Duration Calculus Semantics for Statecharts

Marisa A. Sánchez[1], Pablo Fillottrani[2], and Miguel A. Felder[3]

[1] Dpto. de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur
Avda. Alem 1253, Bahía Blanca
mas@cs.uns.edu.ar

[2] Dpto. de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur
Avda. Alem 1253, Bahía Blanca
prf@cs.uns.edu.ar

[3] Pragma Consultores, Buenos Aires
mfelder@pragma.com.ar

Abstract. Statecharts and Duration Calculus are two formalisms used in the development of reactive systems. Statecharts provide a powerful visual formalism to specify systems. Duration Calculus is a formal logic to specify and reason about temporal requeriments. In this work, we propose a description of statecharts semantics using Duration Calculus. Thus we build a common semantic model for Duration Calculus specifications and statecharts. The formalization is done in two steps. First, the structure of a statechart is represented using Duration Calculus formula. Then, the semantics of the execution of a step is introduced.

Keywords: statecharts semantics, Duration Calculus, Statemate

1 Introduction

Statecharts and Duration Calculus are two formalisms used in the development of reactive systems. Statecharts were developed by David Harel in 1983 as a visual formalism to specify reactive behavior. Statecharts are widely used in the Software Engineering community [1, 2] and different dialects of the language have been employed in several software design notations, including STATEMATE [3] and UML [4]. Also, several papers include definitions of semantics. Some examples are [5–8]. In this work, we are interested in the semantics as implemented in the STATEMATE system [9]. The behavior of a system described in STATEMATE is a set of possible *runs*, each of one representing the response of the system to a sequence of external stimulus generated by the environment. This semantics is of practical interest since it underlies the tools of STATEMATE, and thus it is relevant for system designers as well as for developers of tools for STATEMATE.

We propose a semantics based on the rigorous but informal definition of statecharts given in [9], using Duration Calculus and Mean Value Duration Calculus. Duration Calculus is a real-time interval logic that has been successfully used for requirements specification and design [10–12]. The main reason that influenced our selection of Duration Calculus is that, for the best of our knowledge, the existing approaches to specify semantics do not use a real-time interval logic. In particular, this work can be seen as a link between statecharts and Duration Calculus tools [13]. Dynamic system properties are usually represented with a mixture of timing diagrams, statecharts, differential equations, or other ad hoc representations. For quality assurance purposes, it would be desirable to integrate the analysis of components that may be described using different approaches.

The paper is organized as follows: Section 2 briefly introduces Duration Calculus and Mean Value Calculus. In Section 3 we give a formalization of the structure of statecharts, and in Section 4 we discuss a formalization of the semantics of statecharts. Conclusions are presented in Section 5.

2 A Brief Introduction to Duration Calculus

The original Duration Calculus was introduced by Zhou Chaochen *et al.* [10]. Several extensions have been proposed, notably the Mean Value Calculus [14], that extends Duration Calculus to deal with point intervals. In this work we consider the Mean Value Calculus with continuous semantics.

In both Duration Calculus and Mean Value Calculus, a system is modeled by a number of functions from the time domain $\mathbf{R}^+$ to the Boolean domain $\{0, 1\}$. These functions are called the state variables of the system. For a state variable (or a Boolean combination of state variables) P, its duration in a time interval, written $\int P$ in Duration Calculus, is the integral of P over the time interval. In Mean Value Calculus, durations are replaced by mean values and interval lengths. For a state variable (or a Boolean combination of state variables) P, its mean value, written $\overline{P}$, is the mean value of P over a time interval if the interval is not a point interval, and the value of P at the point otherwise. Hence, for a state variable P and an interval $[a, b]$, the mean value $\overline{P}$ is defined as:

$$\overline{P}[a,b] = \begin{cases} \int_a^b P(t)dt/(b-a) & if\ b > a \quad and \\ P(b) & if\ b = a \end{cases}$$

In the following, we present the syntax and the semantics of Mean Value Duration Calculus based on the presentations given in [14] and [15].

