Hierarchical states in the Compositional Interchange Format

H. Beohar D.E. Nadales Agut D.A. van Beek P.J.L. Cuijpers
Eindhoven University of Technology (TU/e)
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
{H.Beohar, D.E.Nadales.Agut, D.A.van Beek, P.J.L. Cuijpers}@tue.nl

CIF is a language designed for two purposes, namely as a specification language for hybrid systems and as an interchange format for allowing model transformations between other languages for hybrid systems. To facilitate the top-down development of a hybrid system and also to be able to express models more succinctly in the CIF formalism, we need a mechanism for stepwise refinement. In this paper, we add the notion of hierarchy to a subset of the CIF language, which we call hCIF$^{\subset}$. The semantic domain of the CIF formalism is a hybrid transition system, constructed using structural operational semantics. The goal of this paper is to present a semantics for hierarchy in such a way that only the SOS rules for atomic entities in hCIF$^{\subset}$ are redesigned in comparison to CIF. Furthermore, to be able to reuse existing tools like simulators of the CIF language, a procedure to eliminate hierarchy from an automaton is given.

1 Introduction

Modeling languages for hybrid systems, and hybrid automata in particular, are designed to combine computational and physical aspects of a system in one formal model. The compositional interchange format (CIF), presented in [3, 14], is a hybrid modeling language based on hybrid automata [9], but with the semantics defined via structural operational semantics (SOS) [12] rules. One of the primary aims of CIF is to establish inter-operability among a wide range of tools by means of model transformations to and from the CIF. In addition, it is possible to specify hybrid systems using CIF, and perform simulation. The reason for using an SOS semantics in an automaton-based framework is that the model transformations to and from CIF are not only to be executed on 'complete' models, but also on components of bigger models. Thus, it is crucial that bisimulation (equivalence) is a congruence for all the constructs of the CIF. This is guaranteed using the process-tyft format of [11].

The CIF language contains the following features.

- Predicates in the locations of the automata that constrain the initial values and/or initial locations (init predicate), time behavior (invariants and time can progress predicates), and action behavior (invariants).
- Communication among automata using channels and shared variables.
- Scoping operators for declaring variables, actions, and channels.
- An initialization operator for restricting the initial conditions of variables. This allows initialization on a more global level as compared to the init predicates of automata.
- A synchronization operator for executing actions synchronously in parallel automata.
- An urgency operator for declaring actions or channels as urgent.

To facilitate the top-down development of a hybrid system in the CIF formalism we need a mechanism for stepwise refinement. In this work we develop such hierarchical extensions for a subset of CIF called

Submitted to:
SOS 2010

© H. Beohar, D.E. Nadales Agut, D.A.v.Beek, & P.J.L.Cuijpers

hCIF$^{\subset}$, in which we leave out the constructs for scoping, initialization, synchronization and urgency. Nevertheless, the semantics presented here is general enough to allow us to incorporate these concepts in a straightforward manner. In a later phase, we plan to extend hCIF$^{\subset}$ to contain these constructs again. This requires us to already take into account some semantic features that are particular for the CIF (like the so-called variable trajectories, guard trajectories and termination trajectories that we discuss further on in this paper) while other features only appear in a reduced form (like the so-called environment transitions that are only used for establishing termination, and not for establishing consistency of a state).

Stepwise refinement is a framework for designing a system correct by construction. The following steps are involved in the stepwise refinement framework as laid out in [1]. One starts with design of a system at a higher level of abstraction, and usually the model designed at this level is called an abstract model. Then a concrete model is designed by adding more behavior into the abstract model such that the concrete model is a refinement of the abstract model. This process of refining is performed until a desired implementation is reached. Thus, any formalism that supports stepwise refinement, must incorporate the following two main things. First, it should provide a way to add new details in an abstract model. Secondly, it should provide *at-least* sufficient conditions under which a concrete model is a refinement of a given abstract model by construction. In this paper, we consider only the first aspect of stepwise refinement.

Consequently, we introduce a notion of hierarchy in hCIF$^{\subset}$ that allows a straightforward way to add new behavior to a given model. In the past, the following techniques were proposed in order to accommodate stepwise refinement in other formalisms.

- Action refinement [19]. In this approach an action in the alphabet of a process or an automaton is substituted by another process/automaton. However, the setting of action refinement is incompatible with the interleaving models of concurrency as pointed out in [15]. Since the CIF and hCIF$^{\subset}$ formalisms are based on interleaving models of concurrency, we disregard this technique of refinement.

- Statecharts [8]. Statecharts were the first formalism that extended finite state machines with the concept of hierarchy. Conventionally, the semantics of statecharts requires a tree-structure on the set of locations of a statechart. Consequently, additional concepts from tree-structures, like least common ancestors, children of a location, etc., make the semantics complicated. We show in the current work that these additional concepts are unnecessary when reverting to a structural operational semantics. We only need to introduce the notion of a substructure. Other concepts of state-structures, like AND-states and OR-states, can be expressed through the *parallel composition* and the *multiple initial locations* of substructures, respectively. The concepts of history retention and inter-level transitions (not considered in this paper) are not supported directly in hCIF$^{\subset}$, but they can be emulated.

- Hierarchical timed automata [6, Chap 4.] are the extensions of statecharts with a finite set of clock variables modeling real time. Again the semantics of this formalism is based on the concepts of tree structures and for this reason we also disregard this approach. However, there is a common intuition about the passage of time in [6] with the current work. The time can pass in a hierarchical structure only if the time can pass in all the levels of hierarchy, i.e. time transitions must synchronise in all the levels of hierarchy of a hierarchical automaton.

- State refinement operator [15]. State refinement is a binary operator on process algebraic processes written as $p[q]$ where p, q are arbitrary process terms. Informally, it means that p is a state with the substructure q. In other words, a location of an automaton is allowed to contain another automaton

representing its substructure. Furthermore, it was also stated [15] that the above way of introducing hierarchy is compatible with the interleaving models of concurrency. Thus, the present work is motivated by the work carried out in [15], even though the basic entity in our formalism is an automaton rather than an action.

The semantics of the hCIF$^{\subset}$ formalism is a hybrid transition system (HTS) [5] constructed using structural operational semantics (SOS) [12]. The goal of this paper is to show how the semantics of hierarchical automata can be defined using SOS rules, in a compositional way. We do so without introducing the complexity of state tree structures in the formalism, and in such a way that the semantics of other CIF operators remain unaffected. As an additional result, we define an algorithm for eliminating hierarchy, which enable us to reuse existing tools for implementing an hCIF$^{\subset}$ simulator.

The remainder of this paper is organized as follows. First, the subset of CIF , which is extended with hierarchy, is presented in Section 2. Once the basic concepts are introduced, Section 3 introduces the syntactical extensions that are needed to add support for hierarchy in CIF , and we discuss the design-decisions that led to such extensions. The formal semantics of hCIF$^{\subset}$ is presented in Section 4, where we illustrate how the concept of hierarchy can be defined in a compositional and recursive way. In Section 5, we give a procedure to eliminate the hierarchy from a hCIF$^{\subset}$ model. Finally, in Section 6 we make some conclusive remarks and discuss future work.

