A Compositional Semantics for Statecharts*

J. Hooman
S. Ramesh†
W. P. de Roever

Eindhoven University of Technology
Dept. of Mathematics and Computing Science
P.O. Box 513
5600 MB Eindhoven, The Netherlands

Abstract

Statecharts is a behavioral specification language proposed for specifying large real-time, event driven, reactive systems. It is a graphical language based on finite state machines extended with many features like hierarchy, concurrency and broadcast communication. We give a compositional syntax and a denotational semantics for Statecharts.

1 Introduction

This paper concerns the semantics of a specification language for describing real-time reactive systems. Real-time reactive systems usually run forever, interact continuously with their environment, and have critical time requirements. Typical examples are telecommunication networks and avionic systems. Formal description of real-time reactive systems is an important area of research judging by the sheer number of proposed specification languages such as Statecharts [Har87,Har88], Esterel [BC85], Modecharts [JM89], Lustre [BCH85] and Signal [GB86]. All these specification languages are based on operational descriptions that characterize how a system evolves.

The behavioral specification language considered here is Statecharts [Har87]. Realizing the intuitive and pictorial appeal of finite state machines, Statecharts has been designed on the basis of such state machines. But it is free from the limitations of state machines, such as sequentiality, unstructuredness and exponential growth of states when describing concurrency. Indeed, Statecharts are double exponentially more succinct than state machines (Harel). Quoting [Har88],

Statecharts=finite state machines+depth+orthogonality+broadcast communication.

*This work was supported by ESPRIT Project 937: Debugging and Specification of Ada Real-Time Embedded Systems (DESCARTES).

†Supported by the Foundation for Computer Science Research in the Netherlands (NFI) with financial aid from the Netherlands Organization for Scientific Research (NWO).

Depth is achieved in Statecharts by allowing super states containing substates or even complete statecharts. When such a super state is exited all the computations inside are terminated. Super states may consist of orthogonal sub-statecharts which are executed in parallel. Orthogonal components interact with each other and with their environment by means of events which are broadcast throughout the whole system. External events or events generated in one component can cause new events in another component which, in turn, can cause more events. Thus a single event can give rise to a whole chain of events all of which are assumed to take place simultaneously; this assumption is essentially Berry's synchrony hypothesis [BC85] according to which a system is infinitely faster than its environment. It facilitates the specification task by abstracting from internal reaction times.

The synchrony hypothesis might introduce causal paradoxes like an event causing itself. In Esterel [BC85] causal paradoxes are syntactically disallowed whereas in Statecharts causal relationships are respected and paradoxes are removed semantically. Real-time is incorporated in Statecharts by having an implicit clock, by allowing transitions to be triggered by *time-outs* relative to this clock and by requiring that if a transition can be taken then it should be taken immediately.

1.1 Overview of our work

Our aim is to develop a *compositional* denotational semantics for Statecharts. This requires syntactical operators for building large statecharts from smaller ones. Our compositional semantics is based on a syntax for Statecharts which has been proposed in [HGR88]. Section 2.1 first informally introduces Statecharts and Section 2.2 contains this syntax.

The denotations describe observable entities (i.e. the events generated by a statechart), but also denote non-observable entities such as the causal relation between events. These non-observable entities, which can be sensed by a suitable program context, are needed to obtain a compositional semantics. In [HGR88] a compositional semantic model with minimal amount of non-observable entities (i.e. a fully abstract semantics) has been presented. This semantics forms the basis of our axiomatic system. The denotations of this semantics are prefix-closed sets of linear histories; infinite computations are represented by all their finite prefixes. Our semantics forms the basis for a proof system in which Statecharts are related to property based specifications. In order to express liveness properties, we do not use prefix closed sets. Our histories represent complete (possibly infinite) computations. Section 3 contains the details of the modified semantic model.