2.1 Syntax

We assume a countably infinite set of logical variables $\mathcal{V}$ and a countable infinite set of state variables $\mathcal{SV}$. Furthermore, likewise classical logic, we assume a finite set of function symbols and a finite set of predicate symbols.

State expressions: the set of state expressions is defined by the following rules:

1. 0,1, and each $v \in \mathcal{SV}$ are state expressions;
2. if P and Q are state expressions, then $\neg P$ and $P \vee Q$ are state expressions.

Terms: the set of terms is defined by the following rules:

1. if P is a state expression, then $\bar{P}$ is a term;
2. each $v \in \mathcal{V}$ is a term;
3. ℓ is a term;
4. if $r_1, ..., r_n$ are terms and f_i^n is an n-ary function symbol, then $f_i^n(r_1, ..., r_n)$ is a term.

Formulas: the set of formulas is defined by the following rules:

1. if A_i^n is an n-ary predicate symbol and $r_1, ..., r_n$ are n terms then $A_i^n(r_1, ..., r_n)$ is a formula;
2. $true$ is a formula;
3. if r, r' are terms, then $r = r'$ is a formula;
4. if δ is a formula, then $\neg\delta$ is a formula;
5. if δ, σ are formulas, then $\delta \wedge \sigma$ and $\delta; \sigma$ are formulas;
6. if $v \in \mathcal{V}$ and δ is a formula, then $(\forall v)\delta$, is a formula.

The following abbreviations are also frequently used: $\lceil P \rceil^0$ for $\ell = 0 \ \wedge \ \bar{P} = 1$, and $\lceil P \rceil$ for $\ell > 0 \ \wedge \neg(\ell > 0; \lceil P \rceil^0; \ell > 0)$. Informally, this means that $\lceil P \rceil^0$ is true at an interval iff the interval is a point interval and P has the value 1 at that point; and $\lceil P \rceil$ is true at an interval iff the interval is not a point interval and P has the value 1 everywhere inside the interval the interval.

2.2 Semantics

We assume that there is a function $\underline{f_i^n} : \mathbf{R^n} \to \mathbf{R}$ associated with each n-ary function symbol f_i^n, and a relation $\underline{A_i^n} : \mathbf{R^n} \to \{tt, ff\}$ with each n-ary predicate symbol A_i^n. We write $[b, e]$, where $b, e \in \mathbf{R}^+$ and $b \cdot \ e$, for bounded and closed intervals.
The truth of Mean Value Calculus formulas is defined below with respect to an interpretation of state variables and an assignment of logical variables. Let $Time$ be the set of

non-negative reals $[0,\infty)$. Then an interpretation $\mathcal{I}$ over $Time$ is a function $\mathcal{I} : \mathcal{SV} \to (Time \to \{0,1\})$ where, for each $v \in \mathcal{SV}$, the discontinuity points of $\mathcal{I}(v)$ belong to $Time$. Let $Intv$ be the set of bounded and closed $Time$-intervals: $\{[a,b] \mid a \cdot \ \ b \wedge a, b \in Time\}$.

The *semantics of a state expression* P under interpretation $\mathcal{I}$ over $Time$ is a function $[P]_{\mathcal{I}} : Time \to \{0,1\}$ inductively defined by:

$[0]^{\mathcal{I}}(t) = 0,$
$[1]^{\mathcal{I}}(t) = 1,$
$[v]^{\mathcal{I}}(t) = \mathcal{I}(v)(t),$
$[\neg P]^{\mathcal{I}}(t) = 1 - [P]^{\mathcal{I}}(t),$
$[P \wedge Q]^{\mathcal{I}}(t) = \begin{cases} 1 \; if \;\; [P]^{\mathcal{I}}(t) = 1 \; and \; [Q]^{\mathcal{I}}(t) = 1 \\ 0 \; otherwise \end{cases}$

The *semantics of a term* r under interpretation $\mathcal{I}$ over $Time$ and assignment α is the functions $[r]^{\mathcal{I}}_{\alpha} : Intv \to \mathbf{R}$ inductively defined by:

