# The Hierarchical Compositional Interchange Format

D.E. Nadales Agut, D.A. van Beek, H. Beohar, P.J.L. Cuijpers, and J. Fonteijn

Eindhoven University of Technology (TU/e)
The Netherlands

{D.E.Nadales.Agut, D.A.v.Beek, H.Beohar, P.J.L.Cuijpers}@tue.nl

**Abstract.** In computer science, the development of hierarchical automata / statecharts has lead to stepwise development of complex discrete systems. Such a concept is absent in the Compositional Interchange Format (CIF), which is a modelling language based on hybrid automata. In this article we extend the CIF language with the concept of hierarchy, which results in the Hierarchical Compositional Interchange format (HCIF). Syntactically, hierarchy is introduced by adding three concepts to CIF: a hierarchy function from a location to a HCIF composition, a termination predicate, and disruptive edges. The semantics of HCIF is given by means of Structural Operational Semantics rules. The semantics of a hierarchical automaton is defined in a compositional manner, by referring only to the transition system of the substructures, and not to their syntactic representation. This compositional introduction of hierarchy allows us to keep the semantics of the HCIF operators almost unchanged with respect to their CIF versions. Finally, a case-study called Patient Support System is modelled in HCIF to show its applicability[1].

## 1 Introduction

Hierarchy provides a structured and economical description of complex systems [14], which is suitable for incremental (bottom-up) construction of correct systems [2]. It also provides a framework for the development of abstraction and refinement techniques.

The compositional interchange format (CIF) [4, 3, 1], is a modeling language based on hybrid automata, which aims to establish interoperability among a wide range of formalisms and associated tools for the specification of hybrid and timed systems, by means of model transformations to and from CIF. In this way, implementation of many bilateral translators, is avoided. CIF has a formal semantics defined in terms of Structured Operational Semantics (SOS) [11] rules.

In [5], the addition of hierarchy to a subset of CIF is investigated, and as a result it is shown that the SOS rules of atomic entities can be modified without

[1] Work done as part of the European Community's Seventh Framework Programme (FP7/2007-2013) project MULTIFORM contract number FP7-ICT-224249

altering the rules of the CIF operators. However the question remains whether this approach can be extended when more complicated concepts such as invariants[8], synchronization, and control variables [7] are added to the language.

In this paper we develop and extension of the full CIF language with hierarchy, named the Hierarchical Compositional Interchange Format (HCIF), and we apply the extension to a case study to show how the new concepts can be applied.

## 2 Syntax of HCIF

In this section we describe the mathematical syntax of HCIF, and we illustrate it by modelling a controller of a simplified Patient Support System of an MRI scanner, which is discussed in more detail in Section 5. Note that in this section we give an incomplete description of the controller model to illustrate the various concepts involved in the definition of a hierarchical automaton.

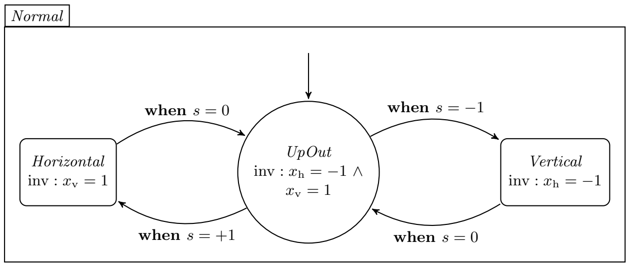


**Fig. 1.** Movement control.

Fig. 1 gives an informal, graphical representation of a HCIF automaton, which models a controller of a patient support table. The control operates in one of the following three modes:

1. *Horizontal*: horizontal movement of the table
2. *UpOut*: table fully up and out
3. *Vertical*: vertical movement of the table

Every location has an initialization predicate, an *invariant* predicate, and a *time can progress predicate* associated to it. The initialization predicate of a location $l$ describes the constraints that the initial values of variables must satisfy for an execution to start in $l$. Such locations for which the initialization predicate is true are called *active locations*. The *invariant* of a location $l$ is a predicate that must hold as long as the system is in $l$. The *time can progress* (tcp) predicate of a location $l$ is a predicate that must hold during time delays, when $l$ is an active location. In Fig. 1, the location *UpOut* has true as the initialization predicate, $x_h = -1 \wedge x_v = 1$ as the invariant. Tcp predicates are in general useful for triggering the execution of an action from a location within a certain period of time. For instance, an action $a$ must be executed when the clock value has reached 2 units of time (See Fig. 3(a)).

*Edges* represent discrete changes in the computational state of a system. An edge has a *source* and a *target* location, and its execution results in a change of active location (unless the edge is a self loop). The automaton of Fig. 1 has four edges in total among the locations *Horizontal*, *Stopped*, and *Vertical*.

Every edge contains a predicate called *guard* that determines when an action can be executed, a predicate called *update* that determines how the model variables can change after performing the action, and a set of *jumping variables* that specify the variables that are changed by the action. Edges are labeled by *actions* that may be used to synchronize the behavior of automata in a parallel composition. In Fig. 1, the edge from the location *UpOut* to the location *Horizontal* has the guard $s = 1$, update predicate true, empty set of jumping variables, and the silent action label $\tau$.

Formally, the set of locations is denoted by $\mathcal{L}$ and the set of actions is denoted by $\mathcal{A}$. The invisible action $\tau$ is a special symbol, which is not present in the set $\mathcal{A}$ and we fix $\mathcal{A}_\tau = \mathcal{A} \cup \{\tau\}$. In HCIF there are three types of variables: regular variables, denoted by the set $\mathcal{V}$; the dotted versions of those variables, which belong to the set $\dot{\mathcal{V}} = \{\dot{x} \mid x \in \mathcal{V}\}$; and the step variables, which belong to the set $\{x^+ \mid x \in \mathcal{V} \cup \dot{\mathcal{V}}\}$. The notation $x^+$ denotes the value of a variable $x$ in the next state. Furthermore, the variables can be classified according to their evolution (i.e. how their values change during time delays). In particular, we distinguish between discrete variables (such as $s$ in Fig. 1), whose values remain constant during time delays, so that the values of their dotted versions are always 0; and continuous variables (such as $x_h$ and $x_v$ in Fig. 1), whose values evolve as a continuous function of time during delays, and whose dotted versions represent their derivatives. Variables can also be constrained by differential algebraic equations, which are specified as predicates (in invariants). The values of the variables belong to the set $\Lambda$ that contains, among others, the sets $\mathbb{B}$ (booleans) and $\mathbb{R}$ (reals). The predicates representing the guards are taken from the set $\mathcal{P}_g$, the tcp, invariants and initializations are taken from the set $\mathcal{P}_t$ and the resets are taken from the set $\mathcal{P}_r$. The exact syntax and semantics of predicates are defined in [1]. The predicates $\mathcal{P}_g, \mathcal{P}_t$ and $\mathcal{P}_r$ are the terms of the language of predicate logic [12], where for $\mathcal{P}_g, \mathcal{P}_t$ the variables are taken from the set $\mathcal{V} \cup \dot{\mathcal{V}}$, and for $\mathcal{P}_r$ the variables are taken from the set $\mathcal{V} \cup \dot{\mathcal{V}} \cup \{x^+ \mid x \in \mathcal{V} \cup \dot{\mathcal{V}}\}$.

