathrm{dial(d)}$, *with the negation of the event trigger of* t_2, $\mathrm{dial(d)} \wedge \mathrm{redial}$, *resulting in* t_1 *having the event trigger* $\mathrm{dial(d)} \wedge \neg\mathrm{redial}$. *The effect is that* t_2 *will be chosen when the* redial *event occurs instead of* t_1.

Example 18 *In Example 10, if transition* t_6 *is given a higher priority than* t_2 *explicitly, then the choice of the* WEAK SYNCHRONOUS EVENT *semantic option always yields a correct behaviour (i.e., the door is not opened when the temperature is above 40°C). Similarly, in Example 12, if transition* t_9 *is given a higher priority than* t_7 *explicitly, then the choice of the* WEAK SYNCHRONOUS VARIABLE *semantic option always yields a correct behaviour.*

3.8 Combo-Step Maximality

The combo-step maximality semantics specifies the extent of a contiguous segment of a big step where computation is carried out based on the statuses of events and/or values of the variables at the beginning of the segment. As specified in Figure 6, the combo-step maximality semantics is relevant for a BSML semantics only if at least one of the *combo-step semantic options*, namely, PRESENT IN NEXT COMBO STEP, GC COMBO STEP, or RHS COMBO STEP, is chosen in the semantics. These options describe how the statuses of events and values of variables change (or not) within a combo step. For example, if a BSML uses the PRESENT IN NEXT COMBO STEP and GC COMBO STEP options, then during a combo step (other than the first combo step of the big step) the statuses of events depend on the generated events of the previous combo step, and the values of variables in GC of transitions depend on the assignments performed in the previous combo step.

Table 11 shows the three semantic options for the combo-step maximality semantic aspect. These options are similar to the three semantic options for the big-step maximality semantics, but specify the scope of a combo step, instead of a big step. In the COMBO SYNTACTIC option, a BSML allows a modeller to designate a basic control state of a model as a *combo stable* control state. During a combo step, once a transition t that enters a combo stable control state is executed, no other transition whose arena overlaps with the arena of t can be taken during that combo step. In the COMBO TAKE ONE option, once a transition t is executed during a combo step, no other transition whose arena overlaps with the arena of t can be executed during that combo step. As such, each *Or* control state can contribute a maximum of one transition to a combo step. The COMBO TAKE MANY option allows a sequence of small steps to continue executing until there are no more enabled transitions to be executed. In practice, we are only aware of BSMLs that use the COMBO TAKE ONE option for the combo step maximality semantics and the TAKE MANY option for the big-step maximality semantics (e.g., RSML [30] and Statemate [19]). As specified in Figure 6, the COMBO TAKE MANY combo-step maximality semantics cannot be chosen together with the TAKE ONE big-step maximality semantics, because a combo step cannot include more small steps than its big step. The same advantages and disadvantages as the ones for the semantic options of the big-step maximality semantic aspect are relevant for the corresponding semantic options of the combo-step maximality semantic aspect.

Scope of a combo step: In the COMBO SYNTACTIC semantic option, the end of a combo step can be traced syntactically, which can be helpful for constructing and understanding a model. The scope of a combo step when the COMBO TAKE ONE or the COMBO TAKE MANY is chosen is more difficult to determine. For example, if the COMBO TAKE MANY combo-step maximality semantics, along with the PRESENT IN NEXT COMBO STEP and GC COMBO STEP semantic options, are chosen, then a combo step of a big step continues until there are no more transitions that are enabled with respect to the generated events and the assignments of the previous combo step. In such a semantics, it is far from clear what the possible combo steps, and thus big steps, of a model are, based on mere review of the syntax of the model.

Example 19 *The model in Figure 21 is meant to swap the values of variables* a *and* b *twice during a big step, maintaining their original values. We choose the* COMBO TAKE ONE *option for the combo step maximality semantics, the* TAKE MANY *option for the big-step maximality semantics, the* SINGLE *concurrency semantics, and the semantics that the statuses of events and the values of variables are fixed during a combo step (i.e., the* RHS COMBO STEP *and the* PRESENT IN NEXT COMBO STEP *semantic options). Upon receiving*

Table 11 Combo-step maximality semantic options.