$[\ell]^{\mathcal{I}}_{\alpha}([b,e]) = e - b$
$[x]^{\mathcal{I}}_{\alpha}([b,e]) = \alpha(x)$
$[\overline{P}]^{\mathcal{I}}_{\alpha}([b,e]) = \begin{cases} \int_b^e [P]^{\mathcal{I}}(t)dt/(e-b) \; if \;\; e - b > 0 \\ [P]^{\mathcal{I}}(e) = 1 \qquad\qquad otherwise \end{cases}$
$[f(r_1, ..., r_n)]^{\mathcal{I}}_{\alpha}([b,e]) = f([r_1]^{\mathcal{I}}_{\alpha}([b,e]), ..., [r_n]^{\mathcal{I}}_{\alpha}([b,e]))$

The *semantics of a formula* ϕ at interval $[b,e] \in Intv$ under interpretation $\mathcal{I}$ over $Time$ and assignment α, written $\mathcal{I}, [b,e] \models_{\alpha} \phi$, is inductively defined by:

$\mathcal{I}, [b,e] \models_{\alpha} t,$
$\mathcal{I}, [b,e] \models_{\alpha} r = r' \Leftrightarrow [r]^{\mathcal{I}}_{\alpha}([b,e]) = [r']^{\mathcal{I}}_{\alpha}([b,e]),$
$\mathcal{I}, [b,e] \models_{\alpha} A(r_1, ..., r_n) \Leftrightarrow ([r_1]^{\mathcal{I}}_{\alpha}([b,e]), ..., [r_n]^{\mathcal{I}}_{\alpha}([b,e]) \in A,$
$\mathcal{I}, [b,e] \models_{\alpha} \neg\phi \Leftrightarrow \neg(\mathcal{I}, [b,e] \models_{\alpha} \phi),$
$\mathcal{I}, [b,e] \models_{\alpha} \phi \wedge \psi \Leftrightarrow (\mathcal{I}, [b,e] \models_{\alpha} \phi) \wedge (\mathcal{I}, [b,e] \models_{\alpha}),$
$\mathcal{I}, [b,e] \models_{\alpha} \phi; \psi \Leftrightarrow \;\; for \; some \; m \in Time \; where \; m \in [b,e], \mathcal{I}, [b,m] \models_{\alpha} \phi \; and \; \mathcal{I}, [m,e] \models_{\alpha} \psi$
$\mathcal{I}, [b,e] \models_{\alpha} \forall x \cdot \phi \Leftrightarrow \forall d \in R, \mathcal{I}, [b,e] \models_{\alpha(x \to d)} \phi.$

We write $\mathcal{I}, [b,e] \models \phi$ to indicate that $\mathcal{I}, [b,e] \models_{\alpha} \phi$ for all assignments α.

3 Formal Description of the Structure of Statecharts

3.1 States and Transitions

In order to describe syntax we need to describe states and transitions. Statecharts states can be partitioned into the following types:

- Compound states (super-states) of type OR that have substates that are related to each other by "exclusive-or" relation.
- Compound states (super-states) of type AND that have orthogonal components that are related by "and" relation.
- Simple states that are those at the bottom of the state hierarchy.
- The root state, i.e. the state with no parent state.

We use constants to represent a given statechart and the states, transitions, events and actions in it. We associate each entity with its statechart using predicates, for example, $State_{COR}(M, S)$ is defined to be true iff S is an or-state of M. S and M are constants. In Table 1 we include all used predicates. Hence, we can define the predicate $State(M, S)$ as

$$State(M, S) \equiv State_{COR}(M, S) \vee State_{CAND}(M, S) \vee State_S(M, S) \vee State_H(M, S)$$

which states whether a state S is a state of statechart M.

