Automatic Synthesis of Real Time Systems *

Jørgen H. Andersen *Kåre J. Kristoffersen* *Kim G. Larsen*
Jesper Niedermann

BRICS †
Department of Math. & Comp. Sc., Aalborg University

Abstract

This paper presents a method for automatically constructing real time systems directly from their specifications. The model–construction problem is considered for implicit specifications of the form:

$$(A_1 \mid \ldots \mid A_n \mid X) \text{ sat } S$$

where S is a real time (logical) specification, $A_1 \ldots A_n$ are given (regular) timed agents and the problem is to decide whether there exists (and if possible exhibit) a real time agent X which when put in parallel with $A_1 \ldots A_n$ will yield a network satisfying S. The method presented proceeds in two steps: first, the implicit specification of X is transformed into an equivalent direct specification of X; second, a model for this direct specification is constructed (if possible) using a direct model construction algorithm. A prototype implementation of our method has been added to the real time verification tool EPSILON.

Introduction

During the last few years the area of real time systems has received a lot of attention from the research community. In particular, a variety of specification formalisms has emerged allowing real time properties to be expressed explicitly. These specification formalisms may roughly be divided into two groups, namely: real time logics (e.g. [RT89, HNSY92]) and real time process algebras (e.g. [Wan90, NRJV90]).

Central to the ongoing research has been the construction of *model–checking* algorithms; i.e. algorithms for deciding whether a given real time system satisfies a given specification. A number of model–checking algorithms exists for real timed logical specifications [ACD90] and more recently algorithms for model–checking [1] timed process algebraic specifications have been given [Cer92, LW93].

*This work has been partially supported by the European Communities under CONCUR2, BRA 7166.

†Basic Research in Computer Science, Centre of the Danish National Research Foundation

[1] In process algebra model–checking consists of checking a suitable behavioural relationship (bisimilarity, say) between the implementation and the specification.

In this work, we deal with the more ambitious goal of *model–construction*: i.e. given a real time specification (logical or process algebraic) we want to automatically synthesize a real time system satisfying the specification (if such a system exists). Moreover, we consider the model–construction problem in the setting of *implicit specifications*, i.e.:

$$(A_1 \mid \ldots \mid A_n \mid X) \text{ sat } S \tag{1}$$

The requirement of (1) represents a certain stage in a top–down development of a network satisfying a given overall specification S: namely, the stage where some components $A_1 \ldots A_n$ have already been constructed, but for the completion of the development one component X remains to be constructed. We call S an implicit specification of X as it specifies the behaviour of X in a certain context.

In this paper we present a method for automatically constructing the component X (if possible) such that (1) is met. Our method is applicable to logical as well as process algebraic specifications and proceeds in two steps: First, the implicit specification S is (effectively) transformed into a *direct* specification S' describing the sufficient and necessary requirement to X in order for (1) to hold; i.e.:

$$X \text{ sat } S' \quad \textit{if and only if} \quad (A_1 \mid \ldots \mid A_n \mid X) \text{ sat } S \tag{2}$$

Second, a real time system satisfying S' is generated (if possible) using a *direct* model–construction algorithm.

Our work can be seen as a real time extension of existing model–constructing algorithms for finite–state systems. For $n = 0$ the model–construction problem for (1) extends classical model–construction methods. For S a process algebraic specification the model–construction problem for (1) is a real time extension of the equation solving problem studied in [LX90b, Shi, Par89, LQ90]. For S a logical specification our work is related to and extends the work on contexts as property transformers studied in [LX91, LS92].

Our method assumes that the network components $A_1 \ldots A_n$ and X are all regular timed agents [Wan90] or equivalently one–clock timed automata [AD90]. For reasons of clarity we have chosen to present our solution method in a somewhat simplified setting:

—**T**he notion of parallel composition used in this paper is simply that of interleaving on actions; however our method extends smoothly to a wide range of existing notions of parallel compositions through parameterization on a so called synchronization function;

—**T**he specification language considered is a timed extension of the well–known Hennessy–Milner Logic [HM85]; however our method extends to a recursive (and very expressive) extension of this logic using already developed and well understood techniques [LX90a, LX91, JLJL93]. Also, our method is applicable to implicit model–construction based on process algebraic specifications by exhibition of suitable characteristic properties.