2 Introduction to CIF

This section presents the syntax and semantics of a subset of CIF [3], because presenting the full syntax and semantics would distract too much from the message of the paper. Still, we have included all the aspects of the CIF that were involved in the design-decisions we made when defining hierarchy.

As was mentioned in the introduction, the CIF language is based on hybrid automata, which model a combination of computational and physical behavior of a system by mixing automata theory with the theory of algebraic differential equations. A key feature of the CIF is that it provides a structural operational semantics for such atomic (hybrid) automata, which makes the definition of more complex compositions of automata easier. The only compositions defined in this paper will be parallel composition and hierarchy, but in [3] many more are described.

Informally, a basic CIF automaton is shown in Figure 1 that models the dynamics of a thermostat in a room. This thermostat can be in one of two computational states. It is either off, or on. This is reflected by the two circles (called *locations*) labeled *Off* and *On*. Next to the label, the locations also contain equations (called *time-can-progress predicates* or *tcp-predicates*) that model the physical behavior of the system while it is in this computational state. In our example, the temperature T will behave according to the differential equation $\dot{T} = -T + 15$ when the thermostat is off and according to $\dot{T} = -T + 25$ when the thermostat is on. The dotted version of variables, like $\dot{T}$ in the example, are used for modeling derivatives. The execution of a calculation generally results in a change of location, which is modeled by an arrow (called *edge*) from one location to another. Edges are labeled by *actions* (in our example *switch-on* and *switch-off*) that may be used to synchronize the behavior of automata in a composition (not shown here). Furthermore, they contain a predicate (called *guard*) that determines under which condition an action can be executed, and a predicate (called *reset*) that determines whether there is a change in any of the model variables. In case of the thermostat, the *switch-on* action can only be executed if the temperature T is lower than 20, and the action results in a change of the variable n to $n + 1$, which counts the number of times the thermostat is switched on. Notation n^+ is used to refer to the value of n after the execution of the action. The guard $n \leq 1000$ disables the *switch-off* action, modeling that the thermostat breaks down

after a thousand switches and leaves the room hot. Every location contains an *initialization predicate*, which determines whether execution can start in that location. In the example, initially the thermostat is switched off, the temperature in the room is $T = 25$ and the counter is set to $n = 0$. This is modeled by an incoming edge in the *Off* location without any origin, labeled by the predicate $T = 25 \wedge n = 0$. The absence of incoming arrows on location *Off* denotes that the initialization predicate is false (which means that execution cannot start on that location).

Formally, the locations of an atomic CIF automata are taken from the set $\mathscr{L}$. Actions belong to the set $\mathscr{A}$. We distinguish the following types of variables: regular variables, denoted by the set $\mathscr{V}$; the dotted versions of those variables, which belong to the set $\dot{\mathscr{V}} \triangleq \{\dot{x} \mid x \in \mathscr{V}\}$; and *step variables*, which belong to the set $\{x^+ \mid x \in \mathscr{V} \cup \dot{\mathscr{V}}\}$. Furthermore, the variables can be classified according their continuous evolution (i.e. how their values change during time delays). In particular, we distinguish between *discrete variables* (n in the previous example), whose values remain constant during time delays, and the value of their dotted versions are always 0; and *continuous variables* (T in the previous example), whose values evolve as a continuous function of time during delays, and whose dotted versions represent their derivatives. Variables are constrained by differential algebraic equations and we implement them as predicates. The values of the variables belong to the set Λ that contains, among else, the sets $\mathbb{B}$, $\mathbb{R}$, and $\mathbb{C}$. Guards, tcp and initialization predicates, and reset predicates are taken from the sets $\mathscr{P}_g$, $\mathscr{P}_t$, and $\mathscr{P}_r$, respectively.

The exact syntax and semantics of predicates are left as a parameter of our theory, as we are not interested in the computational aspects of CIF in this paper. In the examples presented here, and in the tool implementations of CIF, $\mathscr{P}_g$, $\mathscr{P}_t$, and $\mathscr{P}_r$ are terms of the language of predicate logic [13], where for $\mathscr{P}_g$ and $\mathscr{P}_t$ the variables are taken from the set $\mathscr{V} \cup \dot{\mathscr{V}}$, and for $\mathscr{P}_r$ the variables are taken from the set $\mathscr{V} \cup \dot{\mathscr{V}} \cup \{x^+ \mid x \in \mathscr{V} \cup \dot{\mathscr{V}}\}$.

Given these preliminaries, an atomic automaton can be defined as follows.

Definition 2.1 (Atomic automaton)**.** An atomic automaton is a tuple $(V, \text{init}, \text{tcp}, E)$ with a set of locations $V \subseteq \mathscr{L}$; initial and time-can-progress predicates init, tcp: $V \to \mathscr{P}_t$; and a set of edges $E \subseteq V \times \mathscr{P}_g \times \mathscr{A} \times \mathscr{P}_r \times L$.

We use symbol $\mathscr{M}$ to refer to the set of all atomic automata. Atomic automata, as the one shown before, can be used to build more complex models by using the parallel composition operator. CIF includes more operators, but we do not discuss them in this paper. Throughout this work, we use the term *composition* to refer either to an atomic automata, or to a parallel composition of automata, which is denoted as $p \parallel q$, for compositions p and q, where the set $S \subseteq \mathscr{A}$ is the set of actions that must be executed synchronously in both automata. The set $\mathscr{C}$ contains all hCIF$^{\subset}$ compositions, and is formally defined in Section 3.

It is not possible to present the formal semantics of CIF in this paper due to the lack of space. However, if an automaton has no hierarchy (i.e. no location contains a composition), the rules presented here match those of CIF.

After introducing the base language, we are ready to show how it can be extended with hierarchy.

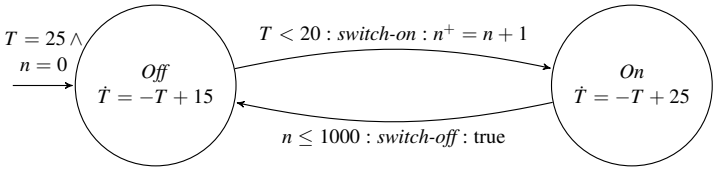


Figure 1: A model of thermostat.

3 Adding hierarchy to CIF

In this section, we show the syntactical extensions that are needed for adding hierarchy to CIF , explaining why every new element is required according to the design-decisions we have taken.

An automaton is said to be *hierarchical* if it contains a composition in at least one of its locations. Extending CIF with hierarchy is easy to achieve with the addition of a hierarchy function h to the elements of an automaton, such that $h(v)$ returns the composition contained in v, which we refer to as *substructure*. We find this hierarchy function to be more suitable for the automata theoretic framework of CIF than the state refinement operator [15], which was conceived for process algebraic setting. This is because, unlike in process algebra, the development of the CIF is aimed at modeling convenience rather than at finding the smallest representation of a given construct.

As an example, suppose we want to extend the model of the thermostat presented in Figure 1, so that it is only switched off after a certain time has elapsed (which allows the room to be heated up). Having hierarchy, a refined model of the thermostat can be elaborated as in Figure 2. We define the hierarchy function h such that $\mathit{Off} \notin \mathrm{dom}(h)$; and $h(\mathit{On})$ returns the automaton shown at the bottom in Figure 2, which initially sets up a clock c (continuous variable), and it waits until the timer expires $\Delta \leq c$ (where $0 < \Delta$) to generate the event *done*.

