PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 10-13

Behavioral Refinements of UML-Statecharts

Nora Szasz Pedro Vilanova

2010

Behavioral Refinements of UML-Statecharts
Szasz, Nora; Vilanova, Pedro
ISSN 0797-6410
Reporte Técnico RT 10-13
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, 2010

Behavioral Refinements of UML-Statecharts

Nora Szasz
Universidad ORT Uruguay
szasz@ort.edu.uy

Pedro Vilanova
KAUST, AMCS, Saudi Arabia
pedro.guerra@kaust.edu.sa

September 2010

Abstract

In [SV08] we have proposed a formalism for specifying the behavior of software product lines using UML-statecharts. Using an order relation among statecharts which represents when a statechart has a more complex structure than another one, we define how to combine diferent extensions of the same statechart into an integral new one. In this paper, we prove that the proposed extension constitutes a behavioral refinement. That is, whenever we allow the possibility of extending a given statechart, the old behaviour is preserved and new behavior may be added.

Resumen

En [SV08] proponemos un formalismo para especificar el comportamiento de lneas de productos de software utilizando UML-statecharts. Mediante una relación de orden entre statecharts que representa cuando un statechart tiene una estructura ms compleja que otro, se define la forma de combinar diferentes extensiones de un mismo statechart en uno nuevo. En este trabajo probamos que la extensión propuesta constituye un refinamiento de comportamiento. Es decir, cada vez que se extiende un statechart, se preserva el comportamiento anterior, con la posibilidad de a;adir nuevo comportamiento.

Keywords: Statecharts, UML, Formal semantics, Software product lines, Variability, Behavioral refinement.

Contents

1 Introduction

The complexity of software systems has generated the need of founding developments on abstract models. Modeling allows, among other things, to verify the systems before their construction and to guide their development by means of techniques such as automatic code generation. In particular, UML-statecharts and interactions are specially meant for software design. Statecharts, originally introduced by Harel [Har87], are used to specify the behavior of class instances (intra-component behaviour). They are compact, expressive and allow describing from simple systems to complex reactive ones.
On the other hand, software reusability has become a major challenge for the software industry. Reusable artifacts increase the productivity by reducing development time. The development of products that vary in more or less peripheral aspects has given rise to the concept of software product lines. A software product line consists of a family of systems that share functionalities and satisfy, in general, the needs of a particular market segment [Gom05, CN02].
In [SV08] we propose a way for specifying the behavior of software product lines using UML-statecharts (statecharts from now on). We use feature diagrams to represent the common and variant functionalities of a family of products, presenting a formal syntax for feature diagrams and configurations based on Czarnecki's work [Cza98, CHE05]. Then, based on von der Beeck's [vdB02] statecharts abstract syntax, we define an order relation for statecharts that represents when a statechart has a more complex structure than another one. Then, we define how to combine different extensions of the same statechart into an integral new statechart. With these notions, given the description of a product line as a feature diagram, we define a statechart with variabilities for the line as a function that associates each feature of the feature diagram with a statechart. The mapping must comply with the hierarchical structure and the feature restrictions, i.e., the more features a product has, the richer the statechart that models it must be. In this way, we can describe the effect that each feature has on the products in which it is present.This definition provides a very simple way to obtain the specification of the behavior of any configuration of the product line as the combination of the statecharts that implement all the features present in the product.
Continuing with that work, in this paper we prove that the structured operational semantics proposed in [vdB02] can be extended using our extension relation without loosing any behaviour. That is, when a statechart is extended, it is still possible to perform the same semantic transitions on it as before. Then, the extension relation can be considered as a behavioral refinement, in the sense that it preserves the same behaviour as before, but adding some new functionality.
Although there exist several extensions of UML models for specification of variability (among other works, [CGW05, Gom05, ZHJ04]), we have not found any formal specification of variabilities for statecharts, with the exception of [GL10]. In that work, the authors define functions that associate statechart components (not statecharts, as our case) to functionalities of a feature diagram. Then, the behavior of a product line is obtained basically by a selection process, and not a combination of statecharts, as in our proposal.
The rest of the article is structured as follows: In section 2 we present the main concepts and definitions for UML-statecharts syntax, following [vdB02]. In section 3 we define an extension relation between statecharts to enrich their structure, and we show how to combine diferent extensions into an integral new statechart. In section 4 we present the formal semantics of statecharts. In section 5 we prove the main result of this paper: the extension relation can be considered as a refinement, in the sense that it preserves the transitions defined in the semantics. Finally, in section 6 we also take into account the actions generated by the transitions, and prove that these actions are preserved by the refinement. We conclude and discuss further work in section 7.

2 Feature Diagrams and UML-Statecharts Syntax

In this section we present the main concepts and definitions for statecharts and feature diagrams, which we will use in the rest of this paper.

2.1 Feature Diagrams

Feature diagrams are used to document features. A feature is a property of a system that directly affects end users. Czarnecki [Cza98, CHE05] proposes three types of features, namely *mandatory*, *optional* and *alternative*. Additionally, a set of constraints over features can be defined. A constraint is a proposition over the set of features. Formally, given $\mathcal{F}$ a set of feature names, we define a feature diagram as a 6-tuple $\Upsilon = \langle \mathsf{L},\mathsf{N},\mathsf{N_C},\mathsf{R_M},\mathsf{R_O},\mathsf{R_A}\rangle$, where: $\mathsf{L} \in \mathcal{F}$, is the product line name (the root of the tree); $\mathsf{N} \subseteq \mathcal{F}$ is the set of features, ($\mathsf{L} \notin \mathsf{N}$); $\mathsf{N_C} \subseteq \mathcal{C}$ is the set of constraints over features, where $\mathcal{C}$ are the propositional calculus formulas with variables $f_i \in \mathcal{F}$ and connectives $\wedge$, $\vee$ and $\neg$; $\mathsf{R_M}, \mathsf{R_O} \subseteq \{\mathsf{L}\}\cup\mathsf{N} \times \mathsf{N}$, are the mandatory and optional *consists of* relations respectively; and $\mathsf{R_A} \subseteq \{\mathsf{L}\}\cup\mathsf{N} \times \mathcal{P}(\mathsf{N})$ is the alternative *consists of* relation. In addition, the union of the relations $\mathsf{R_M}$, $\mathsf{R_O}$ and $\mathsf{R_A}$ must constitute a tree with nodes in $\mathcal{F}$ and root L. The feature diagram must satisfy that there are no cycles.

In order to obtain specific products of a line defined by a feature diagram Υ, we define the possible configurations of Υ as the instances of the tree that are consistent with the relations amongst its features and the constraints of Υ. Formally, given a set of feature names $\mathcal{F}$, a configuration is a 3-tuple $\Phi = \langle \mathsf{P},\mathcal{Y},\mathsf{R}\rangle$, where: $\mathsf{P} \in \mathcal{F}$, is the product name (the root of the tree); $\mathcal{Y} \subseteq \mathcal{F}$, is the set of features of the product ($\mathsf{P} \notin \mathcal{Y}$); and $\mathsf{R} \subseteq \{\mathsf{P}\}\cup\mathcal{Y}\times\mathcal{Y}$ is the *consists of* relation. Additionally, configurations must be trees under the relation R, with root P.

2.2 States and Transitions

In this section we present the main concepts and definitions for statecharts, following [vdB02], which we will use in the rest of this paper.
Let $\mathcal{S}$, $\mathcal{T}$, A, E be countable sets of state names, transition names, actions and events respectively with $\mathsf{E} \subseteq \mathsf{A}$. We denote events and actions by $a, b, c, ..$ and sequences of events as well as actions by $\alpha, \beta, \gamma, ...$ Then the set SC of statecharts is inductively defined by the rules in figure 1, together with the functions name: $\mathsf{SC} \rightarrow \mathcal{S}$ [1], that is, the name of the statechart, and the predicate $\mathsf{wf\text{-}tran}$ (defined below), which decides if a transition is well formed with respect to a set of states. The rules are explained as follows:

Basic Statecharts: $s = [\hat{s},(en,ex)]$ is a basic statechart with name $\hat{s}$, entry sequence of actions en, exit sequence of actions ex and $\mathsf{type}(s) = \mathsf{basic}$.

And-Statecharts: $s = [\hat{s},(s_1,..,s_n),(en,ex)]$ is an and-statechart with name $\hat{s}$, parallel substates $s_1, .., s_n$, $n > 0$, entry sequence of actions en, exit sequence of actions ex and $\mathsf{type}(s) = \mathsf{and}$.

Or-Statecharts: $s = [\hat{s},(s_1,..,s_n),l,T,(en,ex)]$ is an or-statechart with name $\hat{s}$, parallel substates $s_1, .., s_n$, $n > 0$, entry sequence of actions en, exit sequence of actions ex, $l \in \rho$, $\rho = \{1, .., n\}$ is the index of the current active substate of s, T is the set of transitions between its substates, s_1 is the default state of s, $\mathsf{type}(s)$=or. The set of transitions T is included in $\mathsf{T} := \mathcal{T}\times\rho \times \mathcal{P}(\mathcal{S})\times\mathsf{E}\times\mathsf{A}^*\times\mathcal{P}(\mathcal{S})\times\rho\times\mathsf{HT}$, where $\mathsf{HT} = \{\mathsf{none}, \mathsf{shallow}, \mathsf{deep}\}$ are the history types.

[1]Abusing the notation, we will abbreviate $\mathsf{name}(s)$ with $\hat{s}$. The same applies to transitions.

$$\frac{\hat{s} \in \mathcal{S} \quad en, ex \in \mathsf{A}^*}{[\hat{s},(en,ex)] \in \mathsf{SC}}\ \mathsf{Basic} \qquad \frac{\begin{array}{l} s_1, .., s_n \in \mathsf{SC},\ \hat{s}_i \neq \hat{s}_j\ \forall_{i \neq j} \\ \hat{s} \in \mathcal{S},\ \hat{s} \neq \hat{s}_i\ \forall_{i=1..n} \\ en, ex \in \mathsf{A}^* \end{array}}{[\hat{s},(s_1,..,s_n),(en,ex)] \in \mathsf{SC}}\ \mathsf{And}$$

$$\frac{\begin{array}{l} s_1, .., s_n \in \mathsf{SC},\ \hat{s}_i \neq \hat{s}_j\ \forall_{i \neq j} \\ \hat{s} \in \mathcal{S},\ \hat{s} \neq \hat{s}_i\ \forall_{i=1..n} \\ T \subseteq \mathsf{T},\ \mathsf{wf\text{-}tran}(\{s_1, .., s_n\}, t)\ \forall t \in T \\ l \in \{1, .., n\} \\ en, ex \in \mathsf{A}^* \end{array}}{[\hat{s},(s_1,..,s_n),l,T,(en,ex)] \in \mathsf{SC}}\ \mathsf{Or}$$

Figure 1: Syntax of Statecharts

We further define:
$SC_B=\{s\in\mathsf{SC} \mid \mathsf{type}(s)=\mathsf{basic}\}$, $SC_A=\{s\in\mathsf{SC} \mid \mathsf{type}(s)=\mathsf{and}\}$, $SC_O = \{s\in\mathsf{SC} \mid \mathsf{type}(s)=\mathsf{or}\}$.
Given $t = \langle\hat{t},i,S_r,e,\alpha,T_d,j,ht\rangle \in \mathsf{T}$, the following are defined: $\mathsf{name}(t) := \hat{t}$, is the name of the transition t, $\mathsf{sou}(t) := s_i$, $\mathsf{tar}(t) := s_j$, are the source and target states of t respectively, $\mathsf{sou\text{-}r}(t) := S_r$, is the source restriction set, $\mathsf{ev}(t) := e$, is the triggering event of t, $\mathsf{act}(t) := \alpha$, is the action associated to t, $\mathsf{tar\text{-}d}(t) := T_d$, is the target determinator set, $\mathsf{historyType}(t) := ht$, is the history type of t. We say a transition t uses the history mechanism, if $\mathsf{historyType}(t) \in \{\mathsf{deep}, \mathsf{shallow}\}$ (see [vdB02] for more details).
Moreover, we define the predicate $\mathsf{wf\text{-}tran}$: $\mathcal{P}(\mathsf{SC}) \times \mathsf{T}$ which decides if a transition t is well formed with respect to a set of states $s_1, .., s_n$ as $\mathsf{wf\text{-}tran}(\{s_1..s_n\}, t) = (\mathsf{sou}(t), \mathsf{tar}(t) \in \{s_1.., s_n\}) \wedge (\mathsf{sou\text{-}r}(t) \in \mathsf{conf\text{-}all}(\mathsf{sou}(t))) \wedge (\mathsf{tar\text{-}d}(t) \in \mathsf{conf\text{-}all}(\mathsf{tar}(t)))$.
Note that the definition of SC implies that, within a statechart, each substate can be uniquely referred to by its name. This is an important fact regarding configurations, which is the topic of the next section. From now on, when there is no risk of ambiguity, when we denote a state we only show the syntactic elements that are relevant in each case. Moreover, we will abbreviate $s_1, s_2, .., s_k$ with $s_{1..k}$.

