A compositional semantics for statecharts

A compositinal semantics for statecharts

by

G. Huizing
R. Gerth
W.P. de Roever

87/15

december 1987

A Compositional Semantics for Statecharts

C. *Huizing*[*]
R. Gerth
W.P. de Roever

first version May 30, 1987
second version July 4, 1987
third version August 26, 1987

Department of Mathematics & Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

This research was carried out in the context of ESPRIT-project 937, DESCARTES (Development and Specification of Ada Real-Time Embedded Systems).

[*] Electronic mail address: mcvax!eutrc3!wsinkees.UUCP
or wsdckeesh@heithe5.BITNET

Eindhoven University of Technology
Dept. of Mathematics and Computing Science
P.O. Box 513
5600 MB Eindhoven
The Netherlands
All rights reserved
editor: F.A.J. van Neerven

Abstract

We present a denotational, strictly syntax-directed, semantics for Statecharts, a graphical, mixed specification/programming language for real-time, developed by Harel [H].
This requires first of all defining a proper syntax for the graphical language. Apart from more conventional syntactical operators and their semantic counterparts, we encounter unconventional ones, dealing with the typical graphical structure of the language. The synchronous nature of Statecharts makes special demands on the semantics, esp. with respect to the causal relation between simultaneous events, and requires a refinement of our techniques for obtaining a denotational semantics for OCCAM [HGR]. The model presented will serve as a basis for a further study of specification and proof systems within the ESPRIT-project DESCARTES.

Introduction

Statecharts belongs together with Esterel [B], LUSTRE [LUSTRE], SIGNAL [SIGNAL] and an unknown number of local industrial concoctions to the group of mixed specification/programming languages used in development of real-time embedded systems.
Some of these languages (LUSTRE, SIGNAL, Esterél) have no internal notion of time. An external signal needs to be provided as a clock an the system can use it as it likes to, hence various various clock operations can be specified. The disadvantage of this approach is, that time constraints and other specifications w.r.t. the time are not clearly visible in the specification/program. Statecharts adopts the view that these specifications should be visible and hence has an internal notion of time.
Statecharts adopts, like Esterel, the synchrony hypothesis as formulated by Berry [B]. This means that output occurs simultaneously with the input that caused it. If applied without care, this hypothesis can lead to causal paradoxes, such as events disabling their own cause. In Esterel, these paradoxes are circumvented by *syntacticly* forbidding situations in which they can arise. In Statecharts, they are *semantically* impossible, because there the influence of an event is restricted to events that didn't cause it. The semantics of Esterel and Statecharts coincide in the situations that are allowed by Esterel. This restricted influence between events in Statecharts is modelled by applying a partial order on the events that occur simultaneously. This

order describes in which direction events influence each other.
Another problem that arises in giving a compositional semantics of Statecharts, is its graphical nature. For textual languages, defined by means of a proper syntax, it is clear what is demanded of a syntax-directed semantics. It has to be compositional (a homomorphism) with respect to the syntactic operators. For a graphical language, without a proper syntax, this is not so clear.
We succeeded in defining a syntax of Statecharts that makes use of a restricted set of natural operators and primitive objects. These objects and the intermediate results of applications of operators slightly generalise statecharts, by allowing transitions to be incomplete i.e. to have no origin states or no target states yet.
Some syntactic operators lack a clear counterpart in conventional languages. This is because in the graphical representation of Statecharts, the notion of area plays an important role, as it defines a hierarchy of states. Subareas of states are associated with alternative activities or concurrent activities. Transitions leaving a superstate influence the behaviour in all its substates (which are lower in hierarchy). This leads to a semantics in which it is possible to extend the behaviour of some subchart with the behaviour of the state that is put higher in hierarchy.
Unlike Esterel, Statecharts doesn't have a restricted kernel of operations, in terms of which all other features are defined. The designers of Statecharts adopt the view that handy operations should be provided as long as they can be built in. As a consequence, we had to

study a restricted version of Statecharts. The next version of this paper will include the use of variables.

2. Informal introduction to Statecharts

We give a short description of the language Statecharts and an intuitive semantics. For a more basic treatment of this, one is referred to [H] and [HPSS].

Statecharts is a formalism designed to describe the behaviour of *reactive systems* [HP]. A reactive system is a mainly event-driven system, continuously reacting to external and internal stimuli. In contrast to *transformational systems*, that perform transformations on inputs thus producing outputs, reactive systems engage in continuous interactions, *dialogues* so to say, with their environment.

Statecharts generalize Finite State Machines (FSM's), or rather Mealy machines [HU], and arise out of a conscious attempt to free FSM's from two serious limitations: the absence of a notion of hierarchy or modularity and the ability to model concurrent behaviour in a concise way. The external and internal stimuli are called *events* and they cause transitions from one state to the other. We introduce the basic concepts now.

States

In contrast to FSM's, states can be structured as a tree. We call the descendants in such a tree *substates*. A state can be of two types: AND or OR. Being in an OR-state implies being in one of its immediate substates, being in an AND-state implies being in all of its immediate substates at the same time. The latter construction describes concurrency.

Example 1 (see overleaf)

In this picture, S is an OR-state with substates *A* and *B*. Being in state *S* implies being in *A* or *B*, but not in both. *A*,*B* and *T* have no substates, *a* and *b* stand for events that trigger transitions and c is a condition. These events are called *primitive events*, because they have no further structure. They can be generated outside the system, but also by the system itself. E.g. the transition from A to B is triggered when event *a* occurs and condition c is true.

When the system is in *A* and event *a* happens, *A* will go to state *B*, but will stay in S. Whenever it is in *A* or *B* and *b* happens, it will go to *T*. The transition to A is a *default* transition. When the system is in *T* and *b* happens, it will go to S and hence to A.

Example 2 (see overleaf)

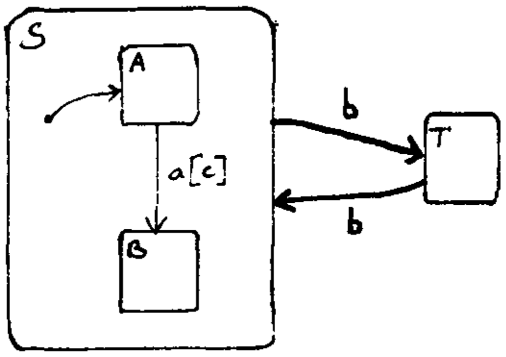
Now, S is an AND-state with immediate substates *A* and *B*. *A* and *B* are OR-states with substates A_1 and A_2 resp. B_1 and B_2. Being in *S* implies being in *A* and *B* simultaneously. when the system is in A_1 and B_2 (and hence also in A, *B* and S) and *b* happens it will go to B_1 and stay in A_1. Now, if *a* happens, it will go *simultaneously* to A_2 and B_2. Notice also the condition *in* (B_1) on the transition from A_2 to A_1. This transition can only be taken if and when the system is in A_2 and B_1 and event *d* occurs.

Transitions

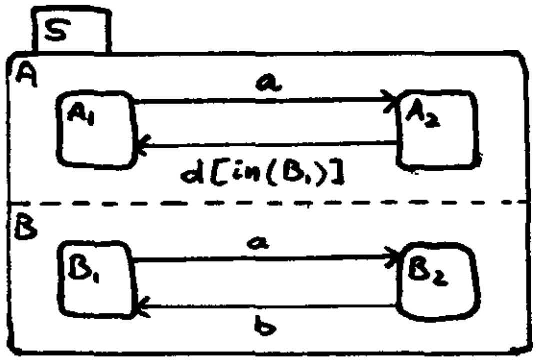
In the examples above we used simple transitions from one state to another like in FSM's.

They can be more complicated, however, going from a set of states to a set of states.

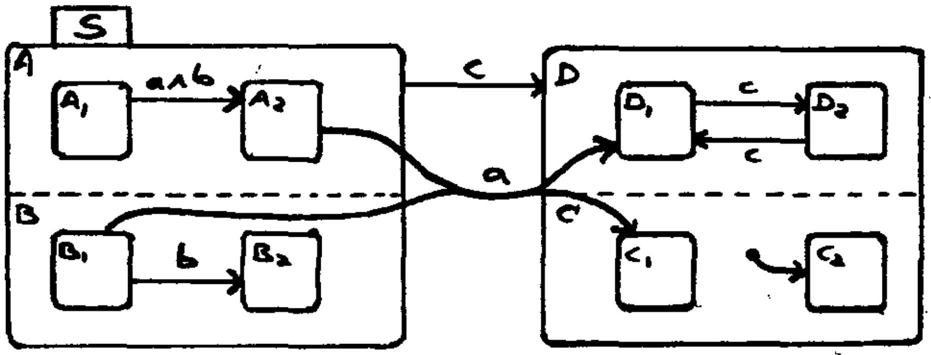
Example 1



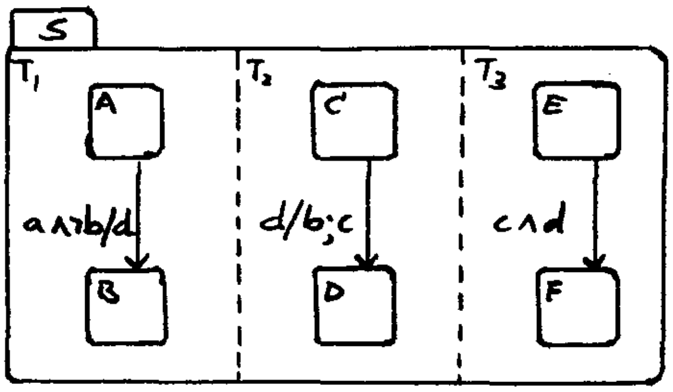
Example 2



Example 3



Example 4



Example 3 (see overleaf)

When the system is in A_2 and B_1 and a happens, it will go to T, and in particular to G and D_1. This is the general case. In this version of the paper, however, we don't allow transitions leaving more than one state. Notice the *compound event* on the transition from A_1 to A_2. Only when a and b occur simultaneously this transition will be triggered.

Actions

In the label of a transition one can specify some events that are generated when the transition is performed. This is called the *action* of a transition. These events take immediately effect and can trigger other transitions.

Example 4 (see overleaf)

When the system is in A, C and E and a occurs, a chain reaction of transitions will be performed. The transition in T_1 will generate event d; this event will trigger the T_2-transition, which on its turn will generate b and c and thus trigger the T_3-transition.

All transitions that are triggered by such a chain reaction are considered to happen at the same time. So in this example, the next state configuration after (A,C,E) is (B,D,F). But see the paragraph on causality.