2 Syntax

2.1 General overview

A statechart can be considered as a tree of states, with the root state as the initial state. The leaves of the tree are basic states, like the states in a finite state machine. Other states

are super states containing their sons (in the tree) as substates. There are two types of super states: AND-states and OR-states. For instance, the statechart in Figure 1 has root

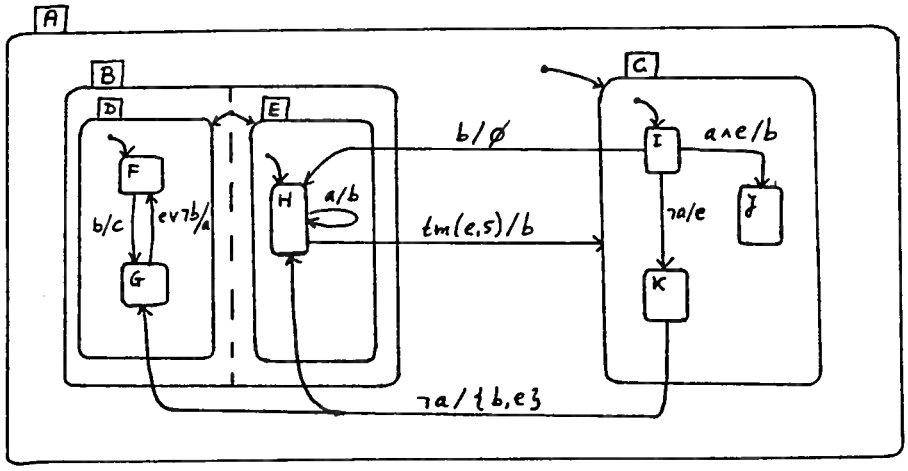

Figure 1:

state A and leaves F, G, H, I, J and K. A is an OR-state, with substates B and C. B is an AND-state (indicated by the dashed line) with D and E as its substates, called orthogonal components, whereas C is an OR-state having I, J and K as its substates. States are entered/exited either explicitly by taking a transition or implicitly because certain other states are entered/exited. Entering an AND-state (OR-state resp.) results in entering *all* (*exactly one* resp.) of its substates implicitly. In Figure 1: entering AND-state B results in entering both D and E, entering OR-state C results in entering exactly one of its substates I, J and K. Similarly, entering an orthogonal component of an AND-state results in implicitly entering all other components of this AND-state. When entering a super state the particular substate(s) which should be entered, is (are) marked by a default arrow, drawn as a transition with no source state, e.g. the default arrow inside C pointing to I.[1] When the transition from I to H is taken, H is entered explicitly and B, D, E, F are entered implicitly. Note that with a forked transition such as the one from K, more than one orthogonal component can be entered explicitly. Transitions between orthogonal components are not allowed (e.g. no transitions between F and H). When a state is exited all its substates are exited implicitly. Exiting an orthogonal component implies an implicit exit of all its orthogonal partners. So the transition from H to C leads to an implicit exit of D (and its substate F or G).

Transitions have labels of the form *'event part/action part'*.[2] The event part is a boolean expression involving atomic events a, b, e ,... - signals without measurable duration. These

[1] Unlike [Har87], we attach a default arrow to every super state; so also to AND-states.

[2] For the sake of simplicity we do not consider the general syntax of labels given in [HPPSS87]. There a

events can be generated by the outside world as an input to the statechart as well as by the statechart itself. The event expression specifies when the transition is enabled. The action part is a set of atomic events which are generated when the transition is taken (a singleton is denoted by its element).

Execution in orthogonal components proceeds concurrently and events generated in one component are broadcast throughout the system, possibly triggering new transitions in other components. This will in general lead to a whole chain of transitions which, by the synchrony hypothesis, take place simultaneously in a single (time) step. The set of transitions taken in a step is a *maximal* set in which there is at most one transition per orthogonal component and there exists a causal relationship between transitions: each transition is enabled by either external events or events generated by other transitions. The general idea is that staying in a state takes some time, whereas taking a transition is instantaneous.
In our example the system can be in the states A, B, D, E, F and H simultaneously. When a is generated externally in this configuration the transition from (and to) state H will generate b, causing a transition from F to G which generates c.

A transitions with event part a is taken when a is generated somewhere in the system. The meaning of $a \wedge b$ (resp. $a \vee b$) is: a transition with this event part is taken in a step if both a and b (resp. a or b) are generated somewhere in the system in this step. λ is a special event which occurs (by definition) in every step. $tm(e, n)$ denotes a *time-out* event which is generated at a particular step if n time steps earlier event e has happened. A transition with event part $\neg a$ is taken in a step if a is not generated at all during this step.[3] In our syntax, negations are immediately succeeded by atomic events, time-outs or λ.

2.2 Syntax of Statecharts

The objects obtained using our syntax are in general unfinished statecharts having arcs without either source or target state. In the sequel we use the word statechart for both unfinished and finished statecharts.
The primitive objects (see Figure 2.2) of our syntax are so called

- Basic Statecharts: $[I, O, S]$, where S is a state name, I a set of incoming arcs and O a set of outgoing arcs. Only the outgoing arcs are labeled with an event/action pair.