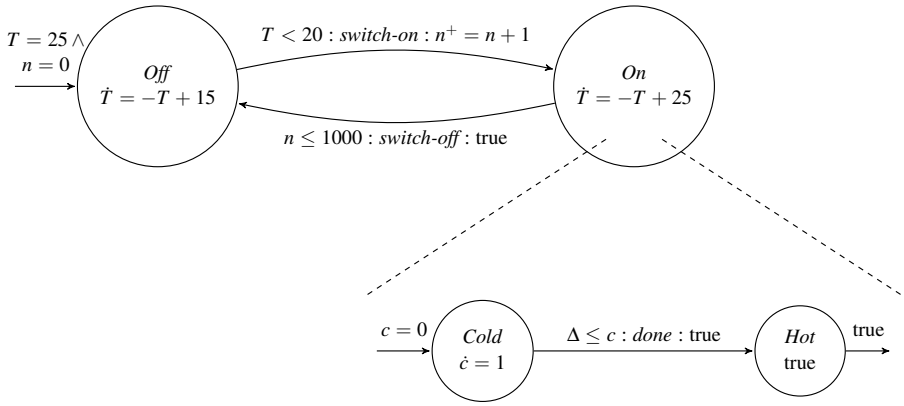


Figure 2: Hierarchical model of the thermostat.

An important thing to note about the substructure in figure 2, is that one of the states contains an outgoing arrow that leads nowhere. The predicate on this arrow is called a *termination predicate* and it is taken, just like tcp and initialization predicates, from the set $\mathscr{P}_t$.

Termination predicates were not part of the atomic automata of CIF before, even though they are common in the general theory of automata. Our reason for adding them is that we need a mechanism to decide when a substructure hands over the control of the execution flow to the superstate to which it belongs.

In most hierarchical formalisms, such as [16] and [10], the actions enabled in the super-automaton are executed regardless of the state of the substructure. The example in Figure 2, however, illustrates that a more general approach, in which the substructure has control over the superstructure, may be useful. The example actually uses the fact that the substructure has control over the superstructure to restrict the behavior of the thermostat in such a way that it is forced to stay in the *On* location for a certain time. The termination mechanism that we have chosen, originates from our desire to be able to (partially) translate sequential function charts [2] to CIF. In that formalism, termination is used as the standard mechanism

to pass control from one chart to the next.

Another mechanism that we need when dealing with hierarchy, is a way to keep track of the so-called *active location* that an automaton is in. Admittedly, the active location of an automaton is a semantic concept, used to keep track of the current state when describing the dynamics of an automaton, and it does not belong in a section describing the syntax. However, in the structured operational semantics of the CIF, the states are formed by pairs of syntactic descriptions and valuations of variables (see section 4 for details), and our solution that deals with the semantic problem of keeping track of the active location, makes use of an auxiliary syntactic construct that one should not use while modeling, but that formally is part of the syntax of CIF.

In the non-hierarchical CIF, the active location of an automaton is fully determined by its initialization predicate. As a result, a state-change in the semantics is modeled by changing this initialization predicate. In the hierarchical CIF, however, changing the initialization predicates of a substructure means changing the hierarchy function h, and this causes so-called history retention; i.e. if a substructure is terminated and the automaton gets back to it later, the substructure will restart where it was ended previously, because h is still in its altered form. From the example in Figure 2, one can see that this is not always desirable. History retention in that example, would make that the thermostat is only forced to linger in the *On* state the first time it is entered. In subsequent visits, the substructure would already be in its terminating state immediately.

The semantics we would like to give to hierarchical CIF, is that a substructure is restarted every time from one of its (original) initial states. Our solution for the history retention problem, is to introduce an auxiliary composition operator $p : \alpha$ which should be read as α is currently in the substructure p. Here, p is an arbitrary composite automaton and α is an atomic hierarchical automaton. Next, the initialization predicate of α can be used to model the active location of the super-automaton α, while the initialization predicate(s) of (the components) p are used to model the active location of the active substructure. We found that the use of this auxiliary operator greatly simplifies the structured operational semantics of hierarchical CIF.

Now that we have informally introduced all the syntactic elements needed for extending CIF with hierarchy, we define the syntax of hierarchical automata as follows.

Definition 3.1. An atomic hierarchical hybrid automaton is a tuple $(V, \text{init}, \text{tcp}, E, \text{term}, h)$ where, $(V, \text{init}, \text{tcp}, E)$ is an atomic CIF automaton, $\text{term} : V \to \mathscr{P}$ is a function that associate to each location a predicate describing the conditions under which a location is final, and $h : V \rightharpoonup \mathscr{C}$ is a (partial) hierarchy function that maps each location with a composite automaton. The set of composite automata $\mathscr{C}$ in hCIF$^{\subset}$ is recursively defined by the following grammar $\mathscr{C} ::= \alpha \mid \mathscr{C} : \alpha \mid \mathscr{C} \parallel_S \mathscr{C}$.

Henceforth, we use Greek letters α and α' to indicate an atomic hierarchical hybrid automaton and Roman letters p, q, p', and q' to indicate any composite automaton in $\mathscr{C}$.

4 Formal semantics of hCIF$^{\subset}$

In this section we illustrate how the concept of hierarchy can be defined in a compositional way, without introducing additional concepts of tree-structures present in the statechart [8] formalism. First, the semantic framework is set up, and then the SOS rules are presented.

The semantics of hCIF$^{\subset}$ compositions (and CIF) is given in terms of SOS rules, which induce hybrid transition systems (HTS) [5]. The states of the HTS are of the form $\langle p, \sigma \rangle$, where $p \in \mathscr{C}$ is a composition and $\sigma : \mathscr{V} \cup \dot{\mathscr{V}} \to \Lambda$ is a function, called *valuation*, which assigns values to the variables. Valuations capture the phenomenon of discrete change in the values of variables caused by the execution of actions

in an automaton. We denote the set of all valuations as Σ. There are three kind of transition in the HTS, namely, *action transitions*, *environment transitions*, and *time transitions*. We describe them in detail next.

Action transition are of the form $\langle p, \sigma \rangle \xrightarrow{a} \langle p', \sigma' \rangle$, and they model the execution of action a by process p in an initial valuation σ, which changes process p into p' and results in a valuation σ'.

Environment transitions are of the form $\langle p, \sigma \rangle \overset{b}{\dashrightarrow} \langle p', \sigma' \rangle$, and in the full CIF language, they are used to model which possible behavior of the environment is consistent with that of the composition p, but cannot be executed by the component itself. In the restricted language hCIF$^{\subset}$, the function of the environment transitions is to indicate that a composition p can initialize to become a composition p' in which an active location is fixed for each (active) substructure. Furthermore, the boolean b indicates whether the initialized substructure can terminate, and thus give back the control over actions to their environment.