Events

In general, the event in the label of a transition has the form of a logic proposition, using conjunction, disjunction and negation. In these formulae, one can use primitive events a,b,c..., but also the structured events *enter*(S) and *exit*(S), denoting the event of entering resp. exiting state S.

Another structured event is the *time-out* event. The expression *time-out*(e,n) stands for the time-out of n time units on event e. A transition labelled with this expression will be triggered when the last occurrence of e was exactly n time units ago. One time unit stands for the time that it costs to take one transition or one chain reaction of transitions. In this version of Statecharts a specification should go with an additional specification relating time units and physical time. Events are instantaneous and transient of nature, such in contrast to the conditions, which represent a more continuous situation. E.g., the event *enter*(S) can only be sensed at the time unit when state S is entered, but the condition *in*(S) is true throughout the time that the system is in the state S, in other words between the occurrence of *enter*(S) and *exit*(S).

Causality

As already mentioned above, transitions can trigger other transitions and all these transitions occur simultaneously. Together with possibility of negation of events and conditions, this can raise causal paradoxes.

If a transition is labelled with $a \wedge \neg b$, this transition will be triggered when a occurs and b does not occur. Suppose this transition generates an event that triggers another transition which, on its turn, generates b. All transitions in this chain reaction are considered to be happening *at the same time*. So b *did* happen and the first transition could not occur, hence the whole chain reaction did not occur, hence...
These kind of paradoxes are avoided by giving the following operational interpretation to chain reactions.

Every time step is subdivided into *micro-steps*, each of wich correspond to the execution of one transition. The events that are generated by a transition can only influence transitions in the following micro-steps. So in the example above, the T_1-transition takes place in the first micro-step, triggering the T_2-transition in the second micro-step. This one generates the events *b* and *c*, but these cannot prevent the T_1-transition anymore, because the latter has taken place in a previous micro-step.

We stress that the micro-steps have nothing to do with time. Their sequential occurrence is only related to the way they can influence each other - no order in time is implied. Maximal sequences of micro-steps are called *macro-steps*; a macro-step corresponds to one step in time. Here, maximal means that the sequence cannot be extended without additional input from the environment. Hence, in example 4 above, the sequence consisting only of the T_1-transition is not maximal, because the T_2-transition is still possible.

3 Syntax

In this chapter we give a non-graphical syntax of statecharts. According to this syntax any statechart is built up from primitive objects and some operators. These operators have a natural relationship with the pictures. The intermediate objects to which the operators are applied are the so-called Unvollendetes. These are incomplete statecharts with transitions without source state(s) or target state(s). Two operators, *concatenation* and *connection* can tie these dangling arrows together, thus creating complete transitions.
Concatenation makes a complete transition *between* two Unvollendetes, which can semantically be compared to sequential composition. *Connection* makes a complete transition *within* one subchart, thus possibly creating loops.
In Statecharts, there are two types of states: the AND-type and the OR-type. Being in an AND-state means being in all of its immediate substates together, being in an OR-state means being in exactly one of its sub-states. *Statification* is the operator that builds such hierarchical structure in statecharts. It puts a subchart inside a primitive state, i.e. a state without substates, thus creating a structured AND- or OR-state. Semantically, it means executing the sub-chart inside, with the possibility of interrupting this execution when one of the (incomplete) transitions leaving the superstate are triggered.
The Unvollendete that Statification puts inside a state is built by the operator *Anding*, if the surrounding state is an AND-state, or by the operator *Orring*, for an OR-state. Anding corresponds to parallel

composition in conventional programming languages. Orring can be compared to non-deterministic choice.
Finally, *Closure* gives the events that are considered internal for the particular subchart, which means that they can only be generated by that statechart. *Hiding* makes the events that are generated inside a statechart or Unvollendete invisible to the outside world. Neither operator has a graphical counterpart in the language as defined in [HPSS].
In the Appendix we give the formal relationship between the objects generated by the syntax and the formal objects representing statecharts as defined in [HPSS].

3.1 Transition Labels

Before we give the definition of Statecharts itself, we need the definition of the labels that can be associated to transitions. Let a set of *elementary events* E_e and a set of states Σ be given.
Define the set of *primitive events* $E_p = E_e \cup \{enter(S), exit(S) \mid S \in \Sigma\}$

Definition

The set of events E is recursively defined by

$\lambda \in E$, the *null* event;

$e \in E_p \rightarrow e \in E$;

$e_1, e_2 \in E_p \rightarrow e_1 \wedge e_2,\ e_1 \vee e_2 \in E$;

$e \in E \rightarrow \neg e \in E$;

$n \in \mathbb{N}\setminus\{0\},\ e \in E, \rightarrow time\text{-}out(e,n) \in E$ □

Remarks: $\neg e$ is here considered as an *event*, in contrast to [S] where it is a condition. Semantically they are the same, i.e. we also have the "not yet" interpretation.

We abbreviate *enter*(S), *exit*(S) and *time-out*(e.n) by resp. *en*(S), *ex*(S) and *tm*(e,n)

tm(e,n) means: time-out of e after n seconds.

Definition

The set of conditions C is recursively defined by

true, *false* $\in$ C;

$c_1, c_2 \in C \rightarrow c_1 \wedge c_2$, $c_1 \vee c_2 \in C$;

$c \in C \rightarrow \neg c \in C$;

$s \in \Sigma \rightarrow$ *in*(S) $\in$ C □

Definition

The set of actions A is recursively defined by:

$\mu \in A$, the *null* action,

$e \in E_p \rightarrow e \in A$,

$a_i \in A$ for $i = 1, \ldots, n \rightarrow a_1, \ldots, a_n \in A$ □

Definition

Lab = $\{e[c]/a \mid e \in E,\ c \in C\ a \in A\}$.

If $e = \lambda$, $c =$ *true*, $a = \mu$, we often omit that part of the label. □

3.2 Unvollendetes

In order to explain the syntax we introduce the notion of *incomplete statechart* or *Unvollendete*, abbreviated as *Unv*. This is a statechart in the process of being built up. It differs from a complete statechart in that it need not have a unique root-state and that it may have so-called incomplete transitions. Incomplete transitions are transitions either without source or without target state(s). These transitions are pictured as dangling arrows. Any statechart can be broken up into Unvollendetes and in chapter 4 we will give the semantics of these

Unvollendetes. Syntactically, an Unvollendete is anything that can be derived from a non-terminal.

Non-terminals

The non-terminals of our syntax are not plain symbols, but they have a structure of there own. They have the form

$\langle I,O \rangle$,

where I is a set of incoming transitions (incomplete transitions without source states) and O is a set of outgoing transitions (incomplete transitions without target states). Every derivation rule in the syntax must be considered as a scheme of rules, one for each appropriate choice of these sets.

Terminal symbols

The terminal symbols are the operators, as usual, and the so-called primitive statecharts. These are Unv's without any complete transition and consisting of only one state. They are denoted by

$[I,O,S]$,

where I and O are as in the non-terminals and S is the name of a state.

Definition

Let T_I be the set of all incoming transitions ranged over by $i,\ldots$; let T_O be the set of all outgoing transitions, ranged over by $o,\ldots$;

$T_I \cap T_O = \emptyset$

Let $E \subset E_e \cup \Sigma$, $I,\ldots \subset T_I$ and $O,\ldots \subset T_O$ and $L\colon\ T_O \rightarrow \mathbf{Lab}$

Then the set of Unvollendetes is defined by

$\mathbf{Unv} = \{U \mid \exists\ I \subset T_I,\ O \subset T_O\colon\ \langle I,O \rangle \overset{*}{\rightarrow} U\ \}$

and the set of Statecharts by

$\mathbf{Stch} = \{V \mid\ B \overset{*}{\rightarrow} V\ \}$

and $\overset{*}{\rightarrow}$ is the derivability relation for the following set of rules:

$B \rightarrow Stat([I_1,O_1,A]\ ,\ \langle\{t\}\ ,\ \emptyset\ \rangle\ ,\ t\ )$

$\langle(I_1 \cup I_2)\backslash\{t_2\}\ ,\ (O_1 \cup O_2)\backslash\{t_1\}\rangle \rightarrow Conc(\langle I_1,O_1\rangle, t_1, t_2, \langle I_2,O_2\rangle)$

with $t_1 \epsilon O_1$ and $t_2 \epsilon I_2$

$\langle I\backslash\{t_2\}\ ,\ O\backslash\{t_1\}\rangle \rightarrow Conn(\langle I,O\rangle, t_1, t_2)$

with $t_1 \epsilon O$ and $t_2 \epsilon I$

$\langle(I_1 \cup I_2)\backslash\{t\}\ ,\ O_1 \cup O_2\rangle \rightarrow Stat([I_1,O_1,A]\ ,\ \langle I_2,O_2\rangle\ ,\ t)$

with $t \epsilon I_2$

$\langle(I_1 \cup I_2)\backslash\{u_1,\ldots u_2\}\ ,\ O_1 \cup O_2\rangle \rightarrow$

$And(\langle I_1,O_1\rangle, \langle I_2,O_2\rangle, (t_1,u_1), \ldots, (t_n,u_n\}))$

$\langle I_1 \cup I_2\ ,\ O_1 \cup O_2\rangle \rightarrow Or(\langle I_1,O_1\rangle, t_2, \langle I_2,O_2\rangle)$

$\langle I\ ,\ O\rangle \rightarrow Close(\langle I,O\rangle\ ,\ E)$

$\langle I\ ,\ O\rangle \rightarrow Hide(\langle I,O\rangle\ ,\ E)$ □

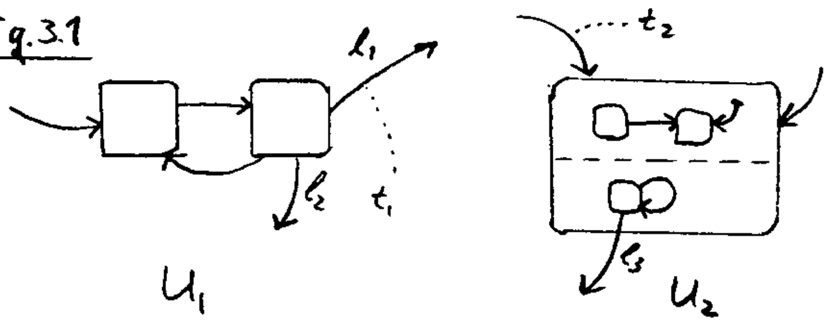
Explanation of the operators

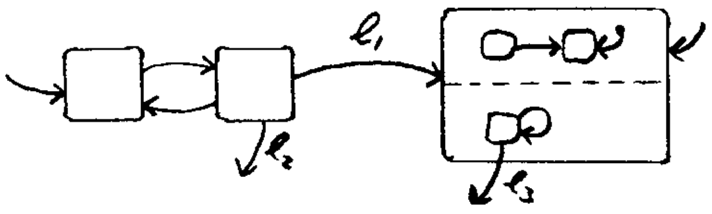
Concatenation ($Conc(U_1, t_1, t_2, U_2)$)