We have the following operators (let B be a basic statechart, U, U_1, U_2 be statecharts, T, T_1, T_2 be transition names and let a be the name of an atomic event):

---

label includes an additional condition part, variable assignments are allowed in action parts and there are special events to signal entry and exit of a state. The proposed axiomatic system can be easily extended to the general case.

[3] There are several possible interpretation of $\neg a$, here we take the approach recently advocated by Pnueli [PS88].

Figure 2: Basic Statechart $[I, O, S]$, with $I = \{T_3, T_6, T_8\}$ and $O = \{T_4, T_7\}$

- Statification: $Stat(B, U, T)$; makes (the state of) B a super state with U inside it and the incoming transition T of U as its default.

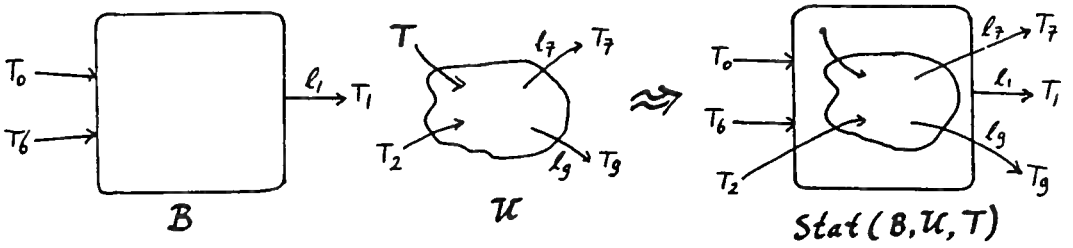

Figure 3: Statification

- Or-construct: $Or(U_1, U_2)$; leads to a statechart which becomes an OR-state after statification.
- And-construct: $And(U_1, U_2)$; yields an AND-state after statification.

In the constructs above both constituents should not have joint incoming or joint outgoing transitions with the same name, except for the AND-construct where joint incoming transitions are allowed.

- Connect: $Connect(U, T_1, T_2)$; results in a chart identical to U except that outgoing arc T_1 and incoming arc T_2 of U are connected to form a single complete transition.

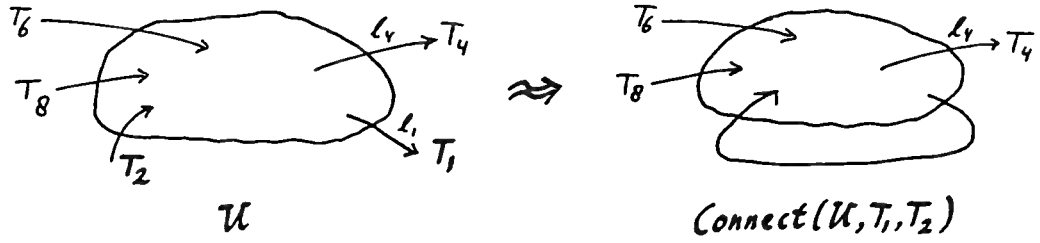

Figure 4: Connection

- Hide-Closure: $HiCl(U,a)$; hides any generation of a by U (Hiding) and makes U insensitive to any a generated by the environment (Closure).

After the informal introduction into the syntax of Statecharts above, we give the formal syntax. First we define the labels that can be associated with the transitions of any statechart and the event expressions used in these labels.

2.2.1 Events

Let E_e be a set of elementary/atomic events. The set of composite events Exp is defined inductively as the least set satisfying:

- $\lambda \in Exp, \neg\lambda \in Exp$.
- if $e \in E_p$ then $e \in Exp, \neg e \in Exp$.
- if $e \in Exp, n \in N \setminus \{0\}$ then $tm(e,n) \in Exp, \neg tm(e,n) \in Exp$.
- if $e_1, e_2 \in Exp$ then $e_1 \vee e_2 \in Exp, e_1 \wedge e_2 \in Exp$.

