# Hierarchical automata as model for statecharts (Extended abstract)

Erich Mikk[1], Yassine Lakhnech[1] and Michael Siegel[2]

[1] Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik
Preußerstr. 1-9, D24105 Kiel, Germany
[2] Weizmann Institute of Science
Department of Applied Mathematics and Computer Science
76100 Rehovot, Israel

**Abstract.** Statecharts are a very rich graphical specification formalism supported by the commercial tool STATEMATE. Statecharts comprises powerful concepts such as interlevel transitions, multiple-source/multiple-target transitions, priority amongst transitions and simultaneous execution of maximal non-conflicting sets of transitions. Every add-on tool which is supposed to be linked with the STATEMATE tool have to deal with the rather involved semantics of these concepts. We propose *extended hierarchical automata* as an intermediate format to facilitate the linking of new tools to the STATEMATE environment, whose main idea is to devise a simple formalism with a more restricted syntax than statecharts which nevertheless allows to capture the richer formalism. We define the format, give operational semantics to it, and translate statecharts to it.

## 1 Introduction

Statecharts are a very rich graphical specification formalism supported by the commercial tool STATEMATE[3]. Statecharts comprises powerful concepts such as interlevel transitions, multiple-source/multiple-target transitions, priority amongst transitions and simultaneous execution of maximal non-conflicting sets of transitions [HN96]. Every add-on tool, such as code generators, test suite generators, simulators, model checkers, etc., which are supposed to be linked with the STATEMATE tool have to deal with the rather involved semantics of these concepts.

In this paper we suggest *extended hierarchical automata* (EHA) as an intermediate format to facilitate the linking of new tools to the STATEMATE environment. The EHA formalism uses single-source/single-target transitions as in usual automata (no interlevel transitions are admitted) and has a simple priority concept which facilitates computing the next step of an EHA when compared to statecharts. So, the main idea is to devise a simple formalism with a more restricted syntax than statecharts which nevertheless allows to capture the richer formalism. Extended hierarchical automata, which are related to the Argos language [Mar91], come with a simple operational semantics which simplifies the implementation of tools for this formalism.

[3] STATEMATE is provided by iLogix: http://www.ilogix.com/company/company.htm

In the current paper we introduce the concept and semantics of EHA and explain the translation of statecharts into EHA. This translation can be used as a generic front-end for the implementation of various tools for the statecharts formalism as supported by the STATEMATE environment.

Before formulating such an intermediate format, its semantics and a translation of statecharts into this format we need to fix a certain semantics for statecharts itself. Since we are interested in add-on tools for STATEMATE we base our work on the recent Harel&Naamad semantics [HN96] which we formalized in [MLPS97].

The main technical problem is to devise a simple formalism which is nevertheless capable to model interlevel transitions. Interlevel transitions (possibly with multiple sources/multiple targets) are transitions which do not respect the hierarchy of states, i.e. those that cross borderlines of states. They can be understood as a powerful goto mechanism which allows to arbitrary movement of control across the state hierarchy. The price of interlevel transitions is their intricate semantics in particular in combination with the priority mechanism as supported by STATEMATE. Interlevel transitions spoil in general a clean decomposition of a system into subsystems (since "dangling" transitions without source or target result) and thus denies a structural operational semantics for statecharts. A rather complicated continuation semantics for statecharts with interlevel transitions can be found in [Hui91].

These semantical problems, amongst others, caused the investigation of other graphical and hierarchical formalisms, such as Argos [Mar91, Mar92], Spec-Charts [NVG91], Modecharts [JM94], and RSML [LHHR94] which forbid the explicit use of interlevel transitions and thereby gain elegant structural operation semantics (SOS). Argos even supports an encoding to simulate the effect of interlevel transitions. The encoding exploits the non-preemptive interrupt mechanism of Argos as well as the synchronicity of input/output events. This encoding does not work for the statecharts formalism as implemented in STATEMATE since it is neither perfectly synchronous (generated events can only be sensed in the next step of the system) nor does it support non-preemptive interrupts (its priority mechanism forbids simultaneous execution of transitions which depart from states which are related in the state hierarchy). After introducing extended hierarchical automata and the translation of statecharts into this formalism we give a more detailed comparison between Argos and our approach in Section 4.

The two main side conditions for the definition of the intermediate format are that it must be as simple as possible but also remain close to the original statecharts formalism. The latter facilitates the actual translation of statecharts into this format (e.g. using a perfectly synchronous formalism will considerably complicate the translation) as well as the correctness proof of the translation. As a result of remaining close to the statecharts formalism the EHA format explains completely formal yet simple to grasp the semantical phenomena of statecharts. These features make EHA a good candidate as intermediate format when implementing new tools for statecharts. So far we have used EHA in the implementation of a compiler from statecharts to Promela, the input language of the SPIN model-checker [Hol91, Hol97].

The paper is organized as follows. In Section 2 we define hierarchical automata and give descriptive style semantics to them. Section 3 introduces the extended version of hierarchical automata, which allows for modeling interlevel transitions. A simple operational semantics is given for EHA which is shown to be compatible with the descriptional one. In Section 4 we translate statecharts with interlevel transitions to extended hierarchical automata.

## 2 Hierarchical automata

In this section we define the class of hierarchical automata (HA) that consists of parallel and hierarchical composition of sequential automata. HA resemble statecharts without interlevel transitions.

### 2.1 Introduction to hierarchical automata

A hierarchical automaton is built as parallel and/or hierarchical composition of sequential automata. HA have maximal parallelism semantics, i.e. parallel automata execute their transitions synchronously. Such a set of synchronously firing transitions constitutes a step (cf. [HN96]). As in statecharts the communication medium are events which are broadcasted instantaneously within the hierarchical automaton. A step consumes input events and produces output events which are inputs for the next step. Hierarchical composition means mapping states of a sequential automaton to another automaton or parallel composition of automata. If a state $s$ of some automaton is mapped to automaton $A$, then we refer to $A$ as the *sub-automaton* of $s$. We also say, that *$s$ is refined by $A$* and that $s$ is a *parent* of $A$.