By concatenation, two Unvollendetes are "sequentially composed". An outgoing transition of U_1 (t_1) is connected to an incoming one of the U_2 (t_2), thus creating a complete transition. (See fig. 3.1, overleaf)

Connection

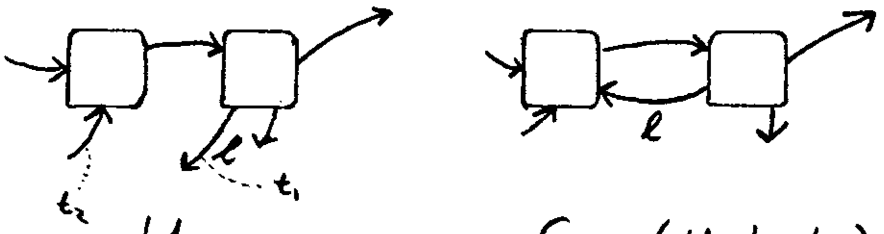
Connection only differs from concatenation by taking only one chart and making the new transition somewhere inside. In fact we don't need concatenation if we have connection and orring (see below), but from the semantic point of view, concatenation is more basic. (See fig. 3.2, overleaf) fig.3.1





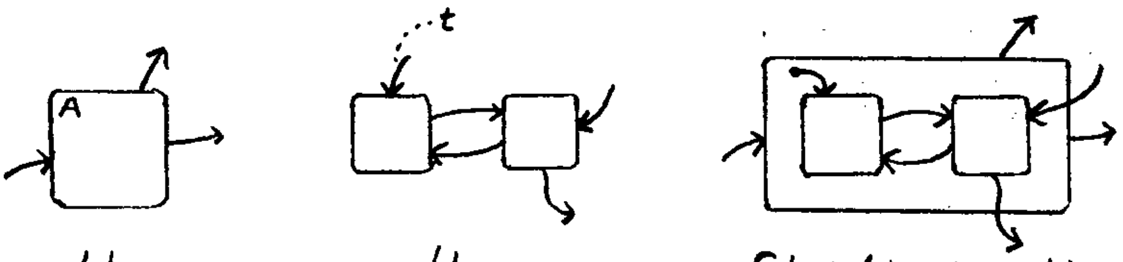
Conc (U_1, t_1, t_2, U_2)

fig.3.2



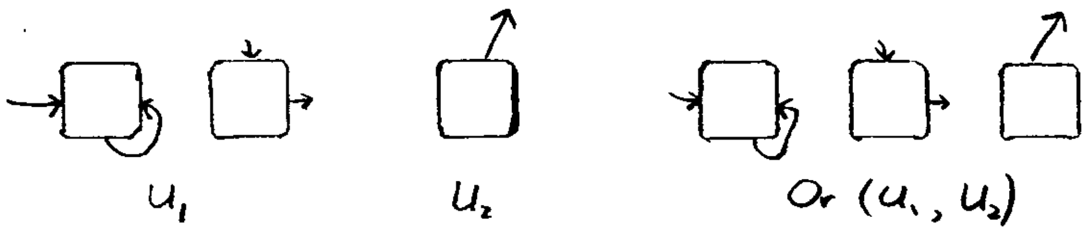
Conn (U, t_1, t_2)

fig.3.3



Stat (U_1, U_2, t)

fig.3.5



Or (U_1, U_2)

<u>Statification</u> Stat(U_1,U_2,t)

This is the hierarchy operator; it has no counterpart in conventional programming languages. It puts an Unvollendete (U_2) inside a state A (the state of a primitive U_1). An explicitly mentioned transition from U_2 (t) becomes the default of A. (See fig. 3.3, overleaf)

<u>Anding</u>

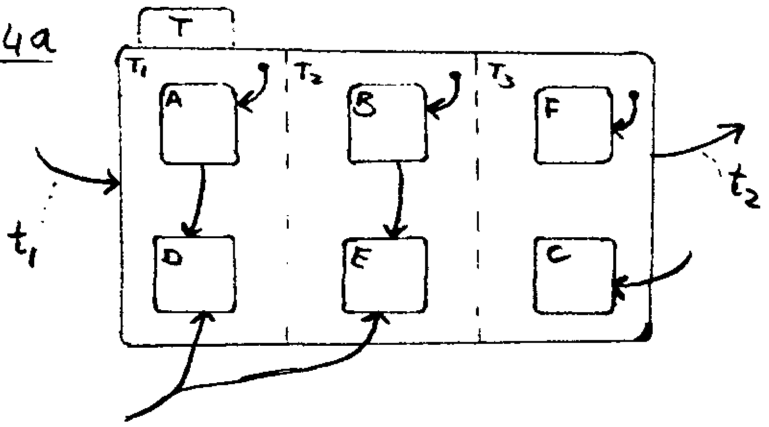
Anding in Statecharts is the parallel composition in conventional programming languages. Two Unvollendetes are put in parallel. At a later stage, they will become orthogonal components of an AND-state. Anding is a binary operator, so if there are to be more than two orthogonal components, it must be applied repeatedly. The semantics counterpart of Anding is associative. Our syntax is more liberal than that of [H'], since it does not prescribe that an orthogonal component must have a unique root state. In the first picture you see a derivation of an AND-state with this restriction of [HPPS] and in the second picture you see a derivation of an AND-state that does not satisfy this restriction. (See fig 3.4, overleaf)

<u>Orring</u>

This is the counterpart of Anding, it puts some subcharts together in non-orthogonal composition, with the intention of statification by an OR-state. It can be compared to non-deterministic choice. (See fig. 3.5, overleaf)

<u>Closure</u>

In [HPSS], the set of primitive events is divided into internal and external events. External events can be generated outside the statechart fig.3.4a



$\mathrm{Stat}([\{t_1\}, \{t_2\}, T], U, *)$

$U = \mathrm{And}(\mathrm{And}(U_1, U_2, \{(t_3, t_4)\}), U_3, \emptyset)$

$U_1 = \mathrm{Stat}([\emptyset, \emptyset, T_1], U_1', t_5)$
$U_2 = \mathrm{Stat}([\emptyset, \emptyset, T_2], U_2', t_6)$
$U_3 = \mathrm{Stat}([\emptyset, \emptyset, T_3], U_3', t_7)$

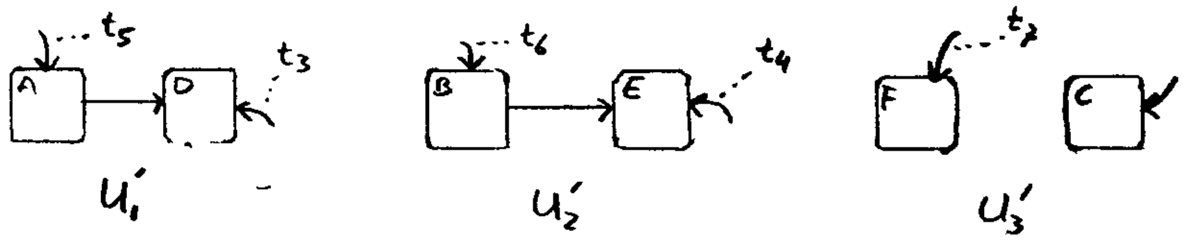
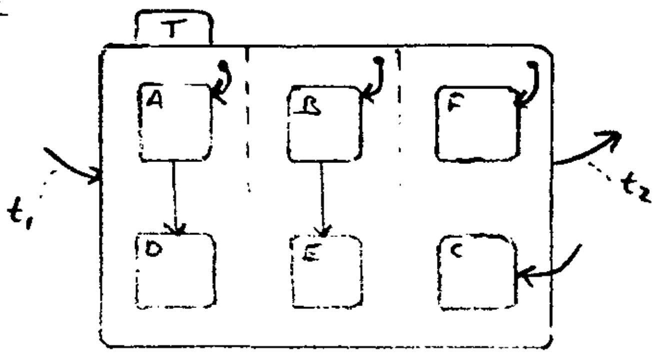


fig.3.4b



$\mathrm{Stat}([\{t_1\}, \{t_2\}, T], V, *)$
$V = \mathrm{And}(\mathrm{And}(U_1', U_2', \{(t_3, t_4)\}), U_3', \emptyset)$.

itself, internal events cannot. For a compositional semantics this distinction is not useful, because events that are internal to the complete statechart, can be external to some subchart.
Therefore, we introduce an operator that declares some events internal to a subchart. This is not hiding and these events are still observable.

<u>Hiding</u>

The hiding operator makes the specified events invisible for the outside world.

4 Semantics

This chapter presents a denotational semantics of statecharts or rather of Unv's. This semantics is compositional (syntax-directed) with regard to the operators defined in chapter 3.
The maximality of the sequences of micro-steps in chapter 2 corresponds to the notion of maximal parallelism as modelled in [HGR,GB] (see also [SM]). The techniques of those papers also apply here.
As Statecharts describes a set of state configurations (as any digital system), a discrete model of time is adequate. Since it is intended to make global time specifications, we use a global notion of time. The simplest domain that gives us these properties is $\mathbb{N}$, but for reasons that will be explained later, we use $\mathbb{Z}$.

4.1 Domain and semantic functions

At first sight, Statecharts are quite different from ordinary programming languages. Simplest to characterise are sequential languages without jump-like constructs. Once jumps enter the picture we have to abandon the idea of giving state transformations for each command in isolation. Traditionally, this is solved using the idea of continuations [SW,M].
It is our aim to give a *compositional* semantics of Statecharts. The semantics of [SW] is only given for full program blocks in which all labels of gotos appear. In our solution jumps (transitions) are made in two stages. In the first stage we have only half jumps, in which the place where we are jumping to or where we come jumping from is not

specified. These are the incomplete transitions in the syntax.

In the semantics, we record the behaviour of a subchart only between such jumps. And we specify for each history the incomplete transition by which it starts and by which it ends. This specification is just the syntactic identification of the transition.

In the second stage, by concatenation or connection these half jumps are made into full jumps by identifying an incoming and an outgoing transition. Now we can also give the full semantics of the jump, as we know where we come from and where we go to. This semantics is just the concatenation of the history that ends in one half of it and the history that starts with the other half. In case of connection, loops can arise, since we jump to the same subchart. Consequently, the semantics of this construct will be characterised by a fixed-point equation.

