Timed Modal Specification — Theory and Tools*

Kārlis Čerāns[1], Jens Chr. Godskesen[2] and Kim G. Larsen[2]

[1] Chalmers University of Technology, Sweden
[2] Aalborg University, Denmark

Abstract. In this paper we present the theory of *Timed Modal Specifications* (TMS) together with its implementation, the tool EPSILON. TMS and EPSILON are timed extensions of respectively *Modal Specifications* [7, 9] and the TAV system [6, 4]. Also, the theory of TMS is an extension of real–timed process calculi with the specific aim of allowing *loose* or *partial* specifications. This allows us to define a notion of *refinement*, generalizing in a natural way the classical notion of bisimulation.

1 Introduction

In this paper we present the theory of *Timed Modal Specifications* (TMS) together with its implementation, EPSILON. TMS and EPSILON are timed extensions of respectively *Modal Specifications* [7, 5, 9] and the TAV system [6, 4].

During the last few years various process calculi have been extended to include real–time in order to handle quantitative aspects of real–time systems. We mention the calculi defined in [14] and the ones defined in [12] and [3]. Common to these real–time calculi is that time is represented by some dense time domain. As argued in [7] process calculi are often too concrete in the sense that when a system has been specified the set of possible implementations are restricted to one and only one equivalence class of processes. Moreover, as correctness is given by the equivalence, the set of possible implementations remains constant under refinement. Hence, stepwise development methodologies are not well supported in classical process calculi. As an example, using the notation from [14], a disposable medium $M_d^{a,b}$ with some delay d between input and output can be specified by

$$M_d^{a,b} \stackrel{def}{=} a.\epsilon(d).\overline{b}$$

meaning that after input a the output $\overline{b}$ is first enabled after d time units. As a specification this may be too precise and perhaps all required of the medium is that it enables its output at some point in the time interval $[e, f]$ after a message

* This work has been supported by the Danish National Science Research Council project DART and the ESPRIT Basic Research Action 7166, CONCUR2.

has been received. Hence, using a suggestive notation, a more loose specification like

$$M^{a,b}_{e,f} \stackrel{def}{=} a.\epsilon[e,f].\bar{b}$$

is needed. Intuitively, we want $M^{a,b}_{e,f}$ to mean that after input a the output $\bar{b}$ may be enabled in the interval $[e,f)$ but is first guaranteed to be enabled after f time units. It is however impossible to give a loose specification, like $M^{a,b}_{e,f}$, using process calculi.

The theory of TMS is an extension of real–timed process calculi with the specific aim of allowing *loose* or *partial* specifications. The looseness of specifications is achieved by introducing two modalities to transitions: a *may* and a *must* modality, denoted by indices $\Diamond$ and $\Box$ respectively on actions. Using modalities the loosely specified media above can be specified by

$$S^{a,b}_{e,f} \stackrel{def}{=} a_{\Box}.(\epsilon(e).\bar{b}_{\Diamond} + \epsilon(f).\bar{b}_{\Box})$$

Obviously we expect $M^{a,b}_{d}$ to implement $S^{a,b}_{e,f}$ whenever $d \in [e,f]$.

Generalizing in a natural way the notion of bisimulation we introduce a *refinement ordering* $\lhd$ between timed modal specifications. Conceptually one may view a modal specifications S as the set of processes satisfying S, and the refinement ordering attempts to capture the corresponding set inclusion between specifications. Intuitively, we expect a modal specification S to refine a specification T when all transition *allowed* by S are also allowed by T, and, conversely, all transitions *required* by T are also required by S. As an example, we expect $S^{a,b}_{e,f} \lhd S^{a,b}_{g,h}$, whenever $g \leq e$ and $f \leq h$. In practical applications it is often advantageous to abstract from certain aspects when analyzing a system. In particular, internal computation will normally be considered unobservable, and in a first analysis of a large combined system explicit timing information may be irrelevant. We therefore introduce notions of refinements abstracting from time and internal computation.

The automatic refinement checking for TMS can be performed through adopting the techniques for checking bisimulation equivalences between networks of regular timed processes, developed in [13] and [10]. The tool EPSILON is based on these techniques. For untimed specifications the algorithms of EPSILON coincide with those of the TAV system.

We intend TMS and EPSILON to be useful during the process of design and implementation. In particular, we want to support system development through *stepwise refinement*. In a stepwise refinement development of a system the initial specification permits a wide range of implementations. An idealized development now consists in a series of small and successive refinements, each restricting the set of permitted implementations, until eventually an implementation can be extracted directly. Each refinement can be relatively small, consisting typically in the replacement of a single component of the current specification with more

concrete ones. To illustrate the first step of an idealized stepwise refinement development, suppose we have an initial specification of disposable media, say $S^{a,b}_{e,f}$. A possible refinement step could be to replace $S^{a,b}_{e,f}$ by[3]

$$(M^{a,c}_d \mid S^{c,b}_{e-d,f-d})\backslash c \tag{1}$$

Obviously, we expect (1) to be a correct refinement of $S^{a,b}_{e,f}$ since the total delay is still within the interval from e to f. Using the verification tool EPSILON we can automatically prove the correctness of this refinement step; more precisely, we can prove $(M^{a,c}_d \mid S^{c,b}_{e-d,f-d})\backslash c \trianglelefteq S^{a,b}_{e,f}$ where $\trianglelefteq$ indicates a refinement abstracting from internal events.

