Electronic Notes in Theoretical Computer Science 65 No. 5 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume65.html 15 pages

Modifying Esterel Concepts to Model Hybrid Systems [1]

Michael Baldamus [3]

Faculty for Informatics
University of Karlsruhe
Karlsruhe, Germany

Thomas Stauner [2,4]

BMW Car IT
Munich, Germany

Abstract

A hallmark of the Esterel language is the combination of perfect synchrony with total orthogonality and powerful constructs for preemption, suspension and trap handling. It is desirable to make this kind of expressiveness available for the description of hybrid systems, that is, systems whose evolution is understood in terms of segment–wise continuous functions over the real time axis. Our approach consists of modifying Esterel concepts, most notably by replacing the discrete time frame by a continuously advancing one. We are then able to state a semantics made up of transitions with closed execution intervals of non–zero length. By an instant we understand an execution interval within this framework. Hybrid signals may change their value during such an instant, non–hybrid ones, that is, classical signals immediately settle to a specific state and keep it the whole time. Time consumption still has to be specified explicitly, namely in that instants reflect jumps among control flow locations defined by **pause** statements; all other statements take no time in the sense that arbitrarily many of them may be sequentially executed regardless of the instant's duration. A transfer of perfect synchrony from the discrete to the continuous is in this way accomplished. We also consider an example, which is from the automotive domain, traces, bisimilarity and compositionality.

[1] This work was supported with funds of the Deutsche Forschungsgemeinschaft within the priority program Design and Design Methodology of Embedded Systems.

[2] During the preparation of this paper, Thomas Stauner was affiliated with the Faculty for Informatics of the Munich University of Technology.
[3] E–Mail: baldamus@ira.uka.de
[4] E–Mail: stauner@in.tum.de

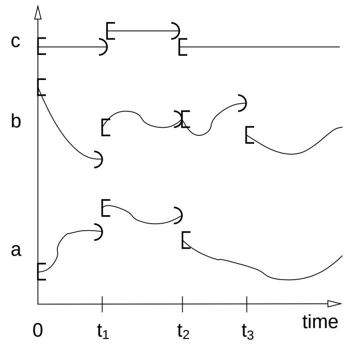

Figure 1. Example trajectories of a hybrid system.

1 Introduction

We speak of a *hybrid system* if variables may undergo continuous evolution interrupted by discontinuous changes at isolated time instants (see Figure 1). Typically hybrid systems are heterogeneous and consist of digital and analog components as well as of components that can not clearly be assigned to one of these two worlds at the beginning of the development process. Their mixed digital/analog nature makes hybrid systems an interdisciplinary topic mainly influenced by computer science and control theory. Demanding examples can, for instance, be found in the automotive domain. While switching dynamics is not new to control theory, the major challenge in hybrid systems is the increasing complexity of logical decision making within them. This situation calls for design methods and programming languages that help to manage complexity and aid in accomplishing reliability in the safety critical context in which hybrid systems often operate. As one step in this direction, we propose a modification of concepts underlying the Esterel programming language [4,3] for the description of hybrid systems.

Hybrid systems are very often best–viewed as real–time embedded systems, the application domain Esterel is primarily intended for. A hallmark of Esterel is the perfect synchrony paradigm — all time consumption has to be made explicit — in combination with total orthogonality and powerful constructs for preemption, suspension and trap handling. Thus it seems to be both natural and desirable to work with hybrid systems in an Esterel–like fashion. The main obstacle is semantic in nature, as the time model behind Esterel is discrete. It turns out, however, that a continuous model can be imposed on the just–mentioned Esterel–typical combination in a relatively canonical manner: What we essentially do is to augment a behavioural semantics with execution intervals of non–zero length. By an instant we understand an interval of this kind. Hybrid signals may evolve continuously during such an instant, non–hybrid ones, that is, classical signals immediately settle to a specific state and keep it until the instant is over. The mixed continuous/discontinuous evolution of hybrid systems exemplified in Figure 1 is thus nicely reflected. Also, time consumption still has to be specified explicitly in that instants reflect jumps to and from control flow locations defined by **pause** statements; all statements in between take no time in the sense that arbitrarily many of them may be sequentially executed regardless of the duration of the instant. Perfect synchrony is in this way indeed transferred from the discrete to the continuous, where downward compatibility is cared for: If one refrains from both using hybrid signals and mentioning continuous time parameters, then what is left behaves in the familiar way, except that the semantics necessarily assigns non–deterministic execution intervals (cf. Section 7).

The overall formalism is rather as a modelling and specification framework than a programming language. That has to do with a new statement, which is of the form **now** «*expression*». Its semantics consists of adjoining the valuation of the expression as an execution condition to every transition whose control flow touches the statement. Such a feature seems to be necessary for being able to describe the evolution of hybrid signals through differential equations (see Section 3); its disadvantage consists of the fact that it is somewhat like an **if** without an **else**, even an implicit one, so that it is possible to write code that is not reactive. The guarantee of reactivity, however, is sometimes considered important in programming embedded systems. This aspect is open for further research. For the moment, we take comfort in the fact that Esterel's guarantees of reactivity and determinism have already been weakened as part of the design of the synchronous modelling and specification language QUARTZ [10]. One might use the name of HYPERQUARTZ for that what is presented here.