2.2.2 Transition Labels

The set of all symbols that can label the transitions of a statechart is the set **Lab** defined as follows:

$$\mathbf{Lab} = \{E/A \mid E \in Exp, A \subseteq E_e, A \text{ is finite}\}$$

If A is a singleton set then we often use the event itself, i.e. E/a abbreviates $E/\{a\}$.[4]

2.2.3 Formal Syntax of Statecharts

Let Σ be the set of all states (or more precisely state names) and $\mathcal{T}_I$ and $\mathcal{T}_O$ be the set of all (names of) incoming and outgoing transitions of any statechart such that $\mathcal{T}_I \cap \mathcal{T}_O = \emptyset$. Also let $L : \mathcal{T}_O \to \mathbf{Lab}$ denote the labeling function that labels all the outgoing transitions. Assume $\mathcal{T} = \mathcal{T}_I \cup \mathcal{T}_O$ and E_e are countable.
The set of statecharts is defined by the following BNF-grammar, where $a \in E_e$, $I \subseteq \mathcal{T}_I$, $O \subseteq \mathcal{T}_O$, I and O are finite, $\{T, T_2\} \subseteq \mathcal{T}_I$, $T_1 \in \mathcal{T}_O$, $S \in \Sigma$.

$$\begin{aligned}
U &::= Disj \mid Conj \\
Disj &::= Prim \mid \mathbf{Or}(Disj,Disj) \mid \mathbf{Connect}(Disj, \mathbf{T_1}, \mathbf{T_2}) \\
Conj &::= \mathbf{And}(Default,Default) \mid \mathbf{And}(Default,Conj) \\
Prim &::= Basic \mid Default \mid \mathbf{HiCl}(U, \mathbf{a}) \\
Default &::= \mathbf{Stat}(Basic,U,\mathbf{T}) \\
Basic &::= [\mathbf{I},\mathbf{O},\mathbf{S}]
\end{aligned}$$

[4] In the original syntax of labels as given in [HPPSS87], the action A is of the form $a_1, \cdots, a_n$ whereas we take A to be the set containing these events.

2.2.4 Syntactic Restrictions

There are certain syntactic conditions to be satisfied by any statechart. In order to describe these conditions we define two functions IN and OUT; for a given statechart U, IN(U) and OUT(U) are the sets of incoming and outgoing transitions of U, respectively.

	IN	OUT
$[I,O,S]$	I	O
$Stat(B,U,T)$	$IN(B) \cup IN(U) \setminus \{T\}$	$OUT(B) \cup OUT(U)$
$Connect(U,T_1,T_2)$	$IN(U) \setminus \{T_2\}$	$OUT(U) \setminus \{T_1\}$
$Or(U_1,U_2)$	$IN(U_1) \cup IN(U_2)$	$OUT(U_1) \cup OUT(U_2)$
$And(U_1,U_2)$	$IN(U_1) \cup IN(U_2)$	$OUT(U_1) \cup OUT(U_2)$
$HiCl(U,a)$	$IN(U)$	$OUT(U)$

Then we have the following syntactic restrictions:

- For $Connect(U,T_1,T_2)$: $T_1 \in OUT(U)$ and $T_2 \in IN(U)$.
- For $Stat(B,U,T)$: $T \in IN(U)$, $IN(B) \cap IN(U) = \emptyset$, and $OUT(B) \cap OUT(U) = \emptyset$.
- For $Or(U_1,U_2)$: $IN(U_1) \cap IN(U_2) = \emptyset$ and $OUT(U_1) \cap OUT(U_2) = \emptyset$.
- For $And(U_1,U_2)$: $OUT(U_1) \cap OUT(U_2) = \emptyset$.

Remarks:

1. In $And(U_1,U_2)$, the intersection of $IN(U_1)$ and $IN(U_2)$ need not be empty. Incoming arcs with identical names are 'merged'.
2. In $Stat(B,U,T)$, there should always be at least one incoming arc to U, to be taken as the default.
3. The *Concat* operation given in [HGR88] has not been provided in our syntax. It can be considered as a derived operation:
   $Concat(U_1,U_2,T_1,T_2) \equiv Connect(Or(U_1,U_2),T_1,T_2)$.

3 Denotational Semantics

As mentioned in the introduction, the semantic model associates with a statechart the set of all (maximal) computation histories representing complete computations. It has been shown in [HGR88] that, besides denotations for events generated at each computation step (the observables) and denotations for entry and exit, the following two additional denotations are necessary and sufficient to obtain a compositional semantics: (1) a set of all events assumed to be generated by the whole system (i.e. statechart together with its environment) at each step and (2) a causality relation between generated events. More precisely, a computation history h of a statechart U is of the form $h = (\hat{s}, i, f, o, s)$ where

- $\hat{s} \in N$ models the start step (N denotes the set of natural numbers).