Table 1. Temporal independent description of a statechart

Predicate	is defined to be true iff
$State(M, S)$	S is a state of M
$State_{COR}(M, S)$	S is an or-state of M
$State_{CAND}(M, S)$	S is an and-state of M
$State_S(M, S)$	S is a simple state of M
$State_I(S, S_i)$	S_i is the default (initial) state of S
$State_H(M, S)$	S is an history state of M
$State_R(M, S)$	S is the root state of M
$Transition(M, T)$	T is a transition of M
$Source(T, S)$	S is the source state of transition T
$Target(T, S')$	S' is the target state of transition T
$Trigger(T, E)$	E is the trigger event of transition T
$Action(T, A)$	A is an action related with transition T
$History(H, S)$	H is a history connector of state S
$Event(M, E)$	E is a trigger event of some transition in M

As another example consider the formula $Transition(M, T)$ which states that transition T is included in statechart M. Also, the predicates $Source(T, S)$, $Target(T, S')$, $Trigger(T, E)$, $Action(T, A)$, and $Guard(T)$, are defined to be true if S, S', E, and A are the source state, the target state, the trigger event, the action, and the guard condition related with T. We use state variables to represent trigger events and actions. Both trigger events and actions can be a conjunction of particles of trigger events or actions. Note that all these predicates are time independent.

Example. Based on the set of predicates available for a static description of a statechart (see Table 1), Figure 2 shows a description of the statechart in Figure 1.

As described in Section 2, Duration Calculus uses states to model behavior of real-time systems. A Boolean state model of a system is a set of Boolean valued functions over time:

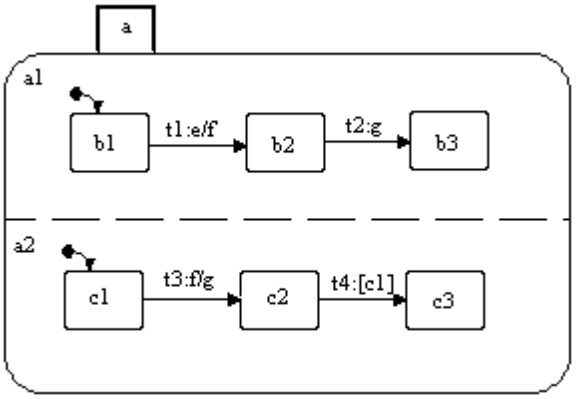

Fig. 1. Example of a Statechart

$State_{CAND}(m, a)$,	$State_{COR}(m, a_1)$,	$State_{COR}(m, a_2)$,	$State_I(a_1, b_1)$,
$State_I(a_2, c_1)$,	$State_S(m, b_1)$,	$State_S(m, b_2)$,	$State_S(m, b_3)$,
$State_S(m, c_1)$,	$State_S(m, c_2)$,	$State_S(m, c_3)$	$Transition(m, t_1)$,
$Transition(m, t_2)$,	$Transition(m, t_3)$,	$Transition(m, t_4)$,	$Source(t_1, b_1)$,
$Source(t_2, b_2)$,	$Source(t_3, c_1)$,	$Source(t_4, c_2)$,	$Target(t_1, b_2)$,
$Target(t_3, c_2)$,	$Trigger(t_1, e)$,	$Trigger(t_2, g)$,	$Trigger(t_3, f)$,
$Action(t_1, f)$,	$Action(t_3, g)$,	$Guard(t_4)$,	$Event(m, e)$,
$Event(m, f)$,	$Event(m, g)$		

Fig. 2. Description of the statechart example

$$Time \rightarrow \{0, 1\}$$

where $Time$ is the set of non-negative reals. Each Boolean-valued function (also called a Boolean state, or state, or state variable) of the system, is a function of a specific aspect of the system behavior, an the whole set of Boolean-valued functions characterizes all the aspects of the behavior. We associate state variables with each state to represent whether the state is active at an instant. This means that when a state is active the respective state variable is true, and false otherwise. We denote constants with $a, b, c, ...$ and the respective states variables with $\mathtt{A}, \mathtt{B}, \mathtt{C}, ...$

Example. Consider the statechart m described in Figure 1. Since the atom $State_S(m, b_1)$ is true, let $\mathtt{B}_1 \in Time \rightarrow \{0, 1\}$ be the state variable associated with b_1, and

$\mathtt{B}_1(t) = 1$, means that $\mathtt{B}_1$ is an active state of the statechart at instant t

$\mathtt{B}_1(t) = 0$, means that $\mathtt{B}_1$ is not an active state of the statechart at instant t

The value of $\mathtt{B}_1$ depends on the actual behavior of the statechart. This is further discussed in Section 4.