Now there is a difference between gotos in conventional languages and transitions in Statecharts, namely in Statecharts the place where a jump can occur is not completely syntacticly determined. Transitions from a superstate can be triggered when execution is anywhere inside that state. Our solution is giving two options at any moment during execution inside a state: exiting by the outside transition or continuing the history generated by the semantics of the interior of the state.

<u>The semantic domain</u>.

The semantics of a (incomplete) statechart, i.e., its denotation, will be a set of <u>histories</u>, each history corresponding to one possible execution.

The set of histories, $\mathbb{H}$, is defined by

$\mathbb{H} = T \cup \{*\} \times (\mathbb{Z} \to \mathbb{C}) \times T \cup \{\perp\}$, where T is the set of edge-identifiers (transitions) and $\mathbb{Z} \to \mathbb{C}$ denotes the set of *partial* functions with indicated domain and codomain.

A history consists of three components. The first component is the incoming transition of the chart by which the execution starts, the third component either equals the outgoing transition by which the execution ends, or equals "$\perp$" in case of an incomplete computation. It is possible that there is no starting transition, indicated by "$*$". This is the case when we have the root state of the complete statechart, or a component of an AND-state that can be started implicitly by an incoming transition of another component.

The second component of the history is a *partial* function that associates to each time unit, a so called clock record.

Execution starts at time unit 0 and ends at the last time unit where the function is defined. The records associated to negative time values contain information about the past, i.e. before the execution of this subchart started. We will need this to describe the occurrence of time-out events

Notation:

- if $f \in \mathbb{Z} \to \mathbb{C}$ then $|f| = \max(\{i \mid f(i) \text{ is defined}\} \cup \{-1\}) + 1$

  $|f|-1$ is the time on which the outgoing transition, if there is one, of this execution occurs.
- $\lambda \in \mathbb{Z} \to \mathbb{C}$ is the function that is nowhere defined; we defined $|f|$ in such a way, that $|\lambda| = 0$.
- the *shift* operator changes the time in a history:

  $shift(f,j)(i+j) = f(i)$

  $|shift(f,j)| = |f|+j$

In order to use fixed-point definitions, we impose the structure of a complete partial order (cpo) on our domain. We use a standard technique as explained in [K&] by defining the Hoare order on prefix-closed sets. We distinguish *extendable* and *finished* computations. Extendable histories correspond to incomplete computations and are charaterised by a bottom outgoing transition ($\perp$). We define the following partial order on histories:

Definition:

$(t_1,f,t_2) \leq (t_1',f',t_2')$ iff

$t_1=t_1' \wedge (t_2=\perp \vee t_2=t_2') \wedge |f|\leq|f'| \wedge \forall i<|f|: f(i) = f'(i)$ □

If $h_1 \leq h_2$ we say that h_1 is a *prefix* of h_2.

Definition:

a set of histories H is prefix-closed iff $\forall h \in H$: $h' \leq h \rightarrow h' \in H$ □

So we define our semantical domain:

Definition:

$\mathbb{D} = \{H \subseteq \mathbb{H} \mid H \text{ is prefix-closed}\}$

$\perp_{\mathbb{D}} = \emptyset$ □

Theorem:

$(\mathbb{D},\subseteq,\perp_{\mathbb{D}})$ is a cpo.

Proof

Standard. □

We define a function that turns a set of histories into the smallest prefix-clodsed set that encloses it:

Definition:

If H is a set of histories, then

$H^{CL} = \bigcap\{H' \mid H \subseteq H', H' \text{is prefix-closed}\}$ □

Before we describe the structure of $\mathbb{C}$, we explain the elementary semantic records.

1. $\mathbb{R} = \{a! \mid a \in E_p\}$

   a! records the fact that event a did happen at a particular time unit.

2. $\mathbb{F} = \{a, \bar{a} \mid a \in E_p\}$

   a and $\bar{a}$ are *claims* that event a did resp did not happen at a particular time. They occur in the semantics of a component that can be influenced *from outside* by the event a. a means: the occurrence of event a is *necessary* for the described behaviour, $\bar{a}$ means: the occurrence of event a is *prohibitive* for the described behaviour.

Now we can define the set of clock records, $\mathbb{C}$:

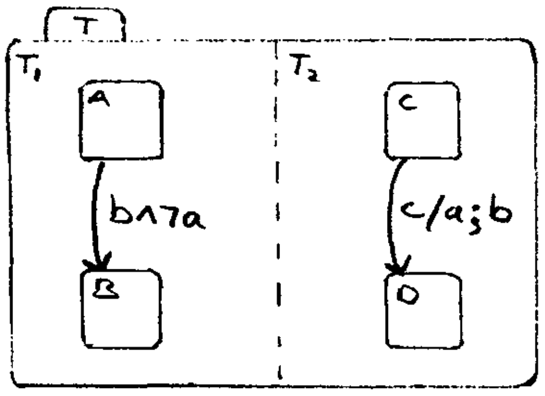
$\mathbb{C} = 2^{\mathbb{F} \cup \mathbb{R}} \times \mathbb{P} \times 2^{\mathbb{F}}$,

where 2^{A} denotes the class of subsets of A.

The first component of a clock record is a set of records and claims that are associated to the transitions that were taken at this time unit. The records give the events that are generated by these transitions and the claims give the events that are necessary resp. prohibitive for these transitions to happen. We call this component the *transition record*.

Unfortunately this information is not sufficient. A transition can influence other transitions of the same time step - by triggering them or by preventing them from being triggered. This influence, however, is restricted. A transition can only influence the transitions that occurred in "later" micro-steps. This is the way causal paradoxes are avoided.

fig 4.1



We have to record this restricted influence too. This leads to the following additional information.

A partial order on the sets of records that are generated by the transitions representing the way they can influence each other. E.g if t_1 causes t_2, then we have $t_1 < t_2$. This means that t_2 can never influence transitions t_3 with $t_3 < t_1$. These relationships can also arise from negative causes: if t_1 prevents t_2, then we also have $t_2 < t_1$, because that is the only way they can occur in the same time step.

Example (see fig. 4.1)

If t_1 and t_2 occur simultaneously, we have $t_1 < t_2$. This means that t_2 can not trigger t_1 even though it generates b. The trigger of t_1 has to come from somewhere else.

This information is represented by a labelled partial order. Each node represents a transition and is labelled with the corresponding sets of events and claims.

Definition

A labelled partial order (lpo) on S is a triple

$(V, <, \ell)$, where

V is a set of vertices

$<$ is an irreflexive partial order on V

$\ell: V \to 2^S$ is a labelling function.

Notation

$\Lambda = (\emptyset, \emptyset, \emptyset)$ is the empty order

$1(S) = (\{v\}, \emptyset, \ell)$ where $\ell(v) = S$; this is the trivial one-node order on S.

In the sequel we assume that the node sets of two lpo's are always disjoint.

So the second component of a clock record is a labelled partial order on the transition records

$$\mathbb{P} = \{(V,<,\ell) \mid \ell\colon V \to 2^{\mathbb{FUR}}\}$$

The third component, called the global record, contains the claims that are not associated to a particular micro step but to the complete macro-step. They are not associated to an *action* performed at the present time step and hence they are not associated to the influence relation of the transition record. They can arise from:

1 The maximality constraint: the sequence of micro-steps that is performed as a macro-step must be maximal in the sense that no additional transitions are possible. These claims give the conditions on the environment that indeed no additional transitions are possible.

2 Time-out events of future transitions: performing a transition with a time-out event in its label lays some claims on the macro-steps in the past. The event must have taken place a specified number of time units ago and may not have taken place since.

3 Conditions of the form *in*(S) on future transitions: this condition is only true if the state S was entered some time ago and not left since.

4.2 Semantics of transitions

Before we define the semantics of subcharts, we define a function that gives the semantics of transitions. When the system is in some state, it can do two things with respect to a transition leaving that state. It can

a) either take the transition; this means that the event expression in the label of the transition should be satisfied and that some events are generated in accordance to the action part of the label.

b) or stay in the state; this means that the event-expression of the transition should not be satisfied.

The history corresponding to a) is produced by the function $\mathcal{T}$, the history corresponding to b) by $\mathcal{W}$.

First we define a restricted version of $\mathcal{T}$ on event expressions.

This function yields a simple kind of histories that gives the conditions on the environment that cause a transition with this event-expression to be triggered. This involves conditions on the present, i.e. the time the transition takes place (time 0) and conditions on the past (time -1, -2, etc) The latter in the case of time-out expressions.

We assume that these expressions are in disjunctive normal form:

$e \equiv \bigvee_i \bigwedge_j p_{ij}$, where p_{ij} is of the form a or $\neg a$, with

$a \in E_p \cup \{\lambda, \bar{\lambda}\} \cup \{tm(e',n), \neg tm(e',n) \mid n \in \mathbb{N},\ e' \text{ in normal form}\}$

EN is the set of the normal form expressions.

Assume that the function $N : E \to EN$ brings a propositional formula into the logically equivalent normal form.

In the following definitions of sets of partial functions we assume that these functions are only defined where their values are specified.

Definition

$\mathcal{T}: EN \to 2^{\mathbb{Z} \to Rec}$ where $Rec = 2^{\mathbb{F} \cup \mathbb{R}}$, is defined recursively:

$\mathcal{T}(a) = \{f \mid f(0) = \{a\}\}$ for $a \in E_p$.

$\mathcal{T}(\neg a) = \{f \mid f(0) = \{\bar{a}\}\}$ for $a \in E_p$.

$\mathcal{T}(\lambda) = \{f \mid f(0) = \emptyset\}$

λ is the null event, the event that always occurs.

$\mathcal{T}(\neg\lambda) = \emptyset$

$\neg\lambda$ never occurs

$\mathcal{T}(e_1 \wedge e_2) = \{f_1 \dot{\cup} f_2 \mid f_i \in \mathcal{T}(e_i)\}$

e.g. $\mathcal{T}(a \wedge b)(0) = \{\{a,b\}\}$.