In general we would like the correctness of a refinement step to be immediately implied by the correctness of the refinement of the replaced component by the one replacing it. That is, we want to support *compositional* verification and hence the refinements to be preserved by composition as much as possible. As an example, we want to be able to infer that the combined medium $(M^{a,c}_d \mid M^{c,b}_{f-d})\backslash c$ is a correct implementation of $S^{a,b}_{e,f}$ directly from (1) and the obvious fact that $M^{c,b}_{f-d} \trianglelefteq S^{c,b}_{e-d,f-d}$. Also, TMS and the various notions of refinements makes our correctness proofs far more *general* than proofs within standard (timed) process calculi. That is, a single proof in TMS may capture a whole (possibly) infinite family of proofs in process calculi. For instance, in the example above the proof of (1) establishes in a single refinement the fact that $(M^{a,c}_d \mid M)\backslash c$ is a correct implementation of $S^{a,b}_{e,f}$ whenever M is chosen from the infinite set $\{M^{c,b}_g \mid g \in [e-d, f-d]\}$.

2 Modal Specifications

Modal Specifications (MS) were introduced in [9] in order to allow loose or partial specifications in a process algebraic framework. Semantically, MS are given an operational interpretation imposing restrictions on the transitions of possible implementations by telling which transitions are *necessary* and which are *admissible*. The transition systems for MS therefore have *two* transition relations: $\longrightarrow_\Box$, describing the *required* transitions, and $\longrightarrow_\Diamond$, describing the *allowed* transitions.

Definition 1. *A modal transition system is a structure* $(\mathcal{S}, \mathcal{A}, \longrightarrow_\Box, \longrightarrow_\Diamond)$, *where* $\mathcal{S}$ *is a set of specifications,* $\mathcal{A}$ *is set of actions and* $\longrightarrow_\Box$, $\longrightarrow_\Diamond \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, *satisfying the condition* $\longrightarrow_\Box \subseteq \longrightarrow_\Diamond$.

The condition $\longrightarrow_\Box \subseteq \longrightarrow_\Diamond$ simply says that anything required is also allowed. Clearly, the more a specification allows and the less it requires the looser the specification is. This idea is formalized in the following notion of refinement

[3] As usual we take $S\backslash c$ to mean S but restricted from c.

Definition 2. *A refinement $\mathcal{R}$ is a binary relation on S such that whenever $S\mathcal{R}T$ and $a \in \mathcal{A}$ then the following holds*

$$\text{Whenever } S \xrightarrow{a}_{\Diamond} S', \text{ then } T \xrightarrow{a}_{\Diamond} T' \text{ for some } T' \text{ with } S'\mathcal{R}T'$$
$$\text{Whenever } T \xrightarrow{a}_{\Box} T', \text{ then } S \xrightarrow{a}_{\Box} S' \text{ for some } S' \text{ with } S'\mathcal{R}T'$$

We say that S refines T whenever (S,T) is contained in some refinement $\mathcal{R}$. In this case we write $S \lhd T$.

The behaviour of processes themselves is assumed to be given in terms of a standard labeled transition system, which may be seen as a special case of MS with all transitions being required (i.e. $\longrightarrow_{\Box}=\longrightarrow_{\Diamond}$). In this case the new notion of refinement coincides with the classical notion of bisimulation.

3 Timed Modal Specifications

The language we use to describe timed processes is the real–time calculus TCCS of Wang [14]. This calculus is essentially CCS [11] extended with a delay construct $\epsilon(d).P$ where d is a non–negative real.

The semantics of TCCS applies the "two–phase functioning principle" outlined in [12]. That is, the behaviour of a real–time system is divided into two phases: one phase in which the components of the system agrees to let time pass, and a second phase in which the system computes by performing (instantaneous) actions. In the operational semantics of TCCS this is reflected by having transitions labeled by either *action names* or *delays* being positive reals.

Similar to the MS extensions of classical Process Algebra we offer in the following a MS extension of the real–time calculus TCCS.

3.1 Informal Semantics

First consider the TCCS process term $a.P$. As a specification this term is quite specific, in that it *requires* an implementation *at any moment* to be able to perform the action a and implement P thereafter. This interpretation is formalized by the following *required* transitions for $a.P$

$$a.P \xrightarrow{a}_{\Box} P \qquad a.P \xrightarrow{\epsilon(d)}_{\Box} a.P \text{ for all } d > 0$$

In the following we shall adopt the notation $a_{\Box}.P$ for $a.P$.

To obtain looseness we introduce a new *may* prefix construct $a_{\Diamond}.P$. As a specification this term will at any moment *allow* (without requiring) an implementation to have an a–transition as long as the result of such a transition implements P. Formally this is reflected in the following transitions

$$a_{\Diamond}.P \xrightarrow{a}_{\Diamond} P \qquad a_{\Diamond}.P \xrightarrow{\epsilon(d)}_{\Box} a_{\Diamond}.P \text{ for all } d > 0$$

Informally, it should be rather clear that $a_\diamond.P$ is a fairly loose specification covering a range of different implementations. In particular, $a_\diamond.P$ will contain as typical implementations the processes $a.P$, nil and, for any d, $\epsilon(d).a.P$.