2.3 Statechart Configurations

The function conf: $\mathsf{SC} \to \mathcal{P}(\mathcal{S})$ gives the current configuration of a statechart s, i.e. the set of the names of all currently active substates within s (also including $\hat{s}$):

$$\begin{array}{ll} \mathsf{conf}([\hat{s}]) & := \{\hat{s}\} \\ \mathsf{conf}([\hat{s}, (s_{1..n}), l, T]) & := \{\hat{s}\} \cup \mathsf{conf}(s_l) \\ \mathsf{conf}([\hat{s}, (s_{1..n})]) & := \{\hat{s}\} \cup \bigcup_{i=1..n} \mathsf{conf}(s_i) \end{array}$$

The function $\mathsf{conf\text{-}all}$: $\mathsf{SC} \to \mathcal{P}(\mathcal{P}(\mathcal{S}))$ computes the set of all potential configurations of a statechart s (complete or incomplete). By potential we mean that we consider not only the current active substate of an or-substate, but all possibilities for choosing such a substate. Note the difference with the definition of conf. The term incomplete denotes a configuration which results from an application of $\mathsf{conf\text{-}all}$ where the recursion terminates before the basic states are reached. This definition is crucial to handle inter-level transitions just like normal transitions on the level of the uppermost states that the inter-level transition exits and enters (see [vdB02] for more details).

$$\begin{array}{ll}
\mathsf{conf\text{-}all}([\hat{s}]) & := \{\{\hat{s}\}\} \\
\mathsf{conf\text{-}all}([\hat{s}, (s_{1..n}), T]) & := \{\{\hat{s}\} \cup c | \exists i \in \{1..n\}.c \in \mathsf{conf\text{-}all}(s_i)\} \cup \{\{\hat{s}\}\} \\
\mathsf{conf\text{-}all}([\hat{s}, (s_{1..n})]) & := \{\{\hat{s}\} \cup \bigcup_{i=1..n} c_i | c_i \in \mathsf{conf\text{-}all}(s_i)\} \cup \{\{\hat{s}\}\}
\end{array}$$

Incomplete configurations are realized in the second and third cases of the definition by the union with the term $\{\{\hat{s}\}\}$. Note that $\forall s \in \mathsf{SC} : \mathsf{conf}(s) \in \mathsf{conf\text{-}all}(s)$.

3 Statecharts with Variabilities

In [SV08] we propose a way for specifying the behavior of software product lines using statecharts, feature diagrams and an order relation between statecharts that represents when a statechart has a more complex structure than another one. We then define how to combine different extensions of the same statechart into an integral new statechart, and use these notions to obtain the specification of the behavior of any configuration of a product line as the combination of the statecharts that implement all the features present in the product.
In this section we summarize the main ideas and results of [SV08]

3.1 Extension Relation

In [SV08] we define a relation $\succ$ between statecharts, such that $s_1 \succ s_2$ (read "s_2 extends s_1") if s_2 enriches states or transitions of s_1 with more complex structures. Basically, we can extend a statechart by either adding a parallel or sequential statechart to it, adding a new transition between two existing states, or adding actions in transitions or entry and exit actions. For this last extension we define the relation $\triangleright$ between sequences of actions as the ordinary subsequence relation between elements of A^*. In this case $\alpha \triangleright \alpha'$ means that α is a subsequence of α'. Here we reformulate de definition of the extension relation by defining first the one-step extension relation $\succ_1$ in figure 2, and then defining $\succ$ as its reflexo-transitive closure in figure 3. In the definitions we assume that the well-formedness conditions of the definitions given in section 2 hold whenever we build a statechart. Furthermore, we use the substitution notation as follows: If t is a term, then $t_{[a/b]}$ is the term which results from replacing all occurences of a in t by b. In particular, note that $(s_{1..k}, \tilde{s})_{[s_t/s'_t]} = (s_{1..k[s_t/s'_t]}, \tilde{s})$ if $t \in \{1..k\}$ and $s_{1..k}, \tilde{s} \in \mathsf{SC}$.

3.2 Intersection

Given a statechart, it can be refined in several ways. The question is whether there is a way to combine different extensions into an integral new statechart. Formally, given $r_1, r_2 \in \mathsf{SC}$ such that $\exists s{\in}\mathsf{SC}.\ s \succ r_1 \wedge s \succ r_2$ we want to define a new statechart $r_1 \Cap r_2$ such that $r_1 \succ r_1 \Cap r_2$ and $r_2 \succ r_1 \Cap r_2$.
The definition of $\Cap$ is simple but quite extensive, and it basically consists of carrying out both extensions on the original statechart, whenever this is possible. For further details, the reader is referred to [SV08].

The following properties of the intersection hold:
1) For all $s, r_1, r_2 \in \mathsf{SC}$ such that $s \succ r_1$ and $s \succ r_2$, $r_1 \succ r_1 \Cap r_2$ and $r_2 \succ r_1 \Cap r_2$.
2) For all $s, r_1, r_2, r_3 \in \mathsf{SC}$ such that $s \succ r_1$ and $s \succ r_2$, if $(r_1 \succ r_3 \wedge r_2 \succ r_3)$, then $r_1 \Cap r_2 \succ r_3$.
3) For all $s, r_1, r_2, r_3 \in \mathsf{SC}$ such that $s \succ r_1$, $s \succ r_2$ and $s \succ r_3$, $(r_1 \Cap r_2) \Cap r_3 = r_1 \Cap (r_2 \Cap r_3)$.

Parallel components

$$\frac{s_1,..,s_n \in SC_B,\quad \hat{s}_i \neq \hat{s}_j\ \forall_{i\neq j},\quad \hat{s}\neq \hat{s}_i\ \forall_i}{[\hat{s}] \succ_1 [\hat{s},(s_1,..,s_n)]}\ \textsf{ext-and}$$

$$\frac{s_i \succ_1 s'_i,\quad \hat{s}'_i \neq \hat{s}_j\ \forall_{j=1..i-1,i+1..n}}{[\hat{s},(s_1,..,s_n)] \succ_1 [\hat{s},(s_1,..,s_n)_{[s_i/s'_i]}]}\ \textsf{inside-and}$$

Sequential components

$$\frac{s' \in SC_B,\quad \hat{s}' \neq \hat{s}}{[\hat{s}] \succ_1 [\hat{s},(s'),1,\emptyset]}\ \textsf{ext-or1} \qquad \frac{s' \in SC_B,\quad \hat{s}_i \neq \hat{s}'\ \forall_i,\quad \hat{s}' \neq \hat{s}}{[\hat{s},(s_1,..,s_n),T] \succ_1 [\hat{s},(s_1,..,s_n,s'),T]}\ \textsf{ext-or2}$$

$$\frac{s_i \succ_1 s'_i,\quad \hat{s}'_i \neq \hat{s}_j\ \forall_{j=1..i-1,i+1,..n}}{[\hat{s},(s_1,..,s_n),l,T] \succ_1 [\hat{s},(s_1,..,s_n)_{[s_i/s'_i]},l,T]}\ \textsf{inside-or}$$

Transitions

$$\frac{t \in \mathsf{T},\quad \textsf{wf-tran}(\{s_1,..,s_n\},t)}{[\hat{s},(s_1,...,s_n),l,T] \succ_1 [\hat{s},(s_1,...,s_n),l,T \cup \{t\}]}\ \textsf{add-trans}$$

Actions

$$\frac{\alpha \triangleright \tilde{\alpha},\quad t = \langle \hat{t},i,S_r,e,\alpha,T_d,j,ht\rangle \in T,\quad t' \in \mathsf{T}}{[\hat{s},(s_1,..,s_n),T] \succ_1 [\hat{s},(s_1,..,s_n),T_{[t/t']}]}\ \textsf{ext-act-trans}$$

$$\text{where } t' = \langle \hat{t},i,S_r,e,\tilde{\alpha},T_d,j,ht\rangle$$

$$\frac{en \triangleright \tilde{en}}{[\hat{s},(en,ex)] \succ_1 [\hat{s},(\tilde{en},ex)]}\ \textsf{ext-act-en} \qquad \frac{ex \triangleright \tilde{ex}}{[\hat{s},(en,ex)] \succ_1 [\hat{s},(en,\tilde{ex})]}\ \textsf{ext-act-ex}$$

Figure 2: One-Step Extension Relation $\succ_1$

$$\frac{s \succ_1 s'}{s \succ s'}\ \textsf{one-step} \qquad \frac{s \in \mathsf{SC}}{s \succ s}\ \textsf{reflexivity} \qquad \frac{s \succ s',\quad s' \succ s'' \quad s, s', s'' \in \mathsf{SC}}{s \succ s''}\ \textsf{transitivity}$$

Figure 3: Extension Relation $\succ$

3.3 Statecharts with Variabilities

Finally, in order to define the behavior of a whole product line, we must describe the effect that each feature has on the products in which it is present. For this, we introduced the set SC* of statecharts with variabilities:
Given a feature diagram $\Upsilon = \langle \mathsf{L},\mathsf{N},\mathsf{N_C},\mathsf{R_M},\mathsf{R_O},\mathsf{R_A}\rangle \in \mathsf{FD}$, a SC* for Υ is a function Ψ: $\mathsf{N} \cup \{\mathsf{L}\} \rightarrow \mathsf{SC}$ that associates each feature of Υ with a statechart. The mapping must comply with the hier-

archical structure and the feature restrictions, i.e., the more features a product has, the richer the statechart that models it must be. Then, given a configuration $C = \langle \mathsf{P}, \mathcal{Y}, \mathsf{R} \rangle$ of Υ, in order to obtain the statechart that implements all the features present in $\mathcal{Y}$, we must just take the intersection of all the statecharts in the image of $\mathcal{Y}$ under Ψ (i.e., $\Psi(\mathcal{Y})$).

4 UML-Statecharts Semantics

In this section we present the formal semantics of the UML statecharts. Then, we prove three important lemmas associated with the extension relation which are fundamental for the main result of this paper.

4.1 Computing the Next State

In [vdB02], the function next is defined, which computes the next state after a UML statechart transition t is executed. Given a UML statechart transition t, with target s, history type ht=$\mathsf{historyType}(t)$ and target determinator N=$\mathsf{tarDet}(t)$ the function next: $\mathsf{HT} \times \mathcal{P}(\mathcal{S}) \times \mathsf{SC} \rightarrow \mathsf{SC}$ computes the UML statechart term s'=$\mathsf{next}(ht, N, s)$ which results after the execution of transition t.

$$\begin{array}{lll}
\mathsf{next}(ht, N, [\hat{s}]) & := [\hat{s}] \\
\mathsf{next}(ht, N, [\hat{s}, (s_{1..n}), l, T]) & := \left\{ \begin{array}{l} [\hat{s}, (s_{1..n})_{[s_j/\mathsf{next}(ht,N,s_j)]}, j, T] \\ \quad \text{if } (\exists \nu \in N,\ j \in \{1..n\}).\nu = \hat{s}_j \\ \mathsf{next\text{-}stop}(ht, [\hat{s}, (s_{1..n}), l, T]) \text{ otherwise} \end{array} \right. \\
\mathsf{next}(ht, N, [\hat{s}, (s_{1..n})]) & := [\hat{s}, (\mathsf{next}(ht, N, s_1), .., \mathsf{next}(ht, N, s_n))]
\end{array}$$

The terms s and s' have identical static structure, only the currently active substate information may change. Here it becomes clear why we need the naming convention: If N contains one of the state names of the substates $s_1, ..., s_n$ then the active state index l is replaced with j and the function is recursively applied to s_j. Then the target determinator information is exploited when zooming into the state hierarchy. Otherwise, function next-stop is called which uses the history type information to determine currently active substates of a state:

$$\mathsf{next\text{-}stop}(ht, [\hat{s}, (s_{1..n}), l, T]) := \left\{ \begin{array}{ll} [\hat{s}, (s_{1..n}), l, T] & \text{if } ht = \mathsf{deep} \\ [\hat{s}, (s_{1..n})_{[s_1/\mathsf{def}(s_1)]}, 1, T] & \text{if } ht = \mathsf{none} \\ [\hat{s}, (s_{1..n})_{[s_l/\mathsf{def}(s_l)]}, l, T] & \text{if } ht = \mathsf{shallow} \end{array} \right.$$

The function next-stop uses the function def: $\mathsf{SC} \rightarrow \mathsf{SC}$ which defines for an $s \in SC_O$ that its currently active substate is given by its default substate:

$$\begin{array}{ll}
\mathsf{def}([\hat{s}]) & := [\hat{s}] \\
\mathsf{def}([\hat{s}, (s_{1..n}), l, T]) & := [\hat{s}, (s_{1..n})_{[s_1/\mathsf{def}(s_1)]}, 1, T] \\
\mathsf{def}([\hat{s}, (s_{1..n})]) & := [\hat{s}, (\mathsf{def}(s_1), ..., \mathsf{def}(s_n))]
\end{array}$$