Options	Definition	Characteristics	Examples
COMBO SYNTACTIC	No two transitions with overlapping arenas that enter designated "combo stable" control states can be taken in a same combo step.	(+) Syntactical scope for combo steps (+) Sequential *Or* transitions in a combo step (-) Non-terminating combo steps	N/A
COMBO TAKE ONE	No two transitions with overlapping arenas can be taken in a same combo step.	(+) Terminating combo steps (+) Unclear, non-syntactical scope for combo steps	RSML [30] and Statemate [19]
COMBO TAKE MANY	No constraint on transitions that can be taken in a combo step.	(+) Sequential *Or* transitions in a combo step (-) Unclear, non-syntactical scope for combo steps (-) Non-terminating combo steps	N/A

the environmental input event $\mathrm{swap\_twice}$*, the model executes transitions* t_1 *and* t_2*, at which point the first combo step concludes. The second combo step starts by first considering the effects of the transitions of the first combo step, i.e., the effect of swapping the values of* a *and* b *and the effect of generating events* $\mathrm{swap\_a}$ *and* $\mathrm{swap\_b}$*, and then executing transitions* t_3 *and* t_4*. At the end of the second combo step the big step concludes and the values of* a *and* b *are the same as their values at the beginning of the big step. If the effect of the assignments of the transitions are not hidden from one another during a combo step, the correct behaviour cannot be achieved. For example, depending on whether* t_1 *or* t_2 *is executed first, both* a *and* b *are assigned the initial value of* b *or* a*, respectively.*[11]

Example 20 *The model in Figure 22 shows a simple model of a system that controls the operation of a chemical plant.*[12] *The operation of the plant relies on two chemical substances* A *and* B*. There are two processes, shown as two Or control states* $\mathrm{Process\_1}$ *and* $\mathrm{Process\_2}$*, which can independently increase the amounts of substances* A *and* B *by one unit or two units, respectively. The two processes may simultaneously request for an increase; i.e., environmental input events* $\mathrm{inc\_one}$ *and* $\mathrm{inc\_two}$ *might be received at the same big step. Variables* a *and* b *represent the amount of requested increase for substance* A *and substance* B*, respectively. Environmental output event* $\mathrm{start\_process(a, b)}$ *instructs a physical component of the plant to increase the amounts of substance* A *and* B*, by amounts* a *and* b*, respectively. Internal event* $\mathrm{process}$ *is meant to instruct the* $\mathrm{Controller}$ *to increase the amounts of the substances. Environmental input event* $\mathrm{end\_process}$ *signifies that the requested amounts of*

[11] As pointed out by one of our reviewers, choosing the TAKE MANY big-step maximality semantics, the MANY concurrency semantics, the PRESENT IN NEXT COMBO STEP event lifeline semantics (or the PRESENT IN REMAINDER event lifeline semantics), and the RHS SMALL STEP assignment memory protocol, also yields the correct behaviour. While such an equivalence of behaviours holds for some models, it does not always hold. For example, if there is a possibility for race conditions (e.g., in Example 20) or if it is important whether a model can reach certain configuration of control states or not, then it is not possible to replace the SINGLE concurrency semantics with the MANY concurrency semantics.

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

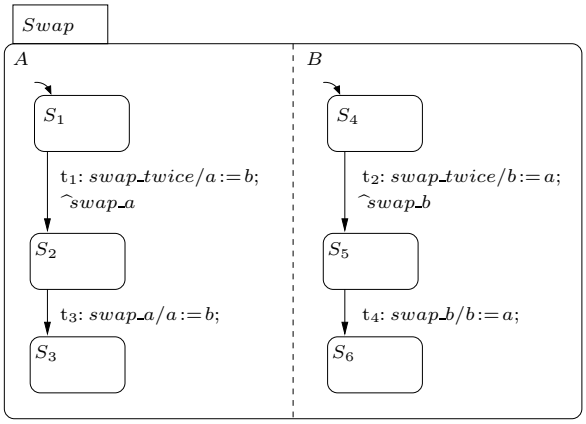

Fig. 21 Swapping a and b twice, using combo steps.