In the concluding remarks the above suggested extensions will be discussed in more details. Also, a prototype implementation of the implicit model–construction method for the full extensions has been given and is available as part of the EPSILON tool [CGL93].

DELAY TRANSITIONS:

$$\langle A, v\rangle \xrightarrow{\epsilon(d)} \langle A, v+d\rangle$$

ACTION TRANSITIONS:

$$\langle \sum_i [l_i, u_i].a_i.A_i, v\rangle \xrightarrow{a_i} \langle A_i, 0\rangle \text{ if } v \in [l_i, u_i]$$

$$\frac{\langle A, v\rangle \xrightarrow{a} \langle A', v'\rangle}{\langle N, v\rangle \xrightarrow{a} \langle A', v'\rangle} \text{ if } N \stackrel{def}{=} A$$

Table 1: Operational Semantics for Regular Timed Agents

1 Timed Processes and Timed Logic

Regular Timed Agents

Let $\mathcal{A}$ be a fixed set of actions range over by $a, b, c, \ldots$. We denote by $\mathbf{R}_{>0}$ the set of positive reals ranged over by $d, d_1, d_2, \ldots, d', d'', \ldots$. Similarly $\mathbf{R}$ denotes the set of non–negative reals, $\mathbf{N}$ denotes the set of natural numbers (including 0), and $\mathcal{D}$ denotes the set $\{\epsilon(d) \mid d \in \mathbf{R}_{>0}\}$. Regular timed agents are terms of the following grammar:

$$A \ ::= \ \sum_{i=1}^{n} [l_i, u_i].a_i.A_i \ \mid \ N$$

where $l_i, u_i \in \mathbf{N}$, $a_i \in \mathcal{A}$ and N ranges over a finite set of agent identifiers For each agent identifier we assume a defining equation $N \stackrel{def}{=} A$. We shall use nil to denote the empty summation, and on occasions we will use the expanded notation $[l_1, u_1].a_1.A_1 + \cdots + [l_n, u_n].a_n.A_n$ for the general summation. Also, we shall omit trailing nil's; hence $[4, 5].a$ denotes the agent $[4, 5].a.nil$. The maximum delay $M(A)$ of an agent A is defined recursively as $M(\sum_{i=1}^{n} [l_i, u_i].a_i.A_i) = \max\{u_i, M(A_i) \mid i = 1 \ldots n\}$, and $M(N) = M(A)$ where $N \stackrel{def}{=} A$.

Intuitively, the term $\sum_{i=1}^{n} [l_i, u_i].a_i.A_i$ describes an agent which is able to perform the action a_i between the time bounds l_i and u_i after which the agent will perform according to A_i. Formally, the semantics of regular timed agents are given in terms of a $\mathcal{A} \cup \mathcal{D}$ labelled transition system, where the *configurations* are pairs of the form $\langle A, v\rangle$, with $v \in \mathbf{R}$ denoting the amount by which the agent A has been delayed. The *transitions* between configurations are either delay– or action–transitions and are given by the rules of Table 1. Thus, we adopt the two–phase functioning priciple [NSY91] present in most real–time process algebras: i.e. the behaviour of a system is regarded as being split in two alternating phases, one where all components agree to let time progress, and one where the components compute.