When a UML statechart transition t is taken, a set of actions is executed. In general, if a transition from state s_i to s_j with action part α is taken, then the sequence of actions $ex :: \alpha :: en$ is executed, with $ex \in \mathsf{exit}(s_i)$, $en \in \mathsf{exit}(s_j)$, where the infix operator :: concatenates sequences of actions.

$$\begin{array}{ll}
\mathsf{exit}([\hat{s}, (en, ex)]) & := \{ex\} \\
\mathsf{exit}([\hat{s}, (s_{1..k}), l, T, (en, ex)]) & := \{ex'::ex \mid ex' \in \mathsf{exit}(s_l)\} \\
\mathsf{exit}([\hat{s}, (s_{1..k}), (en, ex)]) & := \{m_1::..::m_k::ex \mid \exists \text{ bijection } p:\{1..k\}\rightarrow\{1..k\}.m_i \in \mathsf{exit}(s_{p(i)})\forall_i\}
\end{array}$$

$$\begin{array}{ll}
\mathsf{entry}([\hat{s},(en,ex)]) & := \{en\} \\
\mathsf{entry}([\hat{s},(s_{1..k}),l,T,(en,ex)]) & := \{en{::}en' \mid en' \in \mathsf{entry}(s_l)\} \\
\mathsf{entry}([\hat{s},(s_{1..k}),(en,ex)]) & := \{en{::}m_1{::}..{::}m_k \mid \exists \text{ bijection } p{:}\{1..k\}\to\{1..k\}.m_i \in \mathsf{entry}(s_{p(i)})\forall_i\}
\end{array}$$

4.2 Structured Operational Semantics

Given $s \in \mathsf{SC}$, its structured operational semantics (SO semantics) $[\![s]\!]_{aux}$ is given by a Labelled Transition System $(C, L, \longrightarrow, \mathsf{conf}(s))$ where C is a set of state label sets (configurations), $L \subseteq \mathsf{E} \times \mathsf{A}^* \times \{0,1\}$ is the set of labels, $\longrightarrow \subseteq C \times L \times C$ is the transition relation, and $\mathsf{conf}(s)$ is the start configuration.

We define $C := \mathsf{conf\text{-}all}(s)$, where L and $\longrightarrow$ are defined by the rules in figure 4, where $c \overset{e\ f}{\longrightarrow}_{\alpha} c'$ means $(c,(e,\alpha,f),c') \in \longrightarrow$, and $c \overset{e}{\not\rightarrow}{}^{f}$ means $\nexists c', \alpha . c \overset{e}{\rightarrow}_{\alpha} c'$.

Idle

$$\frac{}{[\hat{s}] \overset{e\ 0}{\rightarrow}_{\langle\rangle} [\hat{s}]}\ \mathsf{BAS}$$

Progress

$$\frac{\langle \hat{t}, l, S_r, e, \alpha, T_d, m, ht\rangle \in T, \quad S_r \subseteq \mathsf{conf}(s_l), \quad s_l \overset{e}{\not\rightarrow}{}^{1}}{[\hat{s},(s_{1..k}),l,T] \overset{e\ 1}{\rightarrow}_{ex::\alpha::en} [\hat{s},(s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]}, m, T]}\ \mathsf{OR\text{-}1}$$

where $ex \in \mathsf{exit}(s_s)$ and $en \in \mathsf{entry}(\mathsf{next}(ht, T_d, s_t))$

Propagation of progress

$$\frac{s_l \overset{e\ 1}{\rightarrow}_{\alpha} s'_l}{[\hat{s},(s_{1..k}),l,T] \overset{e\ 1}{\rightarrow}_{\alpha} [\hat{s},(s_{1..k})_{[s_l/s'_l]},l,T]}\ \mathsf{OR\text{-}2}$$

Propagation of stuttering

$$\frac{[\hat{s},(s_{1..k}),l,T] \overset{e}{\not\rightarrow}{}^{1}, \quad s_l \overset{e\ 0}{\rightarrow}_{\langle\rangle} s_l}{[\hat{s},(s_{1..k}),l,T] \overset{e\ 0}{\rightarrow}_{\langle\rangle} [\hat{s},(s_{1..k}),l,T]}\ \mathsf{OR\text{-}3}$$

Composition

$$\frac{\forall_{j\in\{1,\dots,k\}}.s_j \overset{e\ f_j}{\rightarrow}_{\alpha_j} s'_j}{[\hat{s},(s_{1..k})] \overset{e\ \vee_{j=1}^{k} f_j}{\rightarrow}_{\alpha} [\hat{s},(s'_{1..k})]}\ \mathsf{AND} \quad \big(\alpha \in \mathrm{perm}(\alpha_{(i)})\big)$$

Figure 4: Rules of the SO semantics

The stuttering flag f is used to reflect the priority mechanism for statecharts transitions. It also allows the semantics to fullfill the maximality condition of UML statecharts, that is, a

maximal number of non conflicting statechart transitions is taken when a semantic transition is performed. The stuttering flag f can take the values 0 or 1. If f=0, then no statechart transition is taken, only the event e is consumed. If f=1 a statechart transition is taken. The flag is needed to assure that idle steps can only occur if no non-idle step is possible. A key role is played by the stuttering step ($f = 0$), since when no UML statechart transition can be taken, a stuttering step (loop) can be done.
Let us explain the OR rules:
OR-1: This rule models a semantic transition from a statechart transition. If we have a statechart transition of an OR state with trigger e then that state can perform a semantic transition with positive flag and trigger e, given that, the source restriction is a complete current configuration of the currently active substate s_l ($S_r \subseteq \mathsf{conf}(s_l)$) and s_l cannot perform a semantic transition with input e and positive flag.
OR-2: This rule propagates the progress of rule OR-1. If the current active substate can perform a semantic transition with positive flag, then the OR state may perform a transition with positive flag.
OR-3: This rule propagates the negative flag. That is, if the current active substate can perform a semantic transition with negative flag, and if the OR state cannot perform a semantic transition with positive flag, then the OR state can perform a semantic transition to itself with positive flag.

5 Behavioral Refinements

5.1 Extension Relation and Configurations

In this section we prove that the extension relation preserves the set of all possible configurations, which is a key part of the main result of this work.

Lemma 1 *If* $s \succ_1 \tilde{s}$ *then* $\mathsf{conf}(s) \subseteq \mathsf{conf}(\tilde{s})$

Proof. By induction on $s \succ_1 \tilde{s}$.

- ext-and. Assume $s = [\hat{s}]$. Then, we have that $[\hat{s}] \succ_1 [\hat{s}, (s_1, .., s_k)]$, where $s_1, .., s_k \in SC_B$. Then, $\mathsf{conf}([\hat{s}]) = \{\hat{s}\} \subseteq \mathsf{conf}([\hat{s}, (s_1, .., s_k)]) = \{\hat{s}, \hat{s}_1, ..\hat{s}_k\}$.

- inside-and. Assume $s = [\hat{s}, (s_{1..k})]$. Then, we have that $[\hat{s}, (s_{1..k})] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}]$, where $s_i \succ_1 \tilde{s}_i$. Here we use the inductive hypothesis, that is, $\mathsf{conf}(s_i) \subseteq \mathsf{conf}(\tilde{s}_i)$, since $s_i \succ_1 \tilde{s}_i$. We have,

$$\mathsf{conf}([\hat{s}, (s_{1..k})]) = \{\hat{s}\} \cup \bigcup_{j=1..i-1,i+1..k} \mathsf{conf}(s_j) \cup \mathsf{conf}(s_i)$$
$$\mathsf{conf}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}]) = \{\hat{s}\} \cup \bigcup_{j=1..i-1,i+1..k} \mathsf{conf}(s_j) \cup \mathsf{conf}(\tilde{s}_i)$$

  then $\mathsf{conf}([\hat{s}, (s_{1..k})]) \subseteq \mathsf{conf}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}])$.

- ext-or1. Assume $s = [\hat{s}]$. Then, we have that $[\hat{s}] \succ_1 [\hat{s}, (s'), 1, \emptyset]$, where $s' \in SC_B$. Then, $\mathsf{conf}([\hat{s}]) \subseteq \mathsf{conf}([\hat{s}, (s'), 1, \emptyset])$.

- ext-or2. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have that $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k}, s'), l, T]$, where $s' \in SC_B$. Then,

$$\begin{aligned}\mathsf{conf}([\hat{s}, (s_{1..k}), l, T]) &= \{\hat{s}\} \cup \mathsf{conf}(s_l) \\ &= \mathsf{conf}([\hat{s}, (s_{1..k}, s'), l, T])\end{aligned}$$

- inside-or. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have that $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]$, where $s_i \succ_1 \tilde{s}_i$. Here we use the inductive hypothesis, that is, $\mathsf{conf}(s_i) \subseteq \mathsf{conf}(\tilde{s}_i)$, since $s_i \succ_1 \tilde{s}_i$. We have two subcases,

  1. When $i = l$,

  $$\begin{aligned} \mathsf{conf}([\hat{s}, (s_{1..k}), l, T]) &= \{\hat{s}, \mathsf{conf}(s_i)\} \\ \mathsf{conf}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]) &= \{\hat{s}, \mathsf{conf}(\tilde{s}_i)\} \end{aligned}$$

  then $\mathsf{conf}([\hat{s}, (s_{1..k}), l, T]) \subseteq \mathsf{conf}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T])$.

  2. When $i \neq l$,

  $$\begin{aligned} \mathsf{conf}([\hat{s}, (s_{1..k}), l, T]) &= \{\hat{s}, \mathsf{conf}(s_l)\} \\ &= \mathsf{conf}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]) \end{aligned}$$

The cases add-trans, ext-act-en, ext-act-ex and ext-act-trans are trivial because the state structure is not modified, so the current configuration is preserved.

Lemma 2 *If $s \succ_1 \tilde{s}$ then conf-all$(s) \subseteq$ conf-all$(\tilde{s})$*

Proof. By induction on $s \succ_1 \tilde{s}$.

- ext-and. Assume $s = [\hat{s}]$. Then, we have that $[\hat{s}] \succ_1 [\hat{s}, (s_1, .., s_k)]$, where $s_1, .., s_k \in SC_B$. Then,

$$\begin{aligned} \mathsf{conf\text{-}all}([\hat{s}]) &= \{\{\hat{s}\}\} \\ \mathsf{conf\text{-}all}([\hat{s}, (s_1, .., s_k)]) &= \{\{\hat{s}\} \cup \bigcup_{i=1..k} c_i | c_i \in \mathsf{conf\text{-}all}(s_i)\} \cup \{\{\hat{s}\}\} \end{aligned}$$

- inside-and. Assume $s = [\hat{s}, (s_{1..k})]$. Then, we have that $[\hat{s}, (s_{1..k})] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}]$, where $s_i \succ_1 \tilde{s}_i$. Here we use the inductive hypothesis, that is, $\mathsf{conf\text{-}all}(s_i) \subseteq \mathsf{conf\text{-}all}(\tilde{s}_i)$, since $s_i \succ_1 \tilde{s}_i$. We have,

$$\begin{aligned} \mathsf{conf\text{-}all}([\hat{s}, (s_{1..k})]) &= \{\{\hat{s}\} \cup \bigcup_{j=1..k} c_j | c_j \in \mathsf{conf\text{-}all}(s_j)\} \cup \{\{\hat{s}\}\} \\ \mathsf{conf\text{-}all}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}]) &= \{\{\hat{s}\} \cup \bigcup_{j=1..k} c_j | c_j \in \mathsf{conf\text{-}all}(\tilde{s}_j)\} \cup \{\{\hat{s}\}\} \\ &= \{\{\hat{s}\} \cup \bigcup_{j=1..i-1,i+1..k} c_j \cup c | c_j \in \mathsf{conf\text{-}all}(s_j), c \in \mathsf{conf\text{-}all}(s_i)\} \\ &\cup \{\{\hat{s}\} \cup \bigcup_{j=1..i-1,i+1..k} c_j \cup c' | c_j \in \mathsf{conf\text{-}all}(s_j), c' \in \mathsf{conf\text{-}all}(\tilde{s}_i) - \mathsf{conf\text{-}all}(s_i)\} \\ &\cup \{\{\hat{s}\}\} \end{aligned}$$

  then $\mathsf{conf\text{-}all}([\hat{s}, (s_{1..k})]) \subseteq \mathsf{conf\text{-}all}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}])$. Because the first term in the union of the last equality is equal to $\mathsf{conf\text{-}all}([\hat{s}, (s_{1..k})])$.