The idea is that the parent state may delegate its step to its sub-automata. If a state does not have an internal structure we call it a *basic state*. Taking a transition means to leave the source state and enter the target state of the transition; if the source and the target are non-basic states then taking the transition affects the sub-structures of the source and the target as well: together with the source state all sub-states of the source are left and when a target is entered then the initial states of the target's sub-automata are entered.

Besides communication via events, statecharts support synchronization of concurrent components by state references to parallel components. Transitions are guarded by events and such state references. The guard is true if the event is available (i.e. provided by the environment or generated in the previous step) and the state references refer to states that are currently active. Since hierarchical automata are meant to model statecharts, we add this feature to hierarchical automata, too. Transition labels in hierarchical automata are pairs of trigger and action part. On the trigger part, we allow for propositions over event and state names. On the action part we allow for generation of events.

*Example 1. Running example.* We use state transition diagrams to visualize hierarchical automata. We use boxes to denote states, initial states are indicated by double boxes. Transitions are labeled arrows. Sequential automata are surrounded by a dashed line.

The hierarchical structure is indicated by dotted lines. If a state is mapped to more than one sequential automata then they are meant to run in parallel.

The hierarchical automata in Figure 1 depicts a model of a TV-set (this example is inspired by [HdR91]). The root automaton $TV$ may be in states $ON$ or $OFF$. State $OFF$ is refined to automaton $POWER$ and state $ON$ is refined to the parallel composition of automata $IMAGE$ and $SOUND$. Whenever the $TV$ automaton is in the state $ON$, both $SOUND$ and $IMAGE$ are considered active; since $ON$ and $OFF$ are exclusive the system can never be in one of the states of $POWER$ when $TV$ is in state $ON$. The system changes its state upon reception of events from the environment, e.g., the event *on* initiates the state change from $OFF$ to $ON$, which implies deactivation of $POWER$ and activation of $IMAGE$ and $SOUND$.

However, this specification is not quite what we would expect from a TV-set. The state $DISCONNECTED$ is meant to model that the TV-set is not plugged in and only after the event *in* we should expect some activity from the system. Indeed, as described above we can constrain the transition from $OFF$ to $ON$ by a condition restricting its enabledness only to configurations where $STANDBY$ is active. But we would like to have more, e.g., when being in $TV.ON$ (we use the dot notation to select states of automata) upon receiving event *out* we would like to go straight to the sub-state $DISCONNECTED$. In extended hierarchical automata we provide the facility to explicitly specify such target-states of transitions.

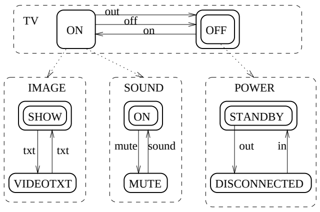


**Fig. 1.** Running example: TV-set as hierarchical automata

## 2.2 Definition of hierarchical automata

**Definition 1.** *Sequential automaton* $A$ is a 4-tuple $(\Sigma, s_0, L, \delta)$ where $\Sigma$ is the set of states of $A$, $s_0$ is the initial state of $A$, $L$ is the set of transition labels of $A$ (we will be more specific about labels later), $\delta \subseteq \Sigma \times L \times \Sigma$ is the transition relation.

Sequential automata can be parallely and hierarchically composed to build hierarchical automata. We assume that sequential automata participating in a hierarchical automaton have mutually disjoint sets of states. We introduce the notion of *composition function* that applies parallel and hierarchical composition to sequential automata.

**Definition 2.** Given a finite set $F = \{A_1, \ldots, A_n\}$ of sequential automata with mutually distinct state spaces $\Sigma_{A_i}$. We call $\gamma : \bigcup_{A \in F} \Sigma_A \to \mathbb{P}(F)$ a *composition function* on $F$ iff

– There exists a unique root automaton: $\exists_1 A \in F . A \notin \bigcup ran(\gamma)$, denoted by $\gamma_{root}$,
– Every non-root automaton has exactly one ancestor state:
$\bigcup ran(\gamma) = F \setminus \{\gamma_{root}\}$,
and $\forall A \in F \setminus \{\gamma_{root}\} . \exists_1 s \in \bigcup_{A' \in F \setminus \{A\}} \Sigma_{A'} . A \in \gamma(s)$,
– The composition function contains no cycles:
$\forall S \subseteq \bigcup_{A \in F} \Sigma_A . \exists s \in S . S \cap \bigcup_{A \in \gamma(s)} \Sigma_A = \emptyset$.

If $\mid \gamma(s) \mid = 1$ then $s$ is refined to a single automaton, if $\mid \gamma(s) \mid > 1$ then $s$ is refined to a parallel composition of automata. Otherwise, $\gamma(s) = \emptyset$, and we call $s$ a *basic state* (denoted by $Basic_\gamma(s)$).

*Example 2.* The composition function in Figure 1 is

$$\begin{aligned}
&\gamma = \{OFF \mapsto \{POWER\}, ON \mapsto \{SOUND, IMAGE\}\} \\
&\cup \{s \mapsto \emptyset \mid s \in \{SHOW, VIDEOTXT, SOUND.ON, \\
&\qquad\qquad MUTE, STANDBY, DISCONNECTED\}\}.
\end{aligned}$$

**Definition 3.** *A hierarchical automaton HA* is given by a triple $(F, E, \gamma)$ where $F = \{A_1, \ldots, A_n\}$ is a set of sequential automata with mutually distinct state spaces, $E$ is a set of events, and $\gamma$ a composition function on $F$.

A label $l$ in a sequential automaton $A \in F$ is a tuple $(ex, ac)$, where $ex$ is a proposition over event and state names (constant *true*, primitive events and/or states connected with $\wedge, \vee, \neg$ ) and $ac \subseteq E$ is a set of event names. Models of $ex$ are tuples $(\mathsf{C}, \mathsf{E})$, where $\mathsf{C}$ is a set of states and $\mathsf{E}$ is a set of events: $(\mathsf{C}, \mathsf{E}) \subseteq \bigcup_{A \in F} \Sigma_A \times E$.