Time transitions are of the form $\langle p, \sigma \rangle \overset{\rho,\theta,\omega}{\longmapsto} \langle p', \sigma' \rangle$, and they model the passage of time in composition p, in an initial valuation σ, which results in a composition p' and valuation σ'. The relation between p and p' is the same as for environment transitions. Function $\rho : \mathbb{T} \to \Sigma$ is called *variable trajectory*, and it models the evolution of variables during the time delay. For each time point $s \in \mathrm{dom}(\rho)$, and for each variable $x \in \mathscr{V} \cup \dot{\mathscr{V}}$, the function application $\rho(s)(x)$ yields the value of variable x at time s. Function $\theta : \mathbb{T} \to 2^{\mathscr{A}}$ is called *guard trajectory*, and it models the evolution of enabled actions during time delays. For each time point $s \in \mathrm{dom}(\theta)$, the function application $\theta(s)$ yields the set of enabled actions of composition p at time s. Lastly, function ω is called *termination trajectory*, and it models the evolution of termination during time delays: for each time point $s \in \mathrm{dom}(\omega)$, composition p' is terminating at time s if and only if $\omega(s)$. For all time transition $\mathrm{dom}(\rho) = [0, t]$, for some time point $t \in \mathbb{T}$, and $\mathrm{dom}(\rho) = \mathrm{dom}(\theta) = \mathrm{dom}(\omega)$.

Guard trajectories were shown to be necessary for the definition of urgency and variable abstraction in CIF and other hybrid formalisms [4, 18]. Termination trajectories allow us to keep track of the possibility of termination over time, and they are essential for constructing the guard trajectories in the rules. Even though these concepts are not necessary for giving semantics to hCIF$^{\subset}$, they allow us to solve the problem of supporting urgency and variable abstraction in a hierarchical setting, and thus our approach can be extended to the whole CIF without modifying the rules.

Even though predicates are abstract entities, we assume that there is a satisfaction relation $\sigma \models e$ is defined, which expresses that predicate $e \in \mathscr{P}_t \cup \mathscr{P}_g \cup \mathscr{P}_r$ is satisfied (i.e. it is true) in valuation σ. For predicate logic, this relation can be defined in a standard way (see [13] for example). For a valuation σ, we define $\sigma^+ \triangleq \{(v^+, c) \mid (v, c) \in \sigma\}$.

Definition 4.1 formalizes the hybrid transition system induced by the SOS rules presented in the next sections.

Definition 4.1. A hybrid transition system (HTS) is a six-tuple of the form $(Q, \mathscr{A}, \to, \longmapsto, \dashrightarrow)$ where, $Q \triangleq \mathscr{C} \times \Sigma$, $\to \subseteq Q \times \mathscr{A} \times Q$, $\longmapsto \subseteq Q \times \big((\mathbb{T} \to \Sigma) \times (\mathbb{T} \to 2^{\mathscr{A}}) \times (\mathbb{T} \to \mathbb{B})\big) \times Q$, and $\dashrightarrow \subseteq Q \times \mathbb{B} \times Q$.

4.1 Hierarchical hybrid automaton

In this section, we give semantics to hierarchical hybrid automata. We use notation α to refer to an atomic automaton $(V, \mathrm{init}, \mathrm{tcp}, E, \mathrm{term}, h)$, and $\alpha[v]$ to refer to the automaton $(V, \mathrm{id}_v, \mathrm{tcp}, E, \mathrm{term}, h)$, where $\mathrm{id}_v(w) \triangleq v = w$.

In absence of hierarchy, an atomic automaton α can perform an action in a location v and initial valuation σ if there is an edge (v, g, a, r, v') such that the following conditions hold:

1. Location v is active ($\sigma \models \text{init}(v)$).
2. Guard g holds ($\sigma \models g$).
3. It is possible to find a new valuation σ' such that the reset predicate is satisfied in valuation $\sigma \cup \sigma'^{+}$ ($\sigma \cup \sigma'^{+} \models r$). We do not write $\sigma' \models r$ since, in general, r refers to the next values of variables, which are contained in σ'^{+}.

The above conditions are summarized in the term $\sigma, \sigma' \models_\alpha (v, g, a, r, v')$, which is syntactically equivalent to condition $(v, g, a, r, v') \in E \wedge \sigma \models \text{init}(v) \wedge \sigma \models g \wedge \sigma'^{+} \cup \sigma \models r$.

Rule 1 describes what is semantically involved with the addition of hierarchy. Firstly, it is necessary to check that the substructure of the initial location, if any, is terminating (condition $\langle h(v), \sigma \rangle \overset{\text{true}}{\dashrightarrow} \langle p, \sigma \rangle \vee v \notin \text{dom}(h)$). Finally, after the action is performed, the substructure in the target location, if present, must be initialized (condition $\langle \alpha, \sigma \rangle \xrightarrow{a} \langle q : \alpha[v'], \sigma' \rangle$). Note that the choice of selecting active locations of substructure $h(v')$ is made upon entering location v'. Example 1 illustrates this phenomenon, which we call *eager choice*.

$$\frac{\sigma, \sigma' \models_\alpha (v, g, a, r, v'), \Big(\langle h(v), \sigma \rangle \overset{\text{true}}{\dashrightarrow} \langle p, \sigma \rangle \vee v \notin \text{dom}(h)\Big), \quad v' \in \text{dom}(h), \langle h(v'), \sigma' \rangle \overset{b}{\dashrightarrow} \langle q, \sigma' \rangle}{\langle \alpha, \sigma \rangle \xrightarrow{a} \langle q : \alpha[v'], \sigma' \rangle} \quad (1)$$

Example 1. Consider the composite automaton shown in Figure 3a in which the tcp function gives the predicate true for all the vertices of the automaton. The idea behind eager choice, is that after the execution of the action a, an initial state of the substructure is picked immediately. This can only result in the left state of the substructure to be picked, because of the value of x is set to 1 during the execution of a. Hence, the action c in the substructure will never be executed. The resulting transition system generated by the SOS is shown in Figure 3b, where the states are depicted as circles and their components are not shown. **End of Example.**

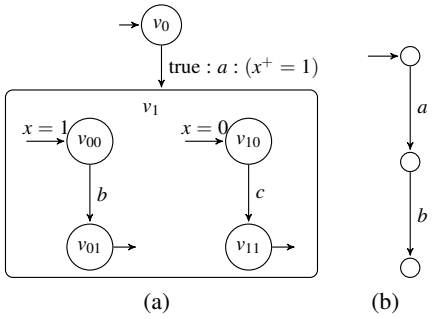


Figure 3: (a) Automaton with two possible initial states. (b) Resulting HTS.

Rule 1 requires as a condition that there is an active substructure in the target location $v' \in \text{dom}(h)$. If this is not the case then no active substructure is prefixed to $\alpha[v]$, as expressed by Rule 2.

$$\frac{\sigma, \sigma' \models_\alpha (v, g, a, r, v'),\ v' \notin \text{dom}(h), \Big(\langle h(v), \sigma \rangle \overset{\text{true}}{\dashrightarrow} \langle p, \sigma \rangle \vee v \notin \text{dom}(h)\Big)}{\langle \alpha, \sigma \rangle \xrightarrow{a} \langle \alpha[v'], \sigma' \rangle} \quad (2)$$

In a hierarchical setting, actions in an automaton can be generated by the substructure of an active location. Rule 3 formalizes this. Note that in the conclusion, $p : \alpha[v]$ reflects the fact that an initial location is chosen in a hierarchical structure if the substructure performs an action.

$$\frac{\sigma \models \text{init}(v), v \in \text{dom}(h), \langle h(v), \sigma \rangle \xrightarrow{a} \langle p, \sigma' \rangle}{\langle \alpha, \sigma \rangle \xrightarrow{a} \langle p : \alpha[v], \sigma' \rangle} \quad (3)$$

In CIF, a time delay is possible in an active location v if there exists a trajectory ρ such that the tcp predicate is satisfied in $[0,t)$. Henceforth, we will use $t, \rho \models_\alpha \langle \text{init}(v), \text{tcp}(v) \rangle$ as an abbreviation of the predicate $\rho(0) \models \text{init}(v) \wedge \text{dom}(\rho) = [0,t] \wedge 0 \leq t \wedge \forall s \in [0,t).[\rho(s) \models \text{tcp}(v)]$.