- ext-or1. Assume $s = [\hat{s}]$. Then, we have that $[\hat{s}] \succ_1 [\hat{s}, (s'), 1, \emptyset]$, where $s' \in SC_B$. Then,

$$\begin{aligned} \mathsf{conf\text{-}all}([\hat{s}]) &= \{\{\hat{s}\}\} \\ \mathsf{conf\text{-}all}([\hat{s}, (s'), 1, \emptyset]) &= \{\{\hat{s}, \hat{s}'\}, \{\hat{s}\}\}, \text{ since } \mathsf{conf\text{-}all}(s') = \{\{\hat{s}'\}\} \end{aligned}$$

- ext-or2. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have that $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k}, s'), l, T]$, where $s' \in SC_B$. Then,

$$\begin{aligned}
\mathsf{conf\text{-}all}([\hat{s}, (s_{1..k}), l, T]) &= \{\{\hat{s}\} \cup c | \exists i \in \{1..k\}.c \in \mathsf{conf\text{-}all}(s_i)\} \cup \{\{\hat{s}\}\} \\
&= \{\{\hat{s}\} \cup c | c \in \mathsf{conf\text{-}all}(s_1)\} \cup \ldots \cup \{\{\hat{s}\} \cup c | c \in \mathsf{conf\text{-}all}(s_k)\} \cup \{\{\hat{s}\}\} \\
\mathsf{conf\text{-}all}([\hat{s}, (s_{1..k}, s'), l, T]) &= \{\{\hat{s}\} \cup c | \exists i \in \{1..k\}.c \in \mathsf{conf\text{-}all}(s_i) \vee c \in \mathsf{conf\text{-}all}(s')\} \cup \{\{\hat{s}\}\} \\
&\quad \text{now using the fact that } \mathsf{conf\text{-}all}(s') = \{\{\hat{s}'\}\}, \\
&= \{\{\hat{s}\} \cup c | c \in \mathsf{conf\text{-}all}(s_1)\} \cup \ldots \cup \{\{\hat{s}\} \cup c | c \in \mathsf{conf\text{-}all}(s_k)\} \\
&\cup \ \{\{\hat{s}, \hat{s}'\}\} \cup \{\{\hat{s}\}\}
\end{aligned}$$

so $\mathsf{conf\text{-}all}([\hat{s}, (s_{1..k}), l, T]) \subseteq \mathsf{conf\text{-}all}([\hat{s}, (s_{1..k}, s'), l, T])$.

- inside-or. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have that $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]$, where $s_i \succ_1 \tilde{s}_i$. Here we use the inductive hypothesis, that is, $\mathsf{conf\text{-}all}(s_i) \subseteq \mathsf{conf\text{-}all}(\tilde{s}_i)$, since $s_i \succ_1 \tilde{s}_i$. We have,

$$\begin{aligned}
\mathsf{conf\text{-}all}([\hat{s}, (s_{1..k}), l, T]) &= \{\{\hat{s}\} \cup c | c \in \mathsf{conf\text{-}all}(s_1)\} \cup \ldots \cup \{\{\hat{s}\} \cup c | c \in \mathsf{conf\text{-}all}(s_i)\} \\
&\cup \ldots \cup \{\{\hat{s}\} \cup c | c \in \mathsf{conf\text{-}all}(s_k)\} \cup \{\{\hat{s}\}\} \\
\mathsf{conf\text{-}all}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]) &= \{\{\hat{s}\} \cup c | c \in \mathsf{conf\text{-}all}(s_1)\} \cup \ldots \cup \{\{\hat{s}\} \cup c | c \in \mathsf{conf\text{-}all}(\tilde{s}_i)\} \\
&\cup \ldots \cup \{\{\hat{s}\} \cup c | c \in \mathsf{conf\text{-}all}(s_k)\} \cup \{\{\hat{s}\}\}
\end{aligned}$$

then $\mathsf{conf\text{-}all}([\hat{s}, (s_{1..k}), l, T]) \subseteq \mathsf{conf\text{-}all}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T])$.

The cases add-trans, ext-act-en, ext-act-ex and ext-act-trans are trivial because the state structure is not modified, so the set of possible configurations is preserved.

5.2 Monotonicity of next Function

In this section we prove that the function next applied to a extended state computes a state which indeed is an extension of the next state computed before the extension, that is, the function next is monotone with respect to the extension relation.
This result is proven in lemma 5. Before, we need two additional lemmas showing that functions def and next-stop are also monotone w.r.t. the extension relation (lemmas 3 and 4 respectively).

Lemma 3 *If $s \succ_1 \tilde{s}$ then $\mathsf{def}(s) \succ_1 \mathsf{def}(\tilde{s})$*

Proof. By induction on $s \succ_1 \tilde{s}$.

- ext-and. Assume $s = [\hat{s}]$. Then, we have that $[\hat{s}] \succ_1 [\hat{s}, (s_1, .., s_k)]$, where $s_1, .., s_k \in SC_B$. We have that $\mathsf{def}([\hat{s}, (s_1, .., s_k)]) = [\hat{s}, (\mathsf{def}(s_1), .., \mathsf{def}(s_k))] = [\hat{s}, (s_1, .., s_k)]$, since $s_1, .., s_k \in SC_B$, which indeed is a one step refinement through ext-and of $\mathsf{def}([\hat{s}]) = [\hat{s}]$.

- inside-and. Assume $s = [\hat{s}, (s_{1..k})]$. Then, we have that $[\hat{s}, (s_{1..k})] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}]$, where $s_i \succ_1 \tilde{s}_i$. Here we use the inductive hypothesis, that is, $\mathsf{def}(s_i) \succ_1 \mathsf{def}(\tilde{s}_i)$, since $s_i \succ_1 \tilde{s}_i$. Then, $\mathsf{def}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}]) = [\hat{s}, (\mathsf{def}(s_1), .., \mathsf{def}(\tilde{s}_i), ..\mathsf{def}(s_k))]$, which indeed is a one step refinement through inside-and of $\mathsf{def}([\hat{s}, (s_{1..k})]) = [\hat{s}, (\mathsf{def}(s_1), .., \mathsf{def}(s_i), .., \mathsf{def}(s_k))]$ by the inductive hypothesis.

- ext-or1. Assume $s = [\hat{s}]$. Then, we have that $[\hat{s}] \succ_1 [\hat{s}, (s'), 1, \emptyset]$, where $s' \in SC_B$. We have that $\mathsf{def}([\hat{s}, (s'), 1, \emptyset]) = [\hat{s}, (s')_{[s'/\mathsf{def}(s')]}, 1, \emptyset] = [\hat{s}, (s'), 1, \emptyset]$ since $s' \in SC_B$, which indeed is a one step refinement of $\mathsf{def}([\hat{s}]) = [\hat{s}]$ through ext-or1.

- $\mathsf{ext\text{-}or2}$. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have that $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k}, s'), l, T]$, where $s' \in SC_B$. We have that $\mathsf{def}([\hat{s}, (s_{1..k}, s'), l, T]) = [\hat{s}, (s_{1..k}, s')_{[s_1/\mathsf{def}(s_1)]}, 1, T]$, which indeed is a one step refinement of $\mathsf{def}([\hat{s}, (s_{1..k}), l, T]) = [\hat{s}, (s_{1..k})_{[s_1/\mathsf{def}(s_1)]}, 1, T]$.

- $\mathsf{inside\text{-}or}$. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have that $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]$, where $s_i \succ_1 \tilde{s}_i$. Here we use the inductive hypothesis, that is, $\mathsf{def}(s_i) \succ_1 \mathsf{def}(\tilde{s}_i)$, since $s_i \succ_1 \tilde{s}_i$. We have two cases:

  1. If $i \neq 1$, then $\mathsf{def}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]) = [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][s_1/\mathsf{def}(s_1)]}, 1, T]$ $= [\hat{s}, (s_{1..k})_{[s_1/\mathsf{def}(s_1)][s_i/\tilde{s}_i]}, 1, T]$, which indeed is a one step refinement of $\mathsf{def}([\hat{s}, (s_{1..k}), l, T]) = [\hat{s}, (s_{1..k})_{[s_1/\mathsf{def}(s_1)]}, 1, T]$

  2. If $i = 1$, then $\mathsf{def}([\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]) = [\hat{s}, (s_{1..k})_{[s_1/\tilde{s}_1][\tilde{s}_1/\mathsf{def}(\tilde{s}_1)]}, 1, T] = [\hat{s}, (s_{1..k})_{[s_1/\mathsf{def}(\tilde{s}_1)]}, 1, T]$, which indeed is a one step refinement of $\mathsf{def}([\hat{s}, (s_{1..k})_{[s_1/\mathsf{def}(s_1)]}, 1, T])$ by the inductive hypothesis.

Extensions $\mathsf{add\text{-}trans}$, $\mathsf{ext\text{-}act\text{-}en}$, $\mathsf{ext\text{-}act\text{-}ex}$ and $\mathsf{ext\text{-}act\text{-}trans}$ are trivial because def does not depend on the set of transitions T nor exit nor entry actions.

Lemma 4 *If $s \succ_1 \tilde{s}$ then $\mathsf{next\text{-}stop}(ht, s) \succ_1 \mathsf{next\text{-}stop}(ht, \tilde{s})$, $ht \in \mathsf{HT}$*

Proof. By induction on $s \succ_1 \tilde{s}$. First, note that $\tilde{s} \in SC_O$, because $s \in SC_O$ by hypothesis. Then only OR rules applies,

- $\mathsf{ext\text{-}or2}$. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k}, s'), l, T]$, where $s' \in SC_B$. We show that, in each case, $\mathsf{next\text{-}stop}(ht, \tilde{s})$ is indeed a one step refinement of $\mathsf{next\text{-}stop}(ht, s)$, through rule $\mathsf{ext\text{-}or2}$.

$$\mathsf{next\text{-}stop}(ht, [\hat{s}, (s_{1..k}, s'), l, T]) = \begin{cases} [\hat{s}, (s_{1..k}, s'), l, T] & \text{if} \quad ht = \mathsf{deep} \\ [\hat{s}, (s_{1..k}, s')_{[s_1/\mathsf{def}(s_1)]}, 1, T] & \text{if} \quad ht = \mathsf{none} \\ [\hat{s}, (s_{1..k}, s')_{[s_l/\mathsf{def}(s_l)]}, l, T] & \text{if} \quad ht = \mathsf{shallow}, l \in \{1..k\} \end{cases}$$

$\mathsf{inside\text{-}or}$. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]$, where $s_i \succ_1 \tilde{s}_i$. We show that, in each case, $\mathsf{next\text{-}stop}(ht, \tilde{s})$ is indeed a one step refinement, through rule $\mathsf{inside\text{-}or}$.

$$\mathsf{next\text{-}stop}(ht, [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]) = \begin{cases} [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T] & \text{if} \quad ht = \mathsf{deep} \\ [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][s_1/\mathsf{def}(s_1)]}, 1, T] & \text{if} \quad ht = \mathsf{none} \\ [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][s_l/\mathsf{def}(s_l)]}, l, T] & \text{if} \quad ht = \mathsf{shallow} \end{cases}$$

The second and third cases require some explanation: In the second one, we need to take into account two subcases:

- $i \neq 1$. It is easy to see that $[\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][s_1/\mathsf{def}(s_1)]}, 1, T] = [\hat{s}, (s_{1..k})_{[s_1/\mathsf{def}(s_1)][s_i/\tilde{s}_i]}, 1, T]$, then $[\hat{s}, (s_{1..k})_{[s_1/\mathsf{def}(s_1)]}, 1, T] \succ_1 [\hat{s}, (s_{1..k})_{[s_1/\mathsf{def}(s_1)][s_i/\tilde{s}_i]}, 1, T]$ through rule $\mathsf{inside\text{-}or}$.
- $i = 1$. We can see that $[\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][\tilde{s}_1/\mathsf{def}(\tilde{s}_1)]}, 1, T] = [\hat{s}, (s_{1..k})_{[s_1/\tilde{s}_1][\tilde{s}_1/\mathsf{def}(\tilde{s}_1)]}, 1, T] = [\hat{s}, (s_{1..k})_{[s_1/\mathsf{def}(\tilde{s}_1)]}, 1, T]$.
  Now, by lemma 3 we conclude that $[\hat{s}, (s_{1..k})_{[s_1/\mathsf{def}(s_1)]}, 1, T] \succ_1 [\hat{s}, (s_{1..k})_{[s_1/\mathsf{def}(\tilde{s}_1)]}, 1, T]$ through rule $\mathsf{inside\text{-}or}$.