*Example 3.* The hierarchical automata depicted in Figure 1 is given by the following triple: $(\{TV, IMAGE, SOUND, POWER\}, \{in, out, on, off, txt, mute, sound\}, \gamma)$, where $\gamma$ is the composition function as in Example 2.

We associate selector functions with transitions: if $(s, l, s') \in \delta_A$ then $source(s, l, s') = s$, $target(s, l, s') = s'$ and $label(s, l, s') = l$; *Source*, *Target* and *Label* denote the usual extension to sets of transitions. We also associate selector functions with labels: if $(ex, ac) \in L_A$ then $expr(ex, ac) = ex$ and $action(ex, ac) = ac$ and respectively *Expr*, *Action* for sets of labels.

### 2.3 Semantics of hierarchical automata

We describe the semantics of HA along the lines of the statecharts semantics given in [HN96]. We associate a Kripke structure to a hierarchical automaton. First we need some preliminary definitions.

**Definition 4.** A composition function $\gamma$ *on* $F = \{A_1, \ldots, A_n\}$ induces a *successor function* $\chi : \bigcup_{A \in F} \Sigma_A \rightarrow \mathbb{P}(\bigcup_{A \in F} \Sigma_A)$ defined by

$$\chi(s) = \{s' \mid \exists A \in F . A \in \gamma(s) \wedge s' \in \Sigma_A\},$$

thus, $s' \in \chi(s)$ iff $s'$ is a state of the direct sub-automaton of $s$.

We denote by $\chi^+$*and* $\chi^*$ *the irreflexive, resp. reflexive transitive closure of* $\chi$. Furthermore, $\chi$ induces an irreflexive *partial order* on states: $s \prec s'$ iff $s \in \chi^+(s')$. We say $s$ is the *parent* of $s'$ (denoted by $parent(s') = s$) iff $s' \in \chi(s)$ and $s$ is an *ancestor* of $s'$ (denoted by $ancestor(s, s')$) iff $s' \in \chi^+(s)$.

*A configuration* describes which states of sequential automata of an hierarchical automaton are simultaneously active.

**Definition 5.** Given a hierarchical automaton $HA = (F, E, \gamma)$. A set $C \subseteq \bigcup_{A \in F} \Sigma_A$ is a *configuration* of $HA$ iff

- Exactly one state of the root automaton $\gamma_{root}$ is in the configuration: $\exists_1\, s \in \Sigma_{\gamma_{root}}.s \in C$,
- Downward closure: $s \in C \wedge A \in \gamma(s) \Rightarrow \exists_1\, s_i \in \Sigma_A.s_i \in C$.

The set of all configurations is denoted by $Conf(\gamma)$. Note, configurations are also upward closed: $s \in C \wedge (\exists\, s'.s \in \chi(s') \Rightarrow s' \in C)$.

For the rest of this section let $HA = (F, E, \gamma)$ be a hierarchical automaton and consider a configuration $\mathsf{C}$ of $HA$ and a set $\mathsf{E}$ of events. A transition $(s, l, s')$ of $A \in F$ is called *enabled in* $(\mathsf{C}, \mathsf{E})$, if $s$ is *active in* $\mathsf{C}$, i.e. $s \in \mathsf{C}$, and $(\mathsf{C}, \mathsf{E}) \models expr(l)$. Let $ET_{(\mathsf{C},\mathsf{E})}$ denote the set of enabled transitions in $(\mathsf{C}, \mathsf{E})$. Then, we call an automaton $A$ of $F$ *enabled in* $(\mathsf{C}, \mathsf{E})$, if there exists a transition $(s, l, s')$ of $A$ which is in $ET_{(\mathsf{C},\mathsf{E})}$. Given transitions $t_1$ and $t_2$, we say that $t_2$ *has higher priority than* $t_1$, if $source(t1) \prec source(t2)$.

An execution step of $HA$ in $(\mathsf{C}, \mathsf{E})$ consists of synchronously firing all the transitions in a so-called *maximal non-conflicting set of transitions* [HN96], i.e. a set of transitions in which every enabled automaton contributes by exactly one transition unless it has an ancestor enabled automaton. Let $trs \subseteq ET_{(\mathsf{C},\mathsf{E})}$ be a set of enabled transitions in $(\mathsf{C}, \mathsf{E})$. A set $trs \subseteq ET_{(\mathsf{C},\mathsf{E})}$ is *maximal and non-conflicting* iff

1. $\mid trs \cap \delta_A \mid \leqslant 1$ for all $A \in F$, i.e. every automaton can contribute at most one transition,
2. $\forall\, t \in ET_{(\mathsf{C},\mathsf{E})}.t \in trs$ iff there does not exist $t' \in ET_{(\mathsf{C},\mathsf{E})}$ with higher priority, i.e. an automaton must contribute a transition to $trs$ if there is no higher priority transition.

Let $HPT_{(\mathsf{C},\mathsf{E})}$ denote the set of all transitions which are maximal and non-conflicting in $(\mathsf{C}, \mathsf{E})$.

Though, we do not define the semantics of statecharts in this paper, it is not difficult to see that the notion of maximal non-conflicting sets as by Harel&Naamad in [HN96] and formalization due to [MLPS97] coincides with the definition above.

*Example 4.* Assume a configuration $\mathsf{C} = \{TV.ON, IMAGE.SHOW, SOUND.ON\}$ and $\mathsf{E} = \{off, txt, mute\}$. Then transitions emerging from $TV.ON, IMAGE.SHOW$ and $SOUND.ON$ are enabled, but the set of maximal non-conflicting transitions consists of a singleton set containing solely the transition from $TV.ON$ to $TV.OFF$.