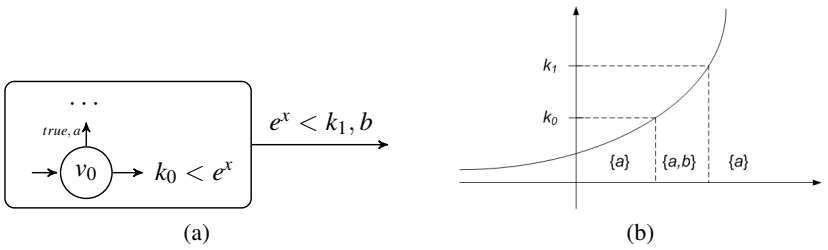
For time delays, in hCIF$^{\subset}$ the substructure must perform a time transition with the same trajectory, and we consider conjunction of all the tcp predicates of all the active locations of an automaton. In this way time passes in an automaton, and also in all of its contained substructures. This is, an automaton and its substructure synchronize in the time delays. In the complete extension of CIF with hierarchy, a similar approach is taken for invariants. Rule 4 models this, where $\text{dom}(\omega) = \text{dom}(\rho)$, $\text{dom}(\theta) = \text{dom}(\rho)$, $\forall_{s\in[0,t]}.\omega(s) = \omega_0(s) \wedge \rho(s) \models \text{term}(v)$, and $\forall_{s\in[0,t]}.\theta(s) = \theta_0(s) \cup \{a \mid (v, g, a, r, v') \in E \wedge \rho(s) \models g \wedge \omega_0(s)\}$. The guard trajectory θ as well as the termination trajectory ω are constructed by using the corresponding trajectories generated by the time transition in the substructure.

We found that this approach is the simple and intuitive: substructures are part of the whole structure, and it is strange from the modeling point of view to have a system in which time can "freeze" for certain parts of the system. Furthermore, since there are several active locations in different levels of hierarchy, it is not clear which one to choose to perform the time delays. The example of the thermostat with hierarchy, depicted in Figure 2, shows the convenience of this decision, since we want the clock to advance while the room heats up.

$$\frac{t, \rho \models_\alpha \langle \text{init}(v), \text{tcp}(v) \rangle,\ \langle h(v), \rho(0) \rangle \overset{\rho,\theta_0,\omega_0}{\longmapsto} \langle p, \rho(t) \rangle}{\langle \alpha, \rho(0) \rangle \overset{\rho,\theta,\omega}{\longmapsto} \langle p : \alpha[v], \rho(t) \rangle} \quad (4)$$

The following example illustrates the need for having termination trajectories to capture properly the set of enabled actions during time delays.

Example 2. Consider the automaton shown in Figure 4a and assume $1 < k_0 < k_1$. Then the set of enabled actions at the active location will depend on the function e^x and this set is illustrated in Figure 4b. In other words, if $0 \leq x < \ln(k_0)$ then the set of enabled actions is $\{a\}$. If $\ln(k_0) \leq x < \ln(k_1)$ then the set of enabled actions is $\{a, b\}$. And if $x \geq \ln(k_1)$ then the set of enabled actions is $\{a\}$. **End of Example.**



(a) (b)

Figure 4: Illustration of the dependence of enabled actions over time.

Rule 5 deals with the case when an initial location v does not contain a substructure, where $\mathrm{dom}(\omega) = \mathrm{dom}(\rho)$, $\mathrm{dom}(\theta) = \mathrm{dom}(\rho)$ and $\forall_{s\in[0,t]}.\omega(s) = \rho(s) \models \mathrm{term}(v)$, and $\forall_{s\in[0,t]}.\theta(s) = \{a \mid (v,g,a,r,v') \in E \wedge \rho(s) \models g\}$.

$$\frac{t,\rho \models_\alpha \langle \mathrm{init}(v), \mathrm{tcp}(v)\rangle,\ v \notin \mathrm{dom}(h)}{\langle \alpha, \rho(0)\rangle \overset{\rho,\theta,\omega}{\longmapsto} \langle \alpha[v], \rho(t)\rangle} \quad (5)$$

In CIF , if an automaton performs an environment transition then an unique active location is chosen. When hierarchy is incorporated, the substructure is initialized as well. This is expressed by Rule 6. The initialized composition p becomes the active substructure of $\alpha[v]$, and the automaton is terminating if the location and the active substructure are. Rule 7 deals with the case where there is no substructure.

$$\frac{\sigma \models \mathrm{init}(v), \langle h(v), \sigma\rangle \overset{b}{\dashrightarrow} \langle p, \sigma'\rangle}{\langle \alpha, \sigma\rangle \overset{\sigma\models \mathrm{term}(v)\wedge b}{\dashrightarrow} \langle p : \alpha[v], \sigma'\rangle} \quad (6) \qquad \frac{\sigma \models \mathrm{init}(v), v \notin \mathrm{dom}(h)}{\langle \alpha, \sigma\rangle \overset{\sigma\models \mathrm{term}(v)}{\dashrightarrow} \langle \alpha[v], \sigma'\rangle} \quad (7)$$

4.2 Automaton postfix operator

We now define the SOS rules for the automaton postfix operator, which helps in defining the overall behavior of a hierarchical automaton.

Intuitively, the composition $p : \alpha$ means that composition p is the active substructure of some initial location $v \in V$ in the automaton α. Note, that whenever the automaton postfix is only used as an auxiliary operator, this initial location will always be uniquely specified.

If we now look at the structure of a state (p, σ) in the hybrid transition system, it becomes clear how the postfix operator helps us to mimick the state-tree structures used in the semantics of statecharts [8]. Figure 5 shows that a composition p in essence *is* a tree, where the postfix operator represents the edges of the tree and the parallel compositions represents the branching. The root of this tree is the active location of the automaton we described, while the leaves are the active substructures where the control over actions currently lies. Indeed, an informal comparison of our semantics to that of statecharts suggest that that the AND-superstates of statecharts are represented as (asynchronous) parallel compositions $\|_\emptyset$, while OR-superstates are represented by having multiple locations for which the initialization predicate holds.

$$((p_1 : \alpha_1) \parallel (p_2 : \alpha_2)) : \alpha_0$$

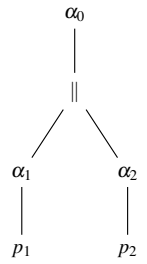


Figure 5: Relation between automaton postfix operator and state tree structures

The semantics of $p : \alpha$ is reminiscent of the sequential composition of untimed process algebra, i.e. the composite automaton p in $p : \alpha$ will perform action transitions until it is terminating, after that the automaton α can perform its action transitions. The difference between the sequential composition

operator and the automaton postfix operator is due to the difference in the passage of time caused by these operators. In the automaton postfix operator, the passage of time is synchronized between the first and second component, whereas in sequential composition the passage of time is not synchronized. There the second component waits for termination of the first, in a similar way as with action transitions.