DELAY TRANSITIONS: $$\langle \overline{A}, \overline{v} \rangle \xrightarrow{\epsilon(d)} \langle \overline{A}, \overline{v} + d \rangle$$
ACTION TRANSITIONS: $$\frac{\langle A_i, v_i \rangle \xrightarrow{a} \langle A_i', 0 \rangle}{\langle A_1 \mid \cdots A_i \cdots \mid A_n, \overline{v} \rangle \xrightarrow{a} \langle A_1 \mid \cdots A_i' \cdots \mid A_n, \overline{v}[v_i := 0] \rangle}$$

Table 2: Operational Semantics for Networks

Timed Networks

Syntactically a timed network agent is a parallel composition of a number of regular timed agents; thus network agents are terms of the following grammar:

$$N \quad ::= \quad (A_1 \mid \ldots \mid A_n)$$

Behaviourally, we shall simply assume that a network agent interleaves components actions, whereas components are required to synchronize with respect to delay. Formally, a *network configuration* is a pair $\langle \overline{A}, \overline{v} \rangle$, where $\overline{A} = A_1 | \ldots | A_n$ is a network and $\overline{v} = (v_1, \ldots, v_n)$ is a delay vector, indicating how much each component of the network has been delayed. The transitions between network configurations are given by Table 2, where for $d \in \mathbf{R}_{>0}$, $\overline{v} + d$ denotes the delay vector $(v_1 + d, \ldots, v_n + d)$, and $\overline{v}[v_i := 0]$ denotes the delay vector obtained by replacing v_i with 0 in the vector $\overline{v}$.

Example 1.1 Consider the network agents $[0,2].a \mid [2,3].b$ and $nil \mid [2,3].b$. The possible configurations involving these two networks are indicated by the two coordinate systems in Figure 1; thus in the left coordinate system the x–axis indicates the delay of $[2,3].b$ and the y–axis gives the delay of $[0,2].a$. Using the inference rules of Tables 1 and 2 we can infer the following transitions from the initial network configuration (see also Figure 1):

$$A = \langle [0,2].a \mid [2,3].b\,, (0,0) \rangle \xrightarrow{\epsilon(1.5)} B = \langle [0,2].a \mid [2,3].b\,, (1.5, 1.5) \rangle \xrightarrow{a}$$
$$C = \langle nil \mid [2,3].b\,, (0, 1.5) \rangle \xrightarrow{\epsilon(1)} D = \langle nil \mid [2,3].b\,, (1, 2.5) \rangle \xrightarrow{b}$$

Timed Logic

The specification language used in this presentation is the Extended Timed Modal Logic introduced in [HLY92], here referred to as TL. The logic is an extension of the well known Hennessy–Milner Logic [HM85], and the formulae of the logic are given by the following abstract syntax:

$$\phi \quad ::= \quad \mathrm{tt} \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \langle a \rangle \phi \mid \exists [l, u] \phi$$

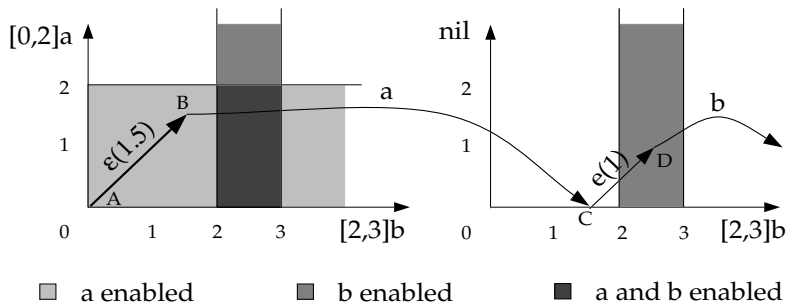


Figure 1: Transition Sequence

We shall freely use ff as abbreviation for $\neg$tt, $\phi_1 \vee \phi_2$ for $\neg(\neg\phi_1 \wedge \neg\phi_2)$, $[a]\phi$ for $\neg\langle a\rangle\neg\phi$ and $\forall[l,u]\phi$ for $\neg\exists[l,u]\neg\phi$.