**Definition 6.** The semantics of $HA$ is a Kripke structure $\mathbf{K} = (\mathbf{S}, \mathbf{s_0}, \xrightarrow{STEP})$ where

– $\mathbf{S} = Conf(\gamma) \times \mathbb{P}(E)$ is the set of states of $\mathbf{K}$,
– $\mathbf{s_0} \in \mathbf{S}$, where $\mathbf{s_0} = (C_0, \emptyset)$ is the initial state of $\mathbf{K}$, where $C_0$ is defined as follows. Let $\gamma_{root} = (\Sigma, s_o, L, \delta)$ and $\Sigma_0 = \bigcup_{A \in F} s_{0_A}$ ($\Sigma_0$ is the set of all initial states of automata in $F$), then $C_0 = (\chi \mid_{\Sigma_0})^*(s_0)$.
– $\xrightarrow{STEP} \subseteq \mathbf{S} \times \mathbf{S}$ is the transition relation of $\mathbf{K}$, where $(\mathsf{C}, \mathsf{E}) \xrightarrow{STEP} (\mathsf{C}', \mathsf{E}')$ iff there exists a maximal non-conflicting set $trs \in HPT_{(\mathsf{C},\mathsf{E})}$, or if $HPT_{(\mathsf{C},\mathsf{E})} = \emptyset$ then $trs = \emptyset$, such that

$$\mathsf{C}' = (\mathsf{C} \setminus \chi^*(Source(trs))) \cup Target(trs) \cup (\chi^+(Target(trs)) \cap \bigcup_{A \in F} s_{0_A})$$

and $\mathsf{E}' = Action(Label(trs))$.

**Proposition 7.** *Given a hierarchical automaton $HA = (F, E, \gamma)$ and its semantics $\mathbf{K} = (\mathbf{S}, \mathbf{s_0}, \xrightarrow{STEP})$. Let $(\mathsf{C}, \mathsf{E})$ be a state of $\mathbf{K}$, let trs be a maximum non-conflicting set $trs \in HPT_{(\mathsf{C},\mathsf{E})}$ and let*

$$\mathsf{C}' = (\mathsf{C} \setminus \chi^*(Source(trs))) \cup Target(trs) \cup (\chi^+(Target(trs)) \cap \bigcup_{A \in F} s_{0_A}).$$

*Then $\mathsf{C}'$ is a configuration: $\mathsf{C}' \in Conf(\gamma)$.*

The proof is by induction over the cardinality of subsets of $HPT_{(\mathsf{C},\mathsf{E})}$. This proposition explains that the transition relation is well-defined.

## 3 Extended hierarchical automata

The operational semantics of statecharts as described in [HN96] and formalized in [MLPS97] is rather involved. This intricacy is mainly caused by the complicated interplay of interlevel transitions and the priority rule of statecharts.

Harel considers interlevel transitions as important concept of the language [Har87]: "... as our methods does not necessarily advocate layer-by-layer development; it is more flexible and encourages interlevel connections too, whenever appropriate." Hence we will not rule them out, instead we present an approach which allows to describe interlevel transitions but nevertheless has simple operational semantics. Since the priority concept of statecharts is very simple in case of non-interlevel transitions, we investigate a reformulation of interlevel transitions which allows to handle them just as ordinary transitions w.r.t. the priority concept.

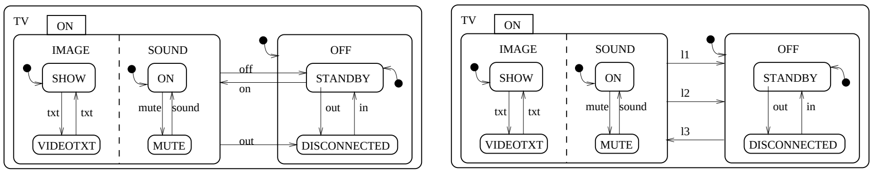


**Fig. 2.** Statechart with interlevel transition (left) and its desired extension.

Consider the left statecharts in Figure 2. Transitions from $TV.ON$ to $DISCONNECTED$, from $TV.ON$ to $STANDBY$ and from $STANDBY$ to $TV.ON$ are interlevel transitions. If any one of those is taken then not only the source state is exited but also the parent of each source state and entering a state means that the parent state is entered, too. Our idea is to lift those transitions to the uppermost states that are exited and entered when that transition is taken. This is done in the right statechart in Figure 2. Now we have to preserve information to restrict enabledness of those transitions: we extend labels by a set of state names, which were the original states of the transition; we call this extension *source restriction*. Furthermore, we have to preserve the information needed to determine the target states: we add another label extension *td* (which we call *target determinator*) that is a set of states that were entered originally. Hence label extensions in the right statecharts are as follows: $sr(l1) = \emptyset$, $td(l1) = \{STANDBY\}$, $sr(l2) = \emptyset$, $td(l2) = \{DISCONNECTED\}$, $sr(l3) = \{STANDBY\}$, $td(l3) = \{SHOW, SOUND.ON\}$. The trigger part is taken over unchanged from the corresponding transitions on the left. In the sequel we apply this idea to hierarchical automata.

*Example 5.* Note, that the specification in Figure 2 more faithfully reflects a conventional TV-set than the hierarchical automaton in Figure 1. Now there is a clear distinction between two groups of events the are responsible for switching in and out and for power supply: the events *on* and *off* switch between active mode and the standby mode, whereas *in* and *out* switch between a plugged-in and plugged-out TV-set. With this particular hierarchy of states the improved specification became possible only with interlevel transitions.

Note, the source restriction extension can already be expressed in HA and statecharts my means of state references in the trigger part of transitions. But target determinator extension has no counterpart in statecharts and HA. Statecharts don't need it due to the presence of interlevel transitions which can explicitly enter the correct target configuration. In this section we work out details of such extension on hierarchical automata. In Section 4 we give a translation from statecharts to extended hierarchical automata.

Another problem with the present semantics is that its descriptional style does not provide algorithmic insight, how one execution step can be computed, i.e., basically how to find a maximal non-conflicting set of transitions. We are looking for an operational description that allows to say for each enabled transition how it participates in a current step. Hence we need conditions on how transitions synchronize within one step in a priority preserving manner. This results in the operational semantics for extended hierarchical automata that is worked out in the sequel.