In order to support compositional verification we want the ability to combine specifications with respect to the process constructs of TCCS. Here we recall that the real–time calculus TCCS applies the *maximal progress* assumption: i.e. if a process is in a state in which it can perform internal computations then time is not allowed to pass. As an example, consider the following combined specification

$$(\overline{a}_\diamond.S \mid a_\diamond.T)\backslash a \tag{2}$$

Implementations of (2) should essentially be of the form $(P \mid Q)\backslash a$, where P implements $\overline{a}_\diamond.S$ and Q implements $a_\diamond.T$. From the discussion above we already know three typical implementations of a may–prefixed term. Hence, typical implementations of (2) can be of the form

$$(\epsilon(d).\overline{a}.P' \mid \epsilon(d').a.Q')\backslash a \tag{3}$$

Now, the operational semantics to be given to (2) should capture precisely these desired implementations. Choosing $d = d' = 0$ in (3), we obtain a desired implementation which — due to the maximal progress assumption — can perform nothing but a τ–transition. Thus, it is clear that $(\overline{a}_\diamond.S \mid a_\diamond.T)\backslash a$ should (at least) allow τ–transitions, whereas delay–transitions can not be required. In general, however, implementations of the form (3) are not immediately able to perform a τ–transition, rather a delay of $max\{d, d'\}$ time units must elapse. Hence, $(\overline{a}_\diamond.S \mid a_\diamond.T)\backslash a$ cannot insist on an immediate τ–transition, and should on the other hand allow implementations to delay. In summary, $(\overline{a}_\diamond.S \mid a_\diamond.T)\backslash a$ is given the following transitions

$$(\overline{a}_\diamond.S \mid a_\diamond.T)\backslash a \xrightarrow{\tau}_\diamond (S \mid T)\backslash a$$

$$(\overline{a}_\diamond.S \mid a_\diamond.T)\backslash a \xrightarrow{\epsilon(d)}_\diamond (\overline{a}_\diamond.S \mid a_\diamond.T)\backslash a$$

Now it is reasonable to expect that $(\overline{a}_\diamond.S \mid a_\diamond.T)\backslash a$ should be equivalent to the specification $\tau_\diamond.((S \mid T)\backslash a)$. This means that the semantics of specifications of the form $\tau_\diamond.S$ should be given by the following two axioms

$$\tau_\diamond.S \xrightarrow{\tau}_\diamond S \qquad\qquad \tau_\diamond.S \xrightarrow{\epsilon(d)}_\diamond \tau_\diamond.S$$

3.2 Formal Syntax and Semantics

After the introductory discussion, we are now ready to formally present the syntax and semantics for TMS. As in CCS, we assume a set $\Lambda = \Delta \cup \bar{\Delta}$ with $\bar{\bar{\alpha}} = \alpha$ for all $\alpha \in \Lambda$, ranged over by α, β representing external actions, and a

$$\frac{-}{a_\Diamond.S \xrightarrow{a}_\Diamond S} \qquad \frac{-}{a_\Box.S \xrightarrow{a}_m S} \qquad \frac{S \xrightarrow{a}_m S'}{\epsilon(0).S \xrightarrow{a}_m S'} \qquad \frac{S \xrightarrow{a}_m S'}{S\backslash L \xrightarrow{a}_m S'\backslash L}\ a \notin L$$

$$\frac{S \xrightarrow{a}_m S'}{S+T \xrightarrow{a}_m S'} \qquad \frac{T \xrightarrow{a}_m T'}{S+T \xrightarrow{a}_m T'} \qquad \frac{S \xrightarrow{a}_m S'}{X \xrightarrow{a}_m S'}\ [X \stackrel{def}{=} S] \in \mathcal{E}$$

$$\frac{S \xrightarrow{a}_m S'}{S|T \xrightarrow{a}_m S'|T} \qquad \frac{T \xrightarrow{a}_m T' \quad S \xrightarrow{\alpha}_m S' \quad T \xrightarrow{\overline{\alpha}}_m T'}{S|T \xrightarrow{a}_m S|T' \quad S|T \xrightarrow{\tau}_m S'|T'}$$

Table 1. Action Rules for TMS ($m \in \{\Box, \Diamond\}$).

distinct symbol τ representing internal actions. As usual $\overline{L} = \{\bar{\alpha} \mid \alpha \in L\}$ for any $L \subseteq \Lambda$. We use $\mathcal{A}ct$ to denote the set $\Lambda \cup \{\tau\}$ ranged over by a, b.