For the interpretation of TL we define the satisfaction relation $\models$ between network configurations K and TL formulae ϕ inductively as follows:

$$
\begin{array}{lll}
i) & K \models \mathrm{tt} & \Leftrightarrow \text{ true} \\
ii) & K \models \phi_1 \wedge \phi_2 & \Leftrightarrow K \models \phi_1 \text{ and } K \models \phi_2 \\
iii) & K \models \neg\phi & \Leftrightarrow \text{ not } K \models \phi \\
iv) & K \models \langle a\rangle\phi & \Leftrightarrow \exists K'.\, K \xrightarrow{a} K' \text{ and } K' \models \phi \\
v) & K \models \exists[l,u]\phi & \Leftrightarrow \exists K', d.\, d \in [l,u] \text{ and } K \xrightarrow{\epsilon(d)} K' \text{ and } K' \models \phi
\end{array}
$$

We shall often write $\overline{A} \models \phi$ for $\langle \overline{A}, \overline{0}\rangle \models \phi$, where $\overline{0}$ is the (initial) delay vector with all components being 0. In this case we say that the network $\overline{A}$ satisfies the property ϕ.

Example 1.2 Consider the network $[0,2].a \mid [2,3].b$ from Example 1.1. Then it is easily seen that this network satisfies the formula $\forall[1,2]\langle a\rangle\forall[1,1]\langle b\rangle$tt. To see this simply observe that whenever $x \in [1,2]$ we can infer the following transition sequence:

$$
\begin{array}{l}
\langle [0,2].a \mid [2,3].b\,,\,(x\,,x)\rangle \xrightarrow{a} \langle nil \mid [2,3].b\,,\,(0,x)\rangle \xrightarrow{\epsilon(1)} \\
\langle nil \mid [2,3].b\,,\,(1,x+1)\rangle \xrightarrow{b}
\end{array}
$$

□

2 Symbolic Processes and Model Checking

Using the by now well–known region technique of Alur and Dill [ACD90] one may obtain an algorithm for model–checking: i.e. an algorithm for deciding whether a given network agent satisfies a TL formula. The region technique provides an abstract interpretation of network agents sufficiently complete that all information necessary for model–checking with respect to TL is maintained. At the same time the abstract interpretation yields a finite–state symbolic representation of networks thus enabling standard algorithmic model–checking techniques to be applied.

For $t \in \mathbf{R}$, let $\lfloor t \rfloor \stackrel{def}{=} \max\{n \in \mathbf{N} \mid n \leq t\}$ denote the integral part of t, and let $\{t\} \stackrel{def}{=} t - \lfloor t \rfloor$ denote its fractional part. We now recall from [ACD90]:

Definition 2.1 *Let $\overline{m} \in \mathbf{N}^n$ be a delay vector. Then $\overline{u}, \overline{v} \in \mathbf{R}^n$ are equivalent with respect to $\overline{m}$, denoted by $\overline{u} \doteq \overline{v}$ if*

- *For each $i = 1 \ldots n$, $u_i > m_i$ iff $v_i > m_i$,*
- *For each $i = 1 \ldots n$ such that $u_i \leq m_i$*
  1. *$\lfloor u_i \rfloor = \lfloor v_i \rfloor$*
  2. *$\{u_i\} = 0$ iff $\{v_i\} = 0$*
- *For each $i, j = 1 \ldots n$ such that $u_i \leq m_i$ and $u_j \leq m_j$, it is the case that $\{u_i\} \leq \{u_j\}$ iff $\{v_i\} \leq \{v_j\}$*

Observe that $\mathbf{R}^n / \doteq$ is finite. For $\overline{v} \in \mathbf{R}^n$, we denote by $[\overline{v}]$ the equivalence class of $\overline{v}$ under $\doteq$. The equivalence classes determined by $\doteq$ are called *regions* (see Figure 2).