In order to formalize constraints that the label extensions must satisfy we need a restriction operation that allows for selecting sub-functions from the tree-like composition function.

**Definition 8.** Given a composition function $\gamma$ on the finite set $F = \{A_1, \ldots, A_n\}$ of sequential automata. *Restriction* of $\gamma$ to state $s$ is a composition function $\gamma\mid_s$

defined as domain restriction to sub-states of $s$: $\gamma \mid_{\chi^*(s)}$. The restriction is extended pointwise to sets of states.

**Definition 9.** *An extended hierarchical automata EHA* is a 3-tuple $(F, E, \gamma)$, where $F = \{A_1, \ldots, A_n\}$ is a set of sequential automata with mutually distinct state spaces, $E$ is a set of events, and $\gamma$ a composition function on $F$.

Labels of the sequential automata are extended as follows: the label $l$ of transition $(s, l, s')$ in the automaton $A \in F$ is a 4-tuple $(sr, ex, ac, td)$, where $sr \in Conf(\gamma \mid_s)$, $td \in Conf(\gamma \mid_{s'})$, and as before $ex$ is a proposition guarding the transition, and $ac$ is a set of generated events.

Given $EHA = (F, E, \gamma)$, the restriction of the composition function to states of some $A \in F$ yields an extended hierarchical automaton $EHA_A = (F_A, E_A, \gamma_A)$ by defining: $F_A = F \setminus \{A_i \mid \Sigma_{A_i} \cap \chi^*(\Sigma_A) = \emptyset\}, E_A = E, \gamma_A = \gamma \mid_{\chi^*(\Sigma_A)}$ . So the composition function $\gamma$ can be seen as assigning EHA (or possibly a parallel composition of EHA) an internal structure to states; state $s$ is refined by (the parallel composition of) $EHA_{A_i}$, for $A_i \in \gamma(s)$. This view suggests to define the semantics of extended hierarchical automata inductively over the composition function which is the main composition of EHA.

### 3.1 Semantics of extended hierarchical automata.

In the following let $EHA = (F, E, \gamma)$ be an extended hierarchical automaton. The semantics of $EHA$ is the Kripke structure $\mathbf{K} = (\mathbf{S}_{\mathbf{S_0}}, \xrightarrow{STEP})$ given in definition Definition 6, however with a step relation as defined in the sequel. In the sequel we refer to states of $\mathbf{K}$ as *status*.

Let $(\mathsf{C}, \mathsf{E})$ be a status of $\mathbf{K}$. A transition $t = (s, (sr, ex, ac, td), s')$ of an automata $A \in F$ is *enabled* in the status $(\mathsf{C}, \mathsf{E})$ (denoted by $enabled_{(\mathsf{C},\mathsf{E})}(t)$) iff $s \in \mathsf{C}$, $sr \subseteq \mathsf{C}$ and $(\mathsf{C}, \mathsf{E}) \models ex$.

**The step relation.** We define the step relation using inference rules. We distinguish between extended hierarchical automata 1) interacting with the environment by receiving events from environment and suppling generated events to the environment (open system approach) and 2) not interacting with the environment (closed system approach). The left rule is for closed systems. The right rule is for open systems, where the environment is allowed to add events after each step.

CLOSED SYSTEMS: OPEN SYSTEMS:

$$\frac{\gamma_{root} :: (\mathsf{C}, \mathsf{E}) \rightarrow (\mathsf{C}', \mathsf{E}')}{(\mathsf{C}, \mathsf{E}) \xrightarrow{STEP} (\mathsf{C}', \mathsf{E}')} \qquad \frac{\gamma_{root} :: (\mathsf{C}, \mathsf{E}) \rightarrow (\mathsf{C}', \mathsf{E}') \wedge \mathsf{E}' \subseteq \mathsf{E}''}{(\mathsf{C}, \mathsf{E}) \xrightarrow{STEP} (\mathsf{C}', \mathsf{E}'')}$$

These rules establish relation $\xrightarrow{STEP}$ based on relation $\rightarrow$; the latter is defined by inference rules given below. An inference in our rule system is denoted by $A :: (\mathsf{C}, \mathsf{E}) \rightarrow (\mathsf{C}', \mathsf{E}')$, where $A \in F$ is a sequential automaton; $(\mathsf{C}, \mathsf{E})$ is a status of the hierarchical automaton $EHA$, where $\mathsf{C}' \in Conf(\gamma \mid_{\Sigma_A})$ (i.e. a configuration of an EHA which has $A$ as root automaton) and $\mathsf{E}' \subseteq E$.

**Progress rule.** If there is an enabled transition an active sequential automaton takes it.

$$\frac{\begin{array}{l}\{s\} = \mathsf{C} \cap \Sigma_A \\ \exists\, tr \in \delta_A . enabled_{(\mathsf{C},\,\mathsf{E})}(tr) \wedge tr = (s, (sr, ex, ac, td), s')\end{array}}{A :: (\mathsf{C}, \mathsf{E}) \to (\{s'\} \cup td, ac)}$$

**Stuttering rule.** If an active sequential automaton does not have an enabled transition and the active state is a basic state then the automaton stutters and consumes events.

$$\frac{\begin{array}{l}\{s\} = \mathsf{C} \cap \Sigma_A \\ Basic_\gamma(s) \\ \forall\, tr \in \delta_A . tr = (s, l, s') \Rightarrow \neg\, enabled_{(\mathsf{C},\,\mathsf{E})}(tr)\end{array}}{A :: (\mathsf{C}, \mathsf{E}) \to (\{s\}, \emptyset)}$$

**Composition rule.** This rule explains how an automaton delegates its step to its sub-automata: if it does not have an enabled transition then its step collects the results of steps performed by its sub-automata. Note, that if the active state is refined into a parallel composition then the components are executed simultaneously and they do not interfere during the step.