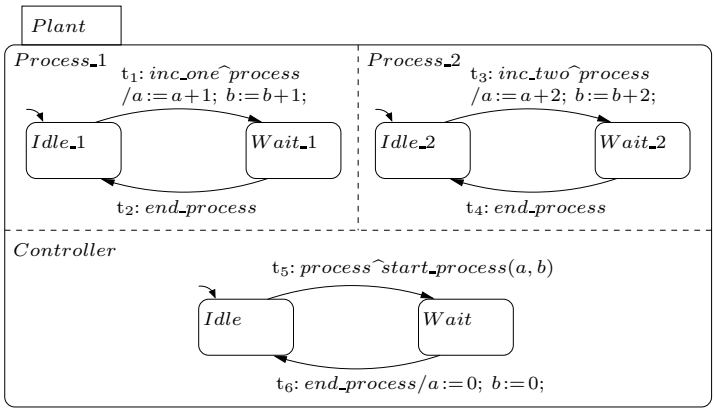

Fig. 22 Controlling the operation of a chemical plant.

the substances have been successfully increased by the physical component of the plant, at which point the system can process new requests.

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

assignment memory protocol, which means that we do not need to choose any semantic option for the combo-step maximality semantic aspect, the additional big step $\langle\{\mathrm{t}_1\}, \{\mathrm{t}_5\}, \{\mathrm{t}_3\}\rangle$ *is possible, which ignores the increase requested by* $\mathrm{Process\_2}$.

Example 21 *In Example 6, we described some possible semantics to make the counter in Example 2 to behave correctly. Another possible semantics is a semantics that subscribes to the* COMBO TAKE ONE *combo-step maximality semantics, the* TAKE ONE *big-step maximality semantics, the* SINGLE *concurrency semantics, and the* PRESENT IN NEXT COMBO STEP *event lifeline semantics.*

Example 22 *Another way to maintain the invariant in Example 13 is to choose the* COMBO TAKE ONE *combo-step maximality semantics, the* TAKE MANY *big-step maximality semantics, and the* RHS COMBO STEP *assignment memory protocol. The execution of the first combo step,* $\{\mathrm{t}_1\}$, $\{\mathrm{t}_3\}$, *results in* $a = 9$ *and* $b = 4$, *and the execution of the second combo step,* $\{\mathrm{t}_2\}$, $\{\mathrm{t}_4\}$, *results in* $a = 27$ *and* $b = 22$. *The order of the execution of* $\{\mathrm{t}_1\}$ *and* $\{\mathrm{t}_3\}$, *and,* $\{\mathrm{t}_2\}$ *and* $\{\mathrm{t}_4\}$, *do not affect the end result. If we choose the* COMBO TAKE MANY *combo-step maximality semantics, then the invariant would be maintained, but the big step concludes with* $a = 21$ *and* $b = 16$*;*

3.9 Summary of Semantics and Notations

In our framework, a BSML is described by, first, describing how its syntax can be translated to our normal-form syntax, and then, enumerating its choice of semantic options. The syntactic translation to our normal-form syntax is straightforward for most BSMLs, as briefly discussed in Section 2.3. In the light of our semantic deconstruction, the specification of the semantics of a BSML is also straightforward. Table 12 shows the specification of the semantics of some of the BSMLs that we have considered throughout the paper. For the sake of brevity, we have not included the External Output Events semantic aspect. Also, we have merged some aspects (e.g., the Enabledness Memory Protocol for Internal Variables in GC merged with Internal Variables in RHS semantic aspects).