Locations can contain other automata (or compositions of them, as we show in Section 5). In Fig. 1 the location *Horizontal* contains the automaton shown in Fig. 2, that defines the horizontal movement of the controller in more detail. Automata that are contained inside other locations are referred to as *sub-automata* or *sub-structure*, and the containing automata are referred to as *super-automata* or *super-structure*. In a HCIF automaton, there are two types of edges, namely, *non-disruptive* edges (for brevity, we refer to a non-disruptive edge as an edge) and *disruptive* edges. Intuitively, an edge can be executed from a location if the sub-structure at that location is terminating, while a disruptive edge can be ex-

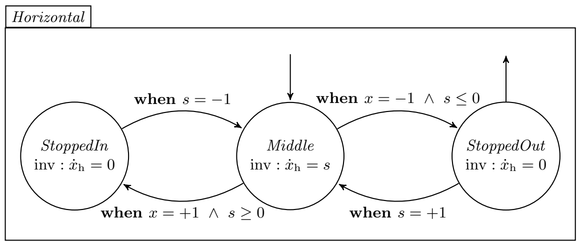


**Fig. 2.** Horizontal movement.

ecuted even if the sub-structure at that location is non-terminating. Note that the conditions under which an edge or a disruptive edge can be executed depend on several factors, which are defined in Section 4.

In addition to initialization, invariant and tcp predicates, each location has a *termination predicate* which defines if execution can terminate in that location. Termination predicates are used for specifying when the super-structure can perform a transition. In the automaton shown in Fig. 1, the $\tau$ transition from the location *Horizontal* to the location *UpOut* can be executed only if the guard $s = 0$ holds, the automaton (Fig. 2) inside the location *Horizontal* has *StoppedOut* as its active location, and the termination predicate holds.

Additional components of an automaton (not shown in the example presented here) include: *control variables*, *synchronizing actions*, and *dynamic type* mappings. Intuitively, controlled variables are those variables that can only be modified by the automaton that declares them, and they do not change arbitrarily after performing an action. The set of synchronizing actions is used to specify which actions are to be synchronized when the automaton is composed in parallel. The concept of dynamic types [9] is used to model constraints in the joint evolution of a variable and its dotted version. In CIF a dynamic type is a set containing pairs of functions, whose domain is a closed range of the form $[0, t]$, with $t \in \mathbb{T}$. Notation $\mathbb{T}$ is used to refer to the set of all time points.

**Definition 1 (Hierarchical automata).** *A hierarchical automaton $\alpha$ is a tuple*

$$(V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, D, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}, \mathrm{term}, \mathrm{h})$$

*where:*

- $V \subseteq \mathcal{L}$ *a set of locations,*
- *initial, invariant, time-can-progress and termination predicates* init*,* inv*,* tcp*,* term$\colon V \to \mathcal{P}_t$*,*
- *a set of edges* $E \subseteq V \times \mathcal{P}_g \times \mathcal{A} \times (2^{\mathcal{V} \cup \dot{\mathcal{V}}} \times \mathcal{P}_r) \times V$*,*
- $D \subseteq E$ *is the set of* disruptive *edges of the automaton,*
- $\mathrm{var}_C \subseteq 2^{\mathcal{V}}$ *is the set of controlled variables,*
- $\mathrm{act}_S \subseteq 2^{\mathcal{A}}$ *is the set of synchronizing actions, and*
- $\mathrm{dtype} : \mathcal{V} \rightharpoonup 2^{(\mathcal{T} \to \Lambda) \times (\mathcal{T} \to \Lambda)}$ *is the dynamic type mapping.*

– $\mathrm{h} : V \rightharpoonup \mathcal{C}$ *is a partial function that associates to some set of locations a substructure. Here,* $\mathcal{C}$ *is the set of all compositions in HCIF (See Definition 2).*

*We use symbol* $\mathcal{M}$ *to refer to the set of all hierarchical automata.*

Using operators, more complex models, referred to as compositions, are possible. They are defined below in the syntax of HCIF compositions. The semantics of the operators is presented in Section 4.1, with the exception of the semantics of the action and variable scope operators. The semantics of these operators is unchanged with respect to the semantics of these operators in CIF, as defined in [1].

**Definition 2 (HCIF compositions).** *The set of compositions* $\mathcal{C}$ *in the HCIF formalism is recursively defined by the grammar below, where* $x \in \mathcal{V}$*,* $e \in \mathcal{E}$*,* $a \in \mathcal{A}$*,* $a_\tau \in \mathcal{A}_\tau$*.*

| | |
|---|---|
| $\mathcal{C} ::= \alpha$ | hierarchical automaton |
| $\mid \mathcal{C} : \alpha$ | automaton postfix operator |
| $\mid \mathcal{C} \parallel \mathcal{C}$ | parallel composition |
| $\mid [\![_{\mathrm{V}}\, x = e, \dot{x} = e :: \mathcal{C}\, ]\!]$ | variable scope operator |
| $\mid [\![_{\mathrm{A}}\, a :: \mathcal{C}\, ]\!]$ | action scope operator |
| $\mid \upsilon_{a_\tau}(\mathcal{C})$ | urgency operator |

Throughout this article, the textual and graphical conventions given in Table 1 are followed.

## 3 Semantic framework

In this section, the semantic framework is set up to properly explain the semantics of HCIF. First we present the concepts of variable valuations and flow trajectories. Next we describe informally hybrid transitions systems, which are used to model the semantics of HCIF compositions. Finally, a formal definition of this semantic model is given.

### 3.1 Preliminaries

Semantically, the *execution* of a system, specified by means of a HCIF composition, causes changes to the values of the variables appearing on it. Thus, in the semantic model it is necessary to represent the values of the variables in a particular instant. For this purpose, we use the concept of *valuation*, which is standard in semantics of processes with data. A valuation $\sigma \colon (\mathcal{V} \cup \dot{\mathcal{V}}) \to \Lambda$ is a function that for each variable returns its corresponding value. We use notation $\Sigma \triangleq (\mathcal{V} \cup \dot{\mathcal{V}}) \to \Lambda$ to refer to the set of all valuations.

Having defined valuations, we introduce the concept of satisfiability. Even though predicates are abstract entities, we assume that a satisfaction relation $\sigma \models u$ is defined, which expresses that predicate $u \in \mathcal{P}$ is satisfied (i.e. it is true) in valuation $\sigma$. For a valuation $\sigma$, we define $\sigma^+ \triangleq \{(v^+, c) \mid (v, c) \in \sigma\}$.

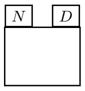
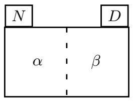
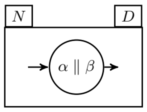
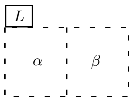
| **Graphical representation** | **Meaning** |
|---|---|
| ○ | Location without any sub-structure |
| →○ | Initial location with init predicate true |
| ○→ | Final location with termination predicate true |
| ▢ | Location containing a sub-structure |
| **when** $g$ **act** $a$ **do** $x := e$ | Edge $(g, a, (\{x\}, x^+ = e))$ |
| ↯ **when** $g$ **act** $a$ **do** $x := e$ | Disruptive edge $(g, a, (\{x\}, x^+ = e))$ |
| **when** $g$ | Edge $(g, \tau, (\emptyset, \text{true}))$ |
| **act** $a$ | Edge $(\text{true}, a, (\emptyset, \text{true}))$ |
|  | Automaton $N$ with declarations $D$ |
|  |  |
|  | AND superstate $L$ containing parallel composition $\alpha \parallel \beta$ |

**Table 1.** Textual and graphical conventions in HCIF.

To model the evolution on the values of variables during time delays we use the concept of *variable trajectories*. A variable trajectory is a function $\rho\colon \mathbb{T} \rightharpoonup \Sigma$ that returns the valuations of the variables at each time point. In other words, $\rho(s)(x)$ is the value of variable $x$ at time $s$. We assume the domain of variable trajectories to be closed intervals, i.e. intervals of the form $[0, t]$, where $t \in \mathbb{T}$.

## 3.2 Hybrid transition systems

The semantics of CIF compositions is given in terms of SOS rules, which induce hybrid transition systems (HTS) [6]. The states of the HTS are of the form $\langle p, \sigma \rangle$, where $p \in \mathcal{C}$ and $\sigma \in \Sigma$ is a valuation. There are three kind of transition in the HTS, namely, *action transitions*, *environment transitions*, and *time transitions*.