We adopt a two–phase syntax to describe networks of regular TMS. First, regular TMS expressions are generated by the following grammar

$$E ::= nil \mid \epsilon(d).E \mid a_\Diamond.E \mid a_\Box.E \mid E + E \mid X$$

where X ranges over a finite set of variables $\mathcal{V}ar$ and d is a positive real. We shall assume that process variables are defined by a recursive equation system

$$\mathcal{E} = \{X \stackrel{def}{=} E_X \mid X \in \mathcal{V}ar\}$$

where all variables in E_X are guarded in the sense that each variable occurrence is within the scope of an action or delay prefix. Networks of regular TMS' are composite expressions of the form $(S_1 \mid \ldots \mid S_n)\backslash L$ where S_i are regular TMS and $\mid$ and $\backslash L$ denote CCS parallel composition and restriction respectively. We will use S and T to range over (networks of regular) specifications.

We offer a modal transition semantics for TMS. We present the transition rules in two groups: rules for actions in Table 1 and rules for delays in Table 2.[4] It should be rather clear from Table 1 and 2 that we indeed have defined a modal transition system, i.e. $\longrightarrow_\Box \subseteq \longrightarrow_\Diamond$.

The side condition for the delay rule of parallel composition is to guarantee that specifications satisfy the following two *maximal progress* assumptions

- *A timed specification will not allow delays if it requires an internal action* τ.
- *A timed specification will not require delays if it allows an internal action* τ.

[4] In Table 2, we use d to stand for a non-zero real, this implies that an $\epsilon(0)$–transition can never be inferred by the inference rules. However, we shall apply the convention that $S \xrightarrow{\epsilon(0)}_m S$ for all S.

$$\frac{-}{nil \xrightarrow{\epsilon(d)}_m nil} \qquad \frac{-}{\alpha_n.S \xrightarrow{\epsilon(d)}_m \alpha_n.S} \qquad \frac{-}{\tau_\Diamond.S \xrightarrow{\epsilon(d)}_\Diamond \tau_\Diamond.S}$$

$$\frac{-}{\epsilon(c+d).S \xrightarrow{\epsilon(d)}_m \epsilon(c).S} \qquad \frac{S \xrightarrow{\epsilon(d)}_m S'}{\epsilon(c).S \xrightarrow{\epsilon(c+d)}_m S'} \qquad \frac{S \xrightarrow{\epsilon(d)}_m S'}{S\backslash L \xrightarrow{\epsilon(d)}_m S'\backslash L}$$

$$\frac{S \xrightarrow{\epsilon(d)}_m S' \qquad T \xrightarrow{\epsilon(d)}_m T'}{S+T \xrightarrow{\epsilon(d)}_m S'+T'} \qquad \frac{S \xrightarrow{\epsilon(d)}_m S'}{X \xrightarrow{\epsilon(d)}_m S'} \; [X \stackrel{def}{=} S] \in \mathcal{E}$$

$$\frac{S \xrightarrow{\epsilon(d)}_m S' \qquad T \xrightarrow{\epsilon(d)}_m T'}{S \mid T \xrightarrow{\epsilon(d)}_m S' \mid T'} \; Sort_{m^c}(d,S) \cap \overline{Sort_{m^c}(d,T)} = \emptyset$$

Table 2. Delay Rules for TMS ($m, n \in \{\Box, \Diamond\}$ and $\Box^c = \Diamond$, $\Diamond^c = \Box$).

The two conditions above are formalized by means of two functions $Sort_\Box$ and $Sort_\Diamond$ defined in Table 3. Intuitively, $Sort_\Box(d,S)$ ($Sort_\Diamond(d,S)$) includes all external actions that S requires (allows) an implementation to enable within d time units, hence the side condition $Sort_\Box(d,S) \cap \overline{Sort_\Box(d,T)} = \emptyset$ ($Sort_\Diamond(d,S) \cap \overline{Sort_\Diamond(d,T)} = \emptyset$) means that implementations of S and T will not necessarily (can not possibly) be able to communicate within d time units.

Definition 3. *Define $Sort_\Box(0,S) = Sort_\Diamond(0,S) = \emptyset$ and define $Sort_\Box(c,S)$ and $Sort_\Diamond(c,S)$ for $d \neq 0$ to be the least sets satisfying the equations*[5] *given in Table 3.*

The following properties are obvious generalizations of properties in [14].

Proposition 4.

(Maximal progress) $\exists S'.\ S \xrightarrow{\tau}_m S'$ *implies* $\neg\exists d, S''.\ S \xrightarrow{\epsilon(d)}_{m^c} S''$

(Time determinism) $S \xrightarrow{\epsilon(d)}_m S'$ *and* $S \xrightarrow{\epsilon(d)}_m S''$ *implies* $S' = S''$

(Percistency) $\exists S', T.\ S \xrightarrow{\epsilon(d)}_m S'$ *and* $S \xrightarrow{\alpha}_m T$ *implies* $\exists T'.\ S' \xrightarrow{\alpha}_m T'$

(Time continuity) $S \xrightarrow{\epsilon(c+d)}_m S''$ *iff* $\exists S'.\ S \xrightarrow{\epsilon(c)}_m S' \xrightarrow{\epsilon(d)}_m S''$

(Transition liveness) $\exists S'.\ S \xrightarrow{\tau}_m S'$ *or* $\exists S'', d > 0.\ S \xrightarrow{\epsilon(d)}_{m^c} S''$

[5] In Table 3, $c \dot{-} d$ is defined to be $c - d$ if $c > d$, 0 otherwise.