The rest of the paper is organised as follows: Section 2 recalls a number of Esterel constructs, which serve as a base of all subsequent sections; Section 3 introduces most of the concepts that make up that what is proposed here; Section 4 is devoted to discussing the most important design decisions behind those concepts; Section 5 completes the picture by introducing the **earliest** qualifier, a hybrid counterpart to the **immediate** qualifier; Section 6 is concerned with an example derived from an electronic car chassis height control; Section 7 presents the most important axioms and rules of the behavioural semantics of the language; Section 8 is concerned with traces, bisimulation and compositionality issues on top of the behavioural semantics; Section 9 provides a brief conclusion for the paper.

Related work.

In the past a number of description techniques for hybrid systems have been proposed. We only want to mention some of the most important pointers to the literature. The interesting common property of the chosen approaches is that they are all based on a global time scale and assume zero reaction time for discrete transitions in the model. In this respect these formalisms rely on basic ideas present in synchronous languages.

The most widespread formalism probably are hybrid automata [2]. Their purpose is the description and automatic verification (of a subclass) of hybrid systems. However, applications suffer from a lack of compositionality of the formalism [9]. Besides that, the verification of hybrid automata suffers from the immanent complexity of the problem. A textual, compositional description technique for hybrid systems, which builds on I/O automata, is proposed by [7]. Verification is a manual activity within this framework.

$stmt$ ::= $stmt_0$ ['||' $stmt$]
$stmt_0$ ::= $stmt_1$ [';' $stmt_0$]
$stmt_1$ ::= '**nothing**' | '**emit**' a | a ':=' $expr$ | '**pause**'
| '**abort**' $stmt$ '**when**' $expr$ '**end**'
| '**weak**' '**abort**' $stmt$ '**when**' ['**immediate**'] $expr$ '**end**'
| '**suspend**' $stmt$ '**when**' ['**immediate**'] $expr$ '**end**'
| '**signal**' a '**in**' $stmt$ '**end**' | '**if**' $expr$ '**then**' $stmt$ '**else**' $stmt$ '**end**'
| '**loop**' $stmt$ '**until**' $expr$ | '[' $stmt$ ']'

Figure 2. Grammar of the core base language. Here and everywhere else, terminals appear within single quotes, and angular brackets indicate optional syntactic components.

Charon [1] and HyCharts [5] support the formal specification of the architecture and behavior of hybrid systems in the style of ROOM [11] and the UML-RT [12]. The *Masaccio* formalism supports the specification of a hybrid system by free nesting of parallel composition and disjunctive composition of atomic discrete and continuous components [6]. In practice the MATLAB/Simulink/Stateflow simulation environment is popular [14]. However, control laws cannot directly be associated with discrete modes in this environment and no formal semantics exists.

2 A Base Language

A grammar of the minimalist core base language employed in the sequel is shown in Figure 2. It is parameterised by a set of signals ranged over by letters from the beginning of the alphabet and their usual notation derivatives. Each individual construct except **until** occurs in Esterel, the informal meaning being the same. Our **if**, however, is somewhat different in the sense that we permit any arbitrary condition over pure and valued signals, where pure signals may occur like variables of type **boolean**. As for types and expressions, we assume that every signal is uniquely associated with one of the types **float**, **integer** and **boolean**. The usual operators over these types are permitted and everything must be well–typed in the expected way.

We employ a number of additional derived constructs, partly occurring in Esterel as well, which resolve to core constructs in canonical ways; see Figure 3. Further constructs such as temporal loops can be made available in the same way.

Modules and local valued signals are the main omissions from the base language. In their cases the formal semantics presented in Section 7 requires additional machinery that would go beyond the space limitations applying to the present paper. The base language as it is suffices as background for explaining every essential aspect of our concepts.

$$
\begin{aligned}
\textit{stmt}_1 ::= \; & \text{‘}\textbf{abort}\text{’}\ \textit{stmt}\ \text{‘}\textbf{when}\text{’}\ \text{‘}\textbf{immediate}\text{’}\ \textit{expr}\ \text{‘}\textbf{end}\text{’} \\
\mid \; & \text{‘}\textbf{await}\text{’}\ [\text{‘}\textbf{immediate}\text{’}]\ \textit{expr} \mid \text{‘}\textbf{sustain}\text{’}\ a \\
\mid \; & \text{‘}\textbf{signal}\text{’}\ a\ [\text{‘,’}\ a]^*\ \text{‘}\textbf{in}\text{’}\ \textit{stmt}\ \text{‘}\textbf{end}\text{’} \mid \text{‘}\textbf{if}\text{’}\ \textit{expr}\ \text{‘}\textbf{then}\text{’}\ \textit{stmt}\ \text{‘}\textbf{end}\text{’} \\
\mid \; & \text{‘}\textbf{while}\text{’}\ \textit{expr}\ \text{‘}\textbf{loop}\text{’}\ \textit{stmt}\ \text{‘}\textbf{end}\text{’} \mid \text{‘}\textbf{loop}\text{’}\ \textit{stmt}\ \text{‘}\textbf{end}\text{’}
\end{aligned}
$$

$$\textbf{abort}\ \textit{stmt}\ \textbf{when immediate}\ \textit{expr}\ \textbf{end} \longrightarrow \begin{array}{l}\textbf{if not}\ \textit{expr}\ \textbf{then} \\ \textbf{abort}\ \textit{stmt}\ \textbf{when}\ \textit{expr}\ \textbf{end}\end{array}$$

$$\textbf{await}\ \textit{expr} \longrightarrow \textbf{loop pause until}\ \textit{expr}$$

$$\textbf{await immediate}\ \textit{expr} \longrightarrow \textbf{while not}\ \textit{expr}\ \textbf{loop pause end}$$

$$\textbf{sustain}\ a \longrightarrow \textbf{loop emit}\ a;\ \textbf{pause end}$$

$$\textbf{signal}\ a_1,\ a_2,\ \ldots,\ a_k\ \textbf{in}\ \textit{stmt}\ \textbf{end} \longrightarrow \begin{array}{l}\textbf{signal}\ a_1\ \textbf{in signal}\ \ a_2,\ \ldots,\ a_k \\ \textbf{in}\ \textit{stmt}\ \textbf{end end}\end{array}$$

$$\textbf{if}\ \textit{expr}\ \textbf{then}\ \textit{stmt}\ \textbf{end} \longrightarrow \textbf{if}\ \textit{expr}\ \textbf{then}\ \textit{stmt}\ \textbf{else nothing end}$$

$$\textbf{while}\ \textit{expr}\ \textbf{loop}\ \textit{stmt}\ \textbf{end} \longrightarrow \textbf{if}\ \textit{expr}\ \textbf{then loop}\ \textit{stmt}\ \textbf{until not}\ \textit{expr}\ \textbf{end}$$

$$\textbf{loop}\ \textit{stmt}\ \textbf{end} \longrightarrow \textbf{while true loop}\ \textit{stmt}\ \textbf{end}$$

Figure 3. Derived base language constructs and rewriting rules for resolving them. The form [...]∗ occurring in the grammar part indicates zero or more occurrences of the phrase in between the brackets.

3 Continuously Advancing Global Time and Hybrid Signals

Our most fundamental modification to the Esterel paradigm consists of replacing the discrete by a continuous global time frame. Hence, there no longer is a discrete clock "tick" anymore but rather a continuously advancing clock. All concurrent activity is kept synchronous in this way. It does not, however, proceed in "lockstep" anymore since the step concept is very different now.

The second essential modification is that hybrid value signals are introduced. These signals are always of type **float** and evolve as functions over the continuous time axis (signals a and b in Fig. 1). Classical value signals are also regarded as such functions, but they must be segment–wise constant (signal c in Fig. 1), where those segments must begin and end at instant boundaries. Even the usual pure signals are understood in this way, this time with the boolean truth values as range: true means present, false means absent.

As in Esterel, time consumption has to be programmed explicitly. We have chosen the **pause** statement as the only time–consuming one. We extend it by an invariant, syntactically **pause** *expr*. Semantically **pause** *expr* permits any non-zero delay during which the invariant specified by the expression is true. This is similar to time steps in hybrid automata: **pause** is non-deterministic with respect to how much time it takes. Obviously the longest delay for which the invariant is true is also permitted by **pause**. This delay can be enforced by iterating the execution of **pause** in a suitable way (see below). If **pause** occurs without a parameter, such as it was introduced in Section 2, then it is equivalent to **pause true**. Loop bodies must still be non–instantaneous.

Our execution model is such that control flow states are seen as sets of labels of **pause** statements. Transitions are tied to execution intervals of non–zero length, reflecting the abandoning of discrete, logical time in favour of real time. They are in general made up of several concurrent transitions at different locations of the respective system. Such constituent transitions have the same execution interval as the respective compound one. Every one involves the time consumption of a specific **pause** statement, which means that a (compound) transition synchronises the execution intervals of all **pause**s reached by it. This aspect goes hand in hand with the above–described non–determinism associated with **pause**. Also, (compound) transitions define what is understood by an instant with respect to non–hybrid valued signals and pure signals.

Here is an example in which a signal is emitted at the first moment a time dependent condition becomes true:

$$\begin{array}{l} \textbf{while not } \mathit{expr}\{\mathrm{t} := \mathrm{l}\} \textbf{ loop} \\ \quad \textbf{pause forall } \mathrm{t}{:}[\mathrm{l},\mathrm{u}[.\textbf{not } \mathit{expr} \\ \textbf{end}; \\ \textbf{emit } a \end{array} \tag{1}$$

The condition is assumed to appear in the form of the expression *expr*, where the reserved identifier t denotes the time parameter. We explain the example in an inside–out fashion. First of all, the quantification involved in the parameter of **pause** is another piece of machinery that we have put into place to capture hybrid systems. We call it a *generic quantification* because its range, which is specified by the construct [l,u[, refers to every execution of **pause**. Specifically, the reserved identifier l refers to the lower bound of any particular execution interval, and the reserved identifier u to its upper bound. The construct [l,u[thus specifies the entire interval except its last moment. In consequence, every execution of the **pause** statement is such that the condition does not hold except at possibly the last moment of the execution interval. Furthermore, the **while** condition specifies that the loop is entered if the condition does not hold at the beginning of the execution interval. This detail, together with the semantics of the loop body, entails that the control flow reaches the loop exit precisely at the first moment where the condition comes true. This point of time is also the beginning of the subsequent transition, which is the one during which the signal is emitted.

The dependency of hybrid valued signals on real time can be specified by means of assignments whose respective right hand side contains other hybrid valued signals and/or the already mentioned identifier t. Also, hybrid signals can be specified to be constant during specific execution intervals by means of assignments whose respective right hand side contains neither of those entities. For two hybrid valued signals i and o, examples are $o = i$ — o evolves like i — and $o = 0$ — o is constantly 0. Another possibility of specifying the dependency of hybrid valued signals on real time consists of referring to their first derivation. The following snippet uses this feature to filter an input signal u by means of the differential equation $\dot{f} = \frac{1}{T}(u - f)$.

$stmt_1$ ::= ‘**pause**’ *expr* | *a* ‘(’ *expr* ‘)’ ‘:=’ *expr* | ‘**now**’ *expr*

Figure 4. Constructs specifically for modelling hybrid systems.

The initialisation for the output *f* is 0 and *T* is a constant. Furthermore, *f* may be reset to the current value of *p* by the input signal *r*:

```
f(l) := 0;
loop
    now change(f) = (1/T) * (u- f);            (2)
    pause true;
    if r then f(l) := p end
end
```

The value of *f* when the code snippet is entered is determined by the assignment to *f*(l); the subsequent course of the output is then determined by the **now**–statement, which refers to the first derivation of the output by means of the **change**–operator. As mentioned in the introduction, the **now**–statement adjoins the parameter predicate as an additional assumption to all transitions whose control flow touches such a statement. Also, there is another yet un–mentioned semantic feature at work: The last and the first values of an hybrid signal in successive execution intervals are the same unless the second interval contains an assignment to the signal. An example of such an evolution is the trajectory of signal *b* at t_2 in Figure 1.

Before we finish this part, we point out that conditions have to be generically quantified unless it is possible to qualify the specific condition with the **earliest** directive explained in the next section. If both any quantification and the directive are missing, then an implicit quantification of the form **forall** t:[l,u] is assumed. In any case hybrid signals and their derivations are considered functions over execution intervals whose higher–order occurrences within conditions are broken down to first–order form by appending (t). The statement **now change**(*f*) = (1/*T*) * (*u*-*f*), which contains higher–order occurrences of the hybrid value signals *f* and *u*, and of the derivation **change**(*f*), thus has to be read as: **now forall** t:[l,u].**change**(*f*)(t) = (1/*T*) * (*u*(t) - *f*(t)). The function character of hybrid signals is also the reason why assignments such as *f*(l) := 0 make sense. They assign a value at a specific point of time, which is determined by the argument on the left hand side. It must be a constant expression over the reserved identifiers l and u, whose value lies within the interval [l,u].

Figure 4 summarises the syntactic features newly introduced thus far.

4 Discussion of Some of the Design Decisions

Before continuing with additional machinery, we discuss some of the design decisions behind that what has been presented in Section 3. First of all, it does not seem viable to understand **pause** *expr* as deterministic in the sense that the statement should terminate when the condition represented by the expression becomes false. The main reason has to do with the orthogonal nature of the language. Specifically, what should be the semantics of something such as

$$a := expr_0;$$
$$[\textbf{pause}\ expr_1 \parallel \textbf{pause}\ expr_2]?$$

Here, an assignment is followed by the parallel composition of two **pause**s whose parameters may in general be different. The question over what time span the assignment should take effect arises whenever $expr_1$ and $expr_2$ stand for conditions of different durations. Solutions to this problem may be possible. There seems to be an un–avoidable element of arbitrariness if one adopts any such approach, however. The technicalities become complicated in all cases.

It seems as if practical examples are more like the signal filter above or the car chassis height control component in Section 6. In all of these cases, it is no problem to accomplish deterministic behaviour by means of the features of the language.

Then, time invariance holds whenever shifting the input along the time axis simply leads to an equal shift of the output. This property of a hybrid systems is practically always desired. One could even go as far as demanding that the formalism itself should guarantee it. We have, nevertheless, not chosen to provide specific support for time invariance at the moment. The reason is that such a feature would probably consist of syntactic restrictions whereas the present work is concerned with the foundations of modifying Esterel concepts to model hybrid systems. Our interest thus lies more in keeping our framework as clean, simple and powerful as possible. We do not foresee any serious difficulty in drawing up a set of restrictions that ensures both time invariance and a sufficiently powerful set of descriptive tools to work with.

Finally, we have chosen to regard both hybrid and classical signals as segment–wise constant functions over the continuous time axis. Another sensible choice would have consisted of simply recording the states of classical signals during any specific instant. We have not opted for this possibility to keep the whole framework more uniform. Uniformity, then, is especially advantageous with respect to considerations on top of the formal semantics, such as defining traces and bisimulation.

5 The earliest Directive

We expect that a very common requirement in using our formalism will consist of ensuring that a condition, once it occurs, will lie at the boundaries of two consecutive instants, so that an immediate reaction to the event can occur during the second one. For this purpose, we introduce three derived constructs, which are supposed to cover most of these situations. There is an **await earliest**, a [**weak**] **abort** . . . **earliest**, and a **suspend** . . . **earliest**; see Figure 5. Each construct makes sense only if the *expr*–component contains the identifier t in unquantified form, although the semantics is always technically well–defined if it does not. As for **await**

$stmt_1$ ::= '**await**' '**earliest**' *expr*
| ['**weak**'] '**abort**' *stmt* '**when**' '**earliest**' *expr* '**end**'
| '**suspend**' *stmt* '**when**' '**earliest**' *expr* '**end**'

Figure 5. Constructs with **earliest**. Like the ones shown in Figure 4, these constructs are introduced specifically for modelling hybrid systems.

earliest, the intuition is that an instant in which it is left is such that the condition holds right at the beginning of the instant, and that this point of time is also the... earliest one in which the condition holds after the control flow has reached the **await**. In particular the instant in which the **await earliest** is entered may be the same as the one in which it is left. The situation is similar whenever a **abort** ...**earliest** is left through abortion, or if and when a **suspend** ...**earliest** comes into effect: The corresponding instant is again such that condition holds right at its beginning, and this point of time is again the earliest one in which the condition holds. We have chosen the word **earliest** for all of that since the similarity to the **immediate** qualifier is obvious.

All constructs with **earliest** can be resolved in a way so that the intended effect is accomplished by the means introduced in Sections 2 and 3; see Figure 6. The resolution of **await earliest** is actually given by the **while** loop from Code Snippet (1) from Section 3; the resolution of [**weak**] **abort** ...**earliest** is such that the statement is placed inside a [**weak**] **abort** ...**immediate** containing an **await earliest** whose task consists of triggering the preemption at the earliest possible moment. The preemption is also triggered once the statement terminates to ensure proper termination of the entire setup. This scheme evidently constitutes a case of causal circularity, as the preemption is triggered from inside. We nevertheless stick to the resolution of [**weak**] **abort** ...**earliest** given in Figure 6. The reason is twofold: First, the semantics presented in Section 7 is well–defined in the presence causal circularity and, at the same time, the meaning it assigns to the resolution of [**weak**] **abort** ...**earliest** is exactly the one that we want to see as the meaning of [**weak**] **abort** ...**earliest** itself. Second, one can always refrain from causal circularity in using the language. Here we are working on the semantic level, that is, one level below that.

These general remarks also apply to the resolution of **suspend**...**earliest**. The scheme itself is somewhat similar to the one used to resolve **abort**...**earliest**. It is obvious, however, that the **await earliest** must occurs inside a loop since it must be possibly to re–suspend. This loop, in turn, is aborted once the statement terminates. A complication consists of the fact that the statement must not terminate if and when it is actually to be suspended. To accommodate this requirement, two distinct signals are used, a for signaling suspension and b for signaling termination, and the loop is preempted on **not** a **and** b.

Traps can also be introduced by relying on causal–circularity. Every **trap** statement is mapped to a **weak abort** ...**immediate** with an enclosing declaration of each trap as a local signal; every statement of the form **exit** a is mapped to **emit** a; **pause**. Exiting a trap then becomes triggering a **weak abort** from within.

```
    await earliest expr
            ↓
while not expr{t := l} loop
    pause forall t:[l,u[.not expr
end

    [weak] abort
        stmt
    when earliest expr end
            ↓
signal a /* fresh */ in
    [weak] abort
        await earliest expr;
        emit a
    ||
        stmt;
        emit a
    when immediate a end
end
```

```
    suspend
        stmt
    when earliest expr end
            ↓
signal a, b /* distinct and fresh */ in
    abort
        loop
            await earliest expr;
            emit a;
            pause;
        end
    when immediate not a and b
||
    suspend
        stmt;
        emit b
    when immediate a
end
```

Figure 6. Rewriting rules for resolving constructs with **earliest**.

Hierarchies of nested trap statements are taken care of by the fact that nested **weak abort**s are hierarchical in the same way. The framework of the present paper, however, supports this entire scheme only for non–valued traps; the reason is that we have not introduced local valued signals.

6 Example

As an example of using our formalism we consider a component of an Electronic Height Control System (EHC) as presented in [13]. The purpose of the EHC system is to adjust the chassis level of a car according to the drivers wishes and according to different driving situations. One component of the system, the control mode automaton, is therefore responsible for determining the driving situation and providing it to other components. We distinguish four modes: driving, standing, engineOff and bend. These modes are hierarchically nested. When the car is travelling through a curve, the active mode is bend. If the car is not in a curve, one of the three other modes is active. If the engine is off mode engineOff is active. Only if the car is not in a bend and the engine is not off, the current velocity is sensed and depending on it the mode is either standing or driving. In particular, switching the engine off immediately causes a change in the control mode to engineOff, irrespective of the velocity. Mode bend has even higher priority than engineOff, that is, the mode changes to bend whenever the car is in a curve irrespective of velocity and engine status. All of this is neatly expressed using nested **suspend**s by the code shown in Figure 7. We make use of a few self–explaining syntactic constructs

```
type ControlMode = enum bend, engineOff, standing, driving end
module ControlModeAutomaton
(in bend, engineOff; in velocity: hybrid float; out controlMode: ControlMode):
begin
    suspend
        suspend
            loop
                while velocity(l) = 0 loop
                    controlMode := controlMode.standing;
                    pause forall t:[l,u[.velocity(t) = 0
                end;
                while not velocity(l) = 0 loop
                    controlMode := controlMode.driving;
                    pause forall t:[l,u[.not velocity(t) = 0
                end
            end
        when immediate engineOff end
    ||
        loop
            if engineOff then controlMode := ControlMode.engineOff end
        end
    when immediate bend end
||
    loop
        if bend then controlMode := ControlMode.bend end
    end
end ControlModeAutomaton.
```

Figure 7. Determining the mode of an electronic car chassis height control.

not introduced so far to give an impression of what practical applications of our formalism could look like. The pure input signals are supposed to be present iff the car is in a curve or the engine is off; the hybrid input signal carries the velocity.

Code Snippet 2 in Section 3 is the body of another module from the EHC. Its purpose is to smooth the chassis level as measured by a sensor. The output signal contains less noise and is used for further control decisions.

7 Behavioural Semantics

The formal semantics we give next is a behavioural semantics in the sense that it consists of *transitions* of the form

$$L \xrightarrow[\beta]{\alpha}_{[l,u]} L',$$

where L is a set of *start labels*, L is a set of *terminal labels*, $[l, u]$ is the *execution interval* of the transition, meaning that l and u are non–negative reals with $l < u$, and α and β are disjoint assignments over $[l, u]$. We call α the *emitted assignment* and β the *assumption* of the transition. Labels are assumed to be uniquely associated with **pause**–statements, so transitions indeed describe jumps among control flow locations. The emitted assignment describes what signals are emitted or provided with a value by the transition; the assumption is an environment of emitted and valued signals in which the transition can take place.

We need to introduce assignments and a few additional technical notions. To begin with, signals have *trajectories*. Specifically, if a is a signal, then a trajectory of a is a function over a closed interval over the non–negative reals into the range of a. — If a is a pure signal, then $\top$ — true — and $\bot$ — false — are understood to be a's range. The domain of a trajectory is called the *trajectory interval*. An *assignment* to a set A of signals is a function on A that yields a trajectory of a for each $a \in A$. We only consider such assignments where all trajectory intervals are pairwise the same. The common trajectory intervals of all trajectories of an assignment is called the *assignment interval*. We also speak of an assignment *over* some interval. Two trajectory assignments are *compatible* if their intervals are the same and if they agree on the intersection of their domains; two assignments are *disjoint* if their domains are disjoint, which means that disjointness implies compatibility. The compatibility of two assignments α and β is denoted by $\alpha \bowtie \beta$. Assignments, and indeed all functions occurring in the present paper, are actually understood in the set–theoretic sense, that is, as sets of argument/value pairs. This allows one to treat assignments set–theoretically. In consequence, we can make use of assertions such as $\alpha \subseteq \beta$, which means that $\mathrm{dom}(\alpha) \subseteq \mathrm{dom}(\beta)$ and that $\alpha(a) = \beta(a)$ for each $a \in \mathrm{dom}(\alpha)$.

The behavioural semantics, then, is given by SOS–style axioms and rules over *judgments* of the form

$$\text{«construct»} \vdash \text{«transition»}.$$

Such an assertion means that the transition belongs to the set of all transitions of the construct. Due to space limitations, we can not present all axioms and rules but only the most important ones; see Figure 8. Of note, the system associates durations with transitions in a bottom up fashion already if they do not touch any **pause** statement. This aspect is purely technical. It does not mean that any other statement than **pause** is time consuming. The system also follows through on our claim of downward compatibility, which means that all features already present in Esterel behave in the familiar way, up to the fact that they operate on instants of varying durations.

The first three axioms make use of meta–level lambda notation of the form

$$\lambda \,\text{«lambda variable»: «range of lambda variable». «valuation»}$$

to specify assignments. Whenever the valuation is constant, that is, whenever it does not depend on the λ–variable, then the variable is denoted by an underscore. A *value assignment* to a set A of signals is a function on A that yields an element of the range of a for each $a \in A$. If α is a trajectory assignment over some interval $[l, u]$ and $t \in [l, u]$, then we denote by $\alpha(t)$ the value assignment given by $\mathrm{dom}(\alpha(t)) = \mathrm{dom}(\alpha)$ and $\alpha(t)(a) = \alpha(a)(t)$ for each $a \in \mathrm{dom}(\alpha(t))$. The evaluation of some expression *expr* with respect to some value assignment α is denoted by $[\![expr]\!]_\alpha$. It is always understood that $\mathrm{dom}(\alpha)$ must encompass the set of free variables of the expression. If *expr* does not depend on hybrid valued signals, then we denote by $[\![expr]\!]_\alpha$ the evaluation of *expr* with respect to $\alpha(t)$ for any $t \in [l, u]$, where $[l, u]$ is the assignment interval of α. This kind of evaluation is well defined because only the trajectories of hybrid valued signals need not be constant. The update and/or extension of a value assignment α at mutually distinct variables $x_1, \ldots, x_k$ with values $v_1, \ldots, v_k$ is denoted by $\alpha[x_1 := v_1, \ldots, x_k := v_k]$. We also use updates of the same form in connection with trajectory assignments, given that no hybrid valued signals are updated. The meaning of such updates is as expected. Finally, $[l, u]$ stands for $[\mathrm{l} := l, \mathrm{u} := u]$ and $[l, t, u]$ for $[\mathrm{l} := l, \mathrm{t} := t, \mathrm{u} := u]$.

8 Traces, Bisimulation and Compositionality

An important aspect of the formal semantics of our formalism is not covered by the behavioural semantics: The effect of assignments beyond the instants in which they occur. A classical valued signal keeps its value until a new assignment to it occurs; a hybrid value signal continues to evolve until there is a new assignment to it, according to differential equations or other types of predicates injected into the control flow by [**weak**] **abort**, **suspend**, **if**, **loop** … **until** or **now**, or freely if their are no such constraints. Also, the terminal value of a hybrid signal during some instant is then the same as its initial value during the following one. Formalising these semantic properties is somewhat tedious and, at the same time, not really instructive beyond the informal explanation just given. For this reason, we skip it.

We continue with the question of compositionality. This issue has posed problems within other hybrid frameworks such as that of hybrid automata [9]. As for the one we are presenting here, we propose to consider it on the basis of bisimilarity. Esterel and, in consequence, our formalism are related to process algebras such as Synchronous CCS (SCCS, [8]), so this approach seems to make sense.

Definition 8.1 (Bisimilarity) Let $stmt_1$ and $stmt_2$ be two statements of the core language. Furthermore, let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the sets of unique **pause** labels of $stmt_1$ and $stmt_2$. Then a binary relation $\mathcal{R}$ on finite subsets of $\mathcal{L}_1$ and $\mathcal{L}_2$ is a *bisimulation* if $L_1 \ \mathcal{R}\ L_2$ implies: (i) $L_1 = \emptyset$ iff $L_2 = \emptyset$. (ii) Whenever $stmt_1 \vdash L_1 \xrightarrow[\beta]{\alpha}_{[l,u]} L_1'$, then $stmt_2 \vdash L_2 \xrightarrow[\beta]{\alpha}_{[l,u]} L_2'$ for some L_2' so that $L_1' \ \mathcal{R}\ L_2'$. (iii) Whenever $stmt_2 \vdash L_2 \xrightarrow[\beta]{\alpha}_{[l,u]} L_2'$, then $stmt_1 \vdash L_1 \xrightarrow[\beta]{\alpha}_{[l,u]} L_1'$ for some L_1' so that $L_1' \ \mathcal{R}\ L_2'$. — We write $stmt_1 \sim stmt_2$ if there is a bisimulation over $\mathcal{L}_1$ and $\mathcal{L}_2$.

$$\textsf{\textbf{emit}}\ a \ \vdash \emptyset \xrightarrow[\emptyset]{\lambda\_:\{a\}.\lambda\_:[l,u].\top}_{[l,u]} \emptyset$$

$$a \ := \ expr \ \vdash \emptyset \xrightarrow[\beta]{\lambda\_:\{a\}.\lambda\_:[l,u].[\![expr]\!]_{\beta[l,u]}}_{[l,u]} \emptyset \quad \text{provided that } a \text{ is not a hybrid valued signal}$$

$$a \ := \ expr \ \vdash \emptyset \xrightarrow[\beta]{\lambda\_:\{a\}.\lambda t:[l,u].[\![expr]\!]_{\beta(t)[l,t,u]}}_{[l,u]} \emptyset \quad \text{provided that } a \text{ is a hybrid valued signal}$$

$$a(expr_1) := expr_2 \vdash \emptyset \xrightarrow[\beta]{\lambda\_:\{a\}.\theta}_{[l,u]} \emptyset \quad \text{provided that } \theta \in ([l,u] \longrightarrow \mathbb{R}) \text{ with } \theta(t) = [\![expr_2]\!]_{\beta(t)[l,t,u]}, \text{ where } t = [\![expr_1]\!]_{\emptyset[l,u]}$$

$$\ell : \ \textsf{\textbf{pause}}\ expr \ \vdash \emptyset \xrightarrow[\beta]{\emptyset}_{[l,u]} \{\ell\} \quad \text{provided that } [\![expr]\!]_{\beta[l,u]} = \top$$

$$\ell : \ \textsf{\textbf{pause}}\ expr \ \vdash \{\ell\} \xrightarrow[\emptyset]{\emptyset}_{[l,u]} \emptyset$$

$$\textsf{\textbf{now}}\ expr \ \vdash \emptyset \xrightarrow[\beta]{\emptyset}_{[l,u]} \emptyset \quad \text{provided that } [\![expr]\!]_{\beta[l,u]} = \top$$

$$\frac{stmt_1 \ \vdash L \xrightarrow[\beta_1]{\alpha_1}_{[l,u]} \emptyset \quad stmt_0 \ \vdash \emptyset \xrightarrow[\beta_2]{\alpha_2}_{[l,u]} L'}{stmt_1;\ stmt_0 \vdash L \xrightarrow[(\beta_1\setminus\alpha_2)\cup(\beta_2\setminus\alpha_1)]{\alpha_1\cup\alpha_2}_{[l,u]} L'} \quad \text{provided that } \alpha_1 \bowtie \alpha_2,\ \beta_1 \bowtie \alpha_2 \text{ and } \beta_2 \bowtie \alpha_1,\ \beta_1 \setminus \alpha_2 \bowtie \beta_2 \setminus \alpha_1$$

$$\frac{stmt_1 \ \vdash L \xrightarrow[\beta]{\alpha}_{[l,u]} L'}{stmt_1;\ stmt_0 \ \vdash L \xrightarrow[\beta]{\alpha}_{[l,u]} L'} \quad \text{provided that } L' \neq \emptyset \qquad \frac{stmt_0 \ \vdash L \xrightarrow[\beta]{\alpha}_{[l,u]} L'}{stmt_1;\ stmt_0 \ \vdash L \xrightarrow[\beta]{\alpha}_{[l,u]} L'} \quad \text{provided that } L \neq \emptyset$$