Here, $\dot{\cup}$ stands for the point-wise union, i.e.

$$
\begin{aligned}
(f_1 \dot{\cup} f_2)(i) &= f_1(i) \cup f_2(i) \text{ if both are defined,}\\
&= f_j(i) \text{ if only } f_j(i) \text{ is defined,}\\
&= \text{undefined otherwise.}
\end{aligned}
$$

$\mathcal{T}(e_1 \vee e_2) = \mathcal{T}(e_1) \cup \mathcal{T}(e_2)$

e.g. $\mathcal{T}(a \vee b)(0) = \{\{a\},\{b\}\}$

Thus far, the function $\mathcal{T}$ produces only claims for one time step of execution. For the time-out expression, however, some claims about the past have to be made.

$$
\begin{aligned}
\mathcal{T}(tm(e,n)) = \; & \{\mathit{shift}(f_0\hat{}\ldots\hat{}f_n,-n) \mid \forall i: |f_i| \leq 1 \wedge \\
& f_0 \in \mathcal{T}(e) \wedge \\
& \quad \forall\, 0<i<n: f_i \in \mathcal{T}(N(\neg e)) \wedge f_n \in \mathcal{T}(\lambda)\}\\
\mathcal{T}(\neg tm(e,n)) = \; & \{\mathit{shift}(f_0\hat{}\ldots\hat{}f_n) \mid \forall i: |f_i| \leq 1 \wedge \\
& [f_0 \in \mathcal{T}(N(\neg e)) \vee \exists\, 0<i<n: (f_i \in \mathcal{T}(e))] \wedge f_n \in \mathcal{T}(\lambda)\} \qquad \square
\end{aligned}
$$

Here the $f_1\hat{}f_2$ denotes concatenation: the present of f_2 starts where f_1 ends and the pastime of f_2 is combined with f_1

o $\quad f_1\hat{}f_2 = f_1 \dot{\cup} \mathit{shift}(f_2,|f_1|)$.

A time-out expression tm(e,n) is satisfied if the last occurrence of e was exactly n time steps ago. This is expressed by $f(-n) = \mathcal{T}(e)(0)$. (e occurred n steps ago) and by $f(i) = \mathcal{T}(N(\neg e))(0)$. (e didn't occur later, i.e. the occurrence at $-n$ was the last occurrence). We have decided that it doesn't matter whether e occurs at the time of the time-out, hence no claims about the present are made ($f(0) = \emptyset$).

The semantics of conditions is defined as follows:

$\mathcal{C}\colon C \to 2^{\mathbb{Z}\to Rec}$

$\mathcal{C}(true) = \{f \mid f(0) = \emptyset\}$

$\mathcal{C}(false) = \emptyset$

$\mathcal{C}(c_1 \wedge c_2) = \{f_1 \cup f_2 \mid f_i \in \mathcal{C}(c_i)\}$

$\mathcal{C}(c_1 \vee c_2) = \mathcal{C}(c_1) \cup \mathcal{C}(c_2)$

$\mathcal{C}(in(S)) = \{f \mid \exists\, n \leq 0\colon f(n) = \{en(S)\} \wedge \forall\, n < i \leq 0\colon f(i) = \{\overline{ex}(S)\}\ \}$

$\mathcal{C}(\neg in(S) = \{f \mid \forall\, n \leq 0\colon f(n) = \{\overline{en}(S)\} \vee$

$[\exists n \leq 0\colon f(n) = \{ex(S)\} \wedge \forall\, n < i \leq 0\colon f(i) = \{\overline{en}(S)\}]\ \}$

In other words, the system is in the state S if it entered S some time ago and has never left it afterwards; the system is not in the state S if it never entered S or has left it some time ago and never entered it afterwards.

The semantics of actions is as follows:

$\mathcal{A}\colon A \to 2^{\mathbb{R}}$

$\mathcal{A}(a) = \{a!\}$ for $a \in E_p$

$\mathcal{A}(a_1;a_2) = \mathcal{A}(a_1) \cup \mathcal{A}(a_2)$ for $a_{1,2} \in A$

$\mathcal{A}(\mu) = \emptyset.$ □

Now we can extend the domain of $\mathcal{T}$ to the set of complete labels, **Lab**, and we extend the codomain to sets of functions in $\mathbb{Z} \to \mathbb{C}$.

$\mathcal{T}$: Lab $\rightarrow 2^{\mathbb{Z} \rightarrow \mathbb{C}}$

$\mathcal{T}(e[c]/a) = \quad \{f \mid \exists\, f_1 \in \mathcal{T}(N(e)),\ f_2 \in \mathcal{C}(N(c)):$

$\qquad f(i) = (\emptyset, \Lambda, f_1(i) \cup f_2(i))$ for $i<0$,

$\qquad f(0) = (f_1(0) \cup f_2(0) \cup \mathcal{A}(a), 1, \emptyset)\}$

$\mathcal{W}$ gives the conditions on the environment that prevents the transition from being triggered:

$\mathcal{W}$: E x C $\rightarrow 2^{\mathbb{Z} \rightarrow \mathbb{C}}$

$\mathcal{W}(e[c]) = \{f \mid \exists\, f_1 \in \mathcal{T}(N(\neg e)),\ f_2 \in \mathcal{C}(N(\neg c)):\ f(i) = (\emptyset, \Lambda, f_1(i) \cup f_2(i)),$

$\qquad$ for all $i\}$

4.3 Semantics of Unvollendetes

A basic semantic notion is the merge of two clock records. Whenever the histories of two charts are combined (*Anding, Statification, Concatenation*), for each time unit the associated clock records should be merged. This means unifying the transition records and the global records. Unifying the partial orders, however, is not enough.
New relationships should be added between transitions that can prevent one another. Hence the merge of two clock records is defined as follows.

Definition:

Let $f_j(i) = (t_j, p_j, w_j)$ and $p_j = (V_j, <_j, \ell_j)$ for $j = 1,2$.

Then we define

$f_1(i) \| f_2(i) =$

$\quad \{(t,p,w) \mid t = t_1 \cup t_2,\ w = w_1 \cup w_2$

$\qquad \forall\, \bar{a} \in w:\ a \notin t \land a! \notin t,$

$\qquad < \;=\; <_1 \cup <_2 \cup <'$ is a p.o., where $<'$ is defined by

$\qquad\quad \forall\, v_1, v_2 \in V:\ \bar{a} \in v_1 \land (a \in v_2 \lor a! \in v_2) \rightarrow v_1 <' v_2,$

$\qquad p = (V_1 \cup V_2, <, \ell_1 \cup \ell_2)\ \}$

and the merge of two histories:

$$
\begin{aligned}
f_1 \| f_2 = \\
&\{f \mid |f|=|f_j| \wedge f(|f_j|-1)=f_j(|f_j|-1) \\
&\qquad \wedge \forall\, i<|f_j|-1: \\
&\qquad\qquad f(i) \in f_1(i)\|f_2(i)\} \text{ if } |f_j|<|f_{3-j}| \text{ or } |f_1|=|f_2|=\infty \\
&= \emptyset \qquad\qquad\qquad \text{if } |f_1| = |f_2| < \infty
\end{aligned}
$$ □

We define the *concatenation* of two histories as follows:

$$(f_1 \mathbin{\hat{}} f_2)(i) = f_1(i) \,\|\, f_2(i-|f_1|)$$

The existence of an irreflexive, transitive partial order with this property guarantees the consistency of the merge. E.g. the transitions labelled $\neg a/b$ and $\neg b/a$ can never be taken in the same time step.

Suppose $f_1(0) = (\{\bar{a}, b!\}, 1, \emptyset)$

and $f_2(0) = (\{\bar{b}, a!\}, 1, \emptyset)$

then $f_1 \| f_2 = \emptyset$.

Note that there is at most one minimal order with the desired properties.

We define the semantic function

$\mathfrak{D}$: Charts $\rightarrow \mathbb{D}$

by induction on the structure of Charts.

Primitives

A primitive has only one state and no complete transitions. Hence, all possible executions consist of some incoming transition, waiting in the state and some outgoing transition. Incomplete executions have no outgoing transitions (but a $\perp$ instead) and the case that the state is never left is expressed by having arbitrary long incomplete executions. The semantics of the outgoing transition is given by the function $\mathcal{T}$, the semantics of the waiting is given by $\mathcal{W}$. Since this waiting is only allowed if none of the outgoing transitions can be taken, it claims that one of the events $e_1,\ldots,e_n$ does not happen or that one of the conditions $c_1,\ldots,c_n$ is not true, where the $e_i[c_i]/a_i$ are the labels of the outgoing transitions.

No semantics is given for the incoming transition, only an identification. In a later stage, this transition will be connected to an outgoing transition of another (or the same) chart. There, this outgoing transition will have a semantics.

$$
\begin{aligned}
&\mathcal{D}([I,O,S]) = \\
&\{(u,f,v) \mid u \in I \cup \{*\} \wedge v \in O \cup \{\perp\} \wedge \\
&\qquad (\exists\, \vec{f}_i : [\ f = f_0\hat{}\ f_1\hat{}\ldots\hat{}f_n \wedge \\
&\qquad\qquad ([v\neq\perp \wedge \exists\, f' \in \mathcal{T}(L(v)) : f_n = f' | n \to f'(n) + ex(S)!\,] \\
&\qquad\qquad\quad \vee\ [v=\perp \wedge f_n \in W]) \wedge \\
&\qquad\qquad \forall\, 0<i<n : f_i \in W \wedge \\
&\qquad\qquad \exists\, f'' : f'' \in W \wedge f_0 = f'' | -1 \to f''(-1) + en(S)!\ ]\}^{CL}
\end{aligned}
$$

where $L(O) = \{e_1[c_1]/a_1,\ldots,e_n[c_n]/a_n\}$,

and $W = \mathcal{W}(e_1 \vee \ldots \vee e_n\ [c_1 \vee \ldots \vee c_n])$;

the +-operator on clock records is defined by:

$(t,p,w) + a = (t \cup \{a\}\ ,\ p\ ,\ w)$;

$f|n\to e$ is the notation for function substitution:

$$
\begin{aligned}
(f|n\to e)(m) &= e && \text{if } m=n \\
&= f(m) && \text{otherwise.}
\end{aligned}
$$

Remember that, if $O=\emptyset$, then $e = \neg\lambda$.

In this definition, we see that for each time-step in the execution a history is generated and these histories are concatenated. Note that they all have length 1, since they are generated by $\mathcal{W}$ and $\mathcal{T}$.

Although all histories are of finite length we can wait forever in this state. This is represented by an infinite chain of histories that have no outgoing transition, but $\perp$.

<u>Concatenation</u>

In the concatenation of two subcharts, new computations become possible. E.g., by entering the first chart, performing a computation that ends in the connecting transition, entering the second chart by this transition and performing a computation there. In our semantics, this corresponds to simply concatenating the histories from the first chart and those from the second chart that end resp. start with the connecting transition.

It is still possible however, to perform a computation in one of the charts in isolation, provided that it doesn't start or end with one of the connecting transitions, because these are no entering or leaving points anymore.