- $i \in \mathcal{T}_I \cup \{\star\}$ is an incoming transition or $\star$ to model an implicit entry.
- $f : N \rightarrow \{(F, C, <) \mid F \subseteq C \subseteq E_e$ and $<$ a total order on $C\}$ records for every step n a triple $(F, C, <)$, where
  - F is a subset of the events generated by U. Considering the chain of transitions in step n, F contains the events which are generated by U, for the first time in this chain.
  - C is the set of events generated by the total system (i.e. U and its environment) in step n.
  - $<$ denotes the causal relationship between events generated by the whole system . If a causes b then $a < b$. If there is no causal relation, then the semantics of U will contain two histories: one with $a < b$ and another with $b < a$.
- $o \in \mathcal{T}_O \cup \{\star, \perp\}$ is an outgoing transition or $\star$ for an implicit exit, or $\perp$ when there is no exit.
- $s \in N \cup \{\infty\}$ denotes the exit step.

Henceforth, h will denote $(\hat{s}, i, f, o, s)$ and similarly for super- and sub-scripts:
h' denotes $(\hat{s}', i', f', o', s')$, h_1 denotes $(\hat{s}_1, i_1, f_1, o_1, s_1)$, etc.
For a function f as above, the fields of $f(n)$ are selected by $f^F(n)$, $f^C(n)$ and $f^<(n)$.
Define $\mathcal{H} = \{h \mid \hat{s} < s,\ o = \perp \leftrightarrow s = \infty$, and $(v \leq \hat{s} \vee v > s) \rightarrow f^F(v) = \emptyset\}$.

Figure 5 shows why F is not equal to the set of *all* events generated by U. If a occurs

Figure 5:

externally, then U_1 generates a at every step, whereas U_2 does not generate a. This difference can not be sensed, however, by other statecharts, because a is generated in the system already. In order to get the same semantics for U_1 and U_2, both have an empty F-set to denote that both are not responsible for the first generation of a.

Our semantic domain is given by $(\mathcal{D}, \sqsubseteq)$, where $\mathcal{D} = \{H \mid H \subseteq \mathcal{H}\}$ and $D_1 \sqsubseteq D_2$ iff $D_1 \subseteq D_2$ for all $D_1, D_2 \in \mathcal{D}$. It is easy to show that our domain is a complete lattice with bottom element $\emptyset$ and top element $\mathcal{H}$. We give the semantics of statecharts by defining a semantic function $\mathcal{M}$ that maps any statechart to an element of $\mathcal{D}$, so to a set of histories. The semantics is an *a priori* semantics that anticipates an arbitrary environment.

Definition 3.1 (Basic) Any general computation history of a basic statechart $[I, O, S]$ enters the chart implicitly (denoted by $\star$) or via one of the arcs in I at a particular time step and starts waiting to exit the statechart from the next step onwards. Then there are three situations possible: it waits forever or it exits the chart at a finite step implicitly (also denoted by $\star$) or by taking one of the outgoing arcs in O; in the latter case, the necessary condition for taking the transition should be true.
Using the predicates *wait* and *fire*, defined below, the semantics is given as follows.

$$\mathcal{M}([I,O,S]) = \{h \in \mathcal{H} \mid i \in (I \cup \{\star\}) \wedge \forall v, \hat{s} < v < s : wait(O,v) \wedge$$
$$[o = \perp \vee (o = \star \wedge f^F(s) = \emptyset) \vee fire(O,s)]\}$$

Let the label of an outgoing transition $t \in O$ be given by: $L(t) = E_t/A_t$.
$wait(O,v)$ characterizes the situation in which none of the transition in O can be taken. Then none of the triggers of these transitions evaluate to true and no event is generated by the statechart. Consequently, *wait* is defined as follows:

$$wait(O,v) \equiv f^F(v) = \emptyset \wedge \bigwedge_{t \in O} \neg inC(E_t, v)$$

where $inC(E_t, v)$ expresses that E_t evaluates to true at step v. It is defined inductively as follows:

$$\begin{aligned}
inC(\lambda, v) &\equiv true \\
inC(e, v) &\equiv e \in f^C(v), \quad \text{for } e \in E_e \\
inC(\neg e, v) &\equiv \neg inC(e, v) \\
inC(e_1 \vee e_2, v) &\equiv inC(e_1, v) \vee inC(e_2, v) \\
inC(e_1 \wedge e_2, v) &\equiv inC(e_1, v) \wedge inC(e_2, v) \\
inC(tm(e,n), v) &\equiv \begin{cases} inC(e, v-n) \wedge \forall v', v-n < v' < v : \neg inC(e, v') & \text{if } v \geq n \\ false & \text{otherwise} \end{cases}
\end{aligned}$$