For model–checking with respect to TL it is important to note that integer delays of equivalent delay vectors are again equivalent. Thus, whenever $\overline{u} \doteq \overline{v}$ then $\overline{u} + n \doteq \overline{v} + n$ whenever $n \in \mathbf{N}$. Hence, we may without ambiguity write $[\overline{u}] + n$ for the region $[\overline{u} + n]$. In general, it can be shown (see e.g. [LW93]) that two equivalent delay vectors $\overline{u}$ and $\overline{v}$ go through the same future regions; i.e. $\{[\overline{u} + d] \mid d \in R\} = \{[\overline{v} + d] \mid d \in R\}$. Moreover, $\overline{u}$ and $\overline{v}$ also agree on the order in which these regions are visited according to the following notion of successor region (see Figure 2):

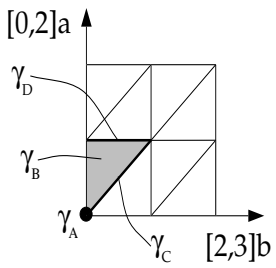
Definition 2.2 *Let $\gamma = [\overline{v}]$ be a region. Then the* successor *region* $\mathrm{succ}(\gamma)$ *is the region $[\overline{v'}]$, where:*

$$v'_i = \begin{cases} v_i + \min\{1 - \{v_j\} \mid j = 1 \ldots n\} & \text{if } \forall i.\{v_i\} > 0 \\ v_i + \min\{1 - \{v_j\} \mid j = 1 \ldots n\}/2 & \text{if } \exists i.\{v_i\} = 0 \end{cases}$$

We denote by $\mathrm{succ}^k(\gamma)$ *the region obtained by applying* succ *k times to γ.*

Now, it may be shown that the future regions from a delay vector $\overline{u}$ are precisely the regions $[\overline{u}], \mathrm{succ}^1([\overline{u}]), \mathrm{succ}^2([\overline{u}]), \mathrm{succ}^3([\overline{u}]), \ldots$ and that they are visited in this order. For γ a region and n a natural number we shall by n_γ denote the unique successor number such that $\gamma + n = \mathrm{succ}^{n_\gamma}(\gamma)$. Thus, when d ranges between two integer bounds l and u the delay vector $\overline{v} + d$ resides in regions between $\mathrm{succ}^{l\lfloor \overline{v} \rfloor}([\overline{v}])$ and $\mathrm{succ}^{u\lfloor \overline{v} \rfloor}([\overline{v}])$. Also, as agents enable actions within integer bounds, two network configurations with identical network agent and equivalent delay vectors agree on the action transitions they can perform in the following sense:

Lemma 2.3 *Let $\overline{A}$ be a network agent and $\overline{u} \doteq \overline{v}$. If $\langle \overline{A}, \overline{u} \rangle \stackrel{a}{\longrightarrow} \langle \overline{A}', \overline{u}' \rangle$, then also $\langle \overline{A}, \overline{v} \rangle \stackrel{a}{\longrightarrow} \langle \overline{A}', \overline{v}' \rangle$ for some $\overline{v}'$ such that $\overline{v}' \doteq \overline{u}'$.*



The figure illustrates four typical dyadic regions:

$$\gamma_A = \{(0,0)\}$$
$$\gamma_B = \{(x,y) \mid 0 < x, y < 1, x < y\}$$
$$\gamma_C = \{(x,y) \mid 0 < x, y < 1, x = y\}$$
$$\gamma_D = \{(x,y) \mid 0 < x < 1, y = 1\}$$

Now using Definition 2.2 it follows that $\mathrm{succ}(\gamma_A) = \gamma_C$ and $\mathrm{succ}(\gamma_B) = \gamma_D$. In general successor regions are determined by following 45^o lines upwards to the right.

Figure 2: Regions and Their Successors

Based on the above observations it can be concluded that network configurations with equivalent delay vectors satisfy the same TL formulae:

Theorem 2.4 *Let $\overline{A}$ be a network agent, $\overline{u}, \overline{v}$ two delay vectors, and ϕ a TL formula. Then if $\overline{u} \doteq \overline{v}$ the following holds:*

$$\langle \overline{A}, \overline{u} \rangle \models \phi \quad \Leftrightarrow \quad \langle \overline{A}, \overline{v} \rangle \models \phi$$