4 Semantic Side Effects

In this section, we describe the *side effects* that arise when a group of semantic options are chosen together, and explain ways to avoid them. The choice of a group of semantic options has a "side effect" when it causes a semantic complication that is not due to the original design of any of the semantic options. A side effect can sometimes be tolerated because the benefit of having a set of semantic options in a BSML outweighs their caused complication.

Complicated event lifeline semantics: To achieve an uncomplicated semantics when choosing the PRESENT IN WHOLE event lifeline semantics, it is recommended to choose the TAKE ONE big-step maximality semantics also, as done in Argos [33]. The TAKE ONE semantic option introduces less complication compared to the other big-step maximality semantics, because the status of an event in a big step can be identified by considering at most one transition of each of the non-overlapping arenas of a model. Similarly, it is recommended to choose the TAKE ONE semantic option, when choosing the STRONG SYNCHRONOUS EVENT semantic option for interface events.

Table 12 Example BSMLs and their semantic options. ([21]: Harel statecharts, [42]: Pnueli and Shalev statecharts, [30]: RSML, [19]: Statemate, [6]: Esterel, [33]: Argos, [22]: SCR, and [3]: reactive modules.)

Semantic Aspects	Semantic Options	[21]	[42]	[30]	[19]	[6]	[33]	[22]	[3]
Big-Step Maximality	SYNTACTIC					✓			
	TAKE ONE	✓	✓				✓	✓	✓
	TAKE MANY			✓	✓				
Concurrency	SINGLE	✓	✓	✓	✓			✓	
	MANY					✓	✓		✓
Small-Step Consistency	SOURCE/DESTINATION ORTHOGONAL								
	ARENA ORTHOGONAL					✓	✓		✓
Preemption	NON-PREEMPTIVE					✓	✓		
	PREEMPTIVE								
(Internal) Event Lifeline	PRESENT IN WHOLE					✓	✓		
	PRESENT IN REMAINDER	✓	✓						
	PRESENT IN NEXT COMBO STEP			✓	✓				
	PRESENT IN NEXT SMALL STEP								
	PRESENT IN SAME								
Environmental Input Events	SYNTACTIC INPUT EVENTS			✓		✓	✓		
	RECEIVED EVENTS AS ENVIRONMENTAL	✓	✓		✓				
	HYBRID INPUT EVENT								
(Interface) Event Lifeline	STRONG SYNCHRONOUS EVENT								
	WEAK SYNCHRONOUS EVENT								
	ASYNCHRONOUS EVENT					✓			
(Internal Variables) Enabledness Memory Protocol	GC/RHS BIG STEP							✓	✓
	GC/RHS COMBO STEP				✓				
	GC/RHS SMALL STEP		✓	✓		✓		✓	
(Interface Variables) Memory Protocol	GC/RHS STRONG SYNCHRONOUS VARIABLE								✓
	GC/RHS WEAK SYNCHRONOUS VARIABLE								
	GC/RHS ASYNCHRONOUS VARIABLE								
Combo-Step Maximality	COMBO SYNTACTIC								
	COMBO TAKE ONE			✓	✓				
	COMBO TAKE MANY								
Order of Small Steps	NONE	✓		✓	✓	✓	✓		
	EXPLICIT ORDERING								
	DATAFLOW		✓					✓	✓
Priority	HIERARCHICAL				✓				
	EXPLICIT PRIORITY								
	NEGATION OF TRIGGERS		✓	✓	✓	✓	✓	✓	✓

Cyclic evaluation orders: To avoid a "cyclic evaluation order" when using the **new_small** operator, as described in Section 3.5, a conservative well-formedness criterion can disallow small steps whose assignments create cyclic evaluation orders. Such a well-formedness criteria depends on the choice of the semantic options for the Small-Step Consistency and Preemption semantic aspects. For example, consider a BSML that subscribes to the ARENA ORTHOGONAL small-step consistency semantics and the PREEMPTIVE preemption semantics. For such a semantics, a conservative well-formedness condition to avoid a cyclic evaluation order is to require that, for a pair of orthogonal control states S_1 and S_2, if the arena of t is S_1, or a descendent of S_1, and t uses `new_small`(u) in the RHS of its assignment a_1 and assigns a value to variable v in assignment a_2, then there is no t' whose arena is S_2, or a descendent of S_2, and uses `new_small`(v) in the RHS of its assignment a_1', together with assigning a value to u in its assignment a_2'.