All the rules presented here are similar to the those presented in the previous section. The difference lies in that in the rules of this section function h is not considered, since there is an active substructure p in the target state of every transitions appearing in the conclusions. Rule 8 models the action transition when the substructure is terminating.

$$\frac{\sigma, \sigma' \models_\alpha (v, g, a, r, v'), \langle p, \sigma\rangle \overset{\text{true}}{\dashrightarrow} \langle p', \sigma\rangle, \langle h(v'), \sigma'\rangle \overset{b}{\dashrightarrow} \langle q, \sigma'\rangle}{\langle p : \alpha, \sigma\rangle \overset{a}{\to} \langle q : \alpha[v'], \sigma'\rangle} \quad (8)$$

Rule 9 also models the action transition generated from a postfix operator when the substructure is terminating and the target location v' does not contain a substructure. Rule 10 models the action transition which is a result of the execution of the substructure.

$$\frac{\sigma, \sigma' \models_\alpha (v, g, a, r, v'), \langle p, \sigma\rangle \overset{\text{true}}{\dashrightarrow} \langle p', \sigma\rangle, v' \notin \text{dom}(h)}{\langle p : \alpha, \sigma\rangle \overset{a}{\to} \langle \alpha[v'], \sigma'\rangle} \quad (9) \qquad \frac{\langle p, \sigma\rangle \overset{a}{\to} \langle q, \sigma'\rangle}{\langle p : \alpha, \sigma\rangle \overset{a}{\to} \langle q : \alpha, \sigma'\rangle} \quad (10)$$

Finally, Rule 11 models the passage of time in an automaton postfix such that the timed transitions are synchronized in every level of hierarchy $p : \alpha$, where, $\text{dom}(\omega) = \text{dom}(\rho)$, $\text{dom}(\theta) = \text{dom}(\rho)$ and $\forall_{s\in[0,t]}.\omega(s) \triangleq \omega_0(s) \wedge \rho(s) \models \text{term}(v)$, and $\forall_{s\in[0,t]}.\theta(s) \triangleq \theta_0(s) \cup \{a \mid (v, g, a, r, v) \in E \wedge \rho(s) \models g \wedge \omega_0(s)\}$.

$$\frac{t, \rho \models_\alpha \langle \text{init}(v), \text{tcp}(v)\rangle, \langle p, \rho(0)\rangle \overset{\rho,\theta_0,\omega_0}{\longmapsto} \langle p', \rho(t)\rangle}{\langle p : \alpha, \rho(0)\rangle \overset{\rho,\theta,\omega}{\longmapsto} \langle p' : \alpha[v], \rho(t)\rangle} \quad (11)$$

Rule 12 models the execution of environment transition in an automaton postfix.

$$\frac{\langle p, \sigma\rangle \overset{b}{\dashrightarrow} \langle p', \sigma'\rangle}{\langle p : \alpha, \sigma\rangle \overset{\sigma\models\text{term}(v)\wedge b}{\dashrightarrow} \langle p' : \alpha[v], \sigma'\rangle} \quad (12)$$

4.3 Parallel composition operator

The parallel composition operator allows synchronisation of equally labeled action transitions between any two components that are specified by the synchronisation set $S \subseteq \mathscr{A}$. Rule 13 models this fact.

$$\frac{\langle p, \sigma\rangle \overset{a}{\to} \langle p', \sigma'\rangle, \langle q, \sigma\rangle \overset{a}{\to} \langle q', \sigma'\rangle, a \in S}{\begin{array}{c}\langle p \parallel_S q, \sigma\rangle \overset{a}{\to} \langle p' \parallel_S q', \sigma'\rangle \\ \langle q \parallel_S p, \sigma\rangle \overset{a}{\to} \langle q' \parallel_S p', \sigma'\rangle\end{array}} \quad (13)$$

Rule 14 models the interleaving of action transitions that do not belong to the synchronisation set S. Note the presence of environment transition in the premise of the following rule. This allows the other component (which does not perform an action transition, in the following case q) to get initialised.

$$\frac{\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle, \langle q,\sigma\rangle \overset{b}{\dashrightarrow} \langle q',\sigma'\rangle, a \notin S}{\begin{array}{c}\langle p \parallel_S q,\sigma\rangle \xrightarrow{a} \langle p' \parallel_S q',\sigma'\rangle \\ \langle q \parallel_S p,\sigma\rangle \xrightarrow{a} \langle q' \parallel_S p',\sigma'\rangle\end{array}} \quad (14)$$

In a parallel composition, time can pass it is can pass in each component individually, as it can be seen in Rule 15, where $\forall s \in [0,t].\big[\theta_{01}(s) = (\theta_0(s) \cap \theta_1(s)) \cup (\theta_0(s) \setminus S) \cup (\theta_1(s) \setminus S)\big]$, and $\forall s \in [0,t].[\omega_{01}(s) = \omega_0(s) \wedge \omega_1(s)]$. The guard trajectory is constructed (same as in CIF) from the guard trajectories of the composite automata interleaving in the parallel composition: at a given time point s, an action is enabled in the parallel composition $p \parallel_S q$ if it is enabled in p and q (regardless of whether the action is in S), or if it is enabled in p or q and is not synchronizing in the other component. The termination trajectory in the parallel composition at a given time point s is the conjunction of the termination trajectories of the respective components at the same time instant s.

$$\frac{\langle p,\rho(0)\rangle \overset{\rho,\theta_0,\omega_0}{\longmapsto} \langle p',\rho(t)\rangle, \langle q,\rho(0)\rangle \overset{\rho,\theta_1,\omega_1}{\longmapsto} \langle q',\rho(t)\rangle}{\begin{array}{c}\langle p \parallel_S q,\rho(0)\rangle \overset{\rho,\theta_{01},\omega_{01}}{\longmapsto} \langle p' \parallel_S q',\rho(t)\rangle \\ \langle q \parallel_S p,\rho(0)\rangle \overset{\rho,\theta_{01},\omega_{01}}{\longmapsto} \langle q' \parallel_S p',\rho(t)\rangle\end{array}} \quad (15)$$

The initialization of a parallel composition (Rule 16) is the initialization of its components. The termination predicate in the parallel composition is the conjunction of the termination predicates of the respective components.

$$\frac{\langle p,\sigma\rangle \overset{b_0}{\dashrightarrow} \langle p',\sigma'\rangle, \langle q,\sigma\rangle \overset{b_1}{\dashrightarrow} \langle q',\sigma'\rangle}{\begin{array}{c}\langle p \parallel_S q,\sigma\rangle \overset{b_0\wedge b_1}{\dashrightarrow} \langle p' \parallel_S q',\sigma'\rangle \\ \langle q \parallel_S p,\sigma\rangle \overset{b_0\wedge b_1}{\dashrightarrow} \langle q' \parallel_S p',\sigma'\rangle\end{array}} \quad (16)$$

4.4 Stateless bisimulation

It is clear from the definition of a HTS (Definition 4.1) that a state in a transition system consists of a process part (a behavioural entity) and a data part (valuation). Furthermore, we know that the stateless bisimulation is the most robust equivalence for the transition systems whose states contains data [11]. This subsection shows that the semantics of hCIF$^{\subset}$ is compositional with respect to stateless bisimulation [11], i.e. stateless bisimulation is a congruence for all operators of hCIF$^{\subset}$.