Hence, the semantics of the concatenation of two subcharts consists of the concatenation of their respective histories together with their own histories, from which the histories that start or end in a connecting transition are deleted.

$$\mathfrak{D}(\underline{\text{conc}}(U_1,t_1,t_2,U_2)) =$$
$$\text{delete}_{t_1t_2}(\text{conc}_{\mathfrak{D}}(\mathfrak{D}(U_1),t_1,t_2,\mathfrak{D}(U_2)))^{CL}$$

where $\text{delete}_{t_1,t_2}(D) = \{(u,f,v) \mid (u,f,v) \in D \wedge u,v \notin \{t_1,t_2\}\}$

and $\text{conc}_{\mathfrak{D}}(D_1,t_1,t_2,D_2) =$
$$\{(u,f_1\hat{\ }f_2,v) \mid (u,f_1,t_1) \in D_1 \wedge (t_2,f_2,v) \in D_2\} \cup D_1 \cup D_2.$$

Connection

Since connection creates a transition from a chart to itself, it can involve repetition.

Definition

$$\mathfrak{D}(\underline{\text{conn}}(U,t_1,t_2)) =$$
$$\text{delete}_{t_1,t_2}(\mu X.\text{conc}_{\mathfrak{D}}(\mathfrak{D}(U),t_1,t_2,X))^{CL},$$

where μ is the least fixed-point operator □

Anding

Anding two charts means executing them in parallel. As we have real-time maximal parallelism, this means merging the clock records that apply to the same time unit. The entering is either explicitly in one component and implicitly in the other one, or by a forked transition that is syntactically specified.

$$\mathfrak{D}(\underline{\text{And}}(U_1,U_2,\{(t_1,w_1),\ldots,(t_n,w_n)\})) =$$
$$\{(u,f,v) \mid \exists\, (u_i,f_i,v_i) \in \mathfrak{D}(U_i):$$
$$(u = u_i \wedge u_{3-i} = *) \vee (u = v_j' \wedge u_1 = t_j \wedge u_2 = w_j)$$
$$\wedge[(|f_1|<|f_2| \wedge v = v_1)$$
$$\vee(|f_2|<|f_1| \wedge v = v_2)]$$
$$\wedge\ f \in f_1 \parallel f_2\}^{CL}$$

Here, $v'_1, \ldots, v'_n$ are the new transitions that replace resp. $(t_1, w_1), \ldots, (t_n, w_n)$.

<u>Statification</u>

$$
\begin{aligned}
&\mathfrak{D}(\underline{\mathrm{Stat}}(U_1, U_2(d)) = \\
&\{(u, f, v) \mid \exists\ (u_i, f_i, v_i) \in \mathfrak{D}(U_i): \\
&\qquad (u = u_1 \land u_2 = d) \lor (u = u_2 \land u_2 \neq d) \\
&\qquad \land[(|f_1| < |f_2| \land v = v_1) \\
&\qquad\quad \lor(|f_2| < |f_1| \land v = v_2)] \\
&\qquad\quad \land f \in f_1 \parallel f_2\}^{\mathrm{CL}}
\end{aligned}
$$

There are three ways to start the execution of a state with inner structure.

1) take a transition explicitly to some states inside; this is represented by the case $u = u_2 \land u_2 \neq d$ in the definition above.

2) take a transition to the outer state and enter some state(s) inside by default; this is represented by the case $u = u_1 \land u_2 = d$.

3) enter the outer state implicity and enter some state(s) inside by default: this has the same representation as 2). The implicit entrance is represented by $u_1 = *$.

Executing a state with inner structure means executing the structure inside and always being prepared to stop the execution when an outgoing transition of the outer state is triggered.

This corresponds to the parallel merge $\parallel$ of the histories of the chart inside and of those of the outer state."Being prepared to stop the execution, etc" just means adding the waiting claims of the outer state to the history and these waiting claims is what the histories of the outer state are built from.

One can leave the execution of a statified chart either by a transition from the outer state ($v = v_1$) or by a transition from the inside chart ($v = v_2$), but never by taking them both.
That is why the two histories that are merged cannot have the same length, unless both are infinite and the chart is not left.
When the statified chart is left, all execution is stopped. This corresponds to the deletion of the records from the longer history that are associated to time units after the time of the leaving transition.
We also delete records that are associated to the <u>same</u> time unit as when the leaving transition takes place. These records come from transitions that should occur simultaneously with the the transition that leaves the complete subchart. It is clear that these transitions are not possible. These records also contain information about the waiting at that moment. If we should preserve this information, it would mean that it is not allowed to leave a subchart as long as internal transitions are possible. This doesn't seem to be a very reasonable semantics.

<u>Closure</u>

There are two ways of closing a statechart. One way is closing for events and one is closing for states. When a statechart is closed for a set of events, this means that these events can now be discarded because claims on the occurrence of events can now be justified. In the semantics this means that we check for each time record in each history if there exists a legal influence ordening between the transitions that gives each internal event a cause.
Closing a statechart for a set of state(names) is obtained by closing it

for the set events of the form *en*(S) and *ex*(S) for every state S in the set of states.

Let $f(i) = (t,p,w)$ and $p = (V,<,\mathcal{L})$

Then we define

$$
\begin{aligned}
Cl(f(i),E) = \{(t',p',w') \mid\ & \forall a \in w\cap E\colon a! \in t \\
& \wedge \exists <'\colon <' \text{ is a minimal p.o. on } V \text{ s.t.} \\
& \qquad < \subseteq <' \wedge \\
& \qquad \forall v_1 \in V\colon a \in v_1 \rightarrow \\
& \qquad\qquad \exists v_2 \in V\colon a! \in v_2 \wedge v_2 <' v_1' \\
& \qquad \wedge p' = (V,<',\mathcal{L}\restriction t') \\
& \wedge t' = t\backslash\{a,\bar{a} \mid a\in E\} \wedge w' = w\backslash\{a,\bar{a} \mid a\in E\} \\
& \wedge f(i) = (t,p,w) \ \}
\end{aligned}
$$

and

$Cl(f,E) = \{f' \mid \forall i\colon f'(i) \in Cl(f(i),E)\}$.

Here, $\mathcal{L}\restriction t'$ stands for $\mathcal{L}$ with a restricted codomain:

$(\mathcal{L}\restriction t')(v) = \mathcal{L}(v) \cap t'$.

Then

$\mathfrak{D}(\underline{\text{Close}}(U,E)) =$

$\{(u,f,v) \mid \exists\ (u,f',v) \in \mathfrak{D}(U) \wedge f\in Cl(f',E')\}$.

where $E' = (E \cap E_p) \cup \{en(S),ex(S) \mid S \in E\}$

<u>Hiding</u>

Hiding some events in a statechart from the outside world is only consistent when the statechart is closed for these events. Hence the hiding operator first closes the statechart for the specified events and then deletes all occurrences of them from the histories.

Let $f(i) = (t,p,w)$ and $p = (V,<,l)$ and $t' = t\backslash\{a! \mid a \in E\}$, then define

$$\mathit{Hi}(f(i),E) = (t' \;,\; (V,<,\mathcal{L}\restriction t') \;,\; w)$$

and

$$\mathit{Hi}(f,E) = f' \text{ iff } \forall i\colon\ f'(i) = \mathit{Hi}(f(i),E)$$

Then

$$\mathfrak{D}(\mathrm{Hide}(U,E)) =$$

$$\{(u,f,v) \mid \exists f' \in \mathit{Cl}(f,E)\colon\ f = \mathit{Hi}(f',E')\}$$

where $E' = (E \cap E_p) \cup \{\mathit{en}(S), \mathit{ex}(S) \mid S \in E\}$.

5 Discussion

In this chapter we discuss the problem of abstraction of the semantics, the future extension of statecharts with variables, and a possible other definition of the semantics with respect to causality between micro-steps.

5.1 Full Abstraction

The presented semantics records many properties of a statechart that we are not directly interested in, but are necessary to define a compositional semantics. The properties we are interested in anyway is called the observable behaviour. The decision what is observable and what not is in principle a free one. Here we adopt the reasonable choice of all (not hidden) occurrences of events, related to the time of their occurrence. In other words, all records of the form $\alpha!$ in the histories are observable, but claims of the form α or $\bar{\alpha}$ and the partial ordering are not. Now we intend to make our semantics fully abstract w.r.t. this notion of observable behaviour. As usual, this means that two programs only have a different semantics if there is a syntactical context in which they have a different observable behaviour. In a formula:

$$\forall P,Q: \mathcal{D}(P)\neq\mathcal{D}(Q) \rightarrow \exists C: \mathcal{O}(C(P))\neq\mathcal{O}(C(Q))$$

where $\mathcal{O}$ associates to each statechart its observable behaviour and C is a syntactical context, a statechart with a hole in which another statechart can be plugged in, thus yielding a complete statechart.

The reader is referred to [HGR] for further details.