Ambiguous dataflow: An ambiguity arises for a dataflow order if a variable is prefixed by the **new** operator but it is assigned values more than once during a big step. A sufficient, but not necessary, condition for an unambiguous DATAFLOW order of small-steps is to require the TAKE ONE big-step maximality semantics with each variable assigned value only by the transitions that have the same arena, as is done in SCR [22, 23] and reactive modules [3]. Similarly, the TAKE ONE semantic option can be chosen together with the GC STRONG SYNCHRONOUS VARIABLE or the RHS STRONG SYNCHRONOUS VARIABLE semantic options for interface variables, to avoid ambiguity in obtaining the value of an interface variable.

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

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

Inconsistent preemption and priority semantics: When the PREEMPTIVE preemption semantics is chosen, the choice of the priority semantics determines whether the interrupt transition has higher or lower priority than non-interrupt transitions. For example, giving the highest priority to a transition whose destination control state is the lowest in the composition tree, i.e., the choice of the DESTINATION-CHILD semantics, has the effect of giving interrupt transition t in Figure 11(b) a higher priority than t', which is an intuitive, desired behaviour. Similarly, the ARENA-PARENT priority semantics gives transition t in Figure 11(a) a higher priority than transition t'.

Conflicting maximality: The choice of the SYNTACTIC semantic option for the big-step maximality semantics together with the choice of the COMBO SYNTACTIC semantic option for the combo-step maximality semantic aspect means that a small step may move a model

to a snapshot where the model resides in a pair of orthogonal control states, one being a **Stable** control state and the other a **Combo Stable** control state. In such a snapshot, it is unclear whether the current combo step has concluded, or not. Alternatively, choosing the TAKE MANY semantic option for the big-step maximality semantic aspect and the COMBO SYNTACTIC semantic option for the combo-step maximality semantic aspect avoids this problem.

5 Related Work

We cover a more comprehensive class of BSMLs and range of BSML semantics than found in related work. Relative to previous comparative studies of different subsets of BSMLs (e.g., statecharts variants [49, 26], Synchronous languages [16], Esterel variants [7, 47], and UML StateMachines [46]), we isolate the essential semantic aspects in a language-independent manner and in terms of the big step as a whole. Huizing and Gerth [26] compare simple BSMLs that have only events, covering most of the event lifeline semantic options and the observability of events among components. In our deconstruction, we are able to describe these options more concisely and place them in the context of other semantics aspects for BSMLs.

By considering a big step as a whole, we have raised the level of abstraction of the semantic variations compared to our previous work on template semantics [37]. The composition operators of template semantics are modelled via our concurrency and consistency, and event lifeline semantic aspects. For example, the *interleaving* and *parallel* composition operators correspond to the SINGLE and MANY semantic options, respectively; and the *rendezvous* composition operator is represented via the PRESENT IN SAME event lifeline semantics and the MANY concurrency semantics. The *interrupt* composition operator is modelled via the small-step consistency and preemption semantic options. By relating parts of the behaviour of composition operators to the step semantic aspects, we provide a foundation for understanding the range of possible composition operators.

6 Conclusion and Future Work

We have presented a novel deconstruction of the semantics of big-step modelling languages into eight high-level, mostly orthogonal semantic aspects. We analyzed the relative advantages and disadvantages of the characteristics of the semantic options of each aspect. The design/choice of a language involves making tradeoffs between different options. Using our aspects, options, as well as the taxonomy of the syntactic constructs of BSMLs, represented conveniently by two feature diagrams and a set of dependencies between their features, our framework empowers requirements engineers and language designers to make such tradeoffs in an informed way. For example, if averting non-determinism is desirable, semantics that permit race conditions, unordered execution of small steps, SINGLE concurrency, non-prioritized transitions, etc. are less suitable choices. SCR [22, 23] is an example of a BSML with simpler semantics than many others because its lack of hierarchical control states means it does not require the semantic aspects of small-step consistency, preemption, and priority.