Action transitions are of the form $\langle p, \sigma \rangle \xrightarrow{a,b,X} \langle p', \sigma' \rangle$. They model the execution of action $a$ by process $p$ in an initial valuation $\sigma$, which changes process $p$ into $p'$ and results in a valuation $\sigma'$. Label $b$ is a boolean that indicates whether action $a$ is synchronizing or not, and label $X$ is the set of controlled variables defined by the environment of $p$ and $p'$.

Time behavior is captured by *time transitions*. Time transitions are of the form $\langle p, \sigma \rangle \overset{\rho,A,\theta,\omega}{\longmapsto} \langle p', \sigma' \rangle$. They model the passage of time in composition $p$, in an initial valuation $\sigma$, which results in a composition $p'$ and valuation $\sigma'$. Label $A$ contains the set of synchronizing actions of $p$ and $p'$. Function $\rho : \mathcal{T} \to \Sigma$

is the variable trajectory. Function $\theta : \mathcal{T} \to 2^{\mathcal{A}}$ is called *guard trajectory.* 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 $\omega$ is called *termination trajectory.* It models the evolution of termination (see below) 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 $t \in \mathcal{T}$, and $\mathrm{dom}(\rho) = \mathrm{dom}(\theta) = \mathrm{dom}(\omega)$. Termination is formally defined next.

**Definition 3 (Termination).** *Given a valuation $\sigma$, we define termination as follows:*

- *An automaton $(V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}, \mathrm{term}, \mathrm{h})$ is terminating in $\sigma$ if there is a location $v \in V$ such that $\sigma \models \mathrm{init}(v)$, $\sigma \models \mathrm{inv}(v)$, $\sigma \models \mathrm{term}(v)$, and if $v \in \mathrm{dom}(\mathrm{h})$ then $\mathrm{h}(v)$ is terminating in $\sigma$.*
- *Composition $p \parallel q$ is terminating in valuation $\sigma$ if $p$ and $q$ are terminating in valuation $\sigma$.*
- *For the remaining operators, termination is defined pointwise.*

Environment transitions are of the form $\langle p, \sigma\rangle \overset{A,b}{\dashrightarrow} \langle p', \sigma'\rangle$. They are used in the semantics to enforce restrictions posed by the environment of a composition on the action behavior of the composition. More specifically, a transition $\langle p, \sigma\rangle \overset{A,b}{\dashrightarrow} \langle p', \sigma'\rangle$ expresses the fact that $p$ is consistent in $\sigma$, and $p'$ is consistent in $\sigma'$. In addition, the role 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 its environment. As before, label $A$ contains the set of synchronizing actions of compositions $p$ and $p'$. Next, *consistency* is defined recursively.

**Definition 4 (Consistency).** *Given a valuation $\sigma$, we define consistency as follows.*

- *An automaton $(V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}, \mathrm{term}, \mathrm{h})$ is consistent in $\sigma$ if there is a location $\ell \in V$ such that $\sigma \models \mathrm{init}(\ell)$ and $\sigma \models \mathrm{inv}(\ell)$, and if $\ell \in \mathrm{dom}(\mathrm{h})$ then $\mathrm{h}(\ell)$ is consistent in $\sigma$.*
- *Composition $p \parallel q$ is consistent in valuation $\sigma$ if $p$ and $q$ are consistent in valuation $\sigma$.*
- *For the remaining operators, consistency is defined pointwise.*

We use notation $\sigma \models p$ to denote that composition $p$ is consistent in valuation $\sigma$. Alternatively, we say that $\sigma$ is consistent with $p$.

Definition 5 formalizes the hybrid transition system induced by the SOS rules presented in the next sections.

**Definition 5 (Hybrid Transition System).** *A hybrid transition system (HTS) is a five-tuple of the form $(Q, \mathcal{A}, \rightarrow, \longmapsto, \dashrightarrow)$ where $Q \triangleq \mathcal{C} \times \Sigma$, $\rightarrow \subseteq Q \times (\mathcal{A}_\tau \times \mathbb{B} \times 2^{\mathcal{V}}) \times Q$, $\longmapsto \subseteq Q \times ((\mathbb{T} \rightharpoonup \Sigma) \times 2^{\mathcal{A}} \times (\mathbb{T} \rightharpoonup 2^{\mathcal{A}}) \times (\mathbb{T} \rightharpoonup \mathbb{B})) \times Q$, $\dashrightarrow Q \times (2^{\mathcal{A}} \times \mathbb{B}) \times Q$.*

## 4 Semantics

In this section we explain the semantics of HCIF both informally by means of examples, and formally by means of SOS rules.

### 4.1 Hierarchical automata

In a hierarchical automaton $\alpha$, an active location $v$ can execute actions at two different levels of abstraction: *external* actions, which are specified as labelled edges from the active location $v$ to an arbitrary location $v'$; and *internal* actions, which are generated by the sub-structure at the location $v$, i.e., $h(v)$. Note that there are different conditions under which an external or internal action can be executed. Next, we explain and formalize these conditions.

Given an initial valuation $\sigma$, an external action can be executed in a location $v$ if there is an edge $(v, g, a, (W, r), v')$ in $\alpha$ such that the following conditions are met:

- Location $v$ is active ($\sigma \models \text{init}(v)$) and the invariant at the location $v$ is satisfied ($\sigma \models \text{inv}(v)$).
- If there is a substructure inside location $v$ ($v \in \text{dom(h)}$), then it is terminating in $\sigma$ or the edge is disruptive.
- Guard $g$ holds ($\sigma \models g$).
- It is possible to find a new valuation $\sigma'$ such that:
  - The invariant of the new location $v'$ holds $\sigma \models \text{inv}(v')$.
  - The reset predicate $r$ is satisfied in valuation $\sigma \cup \sigma'^{+}$ ($\sigma \cup \sigma'^{+} \models r$).
  - $\sigma'$ is consistent with the substructure inside the target location (if any).
  - Controlled variables not in $W$ (the set of jumping variables of the action) are not allowed to change in $\sigma'$ (wrt. $\sigma$).

This is formalized by Rule 1. Some of the above conditions are summarized in the term

$$\sigma, \sigma' \models_\alpha (v, g, a, (W, r), v')$$

that is syntactically equivalent to:

$$(v, g, a, r, v') \in E \wedge\ \sigma \models \text{init}(v) \wedge\ \sigma \models g \wedge\ \sigma \models \text{inv}(v) \wedge\ \sigma' \models \text{inv}(v') \wedge\ \sigma'^{+} \cup \sigma \models r.$$

Henceforth we use notation $\alpha$ to refer to an automaton:

$$(V, \text{init}, \text{inv}, \text{tcp}, E, \text{var}_C, \text{act}_S, \text{dtype}, \text{term}, \text{h})$$

and $\alpha[v]$ to refer to the automaton:

$$(V, \text{id}_v, \text{inv}, \text{tcp}, E, \text{var}_C, \text{act}_S, \text{dtype}, \text{term}, \text{h})$$

where $\text{dom}(\text{id}_v) = V$ and $\text{id}_v(w) \triangleq v = w$. Note that the initialization predicate $\text{id}_v$ encodes the active location, after execution of the first transition. An action specified by an edge $(v, g, a, (W, r), v')$ in an automaton $\alpha$ can be triggered only if the controlled variables of the automaton ($\mathrm{var}_C$) and of the environment ($X$) remain the same in $\sigma$, and $\sigma'$, except if they belong to the set of jumping variables $W$. We use notation $f \upharpoonright_A$ to refer to the domain restriction of function $f$ to the set $A$. Secondly, if the edge is not disruptive, it is necessary to check that the substructure of the initial location, if any, is terminating. This is expressed by condition $(\langle h(v), \sigma\rangle \overset{A_0,b}{\dashrightarrow} \langle p, \sigma\rangle \vee v \notin \mathrm{dom}(h)), (v, g, a, (W, r), v') \in D \vee b)$. Finally, after the action is performed, the substructure in the target location, if present, must be initialized (condition $\langle h(v'), \sigma'\rangle \overset{A_1,b}{\dashrightarrow} \langle q : \alpha[v'], \sigma'\rangle$). The choice of selecting active locations of substructure $h(v')$ is made upon entering location $v'$. Consistency of the substructures is taken care of by the environment transitions.