$$\frac{\begin{array}{l}\{s\} = \mathsf{C} \cap \Sigma_A \\ \forall\, tr \in \delta_A . tr = (s, l, s') \Rightarrow \neg\, enabled_{(\mathsf{C},\,\mathsf{E})}(tr) \\ \gamma(s) = \{A_1, \ldots, A_m\} \neq \emptyset \\ A_1 :: (\mathsf{C}, \mathsf{E}) \to (\mathsf{C}'_1, \mathsf{E}'_1) \\ \ldots \\ A_m :: (\mathsf{C}, \mathsf{E}) \to (\mathsf{C}'_m, \mathsf{E}'_m)\end{array}}{A :: (\mathsf{C}, \mathsf{E}) \to (\{s\} \cup \mathsf{C}'_1 \cup \ldots \cup \mathsf{C}'_m, \mathsf{E}'_1 \cup \ldots \cup \mathsf{E}'_m)}$$

The next proposition explains that the transition relation is well-defined.

**Proposition 10.** *Given an extended hierarchical automaton* $EHA = (F, E, \gamma)$ *and its semantics* $\mathbf{K} = (\mathbf{S}, \mathbf{s_0}, \xrightarrow{STEP})$*. Let* $(\mathsf{C}, \mathsf{E})$ *be a state of* $\mathbf{K}$ *and let* $(\mathsf{C}', \mathsf{E}')$ *be such that* $(\mathsf{C}, \mathsf{E}) \xrightarrow{STEP} (\mathsf{C}', \mathsf{E}')$*. Then* $\mathsf{C}'$ *is a configuration, i.e.* $\mathsf{C}' \in Conf(\gamma)$*.*

This result implies that the semantics treats situations of non-determinism (one automaton may have more than one enabled transition) correctly: every automaton contributes at most one transition to each step of the overall extended hierarchical automaton. That in case of non-determinism all enabled transitions of an automaton (which are in conflict with each other) can contribute to the step relation is immediately evident from the existential quantification in the *progress rule*.

Furthermore, proposition 10 implies that in case of a hierarchical conflict (the source of one transition is an ancestor of the source of another transition) only one transition contributes to the current step. It is not difficult to see from the definition of the semantics, that this is the transition whose source is higher in the state hierarchy; this is in accordance with the priority rule in statecharts.

Note, that not every application of the progress or stuttering rule might contribute to the step relation, but only those that are collected by the composition rule. These superfluous inferences can be avoided by the following traversing strategy through the state hierarchy: beginning in the root we move downwards in the composition function if the current active automaton doesn't have an enabled transition; the downward movement ends if the progress or the stutter rule applies. An application of the progress rule stops further inferences in sub-components.

## 4 The relationship between statecharts and extended hierarchical automata

Now we describe how a sub-dialect of statecharts can be translated to extended hierarchical automata. We deal with a sub-dialect of statecharts, where transition labels are restricted as follows:

- only boolean combinations of predicates *in(st)* are allowed in expression *Cond*; informally, predicate *in(st)* is true iff *st* belongs to the current configuration of the system;
- the only effect of actions is the generation of events.

Since the correspondence between the state hierarchy of statecharts and composition function of hierarchical automata is straightforward, we focus on motivating how transitions of extended hierarchical automata can be used to model interlevel transitions of statecharts.

### 4.1 Transitions in statecharts

*Syntax of transitions.* The syntax of transitions in statecharts as supported by STATEMATE goes far beyond labeled arrows from one state to another. But our interest is the intermediate language that Harel&Naamad use to give semantics to statecharts: in [HN96] transitions of statecharts are transformed into what is called "full compound transitions" (full CT's). The purpose of full CT's is to collect labeling information that is spread over transition segments and to compute in advance sub-configurations to be entered when the transition is taken.

We briefly recall the full CT's below; a formal definition can be found in [MLPS97]. A full CT begins in a set of states and ends in a set of states. The transition can be taken only if its source states belong to the current configuration and if it is taken all target states are entered simultaneously. Only subsets of configurations may form source and target sets of transitions. The target set of states forms a sub-configuration below the scope of the transition.

*Transition scope.* Harel&Naamad associate *scope* to every full CT: "The scope of a full CT *tr* is the lowest OR-state in the hierarchy of states that is a proper common ancestor of all the sources and targets of *tr*." When a transition is taken then the scope is the lowest state in the state hierarchy that is an ancestor of sources and targets of the transition, but is not affected by taking the transition:

the scope has two direct sub-states that are roots of the configuration that are exited and entered, resp..

Assume a transition $tr$, its scope $sc$ and a status $(\mathsf{C}, \mathsf{E})$. Let $Uexit$ and $Uenter$ be direct sub-states of the scope such that $source(tr) \subseteq \chi^*(Uexit)$ and $target(tr) \subseteq \chi^*(Uexit)$. We call $Uexit$ and $Uenter$ *the uppermost states that are exited, resp. entered,* when $tr$ is taken. Let $exit_{(\mathsf{C},\mathsf{E})}$ denote the states which are exited when the transition is taken: $exit_{(\mathsf{C},\mathsf{E})}(tr) = \mathsf{C} \cap \chi^*(Uexit)$. Let $\mathsf{C}'$ be a configuration below $Uenter$, i.e. $\mathsf{C}' \in Conf(\gamma \mid_{Uenter})$, such that $target(tr) \subseteq \mathsf{C}'$. This is the configuration to be entered and we denote it with $enter_{(\mathsf{C},\mathsf{E})}(tr) = \mathsf{C}'$.