Our analysis of the side effects between semantic options allows a requirements engineer to identify the difficulties that may arise in certain combinations of semantic features. For example, the semantics in Example 15, which avoids the undesired non-determinism of the SINGLE concurrency semantics, is not found in an existing BSML. However, a user of

this semantics is warned about the “complicated explicit ordering” side effect described in Section 4.

We have devised a parametric semantic definition schema that formalizes a large subset of the BSML semantics that arise from our deconstruction, while preserving its structure [12]. We believe our work forms a basis for identifying and formally proving semantic properties of a set of semantic options when considered together, as opposed to when considered in isolation, as we described in this paper. Such properties would provide the requirements engineer with a better sense of what are “good” or “risky” combinations of semantic choices to produce a simple, elegant model for a system under study.

In the future, we plan to create tool suites based on the formal semantics of BSMLs to support the analysis of BSML models. We believe that our work can be used to study how semantic choices affect the simplicity and performance of analysis tools.

Acknowledgements

We thank the reviewers for their insightful, detailed comments, which have improved our paper. We also thank the anonymous reviewers of our RE’09 paper, as well as Patrick Heymans, for their helpful comments that have influenced this paper.

References

1. The Esterel v7 reference manual version v7.30, initial IEEE standardization proposal. 2005.
2. Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of message sequence charts. *IEEE Transactions on Software Engineering*, 29(7):623–633, 2003.
3. Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
4. Gérard Berry. A hardware implementation of pure ESTEREL. *Sadhana, Academy Proceedings in Engineering Sciences, Indian Academy of Sciences*, 17(1):95–130, 1992.
5. Gerard Berry. Preemption in concurrent systems. In *Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *LNCS*, pages 72–93. Springer, 1993.
6. Gérard Berry and Georges Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.
7. Frederic Boussinot. Sugarcubes implementation of causality. Technical Report RR-3487, Inria, Institut National de Recherche en Informatique et en Automatique, 1998.
8. Daniel M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, 1984.
9. James Dabney and Thomas L. Harman. *Mastering Simulink*. Pearson Prentice Hall, 2004.
10. Nancy Day. A model checker for statecharts: Linking CASE tools with formal methods. Master’s thesis, University of British Columbia, 1993.
11. Luca de Alfaro and Thomas A. Henzinger. Interface Automata. In *Proceedings of the Joint 8th European Software Engeneering Conference and 9th ACM SIGSOFT Symposium on the Foundation of Software Engeneering (ESEC/FSE-01)*, volume 26 of *Software Engineering Notes*, pages 109–120. ACM Press, 2001.