Similarly, the atom $State_{COR}(m, A_1)$ is true, $\mathtt{A_1} \in Time \rightarrow \{0,1\}$, and:

$\mathtt{A_1}(t) = 1$, means that $\mathtt{A_1}$ is an active state of the statechart at instant t
$\mathtt{A_1}(t) = 0$, means that $\mathtt{A_1}$ is not an active state of the statechart at instant t

We can refine state $\mathtt{A_1}$ in different sub-sates, so $\mathtt{A_1}$ is a state expression of $\mathtt{B_1}$, $\mathtt{B_2}$ and $\mathtt{B_3}$:

$$\mathtt{A_1} \stackrel{\text{def}}{=} (\mathtt{B_1} \vee \mathtt{B_2} \vee \mathtt{B_3}) \wedge \neg(\mathtt{B_1} \wedge \mathtt{B_2}) \wedge \neg(\mathtt{B_1} \wedge \mathtt{B_3}) \wedge \neg(\mathtt{B_2} \wedge \mathtt{B_3})$$

In general, a super-state S of type OR with n sub-sates S_i is represented as:

$$S \stackrel{\text{def}}{=} (\bigvee_{i=1}^{n} S_i) \wedge \bigwedge_{i \neq j} \neg(S_i \wedge S_j)$$

For the case of a super-state S of type AND with n sub-sates S_i we have:

$$S \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} S_i$$

Example. For the AND state $\mathtt{A}$ in Figure 1 we have:

$$\mathtt{A} \stackrel{\text{def}}{=} \mathtt{A_1} \wedge \mathtt{A_2}$$

An initial state may be an OR state, an AND state, or a simple state. The root state of a statechart is always an initial state.

Expression $Event(M, E)$ is defined to be true if event E is included as a trigger event of at least one transition of statechart M:

$$Event(M, E) \equiv (\exists T)(Transition(M, T) \wedge Trigger(T, E))$$

3.2 Configuration of a Statechart

The current state of a statechart is defined as the *configuration* for the statechart at instant t. In [9] a configuration is defined as a maximal set of states that the system can be in simultaneously. Given a root state R, a configuration is a set of states $\mathcal{C}$ obeying the following rules [9]:

- $\mathcal{C}$ contains R.
- If $\mathcal{C}$ contains a state A of type OR, it must also contain exactly one of A's sub-states.
- If $\mathcal{C}$ contains a super-state A of type AND, it must also contain all of A's sub-sates.
- The only states in $\mathcal{C}$ are those that are required by the above rules.

When the system is in any state A, it must also be in A's parent state (unless A is the root state). Hence, to uniquely determine a configuration it is sufficient to know its basic states. In Section 4 we describe how a system evolves from one configuration to another.

3.3 History Connectors

Statecharts feature two kinds of history connectors: H and H^*. Suppose we are executing a transition t whose target is an H history connector h of state S. If S has history, then let S' be the substate of S which the system was in when most recently in S; t is treated as if its target is S'. If h is an H^* history connector, then let S' be the basic configuration relative to S which the system was in when it was most recently in S; t is treated as if its targets are all the states in S'. If the system was never in S, t is treated as if its target is S [9].

In this work we only consider the H type history connectors. Also, we use predicates to describe if a state has this connector. For example, $History(Hist, S)$ is true if $Hist$ is a history connector of state S[1].

4 Semantic Description

In the following discussion, we formalize the execution of a step as described in [9]. A step is initiated when an external event arrives, causing a cascade of subsequent internal events. A step is completed when no more internal events are generated or there are no more transitions triggered by the events that were generated, this means that the system has stabilized in a state. In our semantic we assume an asynchronous time model. In this time model, the system reacts whenever an external change occurs, allowing for several external changes to occur simultaneously. The execution of a step may be viewed as taking zero time as far as the environment is concerned, since during the execution of the step itself no external changes have any effect [9].

Suppose event E is generated by the environment. We should find out which transitions are enabled based on this event. The transitions in the step are executed in sequence, and the next event is considered after all transitions in the step have been fired.