In the third case, we need to take into account two subcases:

- $i \neq l$. It is easy to see that $[\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][s_l/\mathsf{def}(s_l)]}, l, T] = [\hat{s}, (s_{1..k})_{[s_l/\mathsf{def}(s_l)][s_i/\tilde{s}_i]}, l, T]$, then $[\hat{s}, (s_{1..k})_{[s_l/\mathsf{def}(s_l)]}, l, T] \succ_1 [\hat{s}, (s_{1..k})_{[s_l/\mathsf{def}(s_l)][s_i/\tilde{s}_i]}, l, T]$ through rule inside-or.
- $i = l$. We can see that $[\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][s_l/\mathsf{def}(s_l)]}, l, T] = [\hat{s}, (s_{1..k})_{[s_l/\tilde{s}_l][s_l/\mathsf{def}(s_l)]}, l, T] = [\hat{s}, (s_{1..k})_{[s_l/\mathsf{def}(\tilde{s}_l)]}, l, T]$.
  Now, by lemma 3 we conclude that $[\hat{s}, (s_{1..k})_{[s_l/\mathsf{def}(s_l)]}, l, T] \succ_1 [\hat{s}, (s_{1..k})_{[s_l/\mathsf{def}(\tilde{s}_l)]}, l, T]$ through rule inside-or.

- add-trans. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k}), l, T \cup \{t^*\}]$, where $t^* \in \mathsf{T}$. This case is analogous to ext-or2 case, but the refinement is through rule add-trans.

- Extensions ext-act-en, ext-act-ex and ext-act-trans are trivial because next-stop does not depend on the set of transitions T nor exit nor entry actions.

Lemma 5 *If $s \succ_1 \tilde{s}$ then $\mathsf{next}(ht, T_d, s) \succ_1 \mathsf{next}(ht, T_d, \tilde{s})$, $T_d \in \mathsf{conf\text{-}all}(s)$, $ht \in \mathsf{HT}$*

Proof. First note that $T_d \in \mathsf{conf\text{-}all}(s) \Rightarrow T_d \in \mathsf{conf\text{-}all}(\tilde{s})$ by lemma 2. Now we use induction on $s \succ_1 \tilde{s}$.

- ext-and. Assume $s = [\hat{s}]$. Then, we have that $[\hat{s}] \succ_1 [\hat{s}, (s_1, .., s_k)]$, where $s_1, .., s_k \in SC_B$. Then, $\mathsf{next}(ht, T_d, [\hat{s}, (s_1, .., s_k)]) = [\hat{s}, (\mathsf{next}(ht, T_d, s_1), .., \mathsf{next}(ht, T_d, s_k))] = [\hat{s}, (s_1, .., s_k)]$ because $s_1, .., s_k \in SC_B$, which indeed is a one step refinement of s through ext-and.

- inside-and. Assume $s = [\hat{s}, (s_{1..k})]$. Then, we have that $[\hat{s}, (s_{1..k})] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}]$, where $s_i \succ_1 \tilde{s}_i$. Here we use the inductive hypothesis, that is, $\mathsf{next}(ht, T_d, s_i) \succ_1 \mathsf{next}(ht, T_d, \tilde{s}_i)$, since $s_i \succ_1 \tilde{s}_i$ and lemma 2. Then, $[\hat{s}, (\mathsf{next}(ht, T_d, s_1), .., \mathsf{next}(ht, T_d, s_i), .., \mathsf{next}(ht, T_d, s_k))] \succ_1 [\hat{s}, (\mathsf{next}(ht, T_d, s_1), .., \mathsf{next}(ht, T_d, \tilde{s}_i), .., \mathsf{next}(ht, T_d, s_k))]$ through inside-and.

- ext-or1. Assume $s = [\hat{s}]$. Then, we have that $[\hat{s}] \succ_1 [\hat{s}, (s'), 1, \emptyset]$, where $s' \in SC_B$. We have two subcases,

  1. If $\exists \nu \in T_d . \nu = \hat{s}'$, then $\mathsf{next}(ht, T_d, [\hat{s}, (s'), 1, \emptyset]) = [\hat{s}, (s')_{[s'/\mathsf{next}(ht, T_d, s')]}, 1, \emptyset] = [\hat{s}, (s'), 1, \emptyset]$, because $s' \in SC_B$, which indeed is a one step refinement of s through ext-or1.
  2. Otherwise, $\mathsf{next}(ht, T_d, [\hat{s}]) = \mathsf{next\text{-}stop}(ht, [\hat{s}])$. Then, by lemma 4, $\mathsf{next}(ht, T_d, [\hat{s}]) \succ_1 \mathsf{next}(ht, T_d, [\hat{s}, (s'), 1, \emptyset])$.

- ext-or2. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have that $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k}, s'), l, T]$, where $s' \in SC_B$. We have two cases,

  1. If $\exists \nu \in T_d . \nu = \hat{s}_j$, then $\mathsf{next}(ht, T_d, [\hat{s}, (s_{1..k}, s'), l, T]) = [\hat{s}, (s_{1..k}, s')_{[s_j/\mathsf{next}(ht, T_d, s_j)]}, j, T]$, and since $j \in \{1..k\}$, it is a one step refinement of $[\hat{s}, (s_{1..k})_{[s_j/\mathsf{next}(ht, T_d, s_j)]}, j, T]$ through ext-or2.
  2. Otherwise, $\mathsf{next}(ht, T_d, [\hat{s}, (s_{1..k}), l, T]) = \mathsf{next\text{-}stop}(ht, [\hat{s}, (s_{1..k}), l, T])$. Then, by lemma 4, $\mathsf{next}(ht, T_d, [\hat{s}, (s_{1..k}), l, T]) \succ_1 \mathsf{next}(ht, T_d, [\hat{s}, (s_{1..k}, s'), l, T])$.

- inside-or. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have that $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]$, where $s_i \succ_1 \tilde{s}_i$. Here we use the inductive hypothesis, that is, $\mathsf{next}(ht, T_d, s_i) \succ_1 \mathsf{next}(ht, T_d, \tilde{s}_i)$, since $s_i \succ_1 \tilde{s}_i$. We have two cases,

1. If $\exists \nu \in T_d.\nu = \hat{s}_j$, then we need to take account of two subcases:
    1. If $i \neq j$, then $\mathsf{next}(ht, T_d, [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]) = [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][s_j/\mathsf{next}(ht,T_d,s_j)]}, j, T] = [\hat{s}, (s_{1..k})_{[s_j/\mathsf{next}(ht,T_d,s_j)][s_i/\tilde{s}_i]}, j, T]$, which is a one step refinement of $[\hat{s}, (s_{1..k})_{[s_j/\mathsf{next}(ht,T_d,s_j)]}, j, T]$ through inside-or.
    2. If $i = j$, then $\mathsf{next}(ht, T_d, [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]) = [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][s_j/\mathsf{next}(ht,T_d,s_j)]}, j, T] = [\hat{s}, (s_{1..k})_{[s_j/\tilde{s}_j][\tilde{s}_j/\mathsf{next}(ht,T_d,\tilde{s}_j)]}, j, T] = [\hat{s}, (s_{1..k})_{[s_j/\mathsf{next}(ht,T_d,\tilde{s}_j)]}, j, T]$. Then, by inductive hypothesis, $[\hat{s}, (s_{1..k})_{[s_j/\mathsf{next}(ht,T_d,\tilde{s}_j)]}, j, T]$ is a one step refinement of $[\hat{s}, (s_{1..k})_{[s_j/\mathsf{next}(ht,T_d,s_j)]}, j, T]$.
2. Otherwise, $\mathsf{next}(ht, T_d, [\hat{s}, (s_{1..k}), l, T]) = \mathsf{next\text{-}stop}(ht, [\hat{s}, (s_{1..k}), l, T])$. Then, by lemma 4, $\mathsf{next}(ht, T_d, [\hat{s}, (s_{1..k}), l, T]) \succ_1 \mathsf{next}(ht, T_d, [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T])$.

Extensions add-trans, ext-act-en, ext-act-ex and ext-act-trans are trivial because next does not depend on the set of transitions T nor exit nor entry actions.

5.3 Extension Relation as a Behavioral Refinement

In this section we prove the main result of this paper, that is, the extension relation can be considered as a refinement, in the sense that it preserves the transitions defined in the SO semantics.
Due to the priority mechanism for transitions specified by UML, we cannot expect that any extension preserves the semantics. The conflict arises when an inner transition with the same triggering event as an existing outer transition is added to a statechart. As the inner transition has priority over the outer one, the semantics is not preserved, since the outer transition will not take place in the extended statechart. We define an extension to be *safe* if no inner transitions are added with the same event as an existing outer transition. Formally:

Definition 1 *Let* $s = [\hat{s}, (s_{1..k}), l, T]$, $\tilde{s} = [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]$, $s_i \succ_1 \tilde{s}_i$. *We say that* $s \succ_1 \tilde{s}$ *is a **safe extension** iff*

$$\forall e \in \mathsf{E}:\ (\exists s' \in \mathsf{SC}.\ (s \xrightarrow{e}{}^{1} s' \wedge s_i \overset{e}{\not\rightarrow}{}^{1}\ ) \Rightarrow \tilde{s}_i \overset{e}{\not\rightarrow}{}^{1}\ )$$

From now on, we require all extensions to be safe. We now present the main result of this paper:

Theorem 1 $\forall s, s', t \in \mathsf{SC}, \forall e \in \mathsf{E}$, *if* $s \xrightarrow{e}{}^{1} s'$ *and* $s \succ_1 t$, *then* $\exists t'$ *such that* $t \xrightarrow{e}{}^{1} t'$ *and* $s' \succ_1 t'$.

This theorem states that the extension relation is indeed a behavioral refinement, since the extended statechart preserves all the reachable states of the original one. Graphically,

$$\begin{tikzcd}
s \arrow[r, "e^{\,1}"] \arrow[d, phantom, sloped, "\succ_1"] & s' \arrow[d, phantom, sloped, "\succ_1"] \\
t \arrow[r, "e^{\,1}"] & \exists\ t'
\end{tikzcd}$$

Note that the theorem assumes that the stuttering flag is equal to 1. Since a statechart transition takes place when the stuttering flag is equal to 1, we are not interested in idle steps (f=0). As mentioned previously, the stuttering flag is needed to assure that idle steps can only occur if no non-idle step is possible. Basically, it allows the semantics to fullfill the maximality condition of statecharts, since when no statechart transition can be taken, a stuttering step (loop) can be done. It is important to remark again that statechart transitions are different from semantic transitions.

Proof of theorem 1. By induction on $s \succ_1 t$.

- ext-and. Let $[\hat{s}] \succ_1 [\hat{s}, (s_1, .., s_k)]$, where $s_1, .., s_k \in SC_B$. Then $[\hat{\tilde{s}_i}] \overset{e}{\rightarrow}^0 [\hat{\tilde{s}_i}]$ $\forall_{i=1..k}$ because the only rule that can be applied to a BASIC state is BAS. The hypothesis of the theorem does not hold, because we have a transition with flag 0.

- inside-and. Let $[\hat{s}, (s_{1..k})] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}]$, where $s_i \succ_1 \tilde{s}_i$. Let $[\hat{s}, (s_{1..k})] \overset{e}{\rightarrow}^f [\hat{s}, (s'_{1..k})] = s'$ with $s_j \overset{e}{\rightarrow}^{f_j} s'_j$ $\forall j=1, .., k$. By inductive hypothesis: if $s_i \overset{e}{\rightarrow}^1 s'_i$ and $s_i \succ_1 \tilde{s}_i$ then $\exists \tilde{s}'_i$ such that $\tilde{s}_i \overset{e}{\rightarrow}^1 \tilde{s}'_i$ and $s'_i \succ_1 \tilde{s}'_i$. Then we have that:

$$\frac{\forall_{j=1..i-1,i+1..k}. s_j \overset{e}{\rightarrow}^{f_j} s'_j, \qquad \dfrac{s_i \overset{e}{\rightarrow}^1 s'_i, \quad s_i \succ_1 \tilde{s}_i}{\tilde{s}_i \overset{e}{\rightarrow}^1 \tilde{s}'_i}\ \text{I.H.}}{t = [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}] \overset{e}{\rightarrow}^1 [\hat{s}, (s'_{1..k})_{[s'_i/\tilde{s}'_i]}] = t'}\ \text{AND}$$

And $[\hat{s}, (s'_{1..k})] \succ_1 [\hat{s}, (s'_{1..k})_{[s'_i/\tilde{s}'_i]}]$ by inside-and.

- ext-or1. Let $[\hat{s}] \succ_1 [\hat{s}, (\tilde{s}), 1, \emptyset]$, where $\tilde{s} \in SC_B$. Then $[\hat{s}] \overset{e}{\rightarrow}^0 [\hat{s}]$ because the only rule that can be applied to a BASIC state is BAS. The hypothesis of the theorem does not hold.