fig 5.2

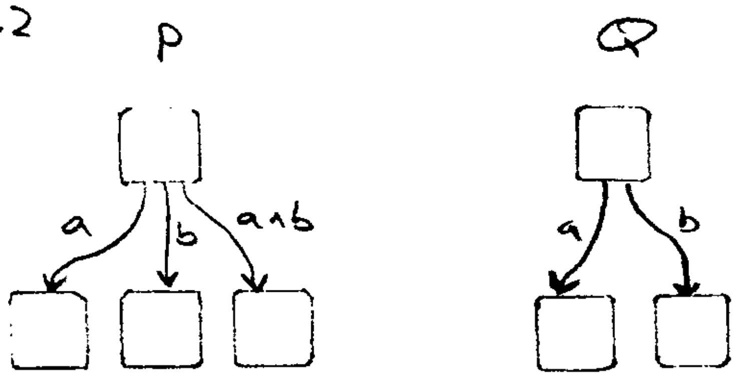


fig 5.1

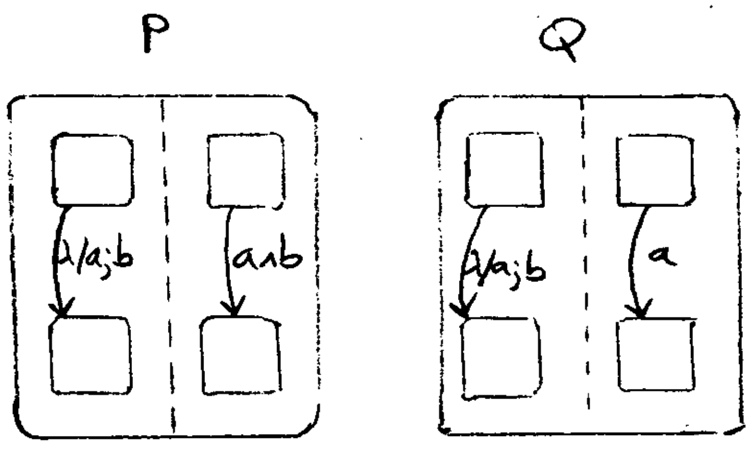


fig. 5.3

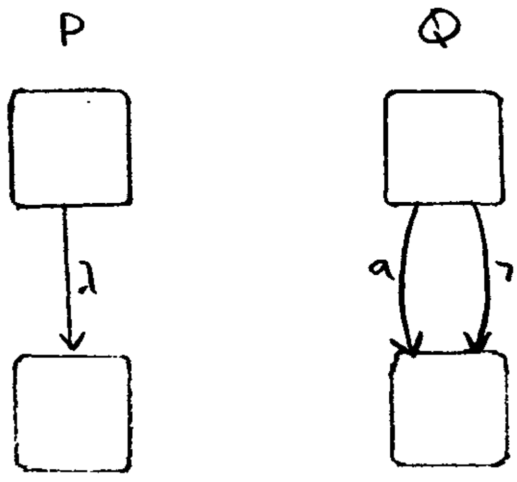
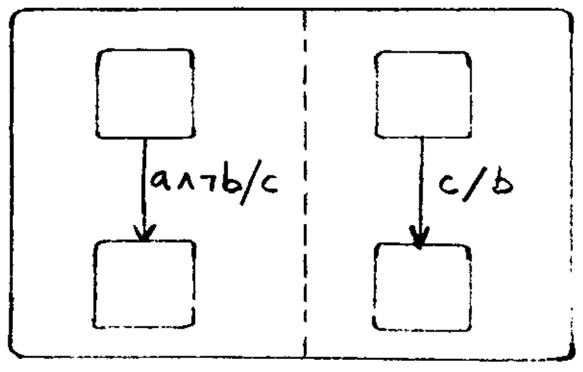


fig. 5.4



We can find many statecharts for which this implication doesnot hold. E.g., in fig. 5.1, P and Q have different semantics, but they will behave equally in any context. The only difference between P and Q - the extra claim b in one transition record of some histories of $\mathfrak{D}(P)$ - is irrelevant, because this claim is already fulfilled due to the presence of $b!$ in the same transition record and its precedence in the partial ordering.

More examples of this *redundancy* of claims *within* a particular history can be found. These redundancies can easily be removed by changing the definition of the merge ($\|$) of two histories. Here, the information that becomes redundant due to the added information (empty labelled nodes and identical nodes in the partial ordering, fulfilled claims) should be removed.

A more complicated kind of redundancy occurs *between* the histories of a particular denotation. E.g., in fig. 5.2, P and Q have different semantics, but cannot be distinguished observably by any context. The history h with $h(0)=(\{a,b\},1,\emptyset)$ in $\mathfrak{D}(P)$ is not present in $\mathfrak{D}(Q)$, but cannot influence the observable behaviour in any context, because any behaviour the history h can generate, can be generated by one of the histories of $D(Q)$ - and vice versa of course.

In [HGR] and [GB], a technique is presented to make comparable models for real-time languages fully abstract. This technique can also be applied here and this will dissolve the kind of redundancy described in this example, but not the one shown in fig. 5.3.

It is quite clear that there does not exist a context that can distinguish P and Q observably. Whether a occurs or not, the system will go from S to T. Yet they have different semantics: some histories of $\mathfrak{D}(P)$ contain an empty transition record and these do not occur in $\mathfrak{D}(Q)$.

To remove this kind of redundancy we now study a generalisation of the technique used in [HGR,GB].

5.2 Variables

The full version of this paper will include the use of variables in the labels of transitions (in conditions and in actions as assignments). This will not involve an essential extension of the model. The same technique used for the condition *in*(S) can be applied here. All changes to variables are signalled in the form of events and the satisfaction of conditions is checked by an inspection of the history.

5.3 Other definition on causality

In the semantics of [HPSS], the influence of a transition is restricted to the transitions that follow it in the sequence of micro-steps building the macro-step. In our compositional semantics, this is modelled by the partial order in the clock record. This solves the causal paradox of the transition annulating its own cause (see fig. 5.4), but this solution is not fully satisfactory. E.g., a transition labelled $\neg a$ can always be taken, even if a happens during that time unit. (It only differs from a transition labelled λ in that it need not be taken when a happens.) Furthermore, the semantics depends heavily on the relative order in which the micro-steps occur, whereas the micro-steps are definitely not observable - they are only introduced to solve the causal problems.

A new version of the operational semantics is under study by Pnueli and others, in which *global contradictions* are not allowed. A global contradiction occurs when two transitions with conflicting labels take place in the same macro-step. E.g., a transition labelled $\neg a$ can never take place in the same macro-step with a transition labelled $\ldots/a$, even if the latter occurs in a later micro-step. This leads to a simpler and more intuitive semantics. The main drawback, however, is that causal paradoxes such as the one in fig. 5.4 now lead to a run time error. There is no acceptable behaviour anymore to associate to these situations and there is no way to detect them syntactically.

We can easily adapt the compositional semantics to model this new operational semantics. All negative claims of the form $\bar{a}$ should be put into the global component of the clock record, even if they come from actual transitions. The partial order is not extended at the merge of histories, because there are no negative claims anymore in the transition record. All other things stay the same.

6 Conclusion

We presented a compositional semantics for the graphical specification/programming language Statecharts, as described in [HPPS]. For this, we had to define a proper generative syntax. The operators in this syntax have simple graphical counterparts as well as a natural semantics. The model extends the model of [HGR] to deal with broadcast and, specifically, with the micro-step semantics of State-charts as described in [HPS]. This is a subtle operational notion to deal with the consequences of the synchrony of action and reaction. The compositional semantics does not model the micro-steps directly, but records only the occurrence relationship between the micro-steps.

This work serves as a basis for extending the work of Hooman on proof-systems for CSP-R [H] and that of Zwiers [Z].

References

[B] Berry G., Cosserat L., The Synchronous Programming Language ESTEREL and its Mathematical Semantics, Seminar on Concurrency, Springer-Verlag, LNCS 197, Science of Programming 1984.

[BG] Gerth R., Boucher A., A Timed Failures Model for Extended Communicating Processes, Proc. ICALP 1986, LNCS 267, pp 95-114, Springer Verlag, Berlin.

[DD] Damm W., Döhmen G. (1987), An axiomatic approach to the specification of distributed computer architectures, LNCS 258, Springer Verlag, Berlin.

[H] Harel D., Statecharts: A visual Approach to Complex Systems, Science of Computer Programming, Vol.**8-3**, pp231-274, 1987.

[HGR] Huizing C., Gerth R., De Roever W.P., (1987), Full Abstraction of a Real-Time Denotational Semantics for an OCCAM-like language, Proc. POPL 1987.

[Ho] Hooman J., A compositional proof theory for real-time distributed message passing, LNCS 259, pp 315-332 (1987).

[HP] Harel D., Pnueli A., On the Development of Reactive Systems, Logic and Models of Concurrent Systems, K.R. Apt Ed., Springer Verlag, Berlin (1985), pp 477-498.

[HPSS] Harel D., Pnueli A., Pruzan-Schmidt J., Sherman R., On the Formal Semantics of Statecharts, Proc. Symposion on Logic in Computer Science 1987 (LICS), pp54-64.

[HU] Hopccroft J.E., Ullman J.D., Introduction to automata theory, languages, and computation, Addison-Wesley, Reading, 1979.

[K&] Koymans R., Shyamasundar R.K., De Roever W.P., Gerth R., Arun-Kumar S. (1986), Compositional Semantics for Real-Time Distributed Computing, Information and Control, to appear.

[LUSTRE] Bergerand J.-L., Caspi P., Halbwachs N., (1985), Outline of a real-time dataflow language, Proc. IEEE-CS Real-Time systems Symposium, San Diego.

[M] Mazurkiewicz A., Proving algorithms by tail functions, Information and Control, **18**, (1971), pp 220-226.

[SIGNAL] Le Guernic P., Beneviste A., Bournal P., Gauthier T., SIGNAL:

A Data Flow Oriented Language For Signal Processing, IRISA Report 246, IRISA, Rennes, France (1985).

[SM] Salwicki A., Müldner T., (1981), On the Algorithmic Properties of Concurrent Programs, LNCS 125, Springer Verlag, New York.

[SW] Strachey C., Wadsworth C.P., Continuations: A Mathematical Semantics for Handling Full Jumps, Technical Monograph PRG-11, Oxford University Computing Laboratory, Oxford.

[Z] Zwiers J., Compositionality and dynamic networks of processes: Investigating verification systems for DNP, Ph.D. Thesis to appear in November, 1987, Eindhoven University of Technology.

Appendix

In [HPSS] the set of statecharts is not defined by a generative grammar, but in a more direct way. We shall call these objects H-statecharts and define the formal relationship between H-statecharts and the elements of the set **Stch**, the expressions generated by the syntax as defined in chapter 3.

Definition:

Let a set of states Σ and a set of labels **Lab** be given.

A **H-statechart** is a quintuple (S,ρ,ψ,δ,T) where

$S \subset \Sigma$ is the set of states;

ρ: $S \to 2^S$ is the hierarchy function;

ψ: $S \to \{\text{AND},\text{OR}\}$ is the type function;

δ: $S \to 2^S$ is the default fucntion;

$T \subseteq S\mathbf{x}\mathbf{Lab}\mathbf{x}S$ is the set of transitions,

with the following restrictions:

(i) $\forall\, s\epsilon S$: $s\notin\rho^+(S)$

(ii) $\forall\, s_1,s_2$: $s_1 \neq s_2 \to \rho(s_1) \cap \rho(s_2) = \emptyset$

(iii) $\forall\, s\epsilon S$: $\delta(s) \subseteq \rho^+(s)$

(iv) $\exists!\, r\epsilon S$: $\rho^*(r) = S \wedge \forall\, t\epsilon T$: $r\notin{}^{<}t \wedge r\notin t^{>}$.

(v) $\forall\, s\epsilon S$: $(\exists\, x\epsilon S$: $s\epsilon\rho(x) \wedge \psi(x)=\text{AND}) \to \forall\, t\epsilon T$: $s\notin{}^{<}t \wedge s\notin t^{>}$

The set of H-statecharts is called **HS**. □

Notation:

if $t\epsilon T$ and $t=(s_1,l,s_2)$, then ${}^{<}t=s_1$, $\hat{t}=l$ and $t^{>}=s_2$.

where ρ^* and ρ^+ are the reflexive resp. irreflexive transitive closure of ρ.