First, we need to determine the transitions whose source state matches the current state configuration $\mathcal{C}$ (at instant t). We call these transitions *active* and we define the expression $Active(T)$ as

$$Active(T) \equiv (Source(T, S) \wedge \lceil S \rceil)$$

Given an event E we need to find those active transitions whose trigger event matches E, and whose guard condition evaluates to true. These transitions are *enabled* at instant t and are defined as:

$$Enabled(T, E) \equiv Event(T, E) \wedge Active(T) \wedge \lceil E \rceil^0 \wedge Guard(T)$$

Given a set of enabled transitions some of them may be in conflict. We can resolve the conflict with help of a priority relation. Before determining which transitions can be effectively be fired we need to describe the concepts related with *conflicting transitions*.

[1] For convenience, $Hist$ is also described as a state of the statechart. Thus, we can easily include a history connector as the target state of a transition.

Two transitions are in conflict if there is some common state that would be exited if any one of them were to be taken. Consider the example in Figure 3 taken from Harel's paper [9]. Transitions t_1 and t_2 are in conflict because they would each imply exiting state A. Also, t_4 is in conflict with all of t_1, t_2 and t_3, since if and when t_4 is taken, the system must have been in state E and thus also in one of its sub-states A, B or C. These conflicts are solved in two different ways. In the first case, if the triggers of both t_1 and t_2 are enabled at the beginning of a step, the system is faced with nondeterminism. In the second case, we should consider that priorities between transitions are determined outside-in; hence, t_4 has priority over t_1, t_2 and t_3.

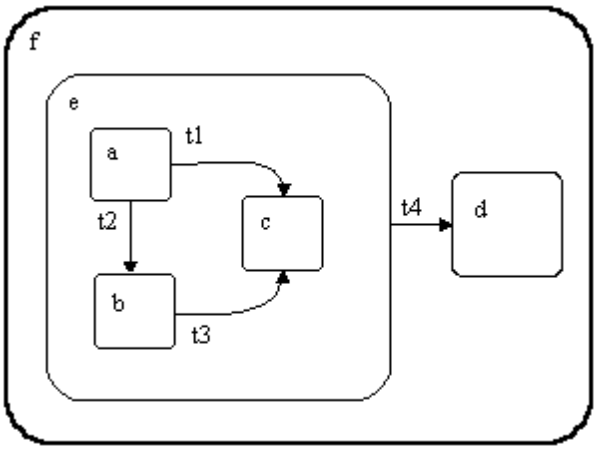

Fig. 3. Conflicting transitions

Given a statechart M, and two conflicting transitions T_i and T_j, T_j has priority over T_i iff the source state of T_j is a super-state that contains the source state of T_i:

$$Priority(T_j, T_i) \equiv Source(T_i, S_i) \wedge Source(T_j, S_j) \wedge SubSate(M, S_i, S_j))$$

Given two states X and S, we say that X is a sub-state of S iff the following holds:

$$SubSate(M, X, S) \equiv ((State_{CAND}(M, S) \wedge (S = \bigwedge_{k=1}^{n} (S_k) \wedge (\exists k \mid (X = S_k \vee SubSate(X, S_k)))) \vee (State_{COR}(M, S) \wedge (S = \bigvee_{k=1}^{n} (S_k) \wedge \bigwedge_{k \neq r} \neg(S_k \wedge S_r) \wedge (\exists k \mid (X = S_k \vee SubSate(X, S_k)))$$

Now we are in conditions to describe when a transition can be fired. Given a configuration C, a transition T can be fired at instant t iff it is enabled and there is no transition with greater priority:

$$Fire(T,E) \equiv Enabled(T,E) \wedge (\nexists T_i) Priority(T_i,T) \wedge$$
$$(\forall T' \mid Source(T',S) \wedge Enabled(T',E) \Rightarrow \neg Fire(T',E))$$

Notice that we have to deal with nondeterminism (see discussion about conflicting transitions). Each model of the previous formula represents a non-deterministic choice. If there is more than one transition enabled with the same source state, then only one should be fired. This is addressed in the last clause of the formula.