$$\frac{\begin{array}{c}\sigma, \sigma' \models_\alpha (v, g, a, (W, r), v'), \sigma \upharpoonright_{(X \cup \mathrm{var}_C)\setminus W} = \sigma' \upharpoonright_{(X \cup \mathrm{var}_C)\setminus W}, \\ \Big(\langle h(v), \sigma\rangle \overset{A_0,b}{\dashrightarrow} \langle p, \sigma\rangle \vee v \notin \mathrm{dom}(h)\Big), (v, g, a, (W, r), v') \in D \vee b, \\ v' \in \mathrm{dom}(h), \langle h(v'), \sigma'\rangle \overset{A_1,b'}{\dashrightarrow} \langle q, \sigma'\rangle\end{array}}{\langle \alpha, \sigma\rangle \xrightarrow{a, a \in \mathrm{act}_S, X} \langle q : \alpha[v'], \sigma'\rangle} \; 1$$

**Remark 1.** Consider the controller automaton in Fig. 1, assuming *UpOut* is an active location with a valuation $\sigma$. The edge labelled **when** $s = +1$ can be executed if there exists a valuation $\sigma'$ such that $\sigma$ satisfies the invariant of the location *UpOut* ($\sigma(x_\mathrm{h}) = -1 \wedge \sigma(x_\mathrm{v}) = 1$), $\sigma'$ satisfies the invariant of the location *Horizontal* ($\sigma'(x_\mathrm{v}) = 1$) and the valuation $\sigma'$ is consistent with the automaton shown in Fig. 2. The consistency of the valuation $\sigma'$ implies that the active location of the automaton in Fig. 2 is *Middle* such that $\sigma' \models \dot{x}_\mathrm{h} = s$.

Now consider the active location of the controller to be *Horizontal* and the active location of the automaton in Fig. 2 to be *Stopped-in*. In this case, the edge labelled **when** $s = 0$ in Fig. 1 cannot be executed even if the guard $s = 0$ is true. This is due to the fact that the composition inside the location *Horizontal* is non-terminating in location *Stopped-in*. External actions, such as the edge labelled **when** $s = 0$, can be executed only if either the sub-structure is terminating or the edge labelled with the external action is specified as disruptive. **End of Remark.**

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

$$\frac{\begin{array}{c}\sigma, \sigma' \models_\alpha (v, g, a, (W, r), v'), \sigma \upharpoonright_{(X \cup \mathrm{var}_C)\setminus W} = \sigma' \upharpoonright_{(X \cup \mathrm{var}_C)\setminus W}, \\ \Big(\langle h(v), \sigma\rangle \overset{A_0,b}{\dashrightarrow} \langle p, \sigma\rangle \vee v \notin \mathrm{dom}(h)\Big), v' \notin \mathrm{dom}(h), (v, g, a, (W, r), v') \in D \vee b\end{array}}{\langle \alpha, \sigma\rangle \xrightarrow{a, a \in \mathrm{act}_S, X} \langle \alpha[v'], \sigma'\rangle} \; 2$$

Besides external actions, an automaton can also execute internal actions, which are triggered by their internal structures. Given an initial valuation $\sigma$, an internal action can be executed in a location $v$ if the following conditions hold:

– $v$ is an initial location ($\sigma \models \mathrm{init}(v)$).
– The invariant associated with location $v$ is satisfied ($\sigma \models \mathrm{inv}(v)$).
– The action performed by the substructure of $v$ results in a new valuation $\sigma'$ that also satisfies the invariant of $v$.

Rule 3 formalizes this. 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. In this rule we ignore the boolean $b$, that indicates whether the action $a$ is synchronizing, in the premise. As a result, the superstructure decides on which actions it wants to synchronize. In other words, the superstructure defines the set of synchronizing actions, independently of the sub-levels.

$$\frac{\begin{array}{c}\sigma \models \mathrm{init}(v), \sigma \models \mathrm{inv}(v), \sigma' \models \mathrm{inv}(v),\\ v \in \mathrm{dom}(h), \langle h(v), \sigma\rangle \xrightarrow{a,b,X\cup \mathrm{var}_C} \langle p, \sigma'\rangle\end{array}}{\langle \alpha, \sigma\rangle \xrightarrow{a,a\in \mathrm{act}_S,X} \langle p : \alpha[v], \sigma'\rangle}\ 3$$

Transition $(h(v), \sigma) \xrightarrow{a,b,X\cup \mathrm{var}_C} (p, \sigma')$ in the premise of the above rule ensures that the control variables inherited from the environment ($X$) as well as the control variables of the automaton ($\mathrm{var}_C$) will not jump arbitrarily when the action is carried out by the substructure.

**Remark 2.** Again consider the model of the controller as given in Figures 1 and 2. Assume that the active location is *Horizontal* and the active location of the sub-structure is *Middle*. The edge labelled **when** $x \geq 1 \wedge s \geq 0$ can be executed from the location *Middle* only if there exists a new valuation $\sigma'$ such that it satisfies the invariant of the locations *Horizontal* and *StoppedIn*. **End of Remark.**

In hierarchical CIF, a time delay is possible in an active location $v$ if there exists a trajectory $\rho$ such that the invariant associated with the active locations is satisfied in time point $[0, t]$, the tcp predicate is satisfied in $[0, t)$ and the dynamic type constraints specified by dtype are satisfied. Henceforth, we use $\rho \models \langle t, v, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, \mathrm{dtype}\rangle$ as an abbreviation of the predicate

$$\begin{array}{l}\rho(0) \models \mathrm{init}(v) \wedge \mathrm{dom}(\rho) = [0, t] \wedge 0 < t \wedge\\ \forall_{s\in[0,t)}.\rho(s) \models \mathrm{tcp}(v) \wedge \forall_{s\in[0,t]}.\rho(s) \models \mathrm{inv}(v) \wedge\\ \forall_{x\in \mathrm{dom}(\mathrm{dtype})}.(\rho \downarrow x, \rho \downarrow \dot{x}) \in \mathrm{dtype}(x).\end{array}$$

For time delays, the substructure (if present) must perform a time transition with the same trajectory. In this way, the invariants and tcp-predicates of the active location of the automaton, and, recursively, of the active locations of its active substructures are considered simultaneously. In this way time passes in an automaton, and also in all of its contained active substructures. In other words, an automaton and its active substructure synchronize on time delays. Rule 4 models this, where $\mathrm{dom}(\omega) = \mathrm{dom}(\rho)$, $\mathrm{dom}(\theta) = \mathrm{dom}(\rho)$, $\forall_{s\in[0,t]}.\omega(s) = (\omega_0(s) \wedge \rho(s) \models \mathrm{term}(v))$, and $\forall_{s\in[0,t]}.\theta(s) = \theta_0(s) \cup \{a \mid (v, g, a, (W, r), v') \in$

$E \wedge \rho(s) \models g \wedge \omega_0(s)\}$. The guard trajectory $\theta$ as well as the termination trajectory $\omega$ are constructed by using the corresponding trajectories generated by the time transition in the substructure. The set of synchronizing actions only takes into account the set $\mathrm{act}_S$ in the superstructure, since the set of synchronizing actions in the substructure does not influences the action synchronizing behavior of its parent. The same approach is taken when computing the set of synchronizing actions in the environment transition in Rule 6.

$$\frac{\rho \models \langle t, v, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, \mathrm{dtype}\rangle, v \in \mathrm{dom}(h) \quad \langle h(v), \rho(0)\rangle \overset{\rho, A, \theta_0, \omega_0}{\longmapsto} \langle p, \rho(t)\rangle}{\langle \alpha, \rho(0)\rangle \overset{\rho, \mathrm{act}_S, \theta, \omega}{\longmapsto} \langle p' : \alpha[v], \rho(t)\rangle}\ 4$$

Rule 5 deals with the case that 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 $\forall_{s\in[0,t]}.\theta(s) = \{a \mid (v, g, a, (W, r), v') \in E \wedge \rho(s) \models g\}$.