Predicate $fire(O,v)$ describes the condition for taking a transition $t \in O$ at step v. Then its trigger E_t evaluates to true, so $inC(E_t, v)$ must hold and all the events in A_t are generated. Furthermore, certain causal relations exist between newly generated events and the events that triggered the transition. These relations are expressed by predicate $rel(E_t, a, v)$, defined below. Consequently, *fire* is defined as follows:

$$fire(O,v) \equiv \bigvee_{t \in O} (o = t \wedge inC(E_t, v) \wedge f^F(v) \subseteq A_t \subseteq f^C(v) \wedge \bigwedge_{a \in A_t} a \in f^F(v) \rightarrow rel(E_t, a, v))$$

Predicate $rel(E_t, a, v)$ provides the necessary causal relation between an event $a \in A_t$ and the events in E_t. For instance, if $E_t \equiv b$ then we need $(b, a) \in f^<(v)$, whereas for $E_t \equiv \neg b$

there should not be any relation between a and b because b does not occur in step v. We define rel inductively as follows:

$$\begin{aligned}
rel(\lambda, a, v) &\equiv true \\
rel(b, a, v) &\equiv \begin{cases} (b,a) \in f^{<}(v), & \text{if } b \not\equiv a \\ false & \text{if } b \equiv a \end{cases} \\
rel(tm(e,n), a, v) &\equiv inC(tm(e,n), v) \\
rel(\neg e, a, v) &\equiv \neg inC(e, v) \\
rel(e_1 \vee e_2, a, v) &\equiv rel(e_1, a, v) \vee rel(e_2, a, v) \\
rel(e_1 \wedge e_2, a, v) &\equiv rel(e_1, a, v) \wedge rel(e_2, a, v)
\end{aligned}$$

Definition 3.2 (Or) The semantics of a *Or* construct is the union of the semantics of its constituents.

$$\mathcal{M}(Or(U_1, U_2)) = \mathcal{M}(U_1) \cup \mathcal{M}(U_2)$$

Definition 3.3 (Connect) Execution of $Connect(U, T_1, T_2)$ consists of first (a) entering U via an arc other than T_2, and then (b) taking transitions as specified by U, indefinitely, or exiting U either via an arc other than T_1, or exiting via T_1, re-entering U via T_2, and repeating (b). Given two sets of histories D_1, D_2, we define $CONC(D_1, D_2, T_1, T_2)$ (informally) as the set of (i) histories from D_1 which do not exit via T_1, and (ii) histories which consist of a history from D_1 exiting via T_1 followed by a history from D_2 entering via T_2.[5]

$$\begin{aligned}
&CONC(D_1, D_2, T_1, T_2) = \\
&\quad \{h \mid h \in D_1 \wedge o \neq T_1\} \cup \\
&\quad \{h \mid \exists h_1 \in D_1, h_2 \in D_2 : \hat{s} = \hat{s}_1 \wedge s_1 = \hat{s}_2 \wedge s = s_2 \wedge i = i_1 \wedge i_2 = T_2 \wedge \\
&\qquad o_1 = T_1 \wedge o = o_2 \wedge f^F = f_1^F \uplus f_2^F \wedge f^C = f_1^C = f_2^C \wedge f^{<} = f_1^{<} = f_2^{<}\}
\end{aligned}$$

Then $\mathcal{M}(Connect(U, T_1, T_2))$ can be obtained by removing the histories with a T_2 entry from the largest set satisfying $D = CONC(\mathcal{M}(U), D, T_1, T_2)$, i.e., the greatest fixed point $\nu_X.CONC(\mathcal{M}(U), X, T_1, T_2))$. (Note that such a set D will not contain histories which exit via T_1, because a T_1-exit leads—by (b) above—to a T_2-entry into U.) It is easy to see that $CONC$ is monotonic in its second argument and hence has a greatest fixed point in our complete lattice (see e.g. [dB80]). This leads to

$$\mathcal{M}(Connect(U, T_1, T_2)) = del_{T_2}(\nu_X.CONC(\mathcal{M}(U), X, T_1, T_2))$$

where $del_{T_2}(D) = \{h \in D \mid h = (\hat{s}, i, f, o, s) \wedge i \neq T_2\}$.