The previous definitions about the transitions that can be fired allow us to describe the statechart configuration that we obtain after executing a step. Let $\mathcal{C}$ and $\mathcal{C}'$ be configurations such that $\mathcal{C}'$ is reached by a step at time t from $\mathcal{C}$ after the execution of a sequence of *micro-steps*: $\mathcal{C}, \mathcal{C}_1, ...\mathcal{C}_i, ..., \mathcal{C}_n, \mathcal{C}'$, $n \geq 1$. Let $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ be configurations such that $\mathcal{C}_{i+1}$ is reached by a micro-step at time t from $\mathcal{C}_i$. The micro-step is defined as a set of transitions $\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_n$ that appear in different orthogonal states whose source states belong from the set of states that characterize configuration $\mathcal{C}_i$ and those transitions that can be fired. Then, a micro-step is initiated when an external event E arrives, and it may be defined as follows:

$$Micro(M,E) \equiv \forall T \mid (Source(T,S) \wedge Target(T,S') \wedge$$
$$Action(T,A) \wedge Fire(T,E)) \Rightarrow Fire(T,E); \lceil \neg S \wedge S' \wedge A \rceil^0$$

which states that all transitions that are fired allow the state changes. Also, the actions related with these transitions are generated as internal events. Note that all the generated internal events (actions) and all the new active states cause a subsequent micro-step.

To consider history connectors, we need to update the history of all parents of the exited states. Thus, we update the previous formula as follows:

$$Micro(M,E) \equiv \forall T \mid (Source(T,S) \wedge Target(T,S') \wedge$$
$$Action(T,A) \wedge Fire(T,E)) \Rightarrow Fire(T,E); \lceil \neg S \wedge S' \wedge A \rceil^0 \wedge$$
$$(\exists S_0 \mid SubState(S,S_0) \Rightarrow LastState(S_0,S)) \vee$$
$$(\nexists S_0 \mid SubState(S,S_0) \Rightarrow LastState(S,S))$$

where $LastState(S_0,S)$ states that S is the substate of S_0 which the system was in when most recently in S_0. If S is a root state, then we use $LastState(S,S)$.

A step is defined as the configuration that is reached when the system is stable. As mentioned below, a system is stable when no more internal events are generated:

$$Stable(M) \equiv (\forall T \mid Action(T,A)) \Rightarrow \lceil \neg A \rceil$$

Finally, the step from configuration C to C', initially fired by the external event E, is defined as:

$$Step(M,E) \equiv Micro(M,E) \wedge Stable(M)$$

Example. Based on the statechart in Figure 1 let $C = \{\mathtt{B}_1, \mathtt{C}_1\}$ be the current configuration. Suppose event e is generated by the environment. Let $\mathtt{E}$ be the state variable associated with e. Hence, we have

$$Step(m, e) \Rightarrow Micro(m, e) \wedge Stable(m)$$

and

$$\begin{gathered} Micro(m, e) \Rightarrow (Source(t_1, b_1) \wedge Target(t_1, b_2) \wedge \\ Action(t_1, f) \wedge Fire(t_1, e)) \Rightarrow \\ Fire(t_1, e); \lceil \neg \mathtt{B}_1 \wedge \mathtt{B}_2 \wedge \mathtt{F} \rceil^0 \wedge LastState(b_1, b_1)) \end{gathered}$$

The above formula implies $Micro(m, f)$:

$$\begin{gathered} Micro(m, f) \Rightarrow (Source(t_3, c_1) \wedge Target(t_3, c_2) \wedge \\ Action(t_3, g) \wedge Fire(t_3, f)) \Rightarrow \\ Fire(t_3, f); \lceil \neg \mathtt{C}_1 \wedge \mathtt{C}_2 \wedge \mathtt{G} \rceil^0 \wedge LastState(c_1, c_1)) \end{gathered}$$

Finally, the truth of the previous formula implies $Micro(m, g)$, and

$$\begin{gathered} Micro(m, g) \Rightarrow (Source(t_2, b_2) \wedge Target(t_2, c_3) \wedge \\ Action(t_2, g) \wedge Fire(t_2, g)) \Rightarrow \\ Fire(t_2, g) \frown \lceil \neg \mathtt{B}_2 \wedge \mathtt{B}_3 \wedge \mathtt{G} \rceil^0 \wedge LastState(b_3, b_3)) \end{gathered}$$

Since no more internal events are generated, the system is stable:

$$Stable(m) \Rightarrow (Action(t_1, f)) \Rightarrow \lceil \neg \mathtt{F} \rceil) \wedge (Action(t_3, g)) \Rightarrow \lceil \neg \mathtt{G} \rceil)$$

This formula completes our formalization of the execution of a step.