Definition 4.2. A symmetric relation $R \subseteq \mathscr{C} \times \mathscr{C}$ is called a stateless bisimulation [11] relation iff the following transfer conditions hold.

- $\forall p,p',\sigma,\sigma',a,q.\Big[\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle \wedge (p,q) \in R \Rightarrow \exists q'.\big[\langle q,\sigma\rangle \xrightarrow{a} \langle q',\sigma'\rangle \wedge (p',q') \in R\big]\Big]$.
- $\forall p,p',\sigma,\sigma',\rho,\theta,\omega,q.\Big[\langle p,\sigma\rangle \overset{\rho,\theta,\omega}{\longmapsto} \langle p',\sigma'\rangle \wedge (p,q) \in R \Rightarrow \exists q'.\big[\langle q,\sigma\rangle \overset{\rho,\theta,\omega}{\longmapsto} \langle q',\sigma'\rangle \wedge (p',q') \in R\big]\Big]$.
- $\forall p,p',\sigma,\sigma',b,q.\Big[\langle p,\sigma\rangle \overset{b}{\dashrightarrow} \langle p',\sigma'\rangle \wedge (p,q) \in R \Rightarrow \exists q'.\big[\langle q,\sigma\rangle \overset{b}{\dashrightarrow} \langle q',\sigma'\rangle \wedge (p',q') \in R\big]\Big]$.

Two composite automata p,q are said to be stateless bisimilar (denoted $p \underline{\leftrightarrow}_s q$) iff there exists a stateless bisimulation relation R such that $(p,q) \in R$.

Theorem 4.3. *Stateless bisimulation is a congruence for all the constructs of* hCIF$^{\subset}$.

Proof. The SOS rules of hCIF$^{\subset}$ are in the *process-tyft* format, which guarantees the congruence for stateless bisimilarity [11]. □

5 Elimination of hierarchy

In this section, we present a technique that converts a hierarchical automaton into an automaton in which the hierarchical function h is empty, such that they are stateless bisimilar. Such techniques in general, are known as *linearization* or *elimination* of operators[17]. The advantage of *flattening* a hierarchical automaton is that it allows the reuse of existing tools like simulators of the CIF language, which are only developed for flat automata.

Definition 5.1. The *depth* $D(p)$ of an composite automaton $p \in \mathscr{C}$ is recursively defined to be the 1 + $\max_{v\in\mathrm{dom}(h)}\ D(h(v))$ when p is a hierarchical automaton of the form $(V, \mathrm{init}, \mathrm{tcp}, E, \mathrm{term}, h)$, and is defined as $\max(D(q), D(r))$ whenever $p = q \parallel_S r$. An automaton is called *well founded* whenever its depth is defined. An automaton of depth 1 (i.e. with $\mathrm{dom}(h) = \emptyset$) is also called a *flat* automaton.

Suppose that we have a procedure $\mathscr{E}$ that turns any composition of flat automata into a stateless bisimilar flat automaton. In particular, suppose that $\mathscr{E}(\alpha \parallel_S \alpha') \leftrightarrows_s \alpha \parallel_S \alpha'$ whenever α and α' are flat automata. Then, we can lift this procedure to any well-founded composite automaton $p \in \mathscr{C}$, by first applying it to all components of p before applying it to p itself. We define $\mathscr{E}(p \parallel_S q) = \mathscr{E}(\mathscr{E}(p) \parallel_S \mathscr{E}(q))$ and $\mathscr{E}((V, \mathrm{init}, \mathrm{tcp}, E, \mathrm{term}, h)) = \mathscr{E}((V, \mathrm{init}, \mathrm{tcp}, E, \mathrm{term}, \tilde{h}))$ with $\tilde{h}(v) = \mathscr{E}(h(v))$, for any p, q and $(V, \mathrm{init}, \mathrm{tcp}, E, \mathrm{term}, h)$ of depth greater than 2. Structural induction on the depth of the composite automaton, combined with the congruence obtained in theorem 4.3, then gives us $\mathscr{E}(p) \leftrightarrows_s p$ for all well-founded composite automata p.

Such a procedure is already known for all the usual operations of the CIF[7], and next, we will give it for hierarchical automata.

Definition 5.2. Let $\alpha \in \mathscr{C}$ be an automaton of depth 2 of the form $(V, \mathrm{init}, \mathrm{tcp}, E, \mathrm{term}, h)$, such that $h(v)$ is a flat automaton for all $v \in \mathrm{dom}(h)$. We define $\mathscr{E}(\alpha) = (\hat{V}, \hat{\mathrm{init}}, \hat{\mathrm{tcp}}, \hat{E}, \hat{\mathrm{term}}, \emptyset)$ where,

- The set $\hat{V}$ of locations of the flat automaton $\mathscr{E}(\alpha)$ is defined by:

$$\hat{V} \triangleq \bigcup_{v\in V} \left\{ (v,w) \middle| \begin{array}{l} \big(v \notin \mathrm{dom}(h) \wedge w = \bot\big) \quad \vee \\ \big(h(v) = (V', \mathrm{init}', \mathrm{tcp}', E', \mathrm{term}', \emptyset) \wedge w \in V'\big) \end{array} \right\}$$

- The predicate-functions $\hat{\square}$, with $\square \in \{\mathrm{init}, \mathrm{tcp}, \mathrm{term}\}$, are defined for each $\hat{v} \in \hat{V}$ by:

$$\hat{\square}(\hat{v}) \triangleq \begin{cases} \square(v), & \text{if } \hat{v} = (v, \bot) \\ \square(v) \wedge \square'(w), & \text{if } (\hat{v} = (v,w) \wedge h(v) = (V', \mathrm{init}', \mathrm{tcp}', E', \mathrm{term}', \emptyset) \wedge w \in V') \end{cases}$$

- The edges $(\hat{v}_0, g', a, r', \hat{v}_1)$ of the flat automaton $\mathscr{E}(\alpha)$ are present (i.e. $(\hat{v}_0, g', a, r', \hat{v}_1) \in \hat{E}$) iff one of the following conditions hold:
  1. $\hat{v}_0 = (v_0, w_0) \wedge \hat{v}_1 = (v_1, w_1)$ for some $v_0, v_1 \in V$ such that $h(v_i) = (V_i', \mathrm{init}_i', \mathrm{tcp}_i', E_i', \mathrm{term}_i', \emptyset)$, $w_i \in V_i'$, for $i \in \{0, 1\}$ and

$$(v_0, g, a, r, v_1) \in E \wedge g' = \big(\mathrm{term}_0'(w_0) \wedge g\big) \wedge r' = \big(r \wedge \mathrm{init}_1'(w_1)^+\big) \quad .$$

  Note that if $w_0 = \bot$ ($w_1 = \bot$) then by defining $\mathrm{term}_0'(w_0) = \mathrm{true}$ ($\mathrm{init}_1'(w_1) = \mathrm{true}$) we get definitions for the simpler cases derived from the above one.