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

In hierarchical CIF, if an automaton performs an environment transition, a unique active location is chosen, and the substructure (if present) is initialized. The environment transition ensures that the active location contains a consistent hierarchical structure (Definition 4). 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.

$$\frac{\sigma \models \mathrm{init}(v), \sigma \models \mathrm{inv}(v), \sigma' \models \mathrm{inv}(v), \sigma \restriction_{\mathrm{var}_C} = \sigma' \restriction_{\mathrm{var}_C}, \quad v \in \mathrm{dom}(h), \langle h(v), \sigma\rangle \overset{A, b}{\dashrightarrow} \langle p', \sigma'\rangle}{\langle \alpha, \sigma\rangle \overset{\mathrm{act}_S, \sigma \models \mathrm{term}(v) \wedge b}{\dashrightarrow} \langle p' : \alpha[v], \sigma'\rangle}\ 6$$

$$\frac{\sigma \models \mathrm{init}(v), \sigma \models \mathrm{inv}(v), \sigma' \models \mathrm{inv}(v), \sigma \restriction_{\mathrm{var}_C} = \sigma' \restriction_{\mathrm{var}_C}, v \notin \mathrm{dom}(h)}{\langle \alpha, \sigma\rangle \overset{\mathrm{act}_S, \sigma \models \mathrm{term}(v)}{\dashrightarrow} \langle \alpha[v], \sigma'\rangle}\ 7$$

## 4.2 Automaton postfix operator

The automaton postfix operator is used to define the semantics of hierarchy. It is not an operator intended to be used for modeling, and therefore we do not illustrate its behavior by means of examples. We limit ourselves to semantic considerations.

Intuitively, the composition $p : \alpha$ means that composition $p$ is the active substructure of some initial location $v \in V$ in the automaton $\alpha$. Note, that whenever the composition $p : \alpha$ is the result of a previous transition in $\alpha$, this initial location is always uniquely defined.

Rule 8 models the action transition taken by automaton $\alpha$ when the active substructure is terminating or when the chosen edge is disruptive, and the target location has a substructure. Rule 9 differs from Rule 8 only in that the target location does not have a substructure.

$$\frac{\begin{array}{c}\sigma, \sigma' \models_\alpha (v, g, a, (W, r), v'), \sigma \restriction_{(X \cup \mathrm{var}_C) \setminus W} = \sigma' \restriction_{(X \cup \mathrm{var}_C) \setminus W}, \\ \langle p, \sigma \rangle \overset{A_0, b}{\dashrightarrow} \langle p', \sigma \rangle, (v, g, a, (W, r), v') \in D \vee b, \\ v' \in \mathrm{dom}(h), \langle h(v'), \sigma' \rangle \overset{A_1, b'}{\dashrightarrow} \langle q, \sigma' \rangle\end{array}}{\langle p : \alpha, \sigma \rangle \xrightarrow{a, a \in \mathrm{act}_S, X} \langle q : \alpha[v'], \sigma' \rangle} \; 8$$

$$\frac{\begin{array}{c}\sigma, \sigma' \models_\alpha (v, g, a, (W, r), v'), \sigma \restriction_{(X \cup \mathrm{var}_C) \setminus W} = \sigma' \restriction_{(X \cup \mathrm{var}_C) \setminus W}, \\ \langle p, \sigma \rangle \overset{A, b}{\dashrightarrow} \langle p', \sigma \rangle, (v, g, a, (W, r), v') \in D \vee b, v' \notin \mathrm{dom}(h)\end{array}}{\langle p : \alpha, \sigma \rangle \xrightarrow{a, a \in \mathrm{act}_S, X} \langle \alpha[v'], \sigma' \rangle} \; 9$$

Rule 10 models the action transition that results from execution of the substructure.

$$\frac{\begin{array}{c}\sigma \models \mathrm{init}(v), \sigma \models \mathrm{inv}(v), \sigma' \models \mathrm{inv}(v), \\ \langle p, \sigma \rangle \xrightarrow{a, b, X \cup \mathrm{var}_C} \langle q, \sigma' \rangle\end{array}}{\langle p : \alpha, \sigma \rangle \xrightarrow{a, a \in \mathrm{act}_S, X} \langle q : \alpha, \sigma' \rangle} \; 10$$

Rule 11 models the passage of time in an automaton postfix such that the timed transitions are (recursively) synchronized in every level of hierarchy of $p : \alpha$, where, $\mathrm{dom}(\omega) = \mathrm{dom}(\rho)$, $\mathrm{dom}(\theta) = \mathrm{dom}(\rho)$ and $\forall_{s \in [0,t]}.\omega(s) = (\omega_0(s) \wedge \rho(s) \models \mathrm{term}(v))$, and $\forall_{s \in [0,t]}.\theta(s) = (\theta_0(s) \cup \{a \mid (v, g, a, (W, r), v') \in E \wedge \rho(s) \models g \wedge \omega_0(s)\})$.

$$\frac{\begin{array}{c}\rho \models \langle t, v, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, \mathrm{dtype} \rangle, \\ \langle p, \rho(0) \rangle \overset{\rho, A, \theta_0, \omega_0}{\longmapsto} \langle p', \rho(t) \rangle\end{array}}{\langle p : \alpha, \rho(0) \rangle \overset{\rho, \mathrm{act}_S, \theta, \omega}{\longmapsto} \langle p' : \alpha[v], \rho(t) \rangle} \; 11$$

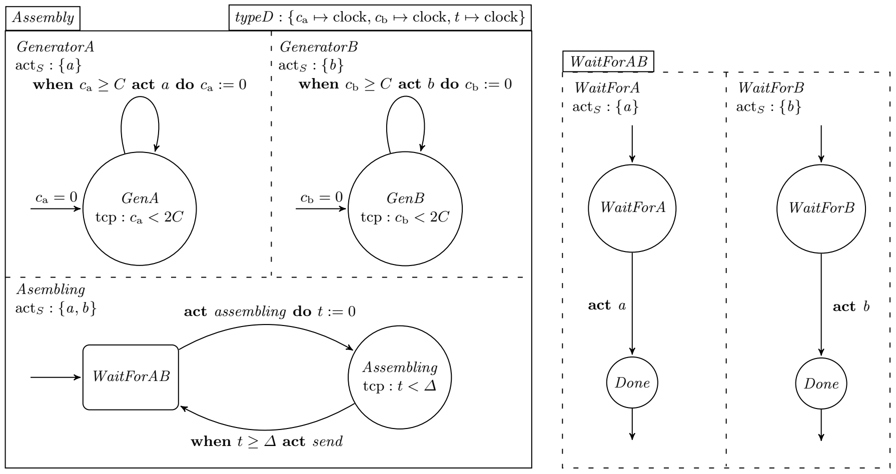
Finally, Rule 12 models the execution of an environment transition in an automaton postfix.

$$\frac{\begin{array}{c}\sigma \models \mathrm{init}(v), \sigma \models \mathrm{inv}(v), \sigma' \models \mathrm{inv}(v), \sigma \restriction_{\mathrm{var}_C} = \sigma' \restriction_{\mathrm{var}_C}, \\ \langle p, \sigma \rangle \overset{A, b}{\dashrightarrow} \langle p', \sigma' \rangle\end{array}}{\langle p : \alpha, \sigma \rangle \overset{\mathrm{act}_S, \sigma \models \mathrm{term}(v) \wedge b}{\dashrightarrow} \langle p' : \alpha[v], \sigma' \rangle} \; 12$$

## 4.3 Parallel composition

The parallel composition operator allows concurrent execution of HCIF compositions. The semantics of parallel composition is equal to the CIF semantics. Action behavior is not affected by the addition of hierarchy. The rules for time and environment transitions are updated to reflect the fact that a parallel composition is terminating only if both components are.