5 Conclusions

In this paper, we presented a new approach to formalize statecharts semantics, as described in [9], based on Duration Calculus.

The combination of statecharts with Duration Calculus allows designers to refine real time requirements and to link statecharts with Duration Calculus tools [13]. Given a statechart, which models a software or hardware system, we obtain a model in Duration Calculus. Then, for a given property specified as a Duration Calculus formula, we determine whether the system satisfies the formula. In particular, we are interested in relating safety analysis with specifications. This work allows to give a common semantic framework to different sources of information, such as Fault Trees and statecharts. In [16], Fault Trees have been given a Duration Calculus semantics. It may then be feasible to check the consistency of systematically derived properties from a Fault Tree against the specification using tools. In this sense, we consider this work as a fundamental step to integrate the analysis of components which may be described using different approaches.

Soundness or completeness results are not proved since we base our semantics on the rigurous but not formal definition given by Harel [9].

References

1. D. Drusinsky and D. Harel. Using Statecharts for Hardware description and Synthesis. *IEEE Transactions on Computer-Aided Design*, 8:798–807, 1989.
2. D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. Statecharts: A visual formalism for complex systems. *Sci. Comput. Prgram.*, 8:231–274, 1987.
3. David Harel and M. Politi. *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*. McGraw Hill, 1998.
4. G. Booch, J. Rumbauch, and I. Jacobson. *The Unified Modeling Language. User Guide*. Object Technology Series. Addison Wesley Longman, Reading, MA, USA, 1998.
5. D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Prgram.*, 8:231–274, 1987.
6. C. Huizing and W. P. de Roever. Introduction to design choices in the semantics of Statecharts. *Information Processing Letters*, 37(4):205–213, 1991.
7. A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In *Proceedings of the Symposium on Theoretical Aspects of Computer Software*, volume 526, pages 244–264. Springer-Verlag, 1991.
8. Andrew C. Uselton and Scott A. Smolka. A compositional semantics for statecharts using labeled transition systems. In *International Conference on Concurrency Theory*, pages 2–17, 1994.
9. D. Harel. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, October 1996.
10. Chaochen Zhou, C.A.R. Hoare, and Anders P. Ravn. A Calculus of Durations. *Information Proc. Letters*, 40(5):269–276, Dec. 1991.
11. A. P. Ravn, H. Rischel, and K. M. Hansen. Specifying and Verifying Requirements of Real-Time Systems. *IEEE Transactions on Software Engineering*, 19(1), January 1993.
12. E. R. Olderog, A. P. Ravn, and J. U. Skakkebaek. *Formal Methods in Real-Time Systems. Trends in Software Engineering*, chapter Refining Systems Requirements to Program Specifications. John Wiley and Sons, 1996.
13. Jens Ulrik Skakkebaek and Natarajan Shankar. Towards a duration calculus proof assistant in pvs. In *Third International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems, Lecture Notes in Computer Science, LNCS 863*, page 660.
14. Zhou Chaochen and Li Xiaoshan. *A Classical Mind. Essays in Honour of C.A.R. Hoare*, chapter A Mean Value Calculus of Durations, pages 431–451. Prentice-Hall, 1994.
15. C. A. Middelburg. Truth of duration calculus formulae in timed frames. In *1517*, page 22. Centrum voor Wiskunde en Informatica (CWI), ISSN 1386-369X, 31 1998.
16. K. M. Hansen, A. P. Ravn, and V. Stavridou. From Safety Analysis to Software Requirements. *IEEE Transactions on Software Engineering*, 24(7), July 1998.