*Example 6.* Consider the statechart in Figure 3 and its counterpart in full CT's language. The scope of the full CT leading from $SHOW$ to $VIDEOTXT$ (denoted by "$SHOW \rightarrow VIDEOTXT$") is the state $IMAGE$, the scope of the full CT $STANDBY \rightarrow \{SOUND.ON, IMAGE.SHOW\}$ is the state $TV$ and the scope of
$TV.ON \rightarrow DISCONNECTED$ is $TV$ as well. We compute the enter and exit states of the transitions $t_1 = STANDBY \rightarrow \{SOUND.ON, IMAGE.SHOW\}$ and $t_2 = STANDBY \rightarrow DISCONNECTED$ w.r.t. status $(\mathsf{C}, \mathsf{E})$, where $\mathsf{C} = \{TV, OFF, STANDBY\}$ and $\mathsf{E} = \{on, out\}$:

$$exit_{(\mathsf{C},\mathsf{E})}(t_1) = \{OFF, STANDBY\}$$
$$enter_{(\mathsf{C},\mathsf{E})}(t_1) = \{TV.ON, IMAGE, SOUND, SHOW, SOUND.ON\}$$
$$exit_{(\mathsf{C},\mathsf{E})}(t_2) = \{STANDBY\}$$
$$enter_{(\mathsf{C},\mathsf{E})}(t_2) = \{DISCONNECTED\}$$

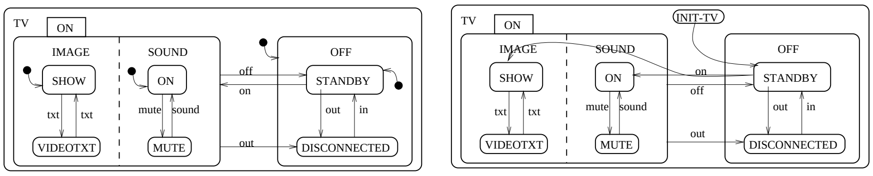


**Fig. 3.** Statechart with interlevel transition (left) and its counterpart in full CT's language (right).

*Step relation.* Like in hierarchical automata a maximal set of non-conflicting transitions must be taken simultaneously, which forms the step. Here, non-conflicting means that the set of transitions respects non-determinism and priorities. The maximality requirement ensures that an enabled non-conflicting transition is taken. Harel&Naamad use scope to compare transitions in order to decide on conflicts, i.e. either non-determinism or priority scheduling. Details can be found in [MLPS97].

*Example 7.* Let the statechart Figure 3 be in the same status as in the previous example: $(\mathsf{C}, \mathsf{E})$, where $\mathsf{C} = \{TV, OFF, STANDBY\}$ and $\mathsf{E} = \{on, out\}$. Then the full CT's departing from $STANDBY$ are in conflict because $STANDBY$ and $OFF$ would be exited if any of them would be taken. For deciding whether it is a priority or non-determinism situation we have to compare their scopes: we have a priority conflict since their scopes are different and one scope is an ancestor of the other. Since the scope of $STANDBY \rightarrow \{SOUND.ON, IMAGE.SHOW\}$ is higher in the state hierarchy ($TV$ is an ancestor of $OFF$) the priority is given to it.

### 4.2 Modeling statecharts' transitions in extended hierarchical automata

Given a statechart $SC$ we translate it into an extended hierarchical automaton $EHA$. Assume that the state hierarchy is translated into the composition function $\gamma$. This translation induces a bijection $\nu$ which maps states of the statecharts to states of the sequential automata of $EHA$. The function $\nu$ also extends to sets of states.

Given a transition $tr$ in a statechart. We associate with $tr$ its scope denoted by $scope$. Let $Uexit$ and $Uenter$ be the uppermost states that are exited and entered, resp., when the transition is taken. We translate $tr$ into a transition $htr$ in an extended hierarchical automaton as follows:

- The source of $htr$ is the state that corresponds to $Uexit$: $source(htr) = \nu(Uexit)$,
- The target state off $htr$ is constructed in a similar way: $target(htr) = \nu(Uenter)$,
- The source restriction of $htr$ is the original source set of $tr$: $sr(htr) = \nu(source(tr))$ (note, that $source(tr)$ may be a set of states); this is a valid entity as source restriction, because the original source set $source(tr)$ is a subset of the sub-configuration below $Uexit$ (property of full CT's language);
- The target determinator is constructed in a similar way: $td(tr) = \nu(target(htr))$; this is a valid entity as target determinator, because $target(tr)$ forms a sub-configuration below $Uenter$ (property of full CT's language);
- The triggering expression of $htr$ becomes a conjunction of event and condition of $tr$.
- The action part of $htr$ becomes the action part of $tr$.

*Example 8.* As seen in Figure 4 the composition function is deduced directly from the statechart in Figure 3. We omit the source restriction and target determinator labels where they are empty. The labels $l0$ to $l4$ are as follows.

- l0: $sr(l0) = \emptyset, td(l0) = \{STANDBY\}, expr(l0) = true$,
- l1: $sr(l1) = \emptyset, td(l1) = \{STANDBY\}, expr(l1) = off$,
- l2: $sr(l2) = \emptyset, td(l2) = \{DISCONNECTED\}, expr(l2) = out$,
- l3: $sr(l3) = \{STANDBY\}, td(l3) = \{SHOW, SOUND.ON\}, expr(l3) = on$.

### 4.3 Comparison with Argos

Rather than using explicit references to source/target configurations as in the advocated approach, Maraninchi [Mar91, Mar92] suggests an encoding of interlevel transitions by means of specific communication pattern between states in the state

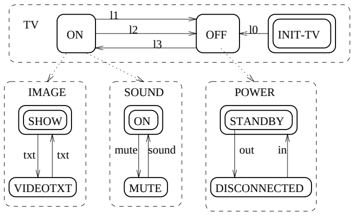


**Fig. 4.** Extended hierarchical automaton.

hierarchy. A transition $tr = (s, (sr, ex, ac, td), s')$ in an EHA can be modeled in Argos as follows: first, all states mentioned in the source restriction $sr$ inform state $s$ (by issuing designated events which are unique to the modeling of this transition) that the system currently resides in a configuration where transition $tr$ is considered enabled. These designated events (one for each state mentioned in $sr$) are sensed as input trigger in the corresponding Argos transition departing from $s$. Upon entering state $s'$ a further unique event is broadcasted which is used to enter the target configuration described by $td$.