- ext-or2. Let $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k}, \tilde{s}), l, T]$, where $\tilde{s} \in SC_B$. We can apply three rules in this case:

  OR-1 rule: Let $[\hat{s}, (s_{1..k}), l, T] \overset{e}{\rightarrow}^1 [\hat{s}, (s_{1..k})_{[s_p/\mathsf{next}(ht,T_d,s_p)]}, p, T] = s'$ by OR-1 rule, where $\langle \hat{t}, l, \_, e, \_, T_d, p, ht \rangle \in T$. Since the extension adds a new state, it cannot be the current active one. Then we choose $t' = [\hat{s}, (s_{1..k[s_p/\mathsf{next}(ht,T_d,s_p)]}, \tilde{s}), t, T]$. By rule OR-1, we can reach the state t' from $t = [\hat{s}, (s_{1..k}, \tilde{s}), l, T]$. That is,

$$\frac{\langle \hat{t}, l, S_r, e, \alpha, T_d, p, ht \rangle \in T, \quad S_r \subseteq \mathsf{conf}(s_l), \quad s_l \overset{e}{\not\rightarrow}^1}{t = [\hat{s}, (s_{1..k}, \tilde{s}), l, T] \overset{e}{\rightarrow}^1 [\hat{s}, (s_{1..k[s_p/\mathsf{next}(ht,T_d,s_p)]}, \tilde{s}), p, T] = t'}\ \text{OR-1}$$

  And $[\hat{s}, (s_{1..k})_{[s_p/\mathsf{next}(ht,T_d,s_p)]}, p, T] \succ_1 [\hat{s}, (s_{1..k[s_p/\mathsf{next}(ht,T_d,s_p)]}, \tilde{s}), p, T]$ by ext-or2.

  OR-2 rule: Let $[\hat{s}, (s_{1..k}), l, T] \overset{e}{\rightarrow}^1 [\hat{s}, (s_{1..k})_{[s_l/s'_l]}, l, T] = s'$, when $s_l \overset{e}{\rightarrow}^1 s'_l$. Then,

$$\frac{s_l \overset{e}{\rightarrow}^1 s'_l}{t = [\hat{s}, (s_{1..k}, \tilde{s}), l, T] \overset{e}{\rightarrow}^1 [\hat{s}, (s_{1..k[s_l/s'_l]}, \tilde{s}), l, T] = t'}\ \text{OR-2}$$

  And we have that $[\hat{s}, (s_{1..k})_{[s_l/s'_l]}, l, T] \succ_1 [\hat{s}, (s_{1..k[s_l/s'_l]}, \tilde{s}), l, T]$ by ext-or2.

  OR-3 rule: Let $[\hat{s}, (s_{1..k}), l, T] \overset{e}{\rightarrow}^0 [\hat{s}, (s_{1..k}), l, T]$, when $[\hat{s}, (s_{1..k}), l, T] \overset{e}{\not\rightarrow}^1$. The hypothesis of the theorem does not hold.

- add-trans. Let $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k}), l, T \cup \{t^*\}]$, where $t^* \in \mathsf{T}$. Again, three rules can be applied:

  OR-1 rule: Let $[\hat{s}, (s_{1..k}), l, T] \overset{e}{\rightarrow}^1 [\hat{s}, (s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]}, m, T] = s'$ by OR-1 rule, where $\langle \hat{t}, l, \_, e, \_, T_d, m, ht \rangle \neq t^* \in T$. Then,

$$\frac{\langle \hat{t}, l, S_r, e, \_, T_d, m, ht \rangle \in T, \quad S_r \subseteq \mathsf{conf}(s_l), \quad s_l \overset{e}{\not\rightarrow}^1}{t = [\hat{s}, (s_{1..k}), l, T \cup \{t^*\}] \overset{e}{\rightarrow}^1 [\hat{s}, (s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]}, m, T \cup \{t^*\}] = t'}\ \text{OR-1}$$

We have that $[\hat{s},(s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]},m,T] \succ_1 [\hat{s},(s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]},m,T \cup \{t^*\}]$ by add-trans.

OR-2 rule: Let $[\hat{s},(s_{1..k}),l,T] \xrightarrow{e}^{1} [\hat{s},(s_{1..k})_{[s_l/s'_l]},l,T]{=}s'$, when $s_l \xrightarrow{e}^{1} s'_l$. Then,

$$\frac{s_l \xrightarrow{e}^{1} s'_l}{t=[\hat{s},(s_{1..k}),l,T\cup\{t^*\}] \xrightarrow{e}^{1} [\hat{s},(s_{1..k})_{[s_l/s'_l]},l,T\cup\{t^*\}]=t'}\ \textsf{OR-2}$$

Here $[\hat{s},(s_{1..k})_{[s_l/s'_l]},l,T] \succ_1 [\hat{s},(s_{1..k})_{[s_l/s'_l]},l,T\cup\{t^*\}]$ by add-trans.

OR-3 rule: Let $[\hat{s},(s_{1..k}),l,T] \xrightarrow{e}^{0} [\hat{s},(s_{1..k}),l,T]$, when $[\hat{s},(s_{1..k}),l,T] \not\xrightarrow{e}^{1}$. The hypothesis of the theorem does not hold.

- inside-or. Here we have that $[\hat{s},(s_{1..k}),l,T] \succ_1 [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i]},l,T']$, where $s_i \succ_1 \tilde{s}_i$. For the sake of clarity, we split the proof in two cases:

1. When $i \neq l$.

OR-1 rule: Let $[\hat{s},(s_{1..k}),l,T] \xrightarrow{e}^{1} [\hat{s},(s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]},m,T]{=}s'$, where $\langle \hat{t},l,\_,\_,\_,T_d,m,ht\rangle \in T$. Here we have two sub cases:

(a) When $i \neq m$.

$$\frac{\langle \hat{t},l,S_r,e,\_,T_d,m,ht\rangle \in T, \quad S_r \subseteq \mathsf{conf}(s_l), \quad s_l \not\xrightarrow{e}^{1}}{t=[\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i]},l,T] \xrightarrow{e}^{1} [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][s_m/\mathsf{next}(ht,T_d,s_m)]},m,T]=t'}\ \textsf{OR-1}$$

Here $[\hat{s},(s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]},m,T] \succ_1 [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][s_m/\mathsf{next}(ht,T_d,s_m)]},m,T]$ by inside-or.

(b) When $i = m$.

$$\frac{\langle \hat{t},l,S_r,e,\_,T_d,m,ht\rangle \in T, \quad S_r \subseteq \mathsf{conf}(s_l), \quad s_l \not\xrightarrow{e}^{1}}{t=[\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i]},l,T] \xrightarrow{e}^{1} [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][\tilde{s}_m/\mathsf{next}(ht,T_d,\tilde{s}_m)]},m,T]=t'}\ \textsf{OR-1}$$

Here $[\hat{s},(s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]},m,T] \succ_1 [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][\tilde{s}_m/\mathsf{next}(ht,T_d,\tilde{s}_m)]},m,T]$ by inside-or, since $\mathsf{next}(ht,T_d,s_m) \succ_1 \mathsf{next}(ht,T_d,\tilde{s}_m)$, by lemma 5.

OR-2 rule: Let $[\hat{s},(s_{1..k}),l,T] \xrightarrow{e}^{1} [\hat{s},(s_{1..k})_{[s_l/s'_l]},l,T]{=}s'$, when $s_l \xrightarrow{e}^{1} s'_l$. Then,

$$\frac{s_l \xrightarrow{e}^{1} s'_l}{t=[\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i]},l,T] \xrightarrow{e}^{1} [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][s_l/s'_l]},l,T]=t'}\ \textsf{OR-2}$$

Since $i{\neq}l$, $[\hat{s},(s_{1..k})_{[s_l/s'_l]},l,T] \succ_1 [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][s_l/s'_l]},l,T]$ by inside-or.

OR-3 rule: The hypothesis of the theorem does not hold.

2. When $i = l$. Note that if $s_i \not\xrightarrow{e}^{1}$ then $\tilde{s}_i \not\xrightarrow{e}^{1}$, because we are dealing with safe refinements. Three OR rules applies:

OR-1 rule: Let $[\hat{s},(s_{1..k}),i,T] \xrightarrow{e}^{1} [\hat{s},(s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]},m,T]{=}s'$, where $\langle \hat{t},i,S_r,e,\_,T_d,m,ht\rangle \in T$. Here we have two sub cases: *(a)* When $i \neq m$.

$$\frac{\langle \hat{t},i,S_r,e,\_,T_d,m,ht\rangle \in T, \quad S_r \subseteq \mathsf{conf}(\tilde{s}_i), \quad \tilde{s}_i \not\xrightarrow{e}^{1}}{t=[\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i]},i,T] \xrightarrow{e}^{1} [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][s_m/\mathsf{next}(ht,T_d,s_m)]},m,T]=t'}\ \textsf{OR-1}$$

And $[\hat{s},(s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]},m,T]\succ_1[\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][s_m/\mathsf{next}(ht,T_d,s_m)]},m,T]$ by inside-or. Note that $S_r\subseteq\mathsf{conf}(s_i)\Rightarrow S_r\subseteq\mathsf{conf}(\tilde{s}_i)$ holds by lemma 1.

(b) When $i=m$.

$$\frac{\langle\hat{t},i,S_r,e,\_,T_d,m,ht\rangle\in T,\quad S_r\subseteq\mathsf{conf}(\tilde{s}_i),\quad \tilde{s}_i \overset{e}{\not\to}{}^1}{t=[\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i]},i,T]\xrightarrow{e}{}^1\ [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][\tilde{s}_m/\mathsf{next}(ht,T_d,\tilde{s}_m)]},m,T]=t'}\ \textsf{OR-1}$$

And $[\hat{s},(s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]},m,T]\succ_1[\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][\tilde{s}_m/\mathsf{next}(ht,T_d,\tilde{s}_m)]},m,T]$ by inside-or, since $\mathsf{next}(ht,T_d,s_m)\succ_1\mathsf{next}(ht,T_d,\tilde{s}_m)$ by lemma 5.

OR-2 rule: Let $[\hat{s},(s_{1..k}),i,T]\xrightarrow{e}{}^1[\hat{s},(s_{1..k})_{[s_i/s_i']},i,T]=s'$, when $s_l\xrightarrow{e}{}^1 s_l'$. Then,

$$\frac{\dfrac{s_i\xrightarrow{e}{}^1 s_i',\quad s_i\succ_1\tilde{s}_i}{\tilde{s}_i\xrightarrow{e}{}^1\tilde{s}_i'}\ \textsf{I.H.}}{t=[\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i]},i,T]\xrightarrow{e}{}^1\ [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][\tilde{s}_i/\tilde{s}_i']},i,T]=t'}\ \textsf{OR-2}$$

Here $[\hat{s},(s_{1..k})_{[s_i/s_i']},i,T]\succ_1[\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][\tilde{s}_i/\tilde{s}_i']},i,T]$ by inside-or.

OR-3 rule: The hypothesis of the theorem does not hold.

ext-act-en, ext-act-ex and ext-act-trans are trivial: In those cases, we apply to s' the same refinement applied to s in order to obtain t'.

6 Statechart Actions

In theorem 1 we proved that the extension relation preserves the transitions defined in the SO semantics, and thus we can consider it as a refinement. The next step is to assure that the set of traces generated by a statechart is preserved in an extended statechart (here informally, we call trace of a statechart the set of actions generated in response of an external event). The study of traces is beyond the scope of this work, but in this section we state a theorem, together with two auxiliary lemmas, which take into account the actions generated by the SO semantics. We show that the possible actions generated by a statechart are preserved by the refinement.

Lemma 6 *If $s\succ_1\tilde{s}$ then $(\forall\alpha\in\mathsf{exit}(s),\ \forall\tilde{\alpha}\in\mathsf{exit}(\tilde{s}))\ .\ \alpha\rhd\tilde{\alpha}$*

Proof. By induction on $s\succ_1\tilde{s}$.

- ext-and. Assume $s=[\hat{s}]$. Then, we have that $[\hat{s}]\succ_1[\hat{s},(s_1,..,s_k)]$, where $s_1,..,s_k\in SC_B$. Then, $\mathsf{exit}(s)=\{ex\}$ and $\mathsf{exit}(\tilde{s})=\{m_1{::}..{::}m_k{::}ex\mid\exists$ bijection $p{:}\{1..k\}\to\{1..k\}.m_i\in\mathsf{exit}(s_{p(i)})\forall_i\}$. Since $s_1,..,s_k\in SC_B$, each $\mathsf{exit}(s_i)$ has only one element $\forall i$. Now, for each permutation $m_{p_1}{::}..{::}m_{p_k}{::}ex\in\mathsf{exit}(\tilde{s})$ the thesis clearly holds.

- inside-and. Assume $s=[\hat{s},(s_{1..k})]$. Then, we have that $[\hat{s},(s_{1..k})]\succ_1[\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i]}]$, where $s_i\succ_1\tilde{s}_i$. Then, $\mathsf{exit}(s)=\{m_1{::}..{::}m_k{::}ex\mid\exists$ bijection $p{:}\{1..k\}\to\{1..k\}.m_i\in\mathsf{exit}(s_{p(i)})\forall_i\}$. Now, fix one bijection (permutation). We get $\mathsf{exit}(s)=\{m_1{::}..{::}m_k{::}ex\mid m_j\in\mathsf{exit}(s_{p_j})\forall_j\}$. Now, $\mathsf{exit}(s)$ is the set of all possible combinations of $m_j\in\mathsf{exit}(s_{p_j})$. Take $i=p_j$ and use the inductive hypothesis, that is, $(\forall m_i\in\mathsf{exit}(s_i),\ \forall\tilde{m}_i\in\mathsf{exit}(\tilde{s}_i))\ .\ m_i\rhd\tilde{m}_i$. Now, the string $m_1{::}..{::}m_i{::}..{::}m_k\rhd m_1{::}..{::}\tilde{m}_i{::}..{::}m_k$ because it is a subsequence. Since the reasoning is valid for any permutation, the thesis holds.