As an illustration, consider the assembly process shown in Fig. 3(a), henceforth referred to as *Assembly*, such that its location *WaitForAB* contains the parallel composition shown in Fig. 3(b). The assembly process initially is in the *WaitForAB* location, and, according to the semantics of atomic automata, it can trigger action *assembling* only if its sub-structure terminates. Since the sub-structure is a parallel composition of two automata, namely *WaitForA* and *WaitForB* (See Fig. 3(b)), the substructure h(*WaitForAB*) can terminate after actions $a$ and $b$ have both been executed; i.e., both automata *WaitForA* and *WaitForB* can terminate. This pattern, in which an action is triggered after a series of parallel processes terminate, can be expressed succinctly using hierarchy. Without support for hierarchy and termination it is necessary to rewrite the parallel processes into a flat automaton.



(a) Assembly process (*Assembly*). (b) Receive process (h(*WaitForAB*)).

**Fig. 3.** Assembly line.

The addition of hierarchy facilitates inter-level synchronization. As an example consider the generator process *GeneratorA* shown in Fig. 3(a), which enables an action $a$ every $C$ time units, when $c_a \geq C$. The action $a$ from the generator synchronizes with action $a$ specified as synchronizing in the automaton *WaitForA*, which is part of the substructure of location *WaitForAB*. This synchronizing behavior is obtained by inclusion of action $a$ in the set of synchronizing actions $\mathrm{act}_S$ of *GeneratorA* ($\{a\}$), and in the set of synchronizing actions of automaton *Assembly* ($\{a, b\}$). Note that strictly speaking, action $a$ need not be defined as synchronizing for automaton *WaitForA*.

Formally, Rule 13 states that two synchronizing actions with the same label can execute in parallel only if they share the same initial and final valuation, and if the action is synchronizing in both compositions. The set of control variables $X$, is propagated from the conclusions to the premises since the control variables in the scope of a parallel composition are shared by both partners. The resulting action transition is also synchronizing which allows action $a$ to synchronise with more than two compositions.

$$\frac{\langle p, \sigma\rangle \xrightarrow{a,\text{true},X} \langle p', \sigma'\rangle, \langle q, \sigma\rangle \xrightarrow{a,\text{true},X} \langle q', \sigma'\rangle}{\begin{array}{c}\langle p \parallel q, \sigma\rangle \xrightarrow{a,\text{true},X} \langle p' \parallel q', \sigma'\rangle \\ \langle q \parallel p, \sigma\rangle \xrightarrow{a,\text{true},X} \langle q' \parallel p', \sigma'\rangle\end{array}}\ 13$$

Rules 14 model interleaving behavior of two compositions when executed in parallel. In these rules, an action can be performed in one of the components ($p$) only if the initial and final valuations are consistent with the other composition ($q$); and if this action is not synchronizing in the other component, which is expressed by the condition $a \notin A$. The environment transition $(q, \sigma) \overset{A,b'}{\dashrightarrow} (q', \sigma')$ is used to obtain the set of synchronizing action labels in composition $q$, to ensure that the initial valuation $\sigma$ is consistent with the active invariants and initialization conditions of $q$, to select an initial location (in case there is more than one in $q$), and to remove any initialization operators from $q$.

$$\frac{\langle p, \sigma\rangle \xrightarrow{a,b,X} \langle p', \sigma'\rangle, \langle q, \sigma\rangle \overset{A,b'}{\dashrightarrow} \langle q', \sigma'\rangle, a \notin A}{\begin{array}{c}\langle p \parallel q, \sigma\rangle \xrightarrow{a,b,X} \langle p' \parallel q', \sigma'\rangle \\ \langle q \parallel p, \sigma\rangle \xrightarrow{a,b,X} \langle q' \parallel p', \sigma'\rangle\end{array}}\ 14$$

Rule 15 models the fact that if two compositions are put in parallel, time can pass $t$ time units only if allowed by both partners. As can be seen in this rule, the set of enabled actions in the parallel composition at any point in time during the delay depends both on the set of enabled actions and the set of synchronizing actions in each component individually. Similarly, the termination trajectory of the parallel composition depends on the termination trajectories of its components, where $\theta_{01} = (\theta_0 \cap \theta_1) \cup (\theta_0 \setminus A_1) \cup (\theta_1 \setminus A_0)$ and $\forall s \in [0, t].[\omega_{01}(s) = \omega_0(s) \wedge \omega_1(s)]$.

$$\frac{\langle p, \rho(0)\rangle \overset{\rho,A_0,\theta_0,\omega_0}{\longmapsto} \langle p', \rho(t)\rangle, \langle q, \rho(0)\rangle \overset{\rho,A_1,\theta_1,\omega_1}{\longmapsto} \langle q', \rho(t)\rangle}{\begin{array}{c}\langle p \parallel q, \rho(0)\rangle \overset{\rho,A_0\cup A_1,\theta_{01},\omega_{01}}{\longmapsto} \langle p' \parallel q', \rho(t)\rangle \\ \langle q \parallel p, \rho(0)\rangle \overset{\rho,A_0\cup A_1,\theta_{01},\omega_{01}}{\longmapsto} \langle q' \parallel p', \rho(t)\rangle\end{array}}\ 15$$

Rule 16 defines the environment transition behavior for parallel composition. The resulting set of synchronizing actions is the union of the synchronizing actions of $p$ and $q$. The conjunction $b_0 \wedge b_1$ models the fact that a parallel composition is terminating if its components are. Note that the end valuations of all transitions match.

$$\frac{\langle p,\sigma\rangle \overset{A_0,b_0}{\dashrightarrow} \langle p',\sigma'\rangle, \langle q,\sigma\rangle \overset{A_1,b_1}{\dashrightarrow} \langle q',\sigma'\rangle}{\begin{array}{c}\langle p \parallel q,\sigma\rangle \overset{A_0\cup A_1,b_0\wedge b_1}{\dashrightarrow} \langle p' \parallel q',\sigma'\rangle \\ \langle q \parallel p,\sigma\rangle \overset{A_0\cup A_1,b_0\wedge b_1}{\dashrightarrow} \langle q' \parallel p',\sigma'\rangle\end{array}} \; 16$$

### 4.4 Urgency operator

By means of the urgency operator it is possible to declare actions as urgent. This means that time cannot pass if an urgent action is enabled. However, urgent actions do not have priority over regular (non-urgent) actions.

For example, consider the model of the controller of Fig. 1 with active location *UpOut*. When a user demands the controller to operate in the horizontal mode, it should react as soon as possible. In other words, the action $\tau$ in the labelled edge **when** $s = +1$ between the locations *UpOut* and *Horizontal* must be urgent. This ensures that time does not pass in the location *UpOut* from the instant when the guard $s = +1$ is enabled.