As mentioned in the introduction this encoding requires a non-preemptive interrupt mechanism (otherwise the states in $sr$ could not inform state $s$ about the current sub-configuration) as well as perfect synchronicity of input/output events. Perfect synchronicity is needed since otherwise the chain reaction "informing $s$ about the current configuration", "performing the transition" and "enforcing the correct target configuration" would not be performed atomically which means that the encoding would not be faithful. Due to the fact that different interrupt mechanisms are employed in statecharts and Argos as well as different assumptions are made about synchronicity of events the translation of statecharts into Argos as intermediate format is rather cumbersome.

However, it is worth noting that the underlying idea how to simulate interlevel transitions are quite similar in Argos and EHA. While we explicitly recall the relevant state information in the respective transition guards, Argos encodes this information by introducing unique communication events for the simulation of each interlevel transition (so the explicit state reference is shifted into the unique identification of sub-configurations by means of sets of communication events).

## 5 Conclusion, applications and future work

We introduced extended hierarchical automata (EHA) as intermediate format for the implementation of add-on tools for statecharts as supported by Statemate. The EHA allow for a simple syntax yet advanced enough to deal with interlevel transitions; we described a translation from statecharts to EHA. A fairly simple operational semantics to EHA was given; through the translation the operational semantics describes the semantics of statecharts as well. The operational semantics of EHA was shown to have the same properties as the Harel&Naamad semantics of statecharts (a maximal non-conflicting set of transitions fires in one step). In the full paper we will show that the semantics of a statechart coincides with the semantics of the corresponding extended hierarchical automata.

Translation of statecharts into extended hierarchical automata removes a number of obstacles when translating statecharts to other languages, e.g., to input languages of model-checkers:

- multiple sources and multiple targets of a transition are eliminated,
- interlevel transitions are eliminated,
- there is no need to associate scopes to transitions,
- priorities are made explicit,

By this we obtain a simple procedure for computing maximal non-conflicting sets of enabled transitions in statecharts.

A useful application of the advocated intermediate format is that it directly implies a compilation schema of statecharts to input languages of transition system based model-checkers. Now its is possible to define a translation such that the size of the code is linear in the size of the statecharts, hence the compiler faithfully passes the structure of the statechart to the model-checker (provided the target system has primitives like sequential and parallel composition, and conditional choice).

This paper does not cover our work [MHLS97] on how extended hierarchical automata are translated to the input language of the SPIN verification system ([Hol91, Hol97]) which allows for simulation and linear temporal logic model-checking. We implemented a partial order reduction technique in our compiler which lead to sequential code and made the partial order reduction technique in SPIN obsolete for our code. Our experiments supported our hypothesis that the sequential code performs better in time and space than the parallel code combined with partial order reduction as implemented in SPIN [HP94]. As a case study we verified safety properties expressed in linear temporal logic of a statecharts model of the production cell [LL95]. Further experiments will show where limits and possible trade-offs are.

Further applications of the format are in, e.g. simulators of statecharts, code generators, and test sequence generators.

We will also use our approach to translate statecharts to symbolic model-checking tools and investigate how BDD-based techniques perform on model-checking statecharts (cf. [HK94]).

**Acknowledgment.** We thank Amir Pnueli, Carsta Petersohn and Willem-Paul de Roever for helpful comments and suggestions relating to this work. We also thank the referees for pointing us to related work on this topic and other suggestions. The work of E. Mikk is partially supported by *Technologiestiftung Schleswig-Holstein* within the CATI project.

## References

[Har87] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[HdR91] C. Huizing and W.-P. de Roever. Introduction to design choices in the semantics of Statecharts. *Information Processing Letters*, 37:205–213, February 1991.

[HK94] J. Helbig and P. Kelb. An OBDD–Representation of Statecharts. In *Proc. of the European Design and Test Conference EDAC'94*, pages 142–149, 1994.

[HN96] D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, Oct 1996.

[Hol91] G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[Hol97] G.J. Holzmann. The Model Checker Spin. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997. Special issue on Formal Methods in Software Practice.

[HP94] G.J. Holzmann and D. Peled. An improvement in formal verification. In *Proc. FORTE94*, Berne, Switzerland, October 1994.

[Hui91] C. Huizing. *Semantics of reactive systems: comparision and full abstraction*. PhD thesis, Technical University Eindhoven, 1991.

[JM94] F. Jahanian and A. Mok. Modechart: a specification language for real-time systems. *IEEE Transactions of Software Engineering*, 20(12):933–947, December 1994.

[LHHR94] N.G. Leveson, M.P.E. Heimdahl, H. Hildreth, and J.D. Reese. Requirements Specification for Process-Control Systems. *IEEE Trans. Soft Eng*, 20(9):684–707, September 1994.

[LL95] C. Lewerentz and T. Lindner. *Formal Development of Reactive Systems: Case Study Production Cell*. Number 891 in LNCS. Springer Verlag, 1995.

[Mar91] F. Maraninchi. The Argos language: Graphical Representation of Automata and Description of Reactive Systems. In *IEEE Workshop on Visual Languages*, Oct 1991.

[Mar92] F. Maraninchi. Operational and Compositional Semantics of Synchronous Automaton Compositions. In *CONCUR'92*, number 630 in Lecture Notes in Computer Science, pages 550–564. Springer-Verlag, 1992.

[MHLS97] E. Mikk, G. J. Holzmann, Y. Lakhnech, and M. Siegel. Implementing Statecharts in Promela/SPIN. Technical report, Manuscript, 1997.

[MLPS97] E. Mikk, Y. Lakhnech, C. Petersohn, and M. Siegel. On formal semantics of Statecharts as supported by STATEMATE. In *2nd BCS-FACS Northern Formal Methods Workshop*. Springer-Verlag, July 97.

[NVG91] S. Narayan, F. Vahid, and D. D. Gajski. System Specification and Synthesis with the SpecCharts Language. In *Proceedings, 1991 IEEE International Conference on Computer-Aided Design (ICCAD '91)*, pages 266–269, November 11-14 1991.