The semantics can also be given as the intersection of approximations:

$$\mathcal{M}(Connect(U, T_1, T_2)) = del_{T_2}(\bigcap_{k \in N} D_k)$$

with $D_0 = \mathcal{H}$, and for $k \geq 0$: $D_{k+1} = CONC(\mathcal{M}(U), D_k, T_1, T_2))$.

[5] *CONC* stands for concatenation.

So the semantics consists of all those histories that exit and re-enter U through the connected arc for a finite/infinite number of times. All the finite histories eventually exit U via an arc other than T_1.

Definition 3.4 (And) A history h from the semantics of $And(U_1, U_2)$ is obtained by combining h_1 from $\mathcal{M}(U_1)$ and h_2 from $\mathcal{M}(U_2)$, provided certain conditions are fulfilled. Since all orthogonal components of an And-construct are entered and exited simultaneously, the entry steps, $\hat{s}_1$ and $\hat{s}_2$, must be equal, and also the exit steps, s_1 and s_2. Furthermore, the claims in h_1 and h_2 about the total system—represented by the C and $<$ components—must be the same. The incoming transition in h, component i, can be either

- a $\star$, denoting an implicit entry, if both i_1 and i_2 are $\star$, or
- a joint incoming transition, so $i = i_1 = i_2$, or
- the incoming transition of one, provided the other is $\star$.

The o component in h, describing the way in which $And(U_1, U_2)$ is exited, can be either

- a $\star$, if $o_1 = o_2 = \star$, denoting an implicit exit, or
- an outgoing transition of one, provided the other is $\star$, or
- a $\perp$, to denote that $And(U_1, U_2)$ is never exited, if both o_1 and o_2 are $\perp$.

This leads to the following definition:

$$\begin{aligned}\mathcal{M}(And(U_1,U_2)) = \{h \mid \exists h_1 \in \mathcal{M}(U_1), h_2 \in \mathcal{M}(U_2) : \hat{s} = \hat{s}_1 = \hat{s}_2 \wedge s = s_1 = s_2 \wedge \\ f^C = f_1^C = f_2^C \wedge f^< = f_1^< = f_2^< \wedge f^F = f_1^F \uplus f_2^F \wedge \\ [(i = i_1 = i_2) \vee (i = i_1 \wedge i_2 = \star) \vee (i = i_2 \wedge i_1 = \star)] \wedge \\ [(o = o_1 \neq \perp \wedge o_2 = \star) \vee (o = o_2 \neq \perp \wedge o_1 = \star) \vee o = o_1 = o_2 = \perp]\}\end{aligned}$$

Definition 3.5 (Statification) The semantics of $Stat(B, U, T)$ is similar to $\mathcal{M}(And(B, U))$, except for the way in which the statified chart is entered; any entry to B leads to U via the default arc T and the direct entry to T is no longer possible. Consequently, the semantics is given as follows:

$$\begin{aligned}\mathcal{M}(Stat(B,U,T)) = \{h \mid \exists h_1 \in \mathcal{M}(B), h_2 \in \mathcal{M}(U) : \hat{s} = \hat{s}_1 = \hat{s}_2 \wedge s = s_1 = s_2 \wedge \\ f^C = f_1^C = f_2^C \wedge f^< = f_1^< = f_2^< \wedge f^F = f_1^F \uplus f_2^F \wedge \\ [(i = i_1 \wedge i_2 = T) \vee (i = i_2 \neq T \wedge i \neq \star \wedge i_1 = \star)] \wedge \\ [(o = o_1 \neq \perp \wedge o_2 = \star) \vee (o = o_2 \neq \perp \wedge o_1 = \star) \vee o = o_1 = o_2 = \perp]\}\end{aligned}$$

Definition 3.6 (Hide and Close) In the semantics of $HiCl(U, a)$ we first require that every occurrence of a is generated by U. Thereafter a is hidden by allowing arbitrary behaviour for it in the new histories as far as the C and $<$ component are concerned, and removing a from the F component. First we define the *pointwise subtraction* of a set-valued function g and a set A, notation $g \dot{-} A$, as follows, $(g \dot{-} A)(v) = g(v) - A$, for all $v \in N$. For a function $f^<$ s.t. $f^<(v) \subseteq E_e \times E_e$, let $f^<|_a$ be defined as $f^<|_a = f^< \dot{-} (E_e \times \{a\}) \dot{-} (\{a\} \times E_e)$. Then the semantics is given by

$$\mathcal{M}(HiCl(U,a)) = del_a(\{h \mid h \in \mathcal{M}(U) \wedge \forall v : a \in f^C(v) \rightarrow a \in f^F(v)\})$$

where $del_a(D) = \{h \in \mathcal{H} \mid \exists h_1 \in D : \hat{s} = \hat{s}_1 \wedge i = i_1 \wedge o = o_1 \wedge s = s_1 \wedge$
$f^F = f_1^F \dot{-} \{a\} \wedge f^< |_a = f_1^< |_a \wedge f^C \dot{-} \{a\} = f_1^C \dot{-} \{a\}\}$.