We define a function $\mathfrak{R}$: **HS** $\to$ **Stch** as follows.

Let $\sigma=(S,\rho,\psi,\delta,T)$ be given.

Define a function $\mathcal{E}$: $S \to$ **Unv** that gives for each substatechart consisting of a state and its interior the associated Unvollendete. Then we can define:

$\mathfrak{R}(\sigma) = \mathcal{E}(r)$ where r is the root state of σ, i.e. $\forall\, s \in S$: $r \notin \rho(s)$.

Define $T_I = \{(t,s) \in T \times S \mid s \in t^{>}\} \cup \{(s_1,s_2) \in S \times S \mid s_2 \in \delta(s_1)\}$

$T_O = \{(s,t) \in S \times T \mid s \in {}^{<}t\}$

$L: T_O \rightarrow \mathbf{Lab}$

$L(s,t) = \hat{t}$

Notation: if $i \in T_I$ and $i=(t,s)$, then $i^{>} = t^{>}$ and $tr(i)=t$;

if $o \in T_O$ and $o=(s,t)$, then ${}^{<}o = {}^{<}o$ and $tr(o)=t$.

T_I and T_O will serve as the set of incoming resp. outgoing transitions for the Unvollendetes we are going to use. Since defaults are made out of incoming transitions, we need some for these purpose.

Define an auxiliary function $\mathcal{E}_p: S \rightarrow \mathbf{Unv}$

$\mathcal{E}_p(s) = [I,O,S]$ with $I = \{(t,s) \in T_I \mid s \in t^{>}$

and $O = \{(s,t) \in T_O \mid s \in {}^{<}t\}$

For $U \in \mathbf{Unv}$, define

$Inc(U) = I$ if $U = \langle I,O \rangle$

$Outg(U) = O$ if $U = \langle I,O \rangle$.

We need a function $\mathcal{E}_i: S \rightarrow \mathbf{Unv}$ that gives for each state the Unvollendete that should be associated to the *interior* of that state. It depends on whether it is an AND-state or an OR-state.

We define $\mathcal{E}$ and $\mathcal{E}_i$ mutually recursively:

$\mathcal{E}(s) = \mathcal{E}_p(s)$ if $\rho(s) = \emptyset$;

$\mathcal{E}(s) = \mathrm{Stat}(\ \mathcal{E}_p(s),\ \mathcal{E}_i(s),\ (s,s')\ )$ if $\rho(s) \neq \emptyset$,

with $(s,s') \in Inc(\mathcal{E}_i(s))$ for some s';

The definition of $\mathcal{E}_i(s)$ is the most complicated.

Let $s \in S$ be given and let $\rho(s) = (s_1,\dots,s_n)$, $n>1$.

Distinguish two cases:

(i) $\underline{\psi(s) = \mathrm{AND}}$

Define a sequence of Unvollendetes $A_1,\dots,A_n$ as follows:

$A_1 = \mathcal{E}(s_1)$

A_j = And(A_{j-1}, $\mathcal{E}(s_j)$, a_j) for $2 \leq j \leq n$

and $a_j = \{(i_1, i_2) \in I_1 x I_2 \mid s_j \in i_2^{>} \wedge \exists\ 1 \leq k < j:\ s_k \in i_1^{>}\}$

and I_1 = *Inc*(A_{j-1}), I_2=*Inc*($\mathcal{E}(s_j)$).

Then $\mathcal{E}_i(s) = A_n$.

(ii) <u>$\psi(s)$ = OR</u>

Let U = Or(...Or($\mathcal{E}(s_1)$, (s_2)),...,$\mathcal{E}(s_n)$)

Let $\{t_1, \ldots, t_n\} = \{t \in T \mid$ LCA(t) = s$\}$. Here, LCA is a function defined in [HPSS]; LCA(t) gives the smallest state that encloses transition t:

Let R = $^{<}t \cup t^{>}$, then LCA(t) = x iff

1. $R \subseteq \rho^{+}(x)$
2. $\psi(x)$ = OR
3. $\forall\ s \in R$: if $\psi(s)$ = OR then $R \subseteq \rho^{+}(s) \rightarrow x \in \rho^{*}(s)$.

Define a sequence of Unvollendetes $B_0, \ldots, B_n$ as follows.

B_0 = U

B_j = Conn(B_{j-1},o_j,i_j) for $1 \leq j \leq n$,

where $tr(i_j) = tr(o_j) = t_j$.

Then $\mathcal{E}_i(s) = B_n$.

Available Reports from the Theoretical Computing Science Group

	Author(s)	Title	Classification EUT	Classification DESCARTES
TIR82.1	R. Kuiper, W.P. de Roever	Fairness Assumptions for CSP in a Temporal Logic Framework		
TIR83.1	R. Koymans, J. Vytopil, W.P. de Roever	Real-Time Programming and Synchronous Message passing (2nd ACM PODC)		
TIR83.2	H. Barringer, R. Kuiper	Towards the Hierarchical, Temporal Logic, Specification of Concurrent Systems		
TIR84.1	R. Gerth, W.P. de Roever	A Proof System for Concurrent Ada Programs (SCP4)		
TIR84.2	R. Gerth	Transition Logic - how to reason about temporal properties in a compositional way (16th ACM FOCS)		
TIR84.3	H.Barringer, R. Kuiper, A. Pnueli	Now you may compose Temporal Logic Specifications (Proc. STOC84)		
TIR84.4	H. Barringer, R. Kuiper	Hierarchical Development of Concurrent Systems in a Temporal Logic Framework		
TIR85.1	W.P. de Roever	The Quest for Compositionality - a survey of assertion-based proof systems for concurrent progams, Part I: Concurrency based on shared variables (IFIP85)		
TIR85.2	O. Grünberg, N. Francez, J. Makowsky, W.P. de Roever	A proof-rule for fair termination of guarded commands (Inf.& Control 1986)		

TIR85.3	F.A. Stomp, W.P. de Roever, R. Gerth	The μ-calculus as an assertion language for fairness arguments (Inf.& Control 1987)		
TIR85.4	R. Koymans, W.P. de Roever	Examples of a Real-Time Temporal Logic Specification (LNCS207)		
TIR85.5	H. Barringer, R. Kuiper, A. Pnueli	A Compositional Temporal Approach to a CSP-like Language		
TIR86.1	R. Koymans	Specifying Message Passing and Real-Time Systems (extended abstract)	CSN86/01	
TIR86.2	J. Hooman, W.P. de Roever	The Quest goes on: A Survey of Proof Systems for Partial Correctness of CSP (LNCS227)	EUT-Report 86-WSK-01	
TIR86.3	R. Gerth, L. Shira	On Proving Communication Closedness of Distributed Layers (LNCS236)	CSN86/07	
TIR86.4	R. Koymans, R.K. Shyamasundar, W.P. de Roever, R. Gerth, S. Arun Kumar	Compositional Semantics for Real-Time Distributed Computing (Inf.&Control 1987)	CSN86/08	
TIR86.5	C. Huizing, R. Gerth, W.P. de Roever	Full Abstraction of a Real-Time Denotational Semantics for an OCCAM-like Language	CSN86/09	PE.01
TIR86.6	J. Hooman	A Compositional Proof Theory for Real-Time Distributed Message Passing	CSN86/10	TR.4-1-1(1)
TIR86.7	W.P. de Roever	Questions to Robin Milner - A Responder's Commentary (IFIP86)	CSN86/11	
TIR86.8	R. Gerth, A. Boucher	A Timed Failures Model for Extended Communicating Processes; extended abstract (ICALP87)	CSN86/12	TR.4-4(1)
TIR86.9	R. Gerth, W.P. de Roever	Proving Monitors Revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)	CSN86/13	

TIR86.10	R. Koymans	Specifying Message Passing Systems Requires Extending Temporal Logic	CSN86/14	PE.02
TIR86.11	H. Barringer, R. Kuiper, A. Pnueli	A Really Abstract Temporal Logic Semantics for Concurrency (Proc. POPL86)		
TIR87.1	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept	CSN87/01	
TIR87.2	R. Kuiper	Enforcing Nondeterminism via Linear Time Temporal Logic Specifications	CSN87/05	
TIR87.3	R. Koymans	Temporele Logica Specificatie van Message Passing en Real-Time Systemen (in Dutch)	CSN87/06	
TIR87.4	R. Koymans	Specifying Message Passing and Real-Time Systems with Real-Time Temporal Logic	CSN87/07	PE.03
TIR87.5	F. A. Stomp, W.P. de Roever	A correctness proof of a distributed minimum-weight spanning tree algorithm		
TIR87.6	J. Hooman	A Compositional Proof System for an OCCAM-like Real-Time Language	CSN87/14	D4-1-2
TIR87.7	C.Huizing, R. Gerth, W.P. de Roever	A Compositional Semantics for Statecharts	CSN87/15	D4-2-1
TIR88.1	R. Gerth, M.Codish, Y.Lichtenstein, E. Shapiro	Fully Abstract Denotational Semantics for Concurrent PROLOG	CSN87/21	
TIR88.2	F.A. Stomp, W.P. de Roever, S.Ramesh	A New Decomposition Principle for Verifying Distributed Algorithms, Formalizing their Designer's Intuition		

86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arum Kumar	Compositional semantics for real-time distributed computing (Inf. & Control 1987)
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing
86/11	W.P. de Roever	Questions to Robin Milner - A responders commentary (IFIP86)
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)
86/14	R. Koymans	Specifying passing systems requires extending temporal logic
87/01	R. Gerth	On the existence of a sound and complete axiomatizations of the monitor concept
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases
87/03	G.J. Houben J. Paredaens	A formal approach to distributed information systems
87/04	T. Verhoeff	Delay-insensitive codes - An overview
87/05	R. Kuiper	Enforcing non-determinism via linear time temporal logic specification

87/06	R. Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch)
87/07	R. Koymans	Specifying message passing and real-time systems with real-time temporal logic
87/08	H.M.J.L. Schols	The maximum number of states after projection
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics
87/10	T. Verhoeff	Three families of maximally nondeterministic automata
87/11	P. Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model
87/12	K.M. van Hee A. Lapinski	OR and AI approaches to decision support systems
87/13	J. van der Woude	Playing with patterns, searching for strings
87/14	J. Hooman	A compositional proof system for an occam-like real-time language
87/15	G. Huizing R. Gerth W.P. de Roever	A compositional semantics for statecharts
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas
87/17	K.M. van Hee G.J. Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples

87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces
87/19	A.J. Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen
87/20	G.J. Houben J. Paredaens	The R^2-Algebra: An extension of an algebra for nested relations
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples