An Operational Semantics and Compiler for Real-Time Specifications

(Submitted For Publication, Please Do Not Distribute)

Carlos Puchol, Douglas A. Stuart and Aloysius K. Mok
{cpg,dastuart,mok}@cs.utexas.edu
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188

Keywords: Formal specification, real-time systems, computer-aided engineering, reactive systems, operational semantics, compilers.

Abstract

The Modechart specification language is a formalism for the specification and implementation of real-time systems. The semantics of the language was formally defined axiomatically in Real-Time Logic. This paper presents the semantics for Modechart in an operational style and a compiler for automatically synthesizing specifications. Modechart is a synchronous language, which assumes the synchronous model of concurrency and broadcast of events. The synchronous model assumes the system to be theoretically much faster than the environment. Modechart allows both deterministic and non-deterministic real-time specifications. The semantics for the class of deterministic specifications is introduced first, followed by the definition of the non-deterministic semantics. The deterministic semantics naturally derives a Modechart compiler, which provides automatic synthesis of formal specifications. An extension to the compiler presented provides support for a subset of the class of non-deterministic specifications, very useful in practice. We characterize this class and show how it can be used in automatic code generation for engineering real-time applications.

1 Introduction

The Modechart specification language is a formally defined language for specifying the behavior of hard real-time systems [JM94]. Modechart is a synchronous language particularly suited for systems with absolute timing constraints. It was designed within the SARTOR project [Mok85], an effort towards an integrated development environment for the design, analysis and implementation of real-time systems. The computation model for the language is based on the *synchrony hypothesis* [Hal93]. The synchrony hypothesis models the system as being theoretically infinitely faster than the environment —it assumes instantaneous broadcast of internal events and immediate (synchronous) response to external inputs.

The first semantics of Modechart [JM94] is an axiomatic semantics defined in Real-Time Logic (RTL, [JM86]). While the rigour of that definition allows for strong analytical capabilities, an operational semantics provides a computational approach to the language, more useful for automatic engineering of real-time applications. Thus, the goal of introducing this operational semantics is to provide a semantics which captures a more practical and intuitive notion of behavior, more suitable for implementation and reference, yet still rigorous and suitable for analysis (such as state-space exploration). The semantics here introduced leads to a compiler for the language, which is presented after characterizing the class of useful programs the compiler translates.

For simplicity, we present the development of the semantics in two stages. The first stage describes the semantics for the class of deterministic specifications. The second stage takes the deterministic semantics and augments it to encompass all Modechart specifications.

This paper is organized as follows. Section 2 briefly introduces Modechart. Section 3 provides the notation and basic definitions used during the rest of the paper. Section 4 defines a two-level transition systems used in the first stage of the semantics, covering the deterministic specifications. Section 5 essentially augments the semantics, adding non-deterministic specifications. Section 6 goes over special cases of specifications that have been considered particularly problematic in the original axiomatic semantics. The compiler based on the deterministic semantics is described in Section 7. Section 8 offers the conclusions obtained from this work and Appendix A outlines the equivalence of both semantics.

2 Overview of Modechart

The Modechart specification language [JM94] is a synchronous language designed for the specification and implementation of real-time systems. Modechart borrows from Statecharts [Har87] the use of hierarchical graphical formalisms to extend conventional state-transition diagrams. Modechart adds absolute and relative timing semantics to Statecharts transitions. A set of tools has been developed for the specification, analysis and implementation of real-time systems within this framework [CHLR93, PMS95].

Modechart specifications are made up of *modes* and *transitions*. Modes can be thought of as hierarchical partitions of the state space, and can be combined by using *parallel* or *serial* composition. The most basic modes are called *atomic* modes. A serial relationship among several modes indicates that the system operates in at most one of the modes at any time. Serial modes designate one mode as their *initial* mode, which is instantaneously entered once the serial mode is entered. Modes can also be in parallel, in which case the system operates in all modes simultaneously.

Transitions allow the system to switch from an origin to a destination mode (possibly at different levels within the hierarchy) and can only take place between modes which have a serial relationship. When a transition occurs, the transition is said to be *taken*. Entry of a serial mode causes the system to enter one of its children. Entry of a parallel mode forces the system to enter all its children. A transition out of one mode requires exit of all of the modes in parallel to it, if any. A transition has an associated expression that, when "triggered," can cause the transition to be taken.

Transition expressions may be formed by disjunctions of *triggering expressions* and *timing expressions*. Triggering expressions are conjunctions of external and internal events of the system (they are triggered iff the conjunction is true). Timing expressions are of the form (lb, ub), where lb is the lower bound and ub is the upper bound (possibly infinite) of a time interval. This time interval denotes the time, since the origin mode of the transition was entered, in which the transition may be triggered. If a mode has at least one transition triggered, one has to be taken (non-deterministically).

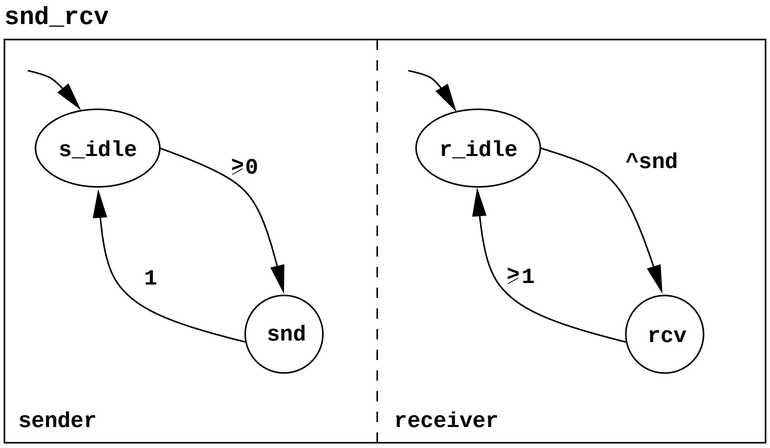

Figure 1: Sender-receiver specification

2.1 Computation Model

The theoretical computation model assumed by the language is based on the synchrony hypothesis, which stipulates that the internal events that take place in a reaction to external stimuli are infinitely fast and occur at discrete intervals (usually called steps) in the timeline. This synchronous model allows for a simpler formal development, without compromising the implementability of the language in practical applications.

The semantics developed in this paper has two levels, namely, the *step* level, and the *reaction* level. The step semantics describes the state changes from one step to the next for an external observer to the system. The reaction semantics is subsumed within the step semantics and describes the sequence of instantaneous state changes within a step (usually called a reaction or a sequence of micro-steps). Two transition systems [Eme90] model each of these semantic levels.

The original definition of Modechart includes support for so-called "actions," which are general computations subject to scheduling and other resource constraints. In general, these actions can be attached to and triggered by events in the system. Actions themselves can influence the behavior of the system. In this paper we define the semantics of "pure" Modechart specifications, which are those whose behavior cannot be influenced by the presence of actions or the (im)possibility of achieving their scheduling or resource constraints, i.e. if present, they are basically considered side effects.

2.2 Examples

Consider specification **snd_rcv** shown in Figure 1. It specifies a sender-receiver pair of synchronous processes. It has a parallel mode containing two children, the sender and the receiver. While the sender is in mode **s_idle** it may take (at any time, including immediately and never) its only transition out to mode **snd**. After exactly one time unit, the system enters the **s_idle** again. While the receiver is in the **r_idle** mode, it may (when the sender enters mode **snd**) take its only transition to mode **rcv**. After *at least* one time unit the receiver may go back to the **r_idle** mode.

Cosider now Figure 2, which shows a specification for a small microwave oven. It has two concurrent processes, **door** to keep track of the state of the door (**open** or **closed**), and **oven**

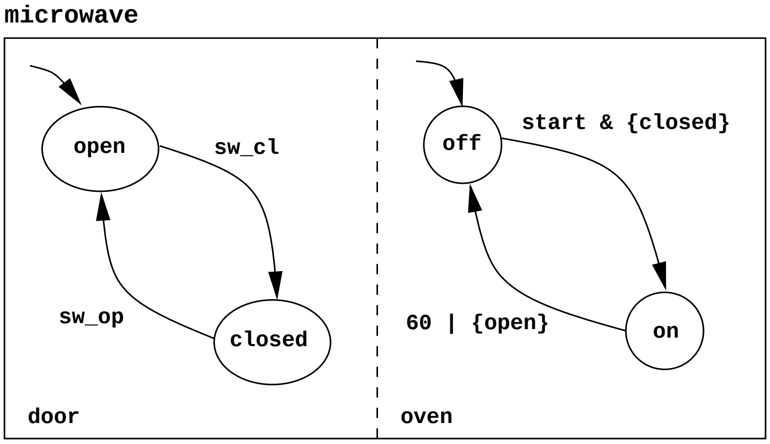

Figure 2: Microwave specification

which runs the the heating element of the oven (**on** or **off**). The oven can only enter the **on** mode if **start** (an external signal corresponding to depressing the start button) occurs and the door is in **closed** mode. The oven enters mode **off** when either a timeout of 60 time units expires or the door is in mode **open** (the door is opened).

3 Modechart Syntax

While Modechart is a graphical language, we now concentrate on introducing the formal syntax underlying the language which is conveyed, sometimes more naturally, with the graphical notation.

Definition. A *modechart* is a 5-tuple $\langle \mathcal{M}, \{S, P, \mathit{initial}\}, \sqsubseteq, \mathcal{T}, \mathcal{E} \rangle$ with the following components:

- $\mathcal{M}$ is a (finite) set of *modes*.
- $S \subset \mathcal{M}$ is a set of *serial* modes and $P \subset \mathcal{M}$ is a set of *concurrent* modes, with $S \cap P = \emptyset$ and $S \cup P \neq \mathcal{M}$. The total function *initial*: $S \rightarrow \mathcal{M}$ denotes a mode (the *default* or *initial* mode) corresponding to a serial mode.
- $\sqsubseteq$ is a complete partial order relation (CPO) on $\mathcal{M}$ which furthermore defines a tree among the elements of $\mathcal{M}$[1]. If $m \sqsubset n$ we say "m is contained in n," "m is a descendant of n" or "n is an ancestor of m." It is required that $S \cup P = \{m \in \mathcal{M} \mid n \sqsubset m,\ \text{for some}\ n \in \mathcal{M}\}$.
- $\mathcal{T} \subset \mathcal{M} \times \mathcal{M} \times MTE$ is a set of *transitions*. For a transition (s, t, e), s is the *source* mode of the transition, t is the *target* mode of the transition and e is the Modechart *transition expression* (MTE) associated with the transition, as defined by the grammar below.
- $\mathcal{E}$ is a (finite) set of external *input signals*.

[1]We often use the relations $\sqsubset, \not\sqsubseteq, \not\sqsubset$ and the set extensions induced from $\sqsubseteq$. In formal terms, we required $\sqsubseteq$ to additionally satisfy the following two properties: $\mathcal{M}$ is directed (every finite non-empty subset of $\mathcal{M}$ has an upper bound in $\mathcal{M}$) and all ancestors of a mode form a chain ($\forall m, n, p \in \mathcal{M} \mid p \sqsubseteq \{m, n\} \Rightarrow m \sqsubseteq n$ or $n \sqsubseteq m$).

Definition. The set of *external output signals* of a modechart specification is defined as a set of labels, and it is formed by the union of four sets of labels:

$$\mathcal{L} = \mathcal{M}^{+} \cup \mathcal{M}^{-} \cup \mathcal{M}^{\perp}$$

where a set of *mode entry* labels M^{+} for $M \subseteq \mathcal{M}$ is defined as follows:

$$M^{+} = \{ m^{+} \mid m \in M \},$$

and m^{+} is an entry label for mode m. The sets of labels M^{-} and $M^{\perp}$ are defined similarly for *exit mode* and *active mode* labels. Thus every mode m in the system has three associated (non-exclusive) outputs which collectively capture the evolution of modes within the system. Labels denote the system entering the mode, m^{+}, exiting the mode, m^{-}, and the mode being active, $m^{\perp}$, respectively. Additionally, each transition in the system has a *transition label* associated with it:

$$\mathcal{M}^{\rightarrow} = \{s \rightarrow t \mid (s,t,e) \in \mathcal{T}, \text{for some } e \text{ in } \mathit{MTE}\}.$$

Definition. The valid Modechart *transition expressions* are defined according to the following BNF grammar:

$$\begin{array}{rcl} \mathit{MTE} & ::= & (\ \mathit{lb}\ ,\ \mathit{ub}\ ) \mid \mathit{TrigC} \\ \mathit{TrigC} & ::= & \mathit{Event}\ \wedge\ \mathit{TrigC} \mid \mathit{Event} \\ \mathit{Event} & ::= & e \mid \overline{e} \mid \{\ \mathit{MSet}\ \} \mid m^{+} \mid m^{-} \mid m \rightarrow n \\ \mathit{MSet} & ::= & m^{\perp}\ ,\ \mathit{MSet} \mid m^{\perp} \end{array}$$

In the grammar above, $lb, ub \in \mathbb{N}$ are called the *lower bound* and the *upper bound* of the transition[2], e denotes the presence of external event e at any time instant, $\overline{e}$ denotes the absence of external event e at any time instant and $m, n \in \mathcal{M}$ are modes of the specification. The expression $m \rightarrow n$ denotes the event of a transition being taken from m to n.

Expressions whose form is (lb, ub) are referred to as *timing* transition expressions while the rest of the expressions are referred to as *triggering* transition expressions. Timing transitions can be unbounded from above.

Figure 1 includes three timing transitions and a triggering transition. The transition labelled by **`^snd`** carries the triggering transition expression **`snd`**$^{+}$, denoting the entrance of mode **`snd`**. In practice, the concrete syntax of timing transitions uses abbreviations. The transition from **`s_idle`** to **`snd`** is $(1,1)$, which is abbreviated to just **`1`**. The transition from **`snd`** to **`s_idle`**, $(0,\infty)$, is denoted with $\geq$**`0`**.

Notation. For convenience, we denote the set of timing expressions as E_{ti} and the set of triggering expressions as E_{tr}. We extend the set of triggering expressions as follows. Two transitions with identical source and destination modes with triggering expressions e and e' are merged into one transition with an expression $e \vee e'$, which is also a triggering expression. We denote the set of possible input events in one time instant or input language with $\Sigma = 2^{\mathcal{E}}$ and the set of possible outputs or output language with $\Sigma_o = 2^{\mathcal{L}}$. We denote their closure or Kleene star with Σ^{*} and Σ_o^{*} respectively.

[2]The upper bound is allowed to be infinite.

Definition. A mode m is *atomic* iff $m \in \mathcal{M} - (S \cup P)$. A mode m is the *root* mode of $\mathcal{M}$ iff $\forall n \in \mathcal{M} :: n \sqsubseteq m$. The function *children*: $\mathcal{M} \rightarrow 2^{\mathcal{M}}$ is defined as follows:

$$children(m) = \{ \, n \mid n \sqsubset m \text{ and } \not\exists n' :: n \sqsubset n' \sqsubset m \, \}.$$

If $n \in children(m)$, we say that n is an *child* of mode m, and m is the *parent* of n. The $\sqsubset$ relation therefore captures the tree of all children (or all parents) of a mode, and the function *children* provides reference to the direct descendants of modes. If $m \not\sqsubseteq n$ and $n \not\sqsubseteq m$, we say that m and n are *unrelated.*

Remark. The least upper bound of any set of modes in $\mathcal{M}$ always exists ($\mathcal{M}$ is directed) and it is unique (by the asymmetry of the inclusion relation). We will usually refer to the least upper bound of two modes, which is defined as follows, for $m, n \in \mathcal{M}$:

$$[\![m, n]\!] = \{ \, l \mid \{m, n\} \sqsubset l \text{ and } \not\exists l' :: \{m, n\} \sqsubset l' \text{ and } l' \sqsubset l \, \}.$$

Graphically, the least upper bound of a set of modes is the smallest mode enclosing them. The concept of least upper bound of a set of modes becomes important for the purpose of developing the semantics modularly, because it delimits the scope of a mode change involving m and n.

Definition. Two unrelated modes $m, n \in \mathcal{M}$ are *concurrent* (or *parallel*), written $m \,\|\, n$, iff $[\![m, n]\!] \in P$. Two unrelated modes $m, n \in \mathcal{M}$ are *sequential,* written $m \sim n$ iff $[\![m, n]\!] \in S$.

Lemma 1 *Let $m, n \in \mathcal{M}$ be any two unrelated modes. Then the following holds:*

$$m \sim n \quad \textit{is equivalent to} \quad \textit{not } (m \,\|\, n)$$

Proof. The least upper bound of a set of modes cannot be atomic, it can only be either serial or parallel. ∎

Definition. (Syntactic correctness of a modechart.) A *syntactically correct* modechart is one for which the following properties hold:

- $\forall s \in S, \exists m \in children(s) :: initial(s) = m$, and,
- $\forall (s, t, e) \in \mathcal{T} :: s \sim t$.

The first condition ensures that every serial mode has one initial mode among its immediate descendants. This was not required in the original definition of the language – we justify imposing it for simplicity, given that it does not constrain the semantics or otherwise the usefulness of the language. The second condition above assures that transitions can only take place between sequential modes. This, while being a semantic requirement it is easily dealt with at the syntactic level.

Consider the (syntactically incorrect) specification in Figure 3, where the transition expressions have been omitted. The set of modes is $\mathcal{M} = \{\mathbf{a}, \ldots, \mathbf{t}\}$, with serial modes being $S = \{\mathbf{b}, \mathbf{c}, \mathbf{e}, \mathbf{f}, \mathbf{o}, \mathbf{s}\}$ and parallel modes being $P = \{\mathbf{a}, \mathbf{d}, \mathbf{q}\}$. The containment relation is depicted in Figure 3-b. It is easy to see that $children(\mathbf{d}) = \{\mathbf{e}, \mathbf{f}\}$ and that $\{\mathbf{d}, \ldots, \mathbf{k}\} \sqsubseteq \mathbf{d}$. The

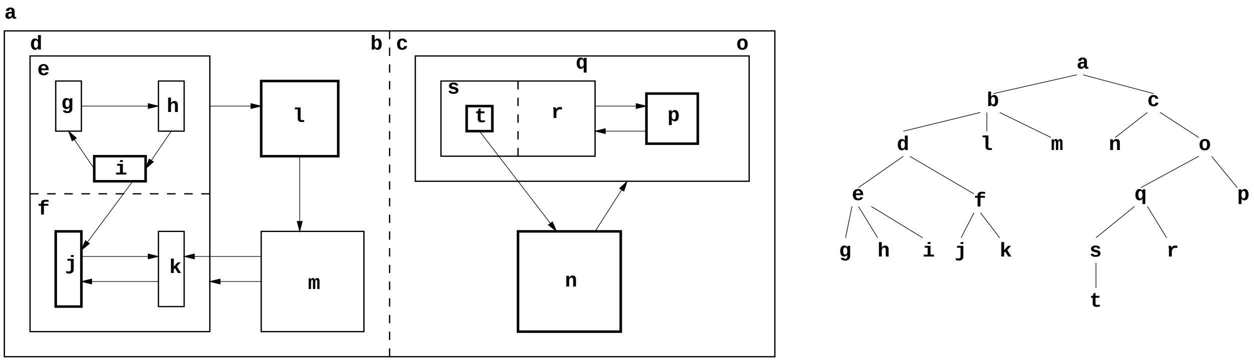

Figure 3: Syntactically incorrect specification and its mode containment relation

initial modes (which are denoted thicker in Figure 3) are correctly designated; no serial mode has two initial children. Consider now the transitions; the transition from **t** to **n** is a correct transition because $[\![\texttt{t}, \texttt{n}]\!] = \texttt{c} \in S$. Similarly, the rest of the transitions are correct, except the transition from **i** to **j**, because $[\![\texttt{i}, \texttt{j}]\!] = \texttt{d} \notin S$.

This ends the definition of the formal syntax of Modechart and the notation that shall be used in the remainder of the paper. The next section concentrates on the deterministic semantics of the language.

4 Deterministic Semantics

In order to define the mathematical semantics of Modechart specifications, we need to formalize a few concepts related to what is meant by behavior of a system. The distinction is made between the *internal* and *external* behavior. From the operational point of view, a system's behavior is nothing but the sum of its transitions from state to state over time. These transitions may make sense only at a certain level; what counts as one transition for one purpose may in fact consist of many steps when viewed in detail. Many transitions at one level may produce no discernible effect at other level. Selecting the appropriate level at which external and internal behaviors are defined is a matter of choice.

In our case, the choice is the synchronous model in which externally the system is seen as continually executing steps of "instantaneous" input/output interactions. Internally, each of this steps is accomplished by a sequence of smaller steps, collectively called a "reaction," which is required to be finite. These steps need not be observable externally; only their accumulated final effect needs to be observable. In Modechart, reactions are formed by sequences of transition "firings," hence reactions capture the essence of the behavior of Modechart specifications.

Specification languages are said to be either "open" or "closed" depending on how the "environment" of the system under specification is modeled. Some languages require that the complete environment surrounding the system under specification be defined before proceeding to analysis or synthesis. The systems modeled with these languages are said to be closed systems and do not allow any external inputs per se (this does not preclude the system from having observable "outputs," however). On the other hand, in open systems the environment is assumed to be free from constraints (although it may still be possible to refine part of the environment).

In Modechart, the environment of the system under specification can be modeled completely within the language. Some of the tools in the Modechart toolset require the system to be closed to operate [JS88]. In the definitions ahead, it is assumed that any input signals not modeled in the specification are unconstrained, thus they can arrive at any time.

Our definition of semantics is in Plotkin's structural operational semantics style [Plo81]. We define the semantics of a Modechart specification by defining a transition system [Eme90] $\langle \Gamma, \mapsto \rangle$ for it. This transition system defines the semantics for an external observer. The carrying set, Γ, defines the possible states of the system. The possible transitions are determined by the label of the transition relation, which is a Σ-indexed set of relations, $\gamma \overset{i}{\mapsto} \gamma'$, mapping state γ to state γ' for each possible input i.

Each of this steps produces an output which is the result of the reaction and it is determined by the specification. This output is considered to be available instantaneously. Defining the semantics of these reactions is the task of the lower semantic level. One of the more important characteristics of the reaction semantics is that the reactions always are causal (the cause of a transition precedes the effect) and terminate (there exist no unbounded loop in the reaction).

The transition system above is defined in terms of a lower level transition system, $\langle \Gamma, \rightarrow \rangle$, whose effects are not observable externally. This structure takes care of instantaneous reactions to external inputs, where no time is consumed in the process, in accordance with the synchrony hypothesis. The possible reactions are determined by the family of relations $\xrightarrow{i}$, for some input i.

As a notational convenience, we drop the input event label from $\rightarrow$ and $\mapsto$ when not ambiguous and we refer to them as just relations throughout the rest of the development, unless otherwise noted.

The top level structure is generic for the complete class of Modechart specifications, that is, only the low level structure will need to be redefined, we include non-deterministic specifications in the semantics. In general, the top level structure defines transitions as labeled pairs of the form $\gamma \xmapsto[o]{i} \gamma'$, which means that "if the system is in state γ, a possible result of external input event i is the output event o and a change in the state to γ'."

Technically, the output in Modechart specifications is formed by the trace of entered, exited and active (labels of) modes at each step, over time. The labels of the system are therefore considered to be the observable behavior of the system. In an implementation, all actual interaction with the environment is typically produced by operations associated with entry and exit events of modes and transitions being taken.

Definition. A non-empty set of modes $M \subseteq \mathcal{M}$ is *sound* iff

$$\forall m, m' \in M,\ m \neq m' :: \ m \sqsubset m' \ \text{ or } \ m' \sqsubset m \ \text{ or } \ m \,||\, m'.$$

Definition. Let $M \subseteq \mathcal{M}$ be a sound set of modes. M is *maximally sound* iff

$$\forall\ m \in (\mathcal{M} - M) :: \ M \ \cup \ \{m\} \ \text{ is not sound.}$$

Definition. Let $M \subseteq \mathcal{M}$ be a sound set of modes. The *closure* of M, $M^{\square}$, is constructed iteratively with the following rules:

1. $M^{\square} \supseteq M$.

2. $\forall m' \in M^{\Box} :: \forall m \in \mathcal{M} ::$ if $m' \sqsubset m$ then $m \in M^{\Box}$.

3. $\forall s \in M^{\Box} \cap S ::$ if $M \cap \mathit{children}(s) = \emptyset$ then $\mathit{initial}(s) \in M^{\Box}$.

4. $\forall p \in M^{\Box} \cap P :: \mathit{children}(p) \subseteq M^{\Box}$.

5. No other modes are in $M^{\Box}$ than those included by rules 1–4.

The closure of a set of modes denotes the "default expansion" of a sound set of modes and yields another sound set of modes that forms a plausible configuration for the system to be in. Specifically, every mode in the original set is also in the closure (rule 1). All ancestors of each mode in the closure are also in the closure (rule 2). If a serial mode is in the closure and none of its children is, then its initial mode is also in the closure (rule 3). If a parallel mode is in the closure, then all it's children are in the closure also (rule 4). Finally, rule 5 establishes that only modes added by the preceding rules can be added to the closure.

For example, in the specification of Figure 3 the closure of the set of modes $\{\mathbf{i}, \mathbf{p}\}$ is the set $\{\mathbf{a}, \mathbf{b}, \mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{j}, \mathbf{i}, \mathbf{c}, \mathbf{o}, \mathbf{p}\}$.

Lemma 2 *Let $M \subseteq \mathcal{M}$ be a set of modes. The following holds:*

$$\exists m, m' \in M,\ m \neq m' ::\ \textit{if}\ \ m \sim m'\ \ \textit{then}\ \ M\ \ \textit{is not sound.}$$

The proof of this lemma is left as an exercise.

Lemma 3 *$M^{\Box}$ is the unique maximally sound subset of $\mathcal{M}$, for all sound $M \subseteq \mathcal{M}$*

Proof. Since M is sound, any two different elements in $M^{\Box}$ are either related by $\sqsubset$ or are concurrent. No rule can introduce modes that are serial with respect to other modes in the closure, therefore $M^{\Box}$ is sound.

It is also the case that it is maximally sound. We prove it by showing that any addition of a mode $m \in \mathcal{M} - M^{\Box}$ yields an unsound subset of modes. We know that since $m \notin M^{\Box}$,

$$\exists m' \in \mathcal{M} - M^{\Box},\ \exists n \in M^{\Box} ::\ m' \in \mathit{children}(n)\ \text{ and }\ m \sqsubseteq m'.$$

We also know that n is a serial mode, since otherwise m' would be in $M^{\Box}$. In addition, we know that $\exists m'' \in \mathit{children}(n) \cap M^{\Box} ::\ m' \neq m''$ by virtue of the definition of $M^{\Box}$.

Now consider the set $N = M^{\Box} \cup \{m\}$. From the definition of $M^{\Box}$, all ancestors of m belong to $M^{\Box}$, in particular m'. Since $m' \sim m''$, then by Lemma 2, N is not sound.

It is also unique. Assume that is not the case and there are two different closures, M_1 and M_2, of mode M. Let $m \in (M_1 - M_2)$. Then, since M_1 is a closure of M, it is the case that $M \cup \{m\}$ is sound. This implies that m belongs to M_2 as well (by the definition of $M^{\Box}$), contradicting the assumption that both closures are different. Symmetrically, the same argument holds for the difference $M_2 - M_1$, hence M_1 and M_2 are in fact one and the same set, contradicting the assumption that the closure is not unique. ∎

Maximally sound subsets form an integral part of the state of Modechart specifications. We define the concept of state in the system at this point, which will become the carrying set of the Kripke structures. A state is a triple that includes a set of "active" modes, a set of "active" transitions, each possibly having an associated clock reading, and a set of "accumulated" labels, which carries the information of the output event to be constructed throughout a reaction.

After defining the concept of state, we define the meaning of transitions being triggered. Timing transitions have an associated clock which, when expired, determines when the transition is triggered. Each timing transition has a clock which is initialized upon entering the source mode of the transition. The clock reading decreases with every time step the system takes, but remains constant throughout a reaction. On the other hand, triggering transitions have expressions, which need to become "true" to be triggered, subject to the current input and current reaction.

Notation. We define the set of "extended" transitions as $\mathcal{T}' = \mathcal{T} \times \mathbb{Z}$, where the integer will denote a *clock reading*, useful only for timing transitions.

Definition. A *state* of a (syntactically correct) modechart specification is a triple $\gamma = (M, T, o) \in 2^{\mathcal{M}} \times 2^{\mathcal{T}'} \times \Sigma_o = \Gamma$ with the following properties:

- M is a maximally sound subset of $\mathcal{M}$, called the set of *active* modes, or a mode *configuration*.
- $T = \{\tau = ((s, t, e), n) \in \mathcal{T}' \mid s \in M\}$ is the set of *active* transitions. We loosely treat elements of T as transitions, when not ambiguous. When e is a timing expression, we call n the *local clock reading* of τ.
- $o \in \Sigma_o$ is the set of internal events (or output).

Definition. A timing transition with clock reading n is said to be *triggered* if $n = 0$. A triggering transition with expression ϵ is *triggered* in the *context* $R \subseteq \mathcal{E} \cup \mathcal{L}$, written $R \vdash \epsilon$ iff ϵ is valid, according to the following recursive set of rules, where e is an external input signal and $m, m_0, \ldots, m_n$ are modes in $\mathcal{M}$:

$$\frac{e \in R}{R \vdash e} \qquad \frac{e \notin R}{R \vdash \overline{e}}$$

$$\frac{m^+ \in R}{R \vdash m^+} \qquad \frac{m^- \in R}{R \vdash m^-}$$

$$\frac{\{\, m_0{}^\perp, m_1{}^\perp, \ldots, m_n{}^\perp \,\} \cap R \neq \emptyset}{R \vdash \{\, m_0, m_1, \ldots, m_n \,\}}$$

$$\frac{R \vdash \epsilon \;\wedge\; R \vdash \epsilon'}{R \vdash (\epsilon \;\wedge\; \epsilon')} \qquad \frac{R \vdash \epsilon \;\vee\; R \vdash \epsilon'}{R \vdash (\epsilon \;\vee\; \epsilon')}$$

We now define several functions whose purpose is to simplify the definition of an internal reaction. These functions capture the evolution of the system throughout a reaction in terms of the transitions and modes involved. The function *texit* defines the set of active transitions with source contained in a given set of modes. The function *tenter* captures all the transitions whose source is in a given set of modes. The function *mexit* defines the "tree" of modes contained or equal to the "source" side of a given transition. Finally, the function *menter* defines the "path" of modes in the "target" side of a given transition.

Definition. Let T be a set of active transitions and M a set of modes. Then we define a function $texit : 2^{\mathcal{T}'} \times 2^{\mathcal{M}} \to 2^{\mathcal{T}'}$ as follows:

$$texit(T, M) = \{ ((s,t,e), n) \in T \mid s \in M \}.$$

Definition. Let M be a set of modes. Then we define a function $tenter : 2^{\mathcal{M}} \to 2^{\mathcal{T}'}$ as follows:

$$tenter(M) = \{ ((s,t,e), n) \mid s \in M\}, \quad \text{with} \quad n = \begin{cases} lb & \text{if } e = (lb, ub) \in E_{ti} \\ \text{undefined} & \text{if } e \in E_{tr}. \end{cases}$$

Definition. Let $\tau = (s,t,e) \in \mathcal{T}$ be a transition. Then we define a function $mexit : \mathcal{T} \to 2^{\mathcal{M}}$ as follows:

$$mexit(\tau) = \{ m \mid m \sqsubseteq n \in children[\![s,t]\!] \text{ and } s \sqsubseteq n \}.$$

Definition. Let $\tau = (s,t,e) \in \mathcal{T}$ be a transition. Then we define a function $menter : \mathcal{T} \to 2^{\mathcal{M}}$ as follows:

$$menter(\tau) = \{ m \mid t \sqsubseteq m \sqsubseteq n \in children[\![s,t]\!] \}.$$

We finally define the relation that captures reactions internally. This relation captures the action of the system "taking" an active transition from one mode configuration to another. We define it using two rules, one for transitions with timing expressions and the other for transitions with triggering expressions, since their conditions are different. A triggered transitions is said to be *taken* when it is selected to be reduced in the transition system $\langle \Gamma, \to \rangle$. We note that there can be several transitions triggered simultaneously, in which a choice is made and one is taken.

Definition. The *reaction* (family of Σ-indexed) binary relation(s), $\to \subseteq \Gamma \times \Sigma \times \Gamma$, for $i \in \Sigma$ is defined as follows:

- Timing transitions:

$$\frac{\tau = ((s,t,e), 0) \in T \text{ and } (s,t,e) \in E_{ti}}{(M, T, o) \xrightarrow{i} (M', T', o')}$$

- Triggering transitions:

$$\frac{\tau = ((s,t,e), n) \in T \text{ and } (s,t,e) \in E_{tr} \text{ and } (i \cup o) \vdash e}{(M, T, o) \xrightarrow{i} (M', T', o')}$$

where:

$$\begin{aligned}
&M_{aux} = M - mexit(\tau),\\
&M_{out} = M \cap mexit(\tau),\\
&M' = (M_{aux} \cup menter(\tau))^{\square},\\
&M_{in} = M' - M_{aux},\\
&\\
&T' = (T - texit(T, mexit(\tau))) \cup tenter(M_{in}),\\
&\\
&o_{aux} = {M_{in}}^{+} \cup {M_{out}}^{-} \cup {M'}^{\perp} \quad \text{and}\\
&o' = o \cup o_{aux}.
\end{aligned}$$

Definition. The *reflexive, transitive closure* of the (family of binary) relation(s) $\xrightarrow{i}$ is defined as follows for a given input i (which is omitted for simplicity), and an integer $n > 0$:

$$\begin{array}{lll} \gamma \rightarrow^{0} \gamma' & \text{iff} & \gamma = \gamma' \\ \gamma \rightarrow^{n} \gamma' & \text{iff} & \exists \gamma'' :: \ \gamma \rightarrow \gamma'' \text{ and } \gamma'' \rightarrow^{n-1} \gamma' \\ \gamma \rightarrow^{*} \gamma' & \text{iff} & \exists n :: \ \gamma \rightarrow^{n} \gamma' \end{array}$$

Definition. A *fixpoint* of the reaction (family of binary) relation(s), $\xrightarrow{i}$, with respect of the state γ and input event i is γ' and is defined as follows (for a given i):

$$\gamma \rightarrow^{\triangle} \gamma' \quad \text{iff} \quad \gamma \rightarrow^{*} \gamma' \ \text{ and } \ \nexists \gamma'' :: \ \gamma' \rightarrow \gamma''.$$

At this point, the low level structure $\langle \Gamma, \rightarrow \rangle$ is fully defined. The class of deterministic Modechart specifications is precisely formed by those specifications that make this structure deterministic (with respect to the fixpoint relation). We now start building the top level structure capturing the observable behavior over time, based on the low level structure just defined. We start by defining the function *tick*, which performs a "time step" in the state of the system, updating the transitions and "clearing" the output event after a reaction has taken place. We then define the step relation (or more specifically the family of binary relations) which forms the top level structure. We thus associate a transition system $\langle \Gamma, \mapsto \rangle$ to every Modechart specification. This structure defines the semantics of the specification.

Definition. We define a function $tick : \Gamma \rightarrow \Gamma$ which given a state, returns the resulting state after one time instant has elapsed:

$$tick(M, T, o) = (M, T', M^{\perp})$$

where

$$T' = \{ \ ((s,t,e), n') \mid ((s,t,e), n) \in T \ \}, \quad \text{with} \quad n' = \begin{cases} n - 1 & \text{if } e \in E_{ti} \\ \text{undefined} & \text{if } e \in E_{tr}. \end{cases}$$

Definition. We define the *step* transition relation $\mapsto \subseteq \Gamma \times \Sigma \times \Sigma_o \times \Gamma$ as follows (for a given i):

$$\frac{\gamma \xrightarrow{i}{}^{\triangle} (M,T,o) \ \text{ and } \ \gamma' = tick(M,T,o)}{\gamma \overset{i}{\underset{o}{\longmapsto}} \gamma'}.$$

where $i \in \Sigma$ is a set of input events. We denote the transitive closure of $\mapsto$ with $\overset{*}{\mapsto}$ for sequences of input events. If $\gamma = \gamma'$ and no transitions in T contain a reference to external events, the modechart is said to *halt* at γ.

Definition. Let $i_0 i_1 \ldots i_n \ldots = I \in \Sigma^*, n \geq 0$, be an input event sequence. Let $\mathcal{S}$ be a modechart with r as root mode. Let $p = \{r\}^{\Box}$ and $\gamma_0 = (p, \ tenter(p), \ p^{+} \ \cup \ p^{\perp} \ )$. Then $\mathcal{S}(I) = o_0 o_1 \ldots o_n \ldots = O \in \Sigma_o^*$ is a *corresponding* output sequence for $\mathcal{S}$ and I iff

$$\exists \gamma_1, \ldots, \gamma_n, \ldots \in \Gamma :: \ \gamma_0 \overset{i_0}{\underset{o_0}{\longmapsto}} \gamma_1 \overset{i_1}{\underset{o_1}{\longmapsto}} \gamma_2, \ \ldots \ , \gamma_n \overset{i_n}{\underset{o_n}{\longmapsto}} \gamma_{n+1}, \ \ldots.$$

The initial state γ_0 is called the *start state* of $\mathcal{S}$.

Definition. Let $\mathcal{S}$ be a proper modechart. Then the *behavior* of $\mathcal{S}$ is the set $\mathcal{B}(\mathcal{S}) = \{ \mathcal{S}(I) \mid I \in \Sigma^* \}$.

Definition. A modechart $\mathcal{S}$ is said to have a *zero cycle* when $(o \cap o_{aux}) - \mathcal{M}^\perp \neq \emptyset$ for some transition $\gamma \xrightarrow{i} \gamma'$ in some derivation of some element of $\mathcal{B}(\mathcal{S})$. Intuitively, a zero cycle-free modechart is one in which no mode can be entered (nor exited) more than once during one reaction.

Lemma 4 *(Termination) Let $\mathcal{S}$ a zero cycle-free modechart with a structure $\langle \Gamma, \mapsto, \rightarrow \rangle$. Let $\gamma \in \Gamma$ be some state of $\mathcal{S}$. Then there exists a $k \geq 0$ such that*

$$\exists \gamma' \in \Gamma \mid \gamma \rightarrow^k \gamma' \qquad \textit{and} \qquad \nexists \gamma'' \in \Gamma \mid \gamma' \rightarrow \gamma''.$$

Proof. Each transition in the system can be taken at most once during one reaction, otherwise at least one mode would be entered more than once, contrary to the assumption the $\mathcal{S}$ does not have zero cycles. Given than the set of transitions is finite, it can be exhausted once at most within one reaction therefore, given an initial state γ, there exists a state γ', such that $\gamma \rightarrow^* \gamma'$ and no transition is triggered in it. Hence there cannot exist a state γ'' such that $\gamma' \rightarrow \gamma''$ (for any given input). Hence γ is a fixpoint of the relation $\rightarrow$, with initial state γ. ∎

Lemma 5 *Let $\mathcal{S}$ be a proper, zero cycle-free modechart specification and let $A = \langle \Gamma, \mapsto \rangle$ and $B = \langle \Gamma, \rightarrow^\triangle \rangle$. Then the following three statements are equivalent: $\rightarrow^\triangle$ is a function, B is deterministic and A is deterministic.*

Proof. If the fixpoint relation is a function for all inputs, B is deterministic. If B is deterministic then A is also deterministic, since the fixpoint relation guarantees the determinism in A. From the definition of A, the only possible element of non-determinism is the fixpoint relation, since the *tick* function does not introduce non-determinism. Therefore, if A is deterministic then $\rightarrow^\triangle$ is a function. ∎

Definition. Let $\mathcal{S}$ be a proper, zero cycle-free modechart with a transition system $\langle \Gamma, \mapsto \rangle$. $\mathcal{S}$ is said to be *deterministic* iff $\langle \Gamma, \mapsto \rangle$ is deterministic, i.e. iff $\forall \omega \in \Sigma^*$,

$$\left.\begin{array}{l} \gamma_0 \xmapsto[\omega_o]{\omega} \gamma \\ \gamma_0 \xmapsto[\omega_o']{\omega} \gamma' \end{array}\right\} \quad \Rightarrow \quad \omega_o = \omega_o' \ \text{ and } \ \gamma = \gamma'$$

where $\gamma_0 \in \Gamma$ is the start state of $\mathcal{S}$, $\gamma, \gamma' \in \Gamma$ and $\omega_o, \omega_o' \in \Sigma_o^*$. Note that by Lemma 5, the low level structure is deterministic for a deterministic modechart.

We now introduce the concept of two transitions being consistent or in conflict, which determines in part whether they can or cannot be taken independently of each other. This concept will be useful later on for refining the concept of deterministic specifications and making specifications, which would otherwise be non-deterministic, deterministic, resulting beneficial in practice. Consistent transitions, we will see, reside in parallel components of a specification and cannot affect each other. Conflicting transitions present a hierarchical relationship among them and in general offer a non-deterministic choice.

Definition. Let $\tau = (s,t,e), \tau' = (s',t',e') \in \mathcal{T}$ be two transitions. Let $l = [\![s,t]\!]$ and $l' = [\![s',t']\!]$. The pair of transitions are said to be *consistent* iff $l \parallel l'$. Otherwise, they are said to be *in conflict.* A set of transitions is consistent iff no two distinct transitions in it are in conflict.

Lemma 6 *(Confluence) Let $\mathcal{S}$ be a deterministic modechart with a transition system $\langle \Gamma, \mapsto \rangle$. Then the fixpoint relation $\to^{\triangle}$ associated with $\mapsto$ is actually a function: it returns a unique state.*

Proof. It is possible that the system is in a reaction with a choice of several triggered transitions to be taken. The operational semantics calls for a choice to be made among them, not necessarily deterministic, and as long as there exists at least one triggered transition in the active transition set, one has to be taken.

Since the sequence of output event sets generated by a reaction is non-decreasing, the clock readings of timing transitions are not decreased and the external input signal set is not changed during reactions, no triggered transition can become "not triggered" within a reaction. This ensures that all triggered transitions are taken within a reaction.

Furthermore, the set of triggered transitions at any point in the reaction is consistent: for each pair of triggered transitions, their source modes (as well as their target modes) are parallel. Taking any of these transitions involves a disjoint set of modes with respect to the modes involved with the rest of triggered transitions. The net result of taking all the transitions is a single state, regardless of the order in which they are taken. This state is the result of the $\to^{\triangle}$ relation, thus it is in fact a deterministic function. From the definition of $\to^{\triangle}$, for any such state γ, it is obvious that $\gamma \to^{\triangle} \gamma$, i.e. it is a fixpoint. ∎

4.1 Building Deterministic Modechart Specifications

The transition system $\langle \Gamma, \to \rangle$ as defined allows more than one transition to be triggered simultaneously during a reaction. However, not all transitions should be allowed to be active simultaneously. Consistent sets of transitions do not present a problem for the semantics, since they can be taken in any sequence "in parallel," without affecting each other. On the other hand, conflicting transitions present choices when triggered simultaneously rendering the specification non-deterministic, when many times the choice is not important or can easily be left to the programmer.

Given that two triggered, conflicting transitions in one state cannot have their source modes share a common immediate parent (no two immediate children can be active simultaneously by the definitions above), their source modes have to be at different "levels." Intuitively, there can be at most one transition triggered per mode level in the hierarchy. Obviously, when more than one of these transitions are active, a definite choice needs to be made in the reaction relation as to which one of them take.

As defined, computations involving conflicting transitions could either be deterministic or non-deterministic. However, we take a conservative approach to this problem by introducing the *strong preemption axiom:* "when a set of pairwise conflicting transitions are triggered, the highest level one is taken first[3]." This axiom makes a choice of giving priority to "higher" transitions over

[3]This is one of the options offered by the RTL semantics, but it is harder to implement in that framework.

related "lower" ones, which get preempted, once the higher transition is taken. This makes all such situations where a set of conflictive transitions is enabled deterministic, thus broadening the class of useful specifications that can be compiled in a deterministic way.

A similar situation can arise with transitions at the same level, i.e. transitions with the same source mode. However, note that this scenario is (behaviorally) equivalent to having extra modes wrapped around the given mode and each transition at each separate level. With the preemption axiom, one can implement *priority among transitions* at the same level this way.

Our solution is to introduce a a predicate *preempted* that establishes a preemption relation among conflicting transitions. A simple extension to this predicate, not shwon here, can handle the introduction of priorities among transitions.

Definition. Let $\tau = ((s,t,e),n)$ be a triggered transition and let T be a set of triggered transitions. We define the predicate *preempted* for $(\tau, T, o) \in \mathcal{T}' \times 2^{\mathcal{T}'} \times \ \mathcal{L} \ \cup \ \mathcal{E}$ to be true iff the following conditions hold:

- $\tau \in T$,
- $\exists \tau' = ((s',t',e'),n') \in T :: n' = 0$ for $\tau' \in \mathcal{E}_{tr}$ or otherwise $o \models e'$ for $\tau' \in \mathcal{E}_{ti}$,
- $s \sqsubset s'$, and
- τ and τ' are in conflict.

The strong preemption axiom then is formally introduced in the semantics. The effect is that transitions can only be taken when they are not preempted by "higher level" transitions.

Definition. The *reaction* relation with preemption is re-defined as follows:

- Timing transitions:

$$\frac{\tau = ((s,t,e),0) \in T \ \text{ and } \ (s,t,e) \in E_{ti} \ \text{ and } \ \neg \mathit{preempted}(\tau, T, o)}{(M, T, o) \xrightarrow{i} (M', T', o')}$$

- Triggering transitions:

$$\frac{\tau = ((s,t,e),n) \in T \ \text{ and } \ (s,t,e) \in E_{tr} \ \text{ and } \ (i \cup o) \vdash e \ \text{ and } \ \neg \mathit{preempted}(\tau, T, o)}{(M, T, o) \xrightarrow{i} (M', T', o')}$$

where the auxiliary notation is the same as before.

To illustrate how useful this axiom is in practice, consider the specification in Figure 4, which is a version of the microwave in Figure 2. In this version, the **oven** has a new child called **beep**. This specification is not deterministic. Suppose the door becomes **open** at exactly 60 steps after the oven is **on**. In this case, the specification calls for a choice of taking the transition to **off** or taking the transition to **beep**.

One engineering solution for this specification would be for the compiler to pick one of the two transitions (randomly at compile time, or at run time or always in a fixed way) and take it. However, while this presents no theoretical difficulty, since taking a choice cannot violate properties known

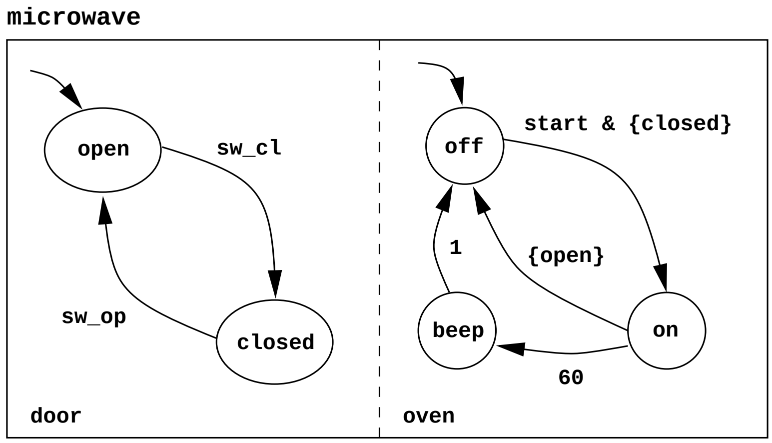

Figure 4: New microwave specification

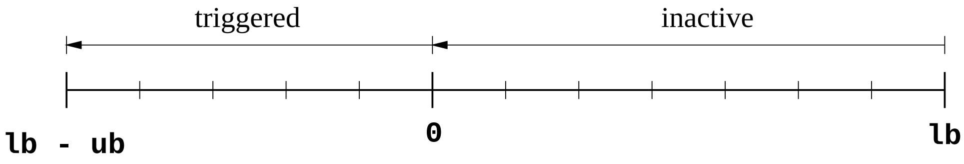

Figure 5: Non-deterministic timing transition representation

of the non-deterministic specification, it arises practical development issues – there is no simple way for a developer of ensuring one decision will take precedence over the other. Furthermore, if a developer makes the choice by denoting one transition to have higher precedence than the other, then the specification becomes deterministic and can be compiled automatically without taking any choices and the behavior is guaranteed regardless of the compiler implementation.

5 Non-Deterministic Semantics

We now proceed to extend the semantics just defined to capture the full set of non-deterministic specifications. In this section, we assume the definitions in the previous sections hold; only the new definitions presented here redefine the previous definitions. We generalize the definition of timing transitions and redefine the $\rightarrow$ relation to handle these extended timing expressions. We then proceed to redefine the transitive closure and the fixpoint of $\rightarrow$.

Notation. We make the following abbreviation: $\Pi = \{\ 0, 1, 2\ \}$ denotes the set of "types of transitions." We introduce the symbol ∞ to denote a "large" positive integer, and define that $\forall n \in \mathbb{N} :: n < \infty$ and $\infty - n = \infty$. We use λ to denote the empty string.

Definition. A *non-deterministic* timing transition is a timing transition such that $lb < ub$. Where $lb \in \mathbb{N}$ and $ub \in \mathbb{N} \cup \{\ \infty\ \}$. A timing transition is *triggered* if its clock reading is at most zero.

The intuitive meaning of these transitions is that they may be taken at least lb time units after becoming active or at most ub time units after becoming active, at which time they are taken (unless some other transition τ is taken and τ de-activates the original transition). To introduce this behavior into the formal semantics defined so far with minimal disruption, the semantics now allows the clock reading to become negative. The idea is illustrated in Figure 5. When a non-deterministic timing transition is initialized, the value of the clock reading is set to the lower bound and is decreased by one each time instant (as before). While its value is greater than zero, the transition relation does not allow the transition to be taken. When its value becomes zero, it may be taken non-deterministically, according to the reaction relation defined below. If it is not taken, its value keeps on decreasing to negative values each step. However, if by the time it reaches the value $(lb - ub)$ it has not been taken (i.e. its "window" is exhausted), the system must take it (if no other choice is available, as defined in the reaction transition).

Therefore, for non-deterministic specifications, the system has the choice of taking a transition if it has been active for at least a period of time equal to the lower bound and at most the upper bound. Furthermore, as it is characterized below, the definition of non-deterministic specifications does not exclude the system from the choice of taking a transition among any number of triggered transitions. Yet, if at least one triggering transition or one exhausted timing transition are triggered, one must be taken.

Definition. We define the labeled reaction relation, $\rightarrow \subseteq \Gamma \times \Sigma \times \Pi \times \Gamma$, as follows for an input event i:

0. Non-deterministic timing transitions:

$$\frac{((s,t,(lb,ub)),n) \in T \quad \text{and} \quad n \leq 0 \quad \text{and} \quad n > lb - ub}{(M,T,o) \xrightarrow{i,0} (M',T',o')}$$

1. Triggering transitions:

$$\frac{((s,t,e),n) \in T \quad \text{and} \quad (i \cup o) \vdash e}{(M,T,o) \xrightarrow{i,1} (M',T',o')}$$

2. Exhausted timing transitions:

$$\frac{((s,t,(lb,ub)),n) \in T \quad \text{and} \quad n = lb - ub}{(M,T,o) \xrightarrow{i,2} (M',T',o')}$$

where M', T', and o' have the same definitions as in the deterministic case.

Definition. The *transitive closure* of the relation $\xrightarrow{i}$ is defined as follows (for an input event i, which we explicitly show):

$$\begin{array}{lll} \gamma \xrightarrow{i,\lambda} \gamma' & \text{iff} & \gamma = \gamma' \\ \gamma \xrightarrow{i,u\omega} \gamma' & \text{iff} & \exists \gamma'',\ \exists u \in \Pi,\ \exists \omega \in \Pi^* :: \quad \gamma \xrightarrow{i,u} \gamma'' \quad \text{and} \quad \gamma'' \xrightarrow{i,\omega} \gamma'. \end{array}$$

Definition. The *fixpoint* of the relation $\xrightarrow{i}$ with respect of the state γ is γ' and is defined as follows:

$$\gamma \xrightarrow{i}{}^{\triangle} \gamma' \quad \text{iff} \quad \exists \omega \in \Pi^* :: \quad \gamma \xrightarrow{i,\omega} \gamma' \quad \text{and} \quad \nexists u \in \{1,2\}, \exists \gamma'' :: \ \gamma' \xrightarrow{i,u} \gamma''.$$

Definition. Let $\mathcal{S}$ be a proper modechart. Then the *set of computations* corresponding to an input event sequence $I \in \Sigma^*$ is the set $\mathcal{S}'(I) = \{ O \mid O = \mathcal{S}(I) \}$.

Definition. Let $\mathcal{S}$ be a proper modechart. Then the *behavior* of $\mathcal{S}$ is the set:

$$\mathcal{B}(\mathcal{S}) = \bigcup_{I \in \Sigma^*} \mathcal{S}'(I).$$

Definition. A proper, zero cycle-free modechart with a transition system $\langle \Gamma, \mapsto \rangle$ as defined above is said to be a *non-deterministic* Modechart specification.

The strong preemption axiom can be applied to the non-deterministic semantics just as easily as to the deterministic semantics. This concludes the presentation of the semantics. We now concentrate on special cases of specifications that where source of obscure of unwanted behaviors in the previous semantics.

6 Special Cases

In the past, a few semantic areas from the original definition of the Modechart language have been identified as being particularly difficult or problematic [Stu95]. Most of these problems stem from the inherent power of RTL. The class of computations allowed by the axioms and formulas derived from some specifications can sometimes be larger than the intended intuitive semantics of said specifications, with sometimes obscure or non-intuitive behavior being allowed. We focus now on how the operational semantics addresses those issues.

Zero cycles: A common semantic error in synchronous systems is related to the instantaneous nature of the reactions. Given that a reaction is considered to take place in zero steps, a cycle in the reaction could in principle produce an unbounded number of events. This phenomenon is defined in the context of Modechart as a *zero cycle*. It is also called an instantaneous loop or zero loop in the context of other synchronous languages. The definition of zero cycles in this semantics provides a compile-time checking procedure for ruling out those specifications with zero cycles.

Causality: A second common semantic problem is the issue of preservation of *causality* in specifications (called non-linearizability in the Modechart literature, and causality errors in others). This problem arises in the axiomatic semantics due to the power of the logic. Computations which are not causally coherent can still satisfy the axioms formulated by a specification. In our semantics, this problem is solved by only allowing the system to react to events that have already taken place or form part of the current input. Transitions can only be triggered within the reaction relation when their expressions are valid in the context of the current reaction input and accumulated output.

Negated events: Other synchronous languages such as ESTEREL [BG92] are more aggressive in allowing the compilation procedures to generate transitions based on the impossibility of certain "future" conditions to falsify or make expressions true (e.g. negated events) however, this approach requires a (sometimes costly) "look-ahead" in the compilation. Allowing these expressions also leads to frequent obscure causality errors (this problem is enhanced by the presence of compile-time non-deterministic tests, which cause the tree of possible "future" computations within a reaction to grow).

Preemption: The problem of *simultaneous conflicting exits*, i.e. when several conflicting transitions are triggered simultaneously, is taken care of in the semantics by way of the strong preemption axiom. The choice in the semantics is done by assigning higher priority to "higher" transitions in the hierarchy or by denoting it explicitly among transitions at the same level (Section 4). This converts a class of non-deterministic specifications to a class of (more useful in practice) deterministic ones by establishing a priority among conflicting transitions.

Implicit exits: The problem of *implicit exits* arises when a mode is restarted (exited and entered) in the same reaction and some of its *new* children are killed instead of *old* children being killed. This situation arises due to the lack of some mechanism to reflect the concept of "instances" of children in the RTL semantics. This problem does not arise in the semantics here defined because of the causal and cumulative nature of the reaction relation.

This semantics therefore covers these special cases in a simple way and introduces solutions for some of them with the strong preemption axiom. We now concentrate on the details of the compilation process.

7 Compiling Modechart Specifications

Once we have defined the operational behavior of the Modechart specification language, we introduce a compiler algorithm for deterministic specifications that produces a finite state machine which is guaranteed to implement the semantics of the specification. This compiler algorithm is essentially an symbolic execution of the two-level structure of the semantics, for all possible inputs. Reactions are compiled into states of a finite state machine and time-taking transitions are the directed edges among the states.

Section 7.2 describes an extension to the compiler to allow for a certain class of non-deterministic specifications which is useful in practice and still allows the compilation of the specification.

7.1 Deterministic Specification Compilation

The deterministic operational semantics presented in Section 4 introduces a relation that captures the essence of the execution of a deterministic Modechart, thus it yields itself naturally into the implementation of a compiler, presented below. In this presentation we do not keep track of the transitions for simplicity in the presentation. It is straightforward to add mechanisms to track them.

Algorithm: CompileMC
Input: MC (deterministic Modechart Spec.), E (input set);
Output: FSM (a Finite State Machine);

m := **closure**(**root**(MC));
% states is a list
% and current_state is a pointer in the list
states := **insert** (NIL, ⟨m, **tenter**(m)⟩);

```
        current_state := head(states);
        done := FALSE;
        while (not done) do
                new_states := ∅;
                for each i in 2^E do
                        new_states ∪ = fixpoint(current_state, i);
                end for
                if (new_states − states = ∅) and (current_state = last(states)) then
                        done := TRUE;
                else
                        states ∪= new_states;
                        current_state := next_state(current_state, states);
                end if
        end while
        output_fsm(states);
end Algorithm
```

We next define the functions **fixpoint** and **next_state**. The function **fixpoint** computes the fixpoint of a reaction, returning the final state after all triggered transitions have been taken with the given input. The function **next_state** picks the next unexplored state out of the state list (the one right after the one from which **current_state** was produced in the state list) and "applies" a time instant to it, producing the new current state to explore.

```
Function: fixpoint
Input: s (state), i (input signal event)
Output: f (state)
        f := s;
        while (any_triggered(f)) do
                f := take(f, any_triggered(f));
        end while
        return f;
end Function
```

The function **take** picks one transition out of all the ones triggered. If any set of pairwise conflicting transitions is triggered, then **take** must pick the transition with highest priority, or otherwise any one of the other consistent transitions that may be left.

It is easy to see that the size of the state space is exponential on the size of the input signal set, the modes in the specification and linear on the timing transition expressions. However, the actual state space and input signal set, as well as the timing expressions, are user specifications, thus it is reasonable to assume that the user will be careful in providing an input to the compiler such that no state space blowup occurs.

Obviously, it is required that the input specification be deterministic so that the **fixpoint** function above is deterministic as well (Lemma 6). Assuming that, the function **fixpoint** is computed in $O(TN \log N)$ time, where T is the size of the transition set, N is the combined size of the state space (inputs times mode combinations).

We assume the set operations are performed with efficient data structures in $O(N \log N)$, including computing the closure of a set of modes, which is linear with the size of the mode set (function **taken**). Assuming an efficient implementation of lists, the **next_state** function has a time complexity of $O(T)$. Considering the above and the fact that $T \ll N$, the time complexity of the algorithm above is approximately $O(N^2 \log N)$.

7.2 Non-Deterministic Specification Compilation

In order to provide support for a larger and more practical class of specifications, we now add a particular type of non-deterministic specifications, the run-time deterministic class of specifications. These are introduced to model the concept of instantaneous non-deterministic decisions within a modechart. They model very practical situations such as loop counters, decisions over contents or types of messages received or, in general control flow *instantaneous* decisions based on data items whose value is determined out of the scope of the modechart being implemented and only available at run-time. It is the specifier's job to decide at which level these transitions are going to be used. Some may decide to model some of this behavior with external events and keep control modes running in parallel to the specification to constrain the possible situations and model the environment more closely, while others may decide to leave more of the data processing as external behavior.

Each immediate child of a serial mode is allowed to have a number of transitions whose expression involve instantaneous evaluations of predicates over *external variables.* These external variables are typed variables whose value can be changed at any time by the environment. The predicates must be exhaustive and mutually exclusive. Intuitively, all the predicates must cover all the range of the variables they test and only one of the predicates must be true at the moment of checking them. We assume they are side-effect free and that the environment remains invariant during the (instantaneous interval of) time in which they are invoked.

For verification purposes, these transitions are replaced by timing transitions of the form $(0,0)$ before any property of the specification is attempted to be proved. For implementation, the functions **take** and **any_triggered** need to be redefined to generate calls to the instantaneous predicates in the transitions. As long as the predicates take an amount of time significantly smaller than the synchronous period, the system will implement the right behavior. The transition expression syntax, the definition of triggered transitions and the reaction relation are augmented accordingly (from the deterministic semantics) to support this class of specifications very easily.

8 Discussion and Future Work

This paper has introduced a formal operational semantics for the class of real-time specifications expressed in the Modechart formalism. This semantics provides the framework for defining a compiler for a subclass of the Modechart language. The original semantics of the Modechart language was described axiomatically in terms of Real-Time Logic. Introducing this semantics provides a more intuitive reference to the language as well as an implementation guide for it. The compiler presented is an efficient implementation of the language. It is based on an exhaustive symbolic execution of the specification within the synchronous model, which compiles communication and concurrency away, providing a simple, efficient finite state machine.

This semantics and compiler are a basis for future work on native code generation for Modechart specifications in a number of parallel and distributed architectures within the SARTOR project. Other areas of future work include support of the complete set of transition expressions originally defined in the language, support for code generation in a variety of general purpose programming languages, more efficient implementation of timing specifications and support for a distributed correctness-preserving implementation of specifications.

References

[BG92] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19:87–152, 1992.

[CHLR93] P. C. Clements, C. L. Heitmeyer, B. G. Labaw, and A.T. Rose. "MT:A Toolset for Specifying and Analyzing Real–Time Systems". In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 12–22, December 1993.

[Eme90] E. A. Emerson. Temporal and modeal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. MIT Press, 1990.

[Hal93] N. Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic, 1993.

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

[JM86] F. Jahanian and A.K. Mok. "Safety Analysis of Timing Properties of Real-Time Systems". *IEEE Transactions on Software Engineering*, 12(9):890–904, September 1986.

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

[JS88] F. Jahanian and D. Stuart. A method for verifying properties of modechart specifications. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1988.

[Mok85] A. Mok. SARTOR –a design environment for real-time systems. In *Proceedings 9th IEEE COMPSAC*, 1985.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[PMS95] C. Puchol, A.K. Mok, and D. Stuart. Compiling Modechart specifications. In *IEEE Real-Time Systems Symposium*, pages 256–265, December 1995.

[Stu95] Douglas A. Stuart. *"Specification and Analysis of Real-Time Systems"*. PhD thesis, The University of Texas at Austin, Department of Computer Sciences, 1995.

A Compatibility of the Semantics

Defining an operational semantics for the Modechart specification language provides a framework for a practical implementation of the language. We outline now the equivalence between the operational semantics and (a simplified version) of the axiomatic semantics presented in the original definition using RTL (Real-Time Logic [JM86]).

The axiomatic semantics essentially defines a set of assertions in the RTL logic for every specification. These formulas follow the tree structure of the specification, thus are defined recursively on this structure. The formulas include an "occurrence" function $\Theta(e, i) = t$ that assigns event occurrences to times, i.e. instance i of event e occurs at time t. The behavior of the specification is the set of solutions (including event occurrence assignments) to the set of assertions for the specification, for each possible input event in the system. We now present a simplified version of how to derive the set of assertions from a given specification.

The following notation is used through the rest of this section. The symbols i, j, m, n, t and t' denote non-negative integers. The symbols M, M' and N denote modes. The expression $M \to N$ denotes some transition from mode M to mode N. The following are notational conveniences (not defined here) that denote RTL formulas:

- $M(t, t')$: the system is in mode M between time t and t'.
- $M[t, t']$: the system enters mode M at time t remains active until and exits at time t'.
- $M[t, t')$: the system enters mode M at time t and remains active until at least time t'.
- $M(t, t']$: the system is in mode M at time t and remains active until and exits at time t'.
- C_i: denotes a formula capturing a triggering expression for transition i.
- B_i: denotes a formula capturing the semantics for bounds of timing transition i.

We emphasize that we are simplifying the semantics for the purposes of presentation. Further details about this semantics, the nature of the logic and formulas displayed below can be found in [JM94, Stu95].

0. For every (unique) triggering transition from M to N with triggering condition C we add the formula:
$$\forall t :: \; M(t, t) \; \wedge \; C \Rightarrow \exists j :: \; \Theta(M \to N, j) = t$$
1. For every (unique) timing transitionfrom M to N with timing condition (r, d) we add the formula:
$$\forall t :: \; M[t, t) \Rightarrow \exists t', \; \exists j :: \; \Theta(M \to N, j) = t' \; \wedge \; B$$
where $B = (t + r \leq t') \; \wedge \; (t' \leq t + d)$, capturing the bounds of the timing condition.
2. Let e_1, e_2 denote two mode transition expressions, a *mutual exclusion* constraint is a formula of the form:
$$\forall i, \forall j :: \; \Theta(e_1, i) \neq \Theta(e_2, j).$$
Every pair of transitions explicitly exiting a mode are subject to a mutual exclusion constraint.
3. If M is a serial mode (with $M_i \in \mathit{children}(M)$) and the system exits M at time t then we add the following assertion:
$$\forall t :: \; M(t, t] \Rightarrow \bigwedge_{i=1}^{n} (M_i(t, t) \Rightarrow M_i(t, t]).$$

4. If M is a parallel mode (with $M_i \in children(M)$) and the system exits M at time t then we add the following assertion:

$$\forall t :: \; M(t,t] \Rightarrow \bigwedge_{i=1}^{n} M_i(t,t].$$

5. Transition assertions for nested modes at level M (where mode M is serial). For each mode $M' \in children(M)$ the following formulas are added:

   - $\forall t :: \; M'(t,t) \Rightarrow \bigwedge_{i=1}^{n}(C_i \Rightarrow (T_i \;\vee\; M'(t,t])),$
   - $\forall t :: \; M'[t,t) \Rightarrow \bigwedge_{i=1}^{m} [(T_{n+i} \;\wedge\; B_i) \;\vee\; (\exists t' :: \; M'[t,t'] \;\wedge\; t' \leq t + d_i)],$
   - $\bigwedge_{i=1}^{n+m} T_i \Rightarrow R_i,$
   - $\forall t :: \; M'(t,t) \Rightarrow \{$ Level M' formulas $\}$.

   Where T_i is a predicate denoting the occurrence of the mode transition event corresponding to the i-th transition from M', and each R_i is the conjunction of:

   - the mode predicates denoting explicit exits for the i-th transitions, and
   - the mode predicates denoting explicit and implicit mode entries for the i-th transition.

6. Transition assertions for nested modes at level M (where mode M is parallel). The following formulas are added:

   - $\forall t :: \; \bigwedge_{i=1}^{n}(C_i \Rightarrow (T_i \;\vee\; M(t,t])),$
   - $\forall t :: \; M[t,t) \Rightarrow \bigwedge_{i=1}^{m} [(T_{n+i} \;\wedge\; B_i) \;\vee\; (\exists t' :: \; M[t,t'] \;\wedge\; t' \leq t + d_i)],$
   - $\bigwedge_{i=1}^{n+m} T_i \Rightarrow R_i,$

   and for each mode $M' \in children(M)$ the following formula is added:

   $$\forall t :: \; M'(t,t) \Rightarrow \{ \text{ Level } M' \text{ formulas } \}.$$

Definition. Let $\mathcal{S}$ be a modechart specification. Then $\mathcal{RTL}(\mathcal{S})$ is the set of RTL assertions associated with $\mathcal{S}$ according to the semantics defined above. Given an input event set sequence $I = i_1, i_2, \ldots$, an output event set sequence $O = o_1, o_2, \ldots$ is a *computation* of $\mathcal{S}$ iff

$$\forall t :: \Theta(i_t, t) \;\wedge\; \Theta(o_t, t) \;\wedge\; \mathcal{RTL}(\mathcal{S}) \quad \text{is satisfiable,}$$

where $\Theta(i_t, t)$ and $\Theta(o_t, t)$ (loosely) denote that all events in the input and output event sets at position t occur precisely at time t.

The semantics is completed by adding a clause to restrict the computations to consider, namely, the *linearizability axiom.* This axiom rules out all computations on which a total order relation capturing a causal relationship among events in a computation cannot be established. These computations are called *linearizable computations* and do not exist in the operational semantics by design. The order relation enforces a causality relationship and also eliminates the need for a series of auxiliary axioms (the *accountability assertions*) that were introduced in the original semantics to rule out computations with spontaneous generation of events. The following theorem (with proof outline) establishes the equivalence of the two semantics.

Theorem 1 *Let $\mathcal{S}$ be a modechart. Let $I \in \Sigma^*$ be an input set sequence, then O be a linerizable computation of $\mathcal{RTL}(\mathcal{S})$ iff $O' \in \Sigma_o^*$ is an output sequence according to the operational semantics and O and O' are the corresponding output sequences.*

Proof outline. Since O is a computation, that is, it satisfies all axioms in $\mathcal{RTL}(\mathcal{S})$, with the output o_t and input i_t corresponding to time t, we can use induction on the structure of the specification and for all points in time. We will construct an output sequence $O' \in \Sigma_o^*$ satisfying the theorem. The assertions corresponding to the system entering the root mode in the first instant are formed by applying either rule 5 or 6 to the root mode (depending on it being serial or parallel). These rules in turn prompt the entry of some or all of their children recursively. The event occurrences in the computation mark which transitions are taken in the transitions systems at the operational level.

The structure of the formulas in rules 5 and 6 capture the hierarchy of $\mathcal{S}$. That allows for the formulas of the children of the root mode to be (or not be) valid initially: all children of parallel modes are entered and at most one of the children of a serial mode is entered (which one in entered is determined by the actual occurrence assignments of events in the computation). Furthermore, since O is linearizable at each time instant, there exists an order relation between all those entry events. This is precisely the order in which the closure operation is calculated and th reaction is formed in the operational semantics.

If any transition belonging to the modes just entered is triggered and it is taken (rules 0, 1 and 2, and ignoring situations of self reinstatement such as self loops), it is taken at this point. The process of constructing O' is repeated just as described above for every time instant, creating each reaction's output.

Similarly, given an output sequence obtained from the operational semantics, we can obtain a corresponding linearizable computation by assigning events to the timeline according to the evolution of the state, first in the reaction transition system, and subsequently on the step transition system for each element of the output sequence. ■