12. Shahram Esmaeilsabzali and Nancy A. Day. Prescriptive semantics for big-step modelling languages. In *Fundamental Approaches to Software Engineering (to appear)*, volume 6013 of *LNCS*, pages 158–172. Springer Verlag, 2010.
13. Shahram Esmaeilsabzali, Nancy A. Day, Joanne M. Atlee, and Jianwei Niu. Big-step semantics. Technical Report CS-2009-05, University of Waterloo, Cheriton School of Computer Science, 2009.
14. Shahram Esmaeilsabzali, Nancy A. Day, Joanne M. Atlee, and Jianwei Niu. Semantic criteria for choosing a language for big-step models. In *17th IEEE International Requirements Engineering Conference*, pages 181–190. IEEE Computer Society Press, 2009.
15. Colin Fidge. A comparative introduction to CSP, CCS and LOTOS. Technical Report 93-24, The university of Queensland, Department of Computer Science, 1994.
16. Nicolas Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers, 1993.
17. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.
18. David Harel and Hillel Kugler. The RHAPSODY semantics of statecharts (or, on the executable core of the UML). In *Integration of Software Specification Techniques for Application in Engineering*, volume 3147 of *LNCS*, pages 325–354. Springer Verlag, 2004.
19. David Harel and Amnon Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.
20. David Harel and Amir Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*. Springer Verlag, 1985.
21. David Harel, Amir Pnueli, J. P. Schmidt, and R. Sherman. On the formal semantics of statecharts. In *IEEE Symposium on Logic in Computation*, pages 54–64, 1987.
22. Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, 1996.
23. Kathryn L. Heninger, John W. Kallander, David L. Parnas, and John E. Shore. Software requirements for the A-7E aircraft. Technical Report 3876, United States Naval Research Laboratory, 1978.
24. Tony Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
25. Tony Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice Hall, 1998.
26. Cornelis Huizing and Rob Gerth. Semantics of reactive systems in abstract time. In *REX Workshop*, volume 600 of *LNCS*, pages 291–314. Springer Verlag, 1992.
27. i Logix Inc. *Statemate 4.0 Analyzer User and Reference Manual*, 1991.
28. Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, SEI, Carnegie Mellon University, 1990.
29. Leslie Lamport and Fred B. Schneider. Pretending atomicity. Technical Report 44, Digital Equipment Corporation, 1989.
30. Nancy G. Leveson, Mats P. E. Heimdahl, Holly Hildreth, and Jon D. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, 20(9):684–707, 1994.
31. Robert J. Lipton. Reduction: A method of proving properties of parallel programs. *Communications of the ACM*, 18:717–721, 1975.
32. Andrea Maggiolo-Schettini, Adriano Peron, and Simone Tini. Equivalences of statecharts. In *International Concurrency Theory Conference*, volume 1119 of *LNCS*, pages

687–702. Springer Verlag, 1996.
33. Florence Maraninchi and Yann Rémond. Argos: an automaton-based synchronous language. *Computer Languages*, 27(1/3):61–92, 2001.
34. Kenneth McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic, 1993.
35. Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3):267–310, 1983. Fundamental study.
36. Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.
37. Jianwei Niu, Joanne M. Atlee, and Nancy A. Day. Template semantics for model-based notations. *IEEE Transactions on Software Engineering*, 29(10):866–882, 2003.
38. OMG. OMG Unified Modeling Language (OMG UML), Superstructure, v2.1.2. 2007. Formal/2007-11-01.
39. David L. Parnas and Jan Madey. Functional documents for computer systems. *Science of Computer Programming*, 25(1):19–23, 1995.
40. Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
41. Amir Pnueli and M. Shalev. What is in a step? In *J.W. De Bakker, Liber Amicorum*, pages 373–400. CWI, 1989.
42. Amir Pnueli and Michal Shalev. What is in a step: On the semantics of statecharts. In *Theoretical Aspects of Computer Software*, volume 526 of *LNCS*, pages 244–264. Springer Verlag, 1991.
43. Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Models for concurrency: Towards a classification. *Theoretical Computer Science*, 170(1–2):297–348, 1996.
44. Sandeep K. Shukla and Michael Theobald. Special issue on formal methods for globally asynchronous and locally synchronous (GALS) systems. *Formal Methods in System Design*, 28(2):91–92, 2006.
45. Signe J. Silver and Janusz A. Brzozowski. True concurrency in models of asynchronous circuit behavior. *Formal Methods in System Design*, 22(3):183–203, 2003.
46. Ali Taleghani and Joanne M. Atlee. Semantic variations among UML StateMachines. In *Model Driven Engineering Languages and Systems, 9th International Conference*, volume 4199 of *LNCS*, pages 245–259. Springer, 2006.
47. Olivier Tardieu. A deterministic logical semantics for pure Esterel. *ACM Transactions on Programming Languages and Systems*, 29(2):8:1–8:26, 2007.
48. Robert van Glabbeek. The linear time – branching time spectrum. In *International Concurrency Theory Conference*, volume 458 of *LNCS*.
49. Michael von der Beeck. A comparison of statecharts variants. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *LNCS*, pages 128–148. Springer, 1994.
50. Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, 1997.