- ext-or1. Assume $s = [\hat{s}]$. Then, we have that $[\hat{s}] \succ_1 [\hat{s}, (s'), 1, \emptyset]$, where $s' \in SC_B$. Then, $\mathsf{exit}(s) = \{ex\}$ and $\mathsf{exit}(\tilde{s}) = \{ex'::ex \mid ex' \in \mathsf{exit}(s')\} = \{ex'::ex\}$ because $s' \in SC_B$.

- ext-or2. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have that $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k}, s'), l, T]$, where $s' \in SC_B$. Then, $\mathsf{exit}(s) = \{ex'::ex \mid ex' \in \mathsf{exit}(s_l)\} = \mathsf{exit}(\tilde{s})$, because $l \in \{1..k\}$.

- inside-or. Assume $s = [\hat{s}, (s_{1..k}), l, T]$. Then, we have that $[\hat{s}, (s_{1..k}), l, T] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T]$, where $s_i \succ_1 \tilde{s}_i$. We distinguish two cases:

  1. If $i \neq l$. Then, $\mathsf{exit}(s) = \{ex'::ex \mid ex' \in \mathsf{exit}(s_l)\} = \mathsf{exit}(\tilde{s})$.
  2. If $i = l$. Then, use the inductive hypothesis, that is, $(\forall m_i \in \mathsf{exit}(s_l),\ \forall \tilde{m}_l \in \mathsf{exit}(\tilde{s}_l))$. $m_l \triangleright \tilde{m}_l$. Then, the string $m_1::..::m_l::..::m_k \triangleright m_1::..::\tilde{m}_l::..::m_k$ because it is a subsequence.

  Here we use the inductive hypothesis, that is, $\mathsf{next}(ht, T_d, s_i) \succ_1 \mathsf{next}(ht, T_d, \tilde{s}_i)$, since $s_i \succ_1 \tilde{s}_i$.

- ext-act-ex. We have three cases:

  1. Assume $s = [\hat{s}, (en, ex)]$. Then, we have that $[\hat{s}, (en, ex)] \succ_1 [\hat{s}, (en, \tilde{ex})]$, where $ex \triangleright \tilde{ex}$. Then, $\mathsf{exit}(s) = \{ex\}$ and $\mathsf{exit}(\tilde{s}) = \{\tilde{ex}\}$.
  2. Assume $s = [\hat{s}, (s_{1..k}), l, T, (en, ex)]$. Then, we have that $[\hat{s}, (s_{1..k}), l, T, (en, ex)] \succ_1 [\hat{s}, (s_{1..k}), l, T, (en, \tilde{ex})]$, where $ex \triangleright \tilde{ex}$. Then, $\mathsf{exit}(s) = \mathsf{exit}(s) = \{ex'::ex \mid ex' \in \mathsf{exit}(s_l)\}$ and $\mathsf{exit}(\tilde{s}) = \mathsf{exit}(s) = \{ex'::\tilde{ex} \mid ex' \in \mathsf{exit}(s_l)\}$.
  3. Assume $s = [\hat{s}, (s_{1..k}), (en, ex)]$. Then, we have that $[\hat{s}, (s_{1..k}), (en, ex)] \succ_1 [\hat{s}, (s_{1..k}), (en, \tilde{ex})]$, where $ex \triangleright \tilde{ex}$. Then, $\mathsf{exit}(s) = \{m_1::..::m_k::ex \mid \exists$ bijection $p{:}\{1..k\} \to \{1..k\}.m_i \in \mathsf{exit}(s_{p(i)}) \forall_i\}$ and $\mathsf{exit}(\tilde{s}) = \{m_1::..::m_k::\tilde{ex} \mid \exists$ bijection $p{:}\{1..k\} \to \{1..k\}.m_i \in \mathsf{exit}(s_{p(i)}) \forall_i\}$.

Extension add-trans, ext-act-en and ext-act-trans are trivial because exit does not depend on the set of transitions T nor entry actions.

Lemma 7 *If $s \succ_1 \tilde{s}$ then $(\forall \alpha \in \mathsf{entry}(s_i),\ \forall \tilde{\alpha} \in \mathsf{entry}(\tilde{s}'_i))$. $\alpha \triangleright \tilde{\alpha}$*

Proof. By induction on $s \succ_1 \tilde{s}$. Analogous to lemma 6.

Theorem 2 *Let $s, s', t, t' \in \mathsf{SC}$ where theorem 1 holds and $s \to_\alpha s'$, $t \to_{\alpha'} t'$ then $\alpha \triangleright \alpha'$.*

Proof. By induction on $s \succ_1 t$.

- ext-and. Let $[\hat{s}] \succ_1 [\hat{s}, (s_1, .., s_k)]$, where $s_1, .., s_k \in SC_B$. The only rule that can be applied is BAS, then the hypothesis of the theorem does not hold.

- inside-and. Let $[\hat{s}, (s_{1..k})] \succ_1 [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}]$, where $s_i \succ_1 \tilde{s}_i$. Let $[\hat{s}, (s_{1..k})] \overset{e\ f}{\longrightarrow}_\alpha [\hat{s}, (s'_{1..k})] = s'$ with $s_j \overset{e\ f_j}{\longrightarrow}_{\alpha_j} s'_j$ $\forall j{=}1, .., k$. By inductive hypothesis: if $s_i \overset{e\ 1}{\longrightarrow}_{\alpha_i} s'_i$ and $s_i \succ_1 \tilde{s}_i$ then $\exists \tilde{s}'_i$ such that $\tilde{s}_i \overset{e\ 1}{\longrightarrow}_{\alpha'_i} \tilde{s}'_i$ and $s'_i \succ_1 \tilde{s}'_i$ where $\alpha_i \triangleright \alpha'_i$. Then we have that:

$$\dfrac{\forall_{j=1..i-1,i+1..k} . s_j \overset{e\ f_j}{\longrightarrow}_{\alpha_j} s'_j, \qquad \dfrac{s_i \overset{e\ 1}{\longrightarrow}_{\alpha_i} s'_i, \quad s_i \succ_1 \tilde{s}_i}{\tilde{s}_i \overset{e\ 1}{\longrightarrow}_{\alpha'_i} \tilde{s}'_i}\ \text{I.H.}}{t = [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}] \overset{e\ 1}{\longrightarrow}_{\alpha'} [\hat{s}, (s'_{1..k})_{[s'_i/\tilde{s}'_i]}] = t'}\ \text{AND}$$

Fix a permutation, then we have $\alpha = \alpha_{p_1}::..::\alpha_{p_j}::..::\alpha_{p_k}$. Take $p_j = i$, then $\alpha_1::..::\alpha_i::..::\alpha_k \triangleright \alpha_1::..::\alpha'_i::..::\alpha_k = \alpha'$. Since the reasoning is valid for all permutations, the thesis holds.

- ext-or1. Let $[\hat{s}]\succ_1[\hat{s},(\tilde{s}),1,\emptyset]$, where $\tilde{s}\in SC_B$. The only rule that can be applied is BAS. The hypothesis of the theorem does not hold.

- ext-or2. Let $[\hat{s},(s_{1..k}),l,T]\succ_1[\hat{s},(s_{1..k},\tilde{s}),l,T]$, where $\tilde{s}{\in}SC_B$. We can apply three rules in this case:

  OR-1 rule: Let $[\hat{s},(s_{1..k}),l,T]\overset{e}{\rightarrow}{}^{1}_{ex::\gamma::en}[\hat{s},(s_{1..k})_{[s_p/\mathsf{next}(ht,T_d,s_p)]},p,T]{=}s'$ by OR-1 rule, where $\langle\hat{t}{,}l{,}\_{,}e{,}\gamma{,}T_d{,}p{,}ht\rangle\in T$, $ex\in\mathsf{exit}(s_l)$ and $en\in\mathsf{entry}(\mathsf{next}(ht,T_d,s_p))$. Since the extension adds a new state, it cannot be the current active one. We have,

$$\frac{\langle\hat{t}{,}l{,}S_r{,}e{,}\gamma{,}T_d{,}p{,}ht\rangle\in T,\quad S_r\subseteq\mathsf{conf}(s_l),\quad s_l\overset{e}{\not\rightarrow}{}^{1}}{t=[\hat{s},(s_{1..k},\tilde{s}),l,T]\overset{e}{\rightarrow}{}^{1}_{ex::\gamma::en}[\hat{s},(s_{1..k[s_p/\mathsf{next}(ht,T_d,s_p)]},\tilde{s}),p,T]=t'}\ \text{OR-1}$$

  then the output actions are unchanged.

  OR-2 rule: Let $[\hat{s},(s_{1..k}),l,T]\overset{e}{\rightarrow}{}^{1}_{\alpha}[\hat{s},(s_{1..k})_{[s_l/s'_l]},l,T]{=}s'$, when $s_l\overset{e}{\rightarrow}{}^{1}_{\alpha}s'_l$. Then,

$$\frac{s_l\overset{e}{\rightarrow}{}^{1}_{\alpha}s'_l}{t=[\hat{s},(s_{1..k},\tilde{s}),l,T]\overset{e}{\rightarrow}{}^{1}_{\alpha}[\hat{s},(s_{1..k[s_l/s'_l]},\tilde{s}),l,T]=t'}\ \text{OR-2}$$

  then the output actions are unchanged.

  OR-3 rule: Let $[\hat{s},(s_{1..k}),l,T]\overset{e}{\rightarrow}{}^{0}[\hat{s},(s_{1..k}),l,T]$, when $[\hat{s},(s_{1..k}),l,T]\overset{e}{\not\rightarrow}{}^{1}$. The hypothesis of the theorem does not hold.

- add-trans. Let $[\hat{s},(s_{1..k}),l,T]\succ_1[\hat{s},(s_{1..k}),l,T\cup\{t^*\}]$, where $t^*{\in}\mathsf{T}$. Again, three rules can be applied:

  OR-1 rule: Let $[\hat{s},(s_{1..k}),l,T]\overset{e}{\rightarrow}{}^{1}_{ex::\gamma::en}[\hat{s},(s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]},m,T]{=}s'$ by OR-1 rule, where $\langle\hat{t}{,}l{,}\_{,}e{,}\gamma{,}T_d{,}m{,}ht\rangle\neq t^*{\in}T$, $ex\in\mathsf{exit}(s_l)$ and $en\in\mathsf{entry}(\mathsf{next}(ht,T_d,s_m))$. Then,

$$\frac{\langle\hat{t}{,}l{,}S_r{,}e{,}\gamma{,}T_d{,}m{,}ht\rangle\in T,\quad S_r\subseteq\mathsf{conf}(s_l),\quad s_l\overset{e}{\not\rightarrow}{}^{1}}{t=[\hat{s},(s_{1..k}),l,T\cup\{t^*\}]\overset{e}{\rightarrow}{}^{1}_{ex::\gamma::en}[\hat{s},(s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]},m,T\cup\{t^*\}]=t'}\ \text{OR-1}$$

  then the output actions are unchanged.

  OR-2 rule: Let $[\hat{s},(s_{1..k}),l,T]\overset{e}{\rightarrow}{}^{1}_{\alpha}[\hat{s},(s_{1..k})_{[s_l/s'_l]},l,T]{=}s'$, when $s_l\overset{e}{\rightarrow}{}^{1}_{\alpha}s'_l$. Then,

$$\frac{s_l\overset{e}{\rightarrow}{}^{1}_{\alpha}s'_l}{t=[\hat{s},(s_{1..k}),l,T\cup\{t^*\}]\overset{e}{\rightarrow}{}^{1}_{\alpha}[\hat{s},(s_{1..k})_{[s_l/s'_l]},l,T\cup\{t^*\}]=t'}\ \text{OR-2}$$

  then the output actions are unchanged.

  OR-3 rule: Let $[\hat{s},(s_{1..k}),l,T]\overset{e}{\rightarrow}{}^{0}[\hat{s},(s_{1..k}),l,T]$, when $[\hat{s},(s_{1..k}),l,T]\overset{e}{\not\rightarrow}{}^{1}$. The hypothesis of the theorem does not hold.