$$\frac{stmt_0 \ \vdash L_1 \xrightarrow[\beta_1]{\alpha_1}_{[l,u]} L_1' \quad stmt \ \vdash L_2 \xrightarrow[\beta_2]{\alpha_2}_{[l,u]} L_2'}{stmt_0 \parallel stmt \vdash L_1 \cup L_2' \xrightarrow[(\beta_1\setminus\alpha_2)\cup(\beta_2\setminus\alpha_1)]{\alpha_1\cup\alpha_2}_{[l,u]} L_1' \cup L_2'} \quad \text{provided that } \alpha_1 \bowtie \alpha_2,\ \beta_1 \bowtie \alpha_2,\ \beta_2 \bowtie \alpha_1 \text{ and } \beta_1 \setminus \alpha_2 \bowtie \beta_2 \setminus \alpha_1$$

$$\frac{stmt_0 \ \vdash L \xrightarrow[\beta]{\alpha}_{[l,u]} L'}{stmt_0 \parallel stmt \ \vdash L \xrightarrow[\beta]{\alpha}_{[l,u]} L'} \qquad \frac{stmt \ \vdash L \xrightarrow[\beta]{\alpha}_{[l,u]} L'}{stmt_0 \parallel stmt \ \vdash L \xrightarrow[\beta]{\alpha}_{[l,u]} L'} \quad \text{provided that } L \neq \emptyset \text{ in both cases}$$

Figure 8. SOS–style axioms and rules.

Compositionality with respect to bisimilarity is usually understood as congruence. This property, in turn, means that bisimilarity is preserved by all constructors of the language under consideration.

Proposition 8.2 *(Compositionality as Congruence) It is indeed the case that the notion of bisimilarity from Definition 8.1 is compositional in this sense. For example,* $\sigma_1 \sim \sigma_2$ *and* $\tau_1 \sim \tau_2$ *implies* $\sigma_1 \parallel \tau_1 \sim \sigma_2 \parallel \tau_2$.

Proof. (Idea) Straightforward but tedious, using standard methods of proving congruence properties of bisimulation. □

9 Conclusion

The basis for the work presented here is the Esterel–typical combination of perfect synchrony, orthogonality and powerful constructs for preemption, suspension and trap handling. We have found that it is indeed possible to integrate these concepts cleanly and sensibly with a continuously advancing global clock and hybrid signals. Important ingredients on the language level are a parameterised **pause** statement with non–deterministic time consumption, a **now** statement for describing the evolution of hybrid signals by means differential equations and an **earliest** directive for placing event occurrences at instant boundaries. On the semantic level, the most important concept consists of regarding an instant as a transition with an execution interval.

References

[1] Alur, R. et al., *Modular Specification of Hybrid Systems in Charon*, in: *Hybrid Systems: Computation and Control*, LNCS 1790 (2000), pp. 6–19, proceedings HSCC '01 workshop.

[2] Alur, R. et al., *The Algorithmic Analysis of Hybrid Systems*, in: *Analysis and Optimization of Systems: Discrete–event Systems*, Lecture Notes in Control and Information Sciences 199 (1994), pp. 331–351, conference proceedings.

[3] Berry, G., *The Constructive Semantics of Esterel*, draft book, version of July 2, 1999.

[4] Berry, G. and G. Gonthier, *The Esterel Synchronous Programming Language*, Science of Computer Programming **19** (1992), pp. 87–152.

[5] Grosu, R., T. Stauner and M. Broy, *A Modular Visual Model for Hybrid Systems*, in: *Formal Techniques in Real–Time and Fault–Tolerant Systems*, LNCS 1486 (1998), pp. 75–91, proceedings FTRTFT '98 symposium.

[6] Henzinger, T., *Masaccio: a Formal Model for Embedded Components*, in: *Theoretical Computer Science*, LNCS 1872 (2000), pp. 549–563.

[7] Lynch, N., R. Segala and F. Vaandrager, *Hybrid I/O Automata Revisited*, in: *Hybrid Systems: Computation and Control*, LNCS 2034 (2001), pp. 403–417, proceedings HSCC '01 workshop.

[8] Milner, R., "Communication and Concurency," Prentice Hall International, 1989.

[9] Müller, O. and T. Stauner, *Modelling and Verification Using Linear Hybrid Automata — a Case Study*, Mathematical and Computer Modelling of Dynamical Systems **6** (2000), pp. 71–89.

[10] Schneider, K., *Embedding Imperative Synchronous Languages in Interactive Theorem Provers*, in: *Applications of Concurrency to System Design* (2001), pp. 143–154, proceedings ICACSD '01.

[11] Selic, B., G. Gullekson and P. Ward, "Real–Time Object–Oriented Modelling," John Wiley & Sons Ltd., 1994.

[12] Selic, B. and J. Rumbaugh, *Using UML for Modeling Complex Real–Time Systems* (1998), http://www.rational.com/media/whitepapers/umlrt.pdf.

[13] Stauner, T., *Specification and Verification of an Electronic Height Control System using Hybrid Automata* (1997), diploma thesis, Munich University of Technology.

[14] The MathWorks Inc., *MATLAB* (2000), http://www.mathworks.com/products/matlab.