Rule 17 specifies that the urgent action operator restricts the time behavior of a composition in such a way that time can pass for as long as no urgent action is enabled.

$$\frac{(p,\sigma) \overset{\rho,A,\theta,\omega}{\longmapsto} (p',\sigma'),\ \forall_{s\in[0,t)} \cdot a \notin \theta(s)}{(\upsilon_a(p),\sigma) \overset{\rho,A,\theta,\omega}{\longmapsto} (\upsilon_a(p'),\sigma')} \; 17$$

The urgency operator affects only the time behavior. Action and environment transitions remain unchanged as expressed by Rules 18 and 19.

$$\frac{(p,\sigma) \xrightarrow{\ell,b,X} (p',\sigma')}{(\upsilon_a(p),\sigma) \xrightarrow{\ell,b,X} (\upsilon_a(p'),\sigma')} \; 18 \qquad \frac{(p,\sigma) \overset{A,b}{\dashrightarrow} (p',\sigma')}{(\upsilon_a(p),\sigma) \overset{A,b}{\dashrightarrow} (\upsilon_a(p'),\sigma')} \; 19$$

## 5 Case-study: Patient Support System

The patient support system (See Fig. 4) is used in medical diagnosis to position a patient in an MRI scanner [13]. The system can be operated in the following modes: vertical mode, horizontal mode and user interface mode. In the vertical mode, the table top on which a patient resides can only move vertically between the bounds depicted in Fig. 4. Similarly, in the horizontal mode, the table top can be moved in or out of the bore, either manually or by means of a motor drive. Furthermore, the system is equipped with a table top release switch for emergency situations. This system is controlled via a user interface that contains a tumble switch to control the movement (both horizontally and vertically) of the table, and a button to enable the start of an initialization sequence. The position of the tumble switch is represented by variable $s$ which can have the values $+1$, $0$ and $-1$. The continuous variables $x_\mathrm{h}$ and $x_\mathrm{v}$ represent the horizontal and vertical position of the table top, respectively.

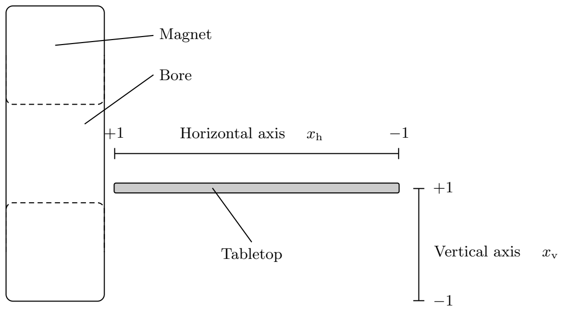
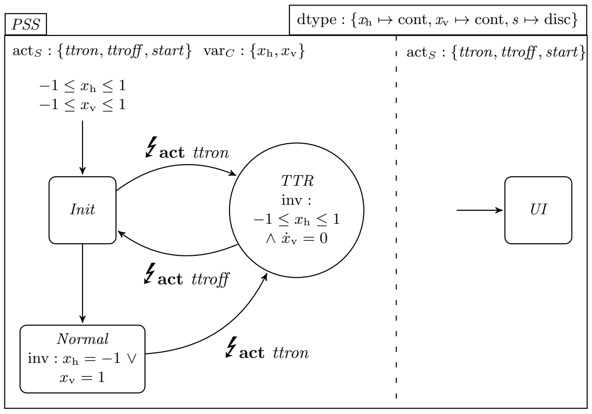


**Fig. 4.** Patient Support System.



PSS
dtype : $\{x_h \mapsto \text{cont}, x_v \mapsto \text{cont}, s \mapsto \text{disc}\}$
$\text{act}_S : \{ttron, ttroff, start\}$ $\text{var}_C : \{x_h, x_v\}$
$\text{act}_S : \{ttron, ttroff, start\}$
$-1 \le x_h \le 1$
$-1 \le x_v \le 1$
Init
**act** ttron
TTR
inv :
$-1 \le x_h \le 1$
$\wedge \dot{x}_v = 0$
**act** ttroff
Normal
inv : $x_h = -1 \vee$
$x_v = 1$
**act** ttron
UI

**Fig. 5.** Patient Support System.

The objective of this case-study is to design a controller that satisfies the following requirements. The table should move up and down, or in and out of the bore, by operating the tumble switch. The table should not move beyond the boundaries shown in Fig. 4. The case-study is specified using a top-down design methodology. In other words, we first model the overall system at a higher level of abstraction in which we identify that the system consists of a controller and a user interface. Furthermore, a controller can run in the following three modes: *Init* mode in which the controller should place the table in the initial position; *Normal* mode in which the controller synchronises with the events of the tumble switch; *TTR* (Table Top Release) mode in which an operator is allowed to override the normal execution of the controller. Fig. 5 shows the model of the system at this level of abstraction. Throughout the complete description of this case-study, it is assumed that only the $\tau$ action is urgent. All other actions, which are the actions generated by the user interface, are non-urgent. This is modeled by $\upsilon_\tau(PSS)$ where *PSS* represents the automaton *PSS* shown in Fig 5.

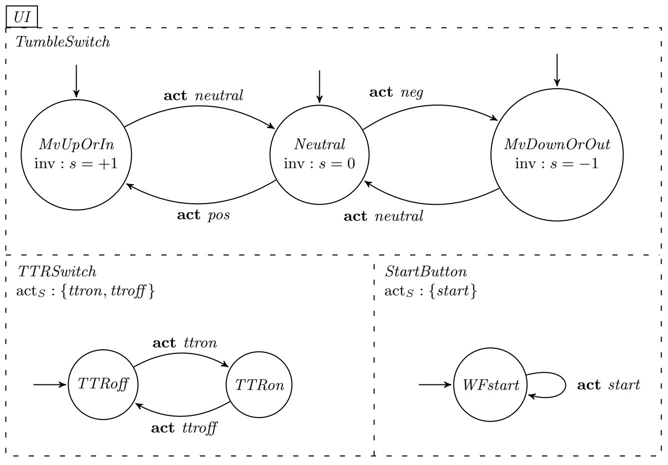


**Fig. 6.** User interface.

*User interface* The user interface consists of three input devices: the tumble switch, the table top release switch and the start button (See Fig. 6). The tumble switch has three positions: *MvUpOrIn*, *Neutral* and *MvDownOrOut*. The *MvUpOrIn* position is used to move the table either up or into the bore, the *MvDownOrOut* position is used to move the table down or out of the bore. When the switch is released, it returns to the neutral position, which enforces actuated (motorized) movement of the table to stop.

The TTR switch can be used to release the table top from the horizontal motor. When the switch is active, the horizontal movement of the table is uncontrolled by the system, so that an operator can manually move the table freely in the horizontal direction. The start button is used to allow initialization of the system.

*Initialization* In the *Init* mode, the position of the patient support system is initialized (Fig. 7). The position of the tumble switch needs to be neutral before initialization begins, and the movement is triggered by pressing the start button. The desired final position of the table is fully retracted and fully up. First, the table is retracted since this is always a safe movement. Then, when the table is fully retracted, the table is moved up until it reaches the top position. The initialization is complete when the tumble switch is in the neutral position, to prevent that the table starts moving immediately after initialization.

*Normal mode* Initially, the system enters the normal mode with the table fully up and retracted, so in an up and out position (Fig. 8). In this intersection point between moving the table horizontally or vertically, holding the tumble switch in the *MvUpOrIn* position triggers horizontal movement of the table into the bore, whereas holding it in a *MvDownOrOut* position triggers vertical, downward movement.