- inside-or. Here we have that $[\hat{s},(s_{1..k}),l,T]\succ_1[\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i]},l,T']$, where $s_i\succ_1\tilde{s}_i$. For the sake of clarity, we split the proof in two cases:

  1. When $i\neq l$.

OR-1 rule: Let $[\hat{s}, (s_{1..k}), l, T] \overset{e}{\rightarrow}{}^{1}_{ex::\gamma::en}[\hat{s}, (s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]}, m, T]{=}s'$, where $\langle \hat{t}, l, \_, \_, \gamma, T_d, m, ht\rangle \in T$, $ex \in \mathsf{exit}(s_l)$ and $en \in \mathsf{entry}(\mathsf{next}(ht, T_d, s_m))$. Here we have two sub cases:

(a) When $i \neq m$.

$$\frac{\langle \hat{t}, l, S_r, e, \gamma, T_d, m, ht\rangle \in T, \quad S_r \subseteq \mathsf{conf}(s_l), \quad s_l \overset{e}{\not\rightarrow}{}^{1}}{t = [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T] \overset{e}{\rightarrow}{}^{1}_{ex::\gamma::en} [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][s_m/\mathsf{next}(ht,T_d,s_m)]}, m, T] = t'} \text{ OR-1}$$

then the output actions are unchanged.

(b) When $i = m$.

$$\frac{\langle \hat{t}, l, S_r, e, \gamma, T_d, m, ht\rangle \in T, \quad S_r \subseteq \mathsf{conf}(s_l), \quad s_l \overset{e}{\not\rightarrow}{}^{1}}{t = [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T] \overset{e}{\rightarrow}{}^{1}_{ex::\gamma::en'} [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][\tilde{s}_m/\mathsf{next}(ht,T_d,\tilde{s}_m)]}, m, T] = t'} \text{ OR-1}$$

By lemma 7, $en \triangleright en'$ since $\mathsf{next}(ht, T_d, s_m) \succ_1 \mathsf{next}(ht, T_d, \tilde{s}_m)$, by lemma 5. Then $ex::\gamma::en \triangleright ex::\gamma::en'$.

OR-2 rule: Let $[\hat{s}, (s_{1..k}), l, T] \overset{e}{\rightarrow}{}^{1}_{\alpha}[\hat{s}, (s_{1..k})_{[s_l/s'_l]}, l, T]{=}s'$, when $s_l \overset{e}{\rightarrow}{}^{1}_{\alpha} s'_l$. Then,

$$\frac{s_l \overset{e}{\rightarrow}{}^{1}_{\alpha} s'_l}{t = [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, l, T] \overset{e}{\rightarrow}{}^{1}_{\alpha} [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][s_l/s'_l]}, l, T] = t'} \text{ OR-2}$$

then the output actions are unchanged.

OR-3 rule: The hypothesis of the theorem does not hold.

2. When $i = l$. Note that if $s_i \overset{e}{\not\rightarrow}{}^{1}$ then $\tilde{s}_i \overset{e}{\not\rightarrow}{}^{1}$, because we are dealing with safe refinements. Three OR rules applies:

OR-1 rule: Let $[\hat{s}, (s_{1..k}), i, T] \overset{e}{\rightarrow}{}^{1}_{ex::\gamma::en}[\hat{s}, (s_{1..k})_{[s_m/\mathsf{next}(ht,T_d,s_m)]}, m, T]{=}s'$, where $\langle \hat{t}, i, S_r, e, \gamma, T_d, m, ht\rangle \in T$, $ex \in \mathsf{exit}(s_l)$ and $en \in \mathsf{entry}(\mathsf{next}(ht, T_d, s_m))$. Here we have two sub cases:

(a) When $i \neq m$.

$$\frac{\langle \hat{t}, i, S_r, e, \gamma, T_d, m, ht\rangle \in T, \quad S_r \subseteq \mathsf{conf}(\tilde{s}_i), \quad \tilde{s}_i \overset{e}{\not\rightarrow}{}^{1}}{t = [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, i, T] \overset{e}{\rightarrow}{}^{1}_{ex'::\gamma::en} [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][s_m/\mathsf{next}(ht,T_d,s_m)]}, m, T] = t'} \text{ OR-1}$$

By lemma 6, $ex \triangleright ex'$. Then $ex::\gamma::en \triangleright ex'::\gamma::en$. Note that $S_r \subseteq \mathsf{conf}(s_i) \Rightarrow S_r \subseteq \mathsf{conf}(\tilde{s}_i)$ holds by lemma 1.

(b) When $i = m$.

$$\frac{\langle \hat{t}, i, S_r, e, \_, T_d, m, ht\rangle \in T, \quad S_r \subseteq \mathsf{conf}(\tilde{s}_i), \quad \tilde{s}_i \overset{e}{\not\rightarrow}{}^{1}}{t = [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i]}, i, T] \overset{e}{\rightarrow}{}^{1}_{ex'::\gamma::en'} [\hat{s}, (s_{1..k})_{[s_i/\tilde{s}_i][\tilde{s}_m/\mathsf{next}(ht,T_d,\tilde{s}_m)]}, m, T] = t'} \text{ OR-1}$$

By lemma 6, $ex \triangleright ex'$. By lemma 7, $en \triangleright en'$ since $\mathsf{next}(ht, T_d, s_m) \succ_1 \mathsf{next}(ht, T_d, \tilde{s}_m)$, by lemma 5. Then $ex::\gamma::en \triangleright ex'::\gamma::en'$.

OR-2 rule: Let $[\hat{s},(s_{1..k}),i,T] \overset{e}{\rightarrow}{}^{1}_{\alpha}[\hat{s},(s_{1..k})_{[s_i/s'_i]},i,T]{=}s'$, when $s_i \overset{e}{\rightarrow}{}^{1}_{\alpha_i} s'_i$. We use the inductive hypothesis,

$$\dfrac{\dfrac{s_i \overset{e}{\rightarrow}{}^{1}_{\alpha_i} s'_i, \quad s_i \succ_1 \tilde{s}_i}{\tilde{s}_i \overset{e}{\rightarrow}{}^{1}_{\alpha'_i} \tilde{s}'_i}\ \textsf{I.H.}}{t = [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i]},i,T] \overset{e}{\rightarrow}{}^{1}_{\alpha'} [\hat{s},(s_{1..k})_{[s_i/\tilde{s}_i][\tilde{s}_i/\tilde{s}'_i]},i,T] = t'}\ \textsf{OR-2}$$

Since $\alpha_i \triangleright \alpha'_i$ by IH, we have $\alpha \triangleright \alpha'$.

OR-3 rule: The hypothesis of the theorem does not hold.

- ext-act-trans. Let $[\hat{s},(s_1,..,s_n),T]\succ_1[\hat{s},(s_1,..,s_n),T_{[r/r']}]$, where $r = \langle \hat{t}, l, S_r, e, \gamma, T_d, p, ht \rangle \in T$, $r' = \langle \hat{t}, l, S_r, e, \tilde{\gamma}, T_d, p, ht \rangle$ and $\gamma \triangleright \tilde{\gamma}$. We can apply three rules in this case:

  OR-1 rule: Let $[\hat{s},(s_{1..k}),l,T] \overset{e}{\rightarrow}{}^{1}_{ex::\gamma::en}[\hat{s},(s_{1..k})_{[s_p/\mathsf{next}(ht,T_d,s_p)]},p,T]{=}s'$ by OR-1 rule, where $\langle \hat{t}, l, \_, e, \gamma, T_d, p, ht \rangle \in T$, $ex \in \mathsf{exit}(s_l)$ and $en \in \mathsf{entry}(\mathsf{next}(ht,T_d,s_p))$. We have,

$$\dfrac{\langle \hat{t}, l, S_r, e, \tilde{\gamma}, T_d, p, ht \rangle \in T, \quad S_r \subseteq \mathsf{conf}(s_l), \quad s_l \overset{e}{\not\rightarrow}{}^{1}}{t = [\hat{s},(s_{1..k},\tilde{s}),l,T] \overset{e}{\rightarrow}{}^{1}_{ex::\tilde{\gamma}::en} [\hat{s},(s_1,..,s_n),T_{[r/r']}] = t'}\ \textsf{OR-1}$$

  then $ex{::}\gamma{::}en \triangleright ex{::}\tilde{\gamma}{::}en$.

  OR-2 rule: Let $[\hat{s},(s_{1..k}),l,T] \overset{e}{\rightarrow}{}^{1}_{\alpha}[\hat{s},(s_{1..k})_{[s_l/s'_l]},l,T]{=}s'$, when $s_l \overset{e}{\rightarrow}{}^{1}_{\alpha} s'_l$. The output actions are unchanged.

  OR-3 rule: Let $[\hat{s},(s_{1..k}),l,T] \overset{e}{\rightarrow}{}^{0}[\hat{s},(s_{1..k}),l,T]$, when $[\hat{s},(s_{1..k}),l,T] \overset{e}{\not\rightarrow}{}^{1}$. The hypothesis of the theorem does not hold.

- ext-act-en and ext-act-ex. These are trivial, because entry and exit actions of the superstate does not take part in the SOS rules.

7 Conclusions and Further Work

We proved that the structured operational semantics for UML Statecharts presented by [vdB02] can be extended using our extension relation without loosing any behaviour. Then, we can state that when a statechart is extended it is still possible to perform the same semantic transitions on it as before. As a consequence of this fact, the extension relation can be considered as a behavioral refinement, thus allowing us to represent the common and variant functionalities of a family of products in conjunction with feature diagrams as an incremetal process of statechart structure enrichment.
As for further work, we are now investigating the possibility of a relaxed definition of conf-all, in order to include incomplete parallel configurations, that is, a configuration in which is known the state of some of the parallel components, and not all of them. That is,

$$\begin{array}{ll}
\mathsf{econf\text{-}all}([\hat{s}]) & := \{\{\hat{s}\}\} \\
\mathsf{econf\text{-}all}([\hat{s},(s_{1..n}),T]) & := \{\{\hat{s}\} \cup c | \exists i \in \{1..n\}. c \in \mathsf{econf\text{-}all}(s_i)\} \cup \{\{\hat{s}\}\} \\
\mathsf{econf\text{-}all}[\hat{s},(s_{1..n})] & := \bigcup_{k=1}^{n}\{\{\hat{s}\} \cup \bigcup_{i=1..k} c_i | c_i \in \mathsf{econf\text{-}all}(s_i)\} \cup \{\{\hat{s}\}\}
\end{array}$$

The first two definitions are analogous to conf-all. The third one, allows as a valid configuration a parallel incomplete one, that is, the union of the configurations of the first k parallel states, for

$k = 1, 2, .., n$. The advantage of this relaxation is that we can obtain a finer extension relation, that is, we can substitute the rule

$$\frac{s_1, .., s_n \in SC_B, \quad \hat{s}_i \neq \hat{s}_j \ \forall_{i \neq j}, \quad \hat{s} \neq \hat{s}_i \ \forall_i}{[\hat{s}] \succ_1 [\hat{s}, (s_1, .., s_n)]} \ \textsf{ext-and}$$

by these two:

$$\frac{s' \in SC_B}{[\hat{s}] \succ_1 [\hat{s}, (s')]} \ \textsf{ext-and1} \qquad \frac{s' \in SC_B \quad \hat{s}' \neq \hat{s}_i \ \forall_i}{[\hat{s}, (s_1,..,s_n)] \succ_1 [\hat{s}, (s_1,..,s_n, s')]} \ \textsf{ext-and2}$$

We conjecture that, with this extension, the main theorems of this paper can be proved without changing the hypothesis. Note that, from the modeling point of view, we can obtain the same statecharts as before.
Moreover, we are working with additional features of UML-statecharts, for example, do actions, transition guards, initial and final states, join/fork pseudostates, junction pseudostate, choice pseudostate and entry/exit pseudostates. We already proved that the semantics of some of them can be equivalently modeled with standard components (for example entry and exit states).

References

[CGW05] M. Cengarle, P. Graubmann, and S. Wagner. Semantics of uml 2.0 interactions with variabilities. In *International Workshop on Formal Aspects of Component Software (FACS05)*, 2005.

[CHE05] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[CN02] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns.* Addison Wesley, 2002.

[Cza98] K. Czarnecki. *Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models.* PhD thesis, Technical University of Ilmenau, 1998.

[GL10] A. Gonzalez and C. Luna. Specification of products and product lines. *CoRR*, abs/1001.4436, 2010.

[Gom05] H. Gomaa. *Designing Software Product Lines with UML.* Addison Wesley, 2005.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *North-Holland*, 1987.

[SV08] N. Szasz and P. Vilanova. Statecharts and variabilities. In P. Heymans, K. C. Kang, A. Metzger, and K. Pohl, editors, *VaMoS*, ICB Research Report, pages 131–140, 2008.

[vdB02] M. von der Beeck. A structured operational semantics for uml-statecharts. *Software and Systems Modeling*, 1(2):130–141, December 2002.

[ZHJ04] T. Ziadi, L. Helouet, and J. Jezequel. Behaviors generation from product lines requirements. In *UML2004 Workshop on Software Architecture Description and UML*, 2004.