2. $\hat{v}_0 = (v, w_0) \wedge \hat{v}_1 = (v, w_1)$ for some $v_0 \in V$ such that

$$h(v) = (V', \text{init}', \text{tcp}', E', \text{term}', \emptyset) \wedge w_0, w_1 \in V' \wedge (w_0, g', a, r', w_1) \in E' \ .$$

Figure 6 shows the resulting automaton after applying the linearization procedure to the thermostat model extended with a clock (Figure 2).

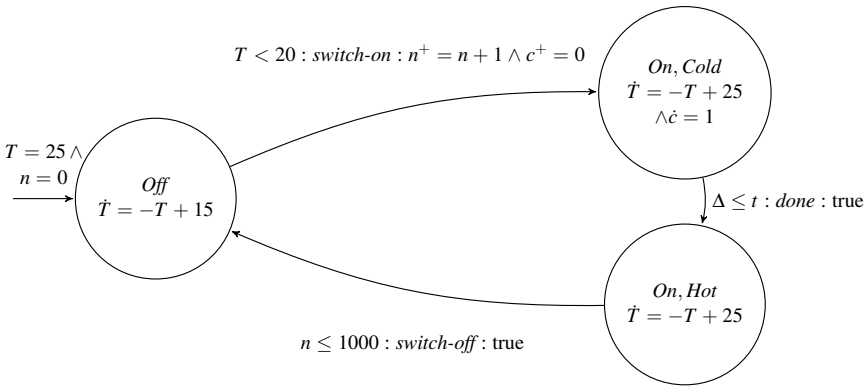


Figure 6: A flat model of thermostat with clock.

Next we prove the correctness of our linearization procedure as defined in Definition 5.2.

Theorem 5.3. *Let $\alpha \in \mathscr{C}$ be an automaton of depth 2 of the form $(V, \text{init}, \text{tcp}, E, \text{term}, h)$, such that $h(v)$ is a flat automaton for all $v \in \text{dom}(h)$, then $\alpha \leftrightarrow_s \mathscr{E}(\alpha)$.*

Proof. Fix $\hat{\alpha} = \mathscr{E}(\alpha) = (\hat{V}, \hat{\text{init}}, \hat{\text{tcp}}, \hat{E}, \hat{\text{term}}, \emptyset)$. It is rather straightforward but tedious to verify that the relation $R = \{(\alpha, \hat{\alpha}), (\alpha[v], \hat{\alpha}[(v, \perp)]), (h(v)[w] : \alpha[v], \hat{\alpha}[(v, w)]) \mid v \in V \ \wedge (v, w) \in \hat{V}\}$, is a witnessing stateless bisimulation. □

6 Conclusions

In this paper we illustrated how to add hierarchy to a subset of the CIF (called hCIF$^{\subset}$) in a compositional way, and we showed that the SOS rules of atomic entities can be modified without altering the rules of the CIF operators. Moreover, the rules are formulated in such a way that the addition of concepts such as urgency and invariants can be incorporated easily without altering the rules presented here. However, the usability of hCIF$^{\subset}$ is not yet investigated and the plan is to evaluate it by performing industrial case-studies within the context of the MULTIFORM project [1] after extending it with the remaining operators of the CIF. A procedure to eliminate hierarchy was given in order to be able to use the existing tools associated with CIF. Note that Definition 5.2 presented here is a relatively inefficient one to implement. It can be further optimized, for example, by disallowing the edges in the set $\hat{E}$ that are never executed for any valuation.

As ongoing work, we are researching a branching version of stateless bisimulation for hybrid transition systems in order to handle τ action as 'invisible'. Using such a notion, it becomes possible to formalize when a stepwise refinement is a correct refinement of an abstract model. Thus we can attack the second aspect of stepwise refinement mentioned in the introduction.

Acknowledgements: The authors would like to thank Jos Baeten, Koos Rooda and Mohammad Mousavi, and the reviewers of SOS, for their constructive comments that very much helped to improve this paper.

This work has been performed as part of the Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems (MULTIFORM) project, supported by the Seventh Research Framework Programme of the European Commission. Grant agreement number: INFSO-ICT-224249.

References

[1] Integrated multi-formalism tool support for the design of networked embedded control systems : Multiform. `http://cms.multiform.bci.tu-dortmund.de/`.

[2] Nanette Bauer, Ralf Huuck, Ben Lukoschus, and Sebastian Engell. S.: A unifying semantics for sequential function charts. in: this volume. In *SoftSpez Final Report, volume 3147 of LNCS*, page 400, 2004.

[3] D. A. van Beek, P. Collins, D. E. Nadales, J.E. Rooda, and R. R. H. Schiffelers. New concepts in the abstract format of the compositional interchange format. In *ADHS 2009*, pages 250–255. IFAC, 2009.

[4] P.J.L. Cuijpers and M.A. Reniers. Lost in translation: Hybrid-time flows vs real-time transitions. In *HSCC 2008*, volume 4981 of *Lecture Notes in Computer Science*, pages 116–129. Springer, 2008.

[5] P.J.L. Cuijpers, M.A. Reniers, and W.P.M.H. Heemels. Hybrid transition systems. Technical Report CS-Report 02-12, TU/e, Eindhoven, Netherlands, 2002.

[6] A. David. *Hierarchical Modeling and Analysis of Timed Systems*. PhD thesis, Uppsala University, November 2003.

[7] Tim Engels. CIF to CIF model transformations. Master's thesis, Eindhoven university of technology, In preparation.

[8] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[9] T.A.H. Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, volume 170 of *NATO ASI Series F: Computer and Systems Science*, pages 265–292. Springer-Verlag, 2000.

[10] Edward A. Lee. Finite state machines and modal models in ptolemy ii. Technical Report UCB/EECS-2009-151, EECS Department, University of California, Berkeley, Nov 2009.

[11] M.R. Mousavi, M.A. Reniers, and J.F. Groote. Notions of bisimulation and congruence formats for SOS with data. *Information and Computation*, 200(1):104–147, 2005.

[12] G.D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[13] John C. Reynolds. *Theories of programming languages*. Cambridge University Press, New York, NY, USA, 1999.

[14] Systems Engineering Group. The Compositional Interchange Format for Hybrid Systems. http://se.wtb.tue.nl/sewiki/cif/start, 2010.

[15] A. Uselton and S. Smolka. State refinement in process algebra. In *Proceedings of the North American Process Algebra Workshop*, Ithaca, New York, August 1993.

[16] Andrew C. Uselton and Scott A. Smolka. A compositional semantics for statecharts using labeled transition systems. In *CONCUR '94: Concurrency Theory*, volume 836 of *Lecture Notes in Computer Science*, pages 2–17. Springer Berlin / Heidelberg, 1994.

[17] Y.S. Usenko. *Linearization in μCRL*. PhD thesis, Technische Universiteit Eindhoven (TU/e), 2002.

[18] D.A. van Beek, P.J.L. Cuijpers, J. Markovski, D.E. Nadales Agut, and J.E. Rooda. Reconciling urgency and variable abstraction in a hybrid compositional setting. In *FORMATS 2010*, volume to appear of *Lecture Notes in Computer Science*. Springer, 2010.

[19] R. J. van Glabbeek. *Comparative concurrency semantics and refinement of actions*. PhD thesis, CWI, 1990.