$$Sort_m(d, nil) = Sort_\Box(d, \alpha_\Diamond.S) = \emptyset$$

$$Sort_\Box(d, \alpha_\Box.S) = \{\alpha\}$$

$$Sort_\Diamond(d, \alpha_m.S) = \{\alpha\}$$

$$Sort_m(d, \epsilon(c).S) = Sort_m(d \dot{-} c, S)$$

$$Sort_m(d, S + T) = Sort_m(d, S) \cup Sort_m(d, T)$$

$$Sort_m(d, S \mid T) = Sort_m(d, S) \cup Sort_m(d, T)$$

$$Sort_m(d, X) = Sort_m(d, S) \quad [X \stackrel{def}{=} S] \in \mathcal{E}$$

Table 3. Definition of $Sort_\Box$ and $Sort_\Diamond$ ($m \in \{\Box, \Diamond\}$).

4 Abstracting Refinements

As already mentioned, TMS together with the two modal transition relations $\longrightarrow_\Box$ and $\longrightarrow_\Diamond$ defined in Table 1 and 2 constitutes a modal transition system. As such, we may readily apply the general notion of refinement from Definition 2 to TMS. However, this refinement will often be too strong in practical applications. In particular, a refinement based directly on $\longrightarrow_\Box$ and $\longrightarrow_\Diamond$ will be completely sensitive to internal computation of systems. In contrast, practical applications often need to abstract away from internal computation of systems. Also, when reasoning about large combined real–time systems, explicit timing information may in a first analysis be unimportant, in which case a time–abstracting refinement will suffice. Though such a refinement yields no information about the timing behaviour of the overall system, it will demand proper interaction between the timing properties of the components of the system.

Abstracting refinements with the above properties will be obtained through the definition of similarly abstracting versions of the modal transition relations $\longrightarrow_\Box$ and $\longrightarrow_\Diamond$. The abstracting transition relations will in all cases be generated by an abstraction function Φ on labels. Now, recall that the modal transitions for TMS are labeled by elements of the following set $\mathcal{L} = \mathcal{A}ct \cup \mathcal{D}elay$ where $\mathcal{D}elay = \{\epsilon(d) \mid d > 0\}$. An *abstraction function* Φ maps sequences of labels into a single label. More precisely, an abstraction function Φ is a *partial* function of the following type $\Phi : \mathcal{L}^* \hookrightarrow \mathcal{L} \cup \{\varepsilon\}$ The partiality of Φ indicates that not all sequences of labels makes sense as abstract actions. Also, $\Phi(s) = \varepsilon$ signifies that s is unobservable.

Given an abstraction function we can now define abstracting transition relations.

Definition 5. *Let Φ be an abstraction function. Then the abstracting transitions relations $\longrightarrow^{\Phi}_{\Box}$ and $\longrightarrow^{\Phi}_{\Diamond}$ are defined as (m ranges over $\Box$ and $\Diamond$): $S \xrightarrow{b}{}^{\Phi}_{m} S'$ whenever $S \xrightarrow{a_1}_m \cdots \xrightarrow{a_n}_m S'$ with $\Phi(a_1 \ldots a_n) \simeq b$ for some $a_1 \ldots a_n$.* [6]

It is easy to verify that TMS equipped with any abstracting transition relations does indeed constitute a modal transition system in the sense of Definition 1. Hence, we may apply the notion of refinement from Definition 2. We now present the abstraction functions which will induce the desired τ– and time–abstracting refinements.

Definition 6. *The τ–abstracting function Φ_τ is defined as follows*

$$\Phi_\tau(\tau^k) = \varepsilon\,,\ k \geq 0$$
$$\Phi_\tau(\tau^{k_0}\epsilon(d_1)\tau^{k_1}\ldots\epsilon(d_n)\tau^{k_n}) = \epsilon(\textstyle\sum_i d_i)\,,\ k_j \geq 0$$
$$\Phi_\tau(\tau^k\alpha\tau^j) = \alpha\,,\ k,j \geq 0$$

Whenever S refines T with respect to Φ_τ we say that S weakly refines T. We write $S \trianglelefteq T$ in this case.

Definition 7. *The time–abstracting function Φ_ϵ is defined as follows*

$$\Phi_\epsilon(s) = \varepsilon\,,\ s \in \mathcal{D}elay^*$$
$$\Phi_\epsilon(s_1 a s_2) = a\,,\ s_1, s_2 \in \mathcal{D}elay^*$$

Whenever S refines T with respect to Φ_ϵ we say that S is a time–abstracted refinement of T. We write $S \overset{\bullet}{\lhd} T$ in this case.

Definition 8. *The τ– and time–abstracting function $\Phi_{\tau\epsilon}$ is defined as follows*

$$\Phi_{\tau\epsilon}(s) = \varepsilon\,,\ s \in (\mathcal{D}elay \cup \{\tau\})^*$$
$$\Phi_{\tau\epsilon}(s_1 \alpha s_2) = \alpha\,,\ s_1, s_2 \in (\mathcal{D}elay \cup \{\tau\})^*$$

Whenever S refines T with respect to $\Phi_{\tau\epsilon}$ we say that S is a weak time–abstracted refinement of T. We write $S \overset{\bullet}{\trianglelefteq} T$ in this case.

In contrast to $\lhd$ and $\trianglelefteq$, $\overset{\bullet}{\lhd}$ and $\overset{\bullet}{\trianglelefteq}$ are *not* preserved by the constructs of TMS. However, we conjecture that the full abstractness result for time abstracted equivalences proved in [10] extends to the above presented notions of time–abstracted refinements. That is, the largest pre–orders contained in $\overset{\bullet}{\lhd}$ and $\overset{\bullet}{\trianglelefteq}$ which are also preserved by parallel composition will be $\lhd$ and $\trianglelefteq$ respectively. The four refinements are related as follows: $\lhd \subseteq \trianglelefteq \subseteq \overset{\bullet}{\trianglelefteq}$ and $\lhd \subseteq \overset{\bullet}{\lhd} \subseteq \overset{\bullet}{\trianglelefteq}$.

[6] For expressions e_1 and e_2, $e_1 \simeq e_2$ holds if both expressions are defined and have the same value.

Example 1. Recall the combined media specification (1) from the introduction[7] *The following abstracting refinements may be deduced*

$$(M_d^{a,c} \mid S_{e-d,f-d}^{c,b})\backslash c \trianglelefteq S_{e,f}^{a,b} \quad (M_d^{a,c} \mid S_{e-d,f-d}^{c,b})\backslash c \stackrel{\bullet}{\trianglelefteq} S_{0,0}^{a,b}$$

The verification tool EPSILON supports automatic verification based on all four types of refinements. More precisely, given two networks of regular TMS in which all delays are naturals, EPSILON contains algorithms for deciding any of the four refinements.[8]

5 The Train Crossing

In this section we apply the tool EPSILON on a small example of a train crossing. Similar examples can be found elsewhere in the literature, e.g. [2].

The Train Crossing consists of four components: the crossing (Cr), a train (T), the gate (G) and the controller (Ct), see Figure 1. When a train approaches the crossing it signals the controller before entering the crossing. The controller begins to close the gate with some delay after the approaching of a train has been signalled and starts opening the gate a few minutes after. The train is assumed to leave the crossing. We use the events down and up to signal when the gate is up and down and we let inside and outside signal when the train enters and leaves the crossing. We force time lock on the external events to ensure they are observed as soon as they are enabled.

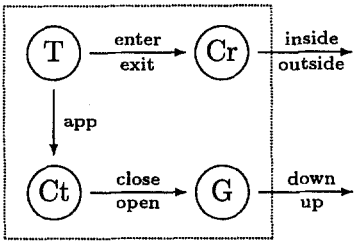

Fig. 1. The Train Crossing.

The short hand for *time lock* on events is

[7] We omit trailing *nil*'s, i.e. a_m abbreviates $a_m.nil$.

[8] Actually, the algorithms would apply also for networks of regular TMS with delays being positive rationals, as we may simply multiply all delays with some rational Q in order to end up with a comparison of processes with delays being naturals.

$$a@S \stackrel{def}{=} a_{\Box}.S + \tau_{\Box}.a@S$$

Due to maximal progress time is blocked until a happens. In the example we heavily use the following very loose specification

$$Uni(L) \stackrel{def}{=} \prod_{a \in L \cup \{\tau\}} a_{\Diamond}.Uni(L)$$

where Π denotes an n–ary parallel composition and $L \subseteq Act$. Another short hand is

$$Until(S, d, e, T) \stackrel{def}{=} S + \epsilon(d).\tau_{\Diamond}.T + \epsilon(e).\tau_{\Box}.T$$

Intuitively a process P satisfies $Until(S, d, e, T)$ if it must satisfy S for d time units, if it can choose to satisfy T instead after d time units and when exactly e time units has passed it must choose to satisfy either S or T. The last short hand is

$$Interval(a, d, e, S) \stackrel{def}{=} \epsilon(d).a_{\Diamond}.S + \epsilon(e).a_{\Box}.S$$

Intuitively, $Interval(a, d, e, S)$ is satisfied by processes being inactive for d time units after which they may enable a, and when e time units has passed a must be enabled. After having performed a the processes must satisfy S.

The Train Crossing is specified in Figure 2 as a parallel composition of the crossing, the gate, a train and the controller. `:=:` is used for equations, `;` denotes the must modality and `?` the may modality. `in(a)` and `out(a)` represents an action and its corresponding co–action and restriction from actions is defined by `\L` where `L` is a list of actions.

The *crossing*, can be in one of two states, either the train is inside the crossing or it is outside. Initially there is no train in the crossing. When a train enters (leaves) the crossing `inside` (`outside`) is signalled immediately. The *gate* is initially opened but the controller can close the gate. When the gate is closed, `down` is signalled immediately and the gate waits to be opened by the controller. As soon as the gate is open up is signalled. It will take at most `X` time units to close (open) the gate. The *train* can be in one of three states. Either it may be approaching, is entering or is leaving the crossing. The state where it may be approaching is the initial one. The train is supposed to enter the crossing in the interval from `A` to `B` after the approaching has been signalled and it is guaranteed to leave the crossing no later than `C` time units after it entered. In the initial state the *controller* waits for the approaching of a train. If a train approaches he starts closing the gate no later than `U` time units after the approaching was signalled. Then he waits for `V` time units before opening the gate.

The specifications used in the analysis can all be found in Figure 3. First we find acceptable values for `X,A,B,C,U` and `V`. Using EPSILON one can prove that

```
OutCross :=: in(enter);inside@InCross
InCross :=: in(exit);outside@OutCross

GateOpen(X) :=: in(close);Until(nil,0,X,GateClosed(X))
GateClosed(X) :=: down@in(open);Until(nil,0,X,up@GateOpen(X))

TrainApp(A,B,C) :=: out(app)?TrainEnter(A,B,C)
TrainEnter(A,B,C) :=: Interval(out(enter),A,B,TrainExit(A,B,C))
TrainExit(A,B,C) :=: Interval(out(exit),0,C,TrainApp(A,B,C))

ContrApp(U,V) :=: in(app);ContrClose(U,V)
ContrClose(U,V) :=: Interval(out(close),0,U,ContrOpen(U,V))
ContrOpen(U,V) :=: V;out(open);ContrApp(U,V)

TC(X,A,B,C,U,V) :=:
(OutCross/GateOpen(X)/TrainApp(A,B,C)/ContrApp(U,V))\L
```

Fig. 2. Specification of the Train Crossing. L is `[enter,exit,close,open,app]`.

```
Spec1 :=: down?inside?outside?up?Spec1

Spec2(M,N) :=: DownUp(M,N)/Uni([inside,outside])
Spec3(M,N) :=: DownUp(M,N)/InOut/MayTime
DownUp(M,N) :=: down?Interval(up,M,N,DownUp(M,N))
InOut :=: inside?outside?InOut
MayTime :=: tau?MayTime

Spec4(P) :=: Down(P)/Uni([inside,outside,up])
Down(P) :=: down?P;Down(P)
```

Fig. 3. Specifications.

`TC(1,3,4,1,1,6)` $\overset{\bullet}{\trianglelefteq}$ `Spec1`. Actually, whenever `U + X < A` and `B + C < V` we can prove, using EPSILON, that `TC(X,A,B,C,U,V)` weak time abstracted refines `Spec1`.

Being more explicit about the timed behaviour of the train crossing it will be of interest to reason about the intervals in which the external events can happen. Suppose we want to know about the relationship between the time moments of the closing and the opening of the gate. `Spec2(M,N)` specifies that the opening of the gate is guaranteed to occur in the interval from `M` to `N` after it was lowered. Whenever $\texttt{M} \leq 5$ and $\texttt{N} \geq 7$ it turns out that

$$\texttt{TC(1,3,4,1,1,6)} \trianglelefteq \texttt{Spec2(M,N)} \tag{4}$$

Hence, due to the strongest of these specifications, `Spec2(5,7)`, the gate must be opened no later than 7 time units after it was closed. Actually, for the values of `M` and `N` mentioned above, we can prove that `TC(1,3,4,1,1,6)` weak refines the even stronger property `Spec3(M,N)`.

```
FastContr :=: in(app);out(close);out(open);FastContr
SlowContr :=: in(app);1;out(close);6;out(open);SlowContr
```

Fig. 4. Implementations.

Under the assumption that a proof of `TC(1,3,4,1,1,6)` $\trianglelefteq$ `Spec3(M,N)` has already been given a direct proof of (4) is not needed. As $\trianglelefteq$ is preserved by parallel composition, we can obtain the result in an alternative and compositional manner. It suffices to prove that `InOut/MayTime` weak refines `Uni([inside,outside])` in order to immediately conclude `Spec3(M,N)` $\trianglelefteq$ `Spec2(M,N)`. The rest of the proof is due to transitivity of $\trianglelefteq$.

Suppose we want to determine the values of `P` for which `TC(1,3,4,1,1,6)` weak refines `Spec4(P)`. Intuitively, `Spec4(P)` specifies that the frequency between any two consecutive closings of the gate must be at least `P` time units. Whenever $\texttt{P} \leq 5$, we can prove

$$\texttt{TC(1,3,4,1,1,6)} \trianglelefteq \texttt{Spec4(P)} \tag{5}$$

We can either prove (5) directly in EPSILON or alternatively, for any $\texttt{P} \leq 5$, we could prove instead that

$$\texttt{Spec2(P,7)} \trianglelefteq \texttt{Spec4(P)} \tag{6}$$

and then take advantage of the transitivity of $\trianglelefteq$. Due to compositionality, (6) holds since `DownUp(P,7)` weak refines `Down(P)/Uni([up])` for any $\texttt{P} \leq 5$.

Implementations of the Train Crossing may now be found simply by substituting each of the four components with some timed process (strongly or weakly) refining the component. Due to the looseness of the specification of the components[9] each component will have several inequivalent implementations. For instance, as implementation of the controller one could choose one of the processes in Figure 4. Clearly `FastContr` and `SlowContr` are inequivalent and obviously both

[9] Except for the Crossing of course.

`FastContr` and `SlowContr` refines `ContrApp(1,6)`. Finally, to emphasize the generality of correctness proofs carried out within the framework of TMS, it follows that no matter which implementation we decide upon it is ensured, as a consequence of the compositionality of verification in TMS, that any such implementation is guaranteed to satisfy all the properties above refined by `TC(1,3,4,1,1,6)`.

6 Algorithms for Refinement Checking

In the full version of this paper we give a description of the algorithms for refinement checking (i.e. with respect to $\lhd$, $\trianglelefteq$, $\overset{\bullet}{\lhd}$ and $\overset{\bullet}{\trianglelefteq}$) as they are implemented in the verification tool EPSILON. The description is based on timed graphs and the associated region graphs technique [1].

One should observe that the timed modal specifications to be checked under the refinement relations are essentially infinite state (due to the density of the underlying time domain — non–negative reals), so we cannot perform the needed refinement checking directly from their definitions. However, there are some already known techniques which have proven useful in deciding various algorithmic properties of quantitative timed systems over dense time, most notably the *region graph technique* due to [1]. It turns out to be possible to make some of these methods work also for solving the modal refinement checking problems. In fact, pragmatically the refinement problems for TMS can be solved in a quite similar way, as the corresponding *bisimulation equivalence* problems for the networks of regular timed processes, investigated in [13] ("time-sensitive" cases) and [10] (time-abstracted cases).

Given two timed modal specifications S and T (assumed, as usual, to have *integral* delay constants in their definitions[10]) and an instance of the refinement relation `ref` ($\lhd$, $\overset{\bullet}{\lhd}$, $\trianglelefteq$ or $\overset{\bullet}{\trianglelefteq}$), the deciding algorithm follows the following principal schema:

- define a symbolic (finite) transition system $\mathcal{T}_{S,T}$ (usually with several transition relations) characterizing the transition systems of the given modal specifications S and T (in the time abstracted cases ($\overset{\bullet}{\lhd}$ and $\overset{\bullet}{\trianglelefteq}$) it is possible to define symbolic transition systems for specifications S and T separately, and combine (multiply as labeled graphs) them afterwards);
- define a functional $\mathcal{F}^{\mathrm{ref}}_{S,T}$ on the powerset of $\mathcal{X}_{S,T}$ (the set of *states* of $\mathcal{T}_{S,T}$) having the set $R_{S,T} \subseteq \mathcal{X}_{S,T}$ of "good" (= representing the refinement pairs) symbolic states as a *greatest fixpoint* of $\mathcal{F}^{\mathrm{ref}}_{S,T}$;

[10] Observe that in all examples in Section 5 only integral delay constants are found in the specifications. The refinement checking for rational timed specifications can be reduced to the integral case by a simple scaling of the time axis. It should also be clear that the integrality of the initial delay bounds still admits the runtime delays of arbitrary *real* length.

- decide whether the initial state of the symbolic transition system $\mathcal{T}_{S,T}$ falls into the subset $R_{S,T}$, using for this purpose efficient local correctness checking technique over $\mathcal{T}_{S,T}$ and $\mathcal{F}^{\mathtt{ref}}_{S,T}$, as described in [8].

In the full paper, we provide the details regarding the computation of symbolic transition systems and relations $\mathcal{F}^{\mathtt{ref}}$ for all four refinements `ref`. Regarding the correctness checking technique we just note that it runs with polynomial worst–case time complexity in the size of symbolic transition system. In "good" cases the "locality" of the used technique gives considerable improvements wrt. the global techniques since it will at most explore the part of $\mathcal{X}_{S,T}$ which is accessible from the initial symbolic state, and which is normally considerably smaller than $\mathcal{X}_{S,T}$ itself. For more information see [8].

References

1. R. Alur and D. Dill. Automata for modelling real–time systems. In *ICALP'90*, volume 443 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.
2. R. Alur and D. Dill. The theory of timed automata. In *Real–Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.
3. J. Baeten and J. Bergstra. Real time process algebra. Technical Report P8916, University of Amsterdam, 1989.
4. A. Børjesson, K. Larsen, and A. Skou. Generality in design and compositional verification using tav. In *Proceedings of FORTE'92*, 1992.
5. G. Boudol and K. Larsen. Graphical versus logical specifications. In *Proceedings of CAAP'90*, volume 431 of *Lecture Notes in Computer Science*, 1990.
6. J. Godskesen, K. Larsen, and M. Zeeberg. TAV (tools for automatic verification) — users manual. Technical report, University of Aalborg, Denmark, 1989.
7. K. Larsen. Modal specifications. In *Proceedings of Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, 1990.
8. K. Larsen. Efficient local correctness checking. In *Proceedings of CAV'92*, 1992.
9. K. Larsen and B. Thomsen. A modal process logic. In *Proceedings LICS'88*, 1988.
10. K. Larsen and Y. Wang. Time abstracted bisimulation: Implicit specifications and decidability. Submitted for MFPS'93.
11. R. Milner. *Communication and Concurrency.* Series in Computer Science. Prentice–Hall International, 1989.
12. X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. In *Real–Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.
13. K. Čerāns. Decidability of bisimulation equivalences for processes with parallel timers. In *Proceedings of CAV'92*, 1992.
14. Y. Wang. Real–time behaviour of asynchronous agents. In *Proceedings of CONCUR'90*, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.