4 Related Work

A denotational semantics has been given for the graphical, state-based, specification language Statecharts. This semantics serves as a basis for a compositional axiomatisation of Statecharts in terms of a logic which is strong enough to express both safety and liveness properties. In contrast with [HGR88], we did not aim at full abstractness; our purpose was to give the semantics of a small subset of Statecharts while maintaining the essentials of a event-driven synchronous language. The compositional syntax and the main primitives of our denotations are derived from [HGR88]. An operational, non-compositional, semantics for Statecharts has been given in [HPPSS87].

In [Gon88] a related denotational semantics has been given for the non-graphical synchronous language Esterel. An operational description for this language can be found in [BC85]. Related real-time models have been given for extensions of CSP. [KSR+88] contains a denotational semantics for a real-time version of CSP, based on the linear history semantics of [FLP84]. Huizing extends the same model to achieve a fully abstract semantics [HGR87] for an OCCAM-like language. Reed and Roscoe [RR88] give a hierarchy of timed models for CSP, based on a complete metric space structure. A fully abstract timed failure semantics for an extended CSP language has been developed in [GB87].

References

[BC85] B. Berry and L. Cosserat. The synchronous programming language Esterel and its mathematical semantics. In *Proceedings CMU Seminar on Concurrency*, pages 389–449. LNCS 197, Springer-Verlag, 1985.

[BCH85] J.-L. Bergerand, P. Caspi, and N. Halbwachs. Outline of a real-time data flow language. In *Proceedings IEEE Real-Time Systems Symposium*, 1985.

[dB80] J. de Bakker. *Mathematical Theory of Program Correctness.* Prentice-Hall, 1980.

[FLP84] N. Francez, D. Lehman, and A. Pnueli. A linear history semantics for distributed programming. *Theoretical Computer Science*, 32, 1984.

[GB86] P. le Guernic and A. Benveniste. Real-time, synchronous, data-flow programming: The language signal and its mathematical semantics. Technical Report 620, INRIA, Rennes, 1986.

[GB87] R. Gerth and A. Boucher. A timed failures model for extending communicating processes. In *Proceedings in the 14th International Colloquium on Automata, Languages and Programming*, pages 95–114. LNCS 267, Springer-Verlag, 1987.

[Gon88] G. Gonthier. *Sémantiques et modèles d'exécution des langages réactifs synchrones; Application à ESTEREL.* PhD thesis, University of Orsay, 1988.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[Har88] D. Harel. On visual formalisms. *Communications of the ACM*, 31:514 – 530, 1988.

[HGR87] C. Huizing, R. Gerth, and W.P. de Roever. Full abstraction of a real-time denotational semantics for an OCCAM-like language. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 223–237, 1987.

[HGR88] C. Huizing, R. Gerth, and W.P. de Roever. Modelling statecharts behaviour in a fully abstract way. In *Proceedings of the 13th Colloquium on Trees in Algebra and Programming*, pages 271–294. LNCS 299, Springer-Verlag, 1988.

[HPPSS87] D. Harel, A. Pnueli, J. Pruzan-Schmidt, and R. Sherman. On the formal semantics of Statecharts. In *Proceedings Symposium on Logic in Computer Science*, pages 54–64, 1987.

[JM89] F. Jahanian and A. Mok. Modechart, a specification language for real-time systems. *IEEE Transactions on Software Engineering*, to appear, 1989.

[KSR+88] R. Koymans, R.K. Shyamasundar, W.P. de Roever, R. Gerth, and S. Arun-Kumar. Compositional semantics for real-time distributed computing. *Information and Computation*, 79(3):210–256, 1988.

[PS88] A. Pnueli and M. Shalev. What is in a step. Draft, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, 1988.

[RR88] G. Reed and A. Roscoe. Metric spaces as models for real-time concurrency. In *Proceedings of the 3th Workshop on the Mathematical Foundations of Programming Languages Semantics 87*, 1988.