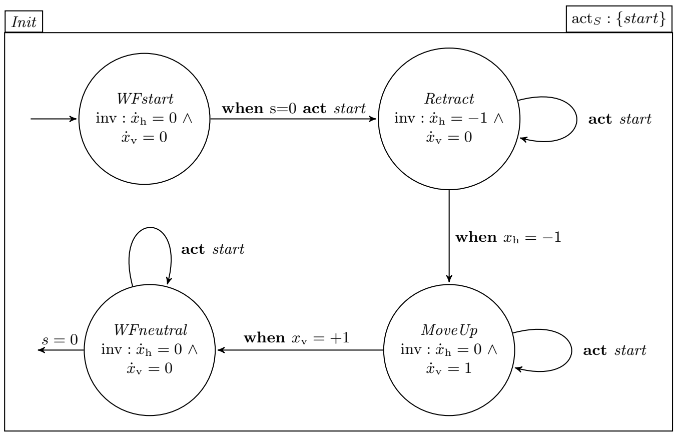

Init
act$_S$ : {*start*}
*WFstart*
inv : $\dot{x}_h = 0 \wedge \dot{x}_v = 0$
**when** s=0 **act** *start*
*Retract*
inv : $\dot{x}_h = -1 \wedge \dot{x}_v = 0$
**act** *start*
**when** $x_h = -1$
**act** *start*
*WFneutral*
inv : $\dot{x}_h = 0 \wedge \dot{x}_v = 0$
$s = 0$
**when** $x_v = +1$
*MoveUp*
inv : $\dot{x}_h = 0 \wedge \dot{x}_v = 1$
**act** *start*

**Fig. 7.** Initialization.

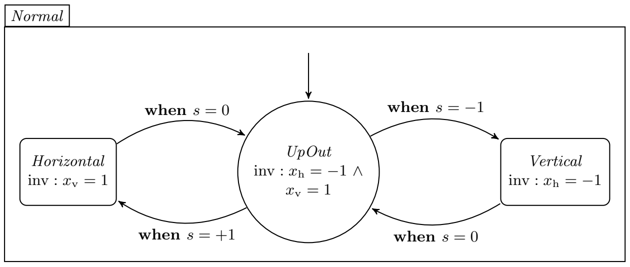


**Fig. 8.** Normal movement control.

A system requirement is that between switching from horizontal to vertical movement, and vice versa, the position of the tumble switch must be neutral. This to prevent the table from continuing movement unexpectedly in a different direction. Figures 9 and 10 show the horizontal and vertical movement of the system in more detail.

## 6 Concluding remarks

In this article we have presented the syntax and semantics of HCIF, which extends CIF with hierarchy in a compositional manner, so that only the SOS rules for an automaton and for the time transitions of parallel composition need to be adapted.

We conjecture that we are able to transform a HCIF composition into a bisimilar CIF specification on the condition that the effective set of controlled variables and the effective dynamic type of the variables is independent of the active locations of the automata, and is thus statically defined. This condition is needed because in HCIF, the dynamic type of variables and the set of control

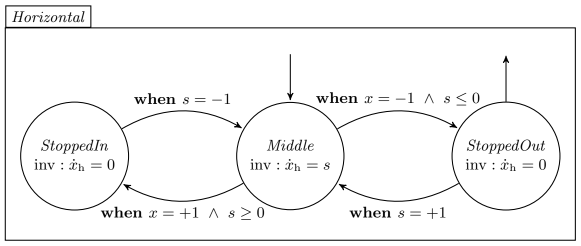

Horizontal
when $s = -1$
when $x = -1 \wedge s \leq 0$
StoppedIn
inv : $\dot{x}_{\mathrm{h}} = 0$
Middle
inv : $\dot{x}_{\mathrm{h}} = s$
StoppedOut
inv : $\dot{x}_{\mathrm{h}} = 0$
when $x = +1 \wedge s \geq 0$
when $s = +1$

**Fig. 9.** Horizontal movement control.

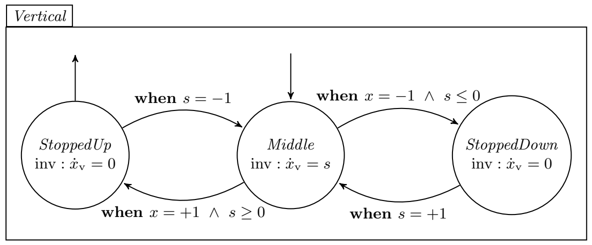

Vertical
when $s = -1$
when $x = -1 \wedge s \leq 0$
StoppedUp
inv : $\dot{x}_{\mathrm{v}} = 0$
Middle
inv : $\dot{x}_{\mathrm{v}} = s$
StoppedDown
inv : $\dot{x}_{\mathrm{v}} = 0$
when $x = +1 \wedge s \geq 0$
when $s = +1$

**Fig. 10.** Vertical movement control.

variables can change per location, since the substructures of different locations may have different dynamic types and different control variables. Future work includes proving that HCIF is more expressive than CIF, and defining the subset of HCIF that can be translated to CIF. These transformations are important to reuse existing tools for CIF, including model transformations.

In addition, we observe that a liberal interpretation of a hierarchical HCIF automaton as an n-ary operator (where n represents the number of locations in the super-automaton) places our semantic rules within the congruence format of [10]. This means that the replacement of a sub-automaton by an equivalent one (modulo stateless bisimulation equivalence) will lead to an equivalent behavior of the super-automaton, which is a fundamental property for compositional reasoning.

## References

1. J. Baeten, D. van Beek, D. Hendriks, A. Hofkamp, D. N. Agut, J. Rooda, and R. Schiffelers. Definition of the compositional interchange format. Technical Report Deliverable D1.1.2, Multiform, 2010.
2. A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.
3. D. A. v. Beek, P. Collins, D. E. Nadales, J. Rooda, and R. R. H. Schiffelers. New concepts in the abstract format of the compositional interchange format. In

A. Giua, C. Mahuela, M. Silva, and J. Zaytoon, editors, *3rd IFAC Conference on Analysis and Design of Hybrid Systems*, pages 250–255, Zaragoza, Spain, 2009.

4. D. A. v. Beek, M. A. Reniers, R. R. H. Schiffelers, and J. E. Rooda. Foundations of an interchange format for hybrid systems. In A. Bemporad, A. Bicchi, and G. Butazzo, editors, *Hybrid Systems: Computation and Control, 10th International Workshop*, volume 4416 of *Lecture Notes in Computer Science*, pages 587–600, Pisa, 2007. Springer-Verlag.
5. H. Beohar, D. E. Nadales Agut, D. A. van Beek, and P. J. L. Cuijpers. Hierarchical states in the compositional interchange format. *Electronic Proceedings in Theoretical Computer Science*, 32:42–56, 2010.
6. P. J. L. Cuijpers, M. A. Reniers, and W. P. M. H. Heemels. Hybrid transition systems. Technical Report CS-Report 02-12, Eindhoven University of Technology, Department of Computer Science, The Netherlands, 2002.
7. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer-Verlag, 2005.
8. T. A. Henzinger. The theory of hybrid automata. In M. K. Inan and R. P. Kurshan, editors, *Verification of Digital and Hybrid Systems*, volume 170 of *NATO ASI Series F: Computer and Systems Science*, pages 265–292. Springer-Verlag, New York, 2000.
9. N. Lynch, R. Segala, and F. Vaandrager. Hybrid i/o automata revisited. In *Proceedings Fourth International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, pages 403–417. Springer-Verlag, 2001.
10. 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):107–147, 2005.
11. G. D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.
12. J. C. Reynolds. *Theories of programming languages*. Cambridge University Press, New York, NY, USA, 1999.
13. R. J. M. Theunissen, M. Petreczky, R. R. H. Schiffelers, D. A. v. Beek, and J. E. Rooda. Application of supervisory control synthesis to MRI scanners: improving evolvability. SE Report 2010-06, System Engineering Group, Department of Mechanical Engineering, Eindhoven university of technology, Eindhoven, 2010.
14. A. E. Uselton and S. A. Smolka. State Refinement in Process Algebra. Technical report, Stony Brook university, NY, 1993.