To obtain the model–checking algorithm we extract a finite–state symbolic semantics of network configurations, by identifying configurations with equivalent delay vector. Thus *symbolic* network configurations (or simply symbolic states) are pairs of the form $[\overline{A}, \gamma]$, where $\overline{A}$ is a network agent and γ is a $\doteq$–equivalence class with respect to the delay vector $M(\overline{A}) = (M(A_1), \ldots, M(A_n))$. As network agents have only finitely many sub-terms [2], and there are only finitely many regions with respect to $M(\overline{A})$ it follows that the set of symbolic states reachable from $[\overline{A}, \gamma]$ is finite. The transitions between symbolic states are either un–quantified delay transitions (labelled χ) or action transitions and are defined by the axiom and rule of Table 3. Moreover, symbolic transitions may be computed effectively. This rests on an effective representation of regions [3] allowing effective computation of a representative of a region as well as effective computation of the region from a delay vector. Also, due to Lemma 2.3, it suffices to consider a single representative $\overline{v}$ of γ when inferring symbolic action transitions.

We may now give an alternative interpretation of TL based on the above symbolic semantics of networks.

$$\begin{array}{rll}
i) & [\overline{A}, \gamma] \models \mathrm{tt} & \Leftrightarrow \text{ true} \\
ii) & [\overline{A}, \gamma] \models \phi_1 \wedge \phi_2 & \Leftrightarrow [\overline{A}, \gamma] \models \phi_1 \text{ and } [\overline{A}, \gamma] \models \phi_2 \\
iii) & [\overline{A}, \gamma] \models \neg\phi & \Leftrightarrow \text{ not } [\overline{A}, \gamma] \models \phi \\
iv) & [\overline{A}, \gamma] \models \langle a \rangle \phi & \Leftrightarrow \exists [\overline{B}, \eta]. [\overline{A}, \gamma] \xrightarrow{a} [\overline{B}, \eta] \text{ and } [\overline{B}, \eta] \models \phi \\
v) & [\overline{A}, \gamma] \models \exists [l, u] \phi & \Leftrightarrow \exists l_\gamma \leq k \leq u_\gamma. [\overline{A}, \mathrm{succ}^k(\gamma)] \models \phi
\end{array}$$

[2] with the usual application of unfolding in the case of recursive definition

[3] The obvious effective representation of a region is as a linear inequation system. An alternative effective representation where each region has a canonical representation is given in [GL94, God94].

DELAY TRANSITIONS: $$[\overline{A}, \gamma] \xrightarrow{\chi} [\overline{A}, \mathrm{succ}(\gamma)]$$
ACTION TRANSITIONS: $$\frac{\langle \overline{A}, \overline{v} \rangle \xrightarrow{a} \langle \overline{B}, \overline{u} \rangle}{[\overline{A}, [\overline{v}]] \xrightarrow{a} [\overline{B}, [\overline{u}]]}$$

Table 3: Symbolic Semantics of Networks

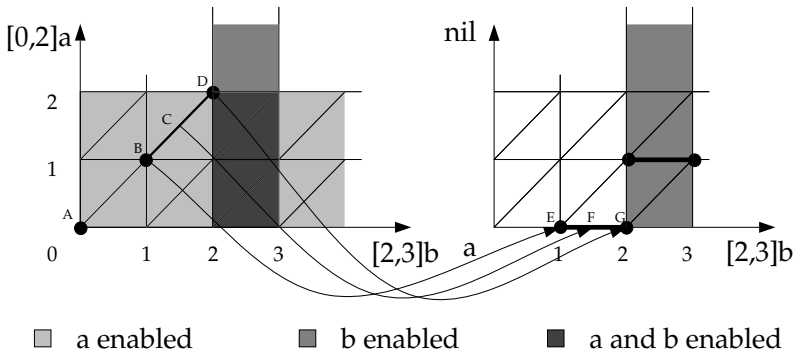


Figure 3: Symbolic Transitions

Clearly, due to the finite–state nature of the symbolic semantics of networks, the above symbolic interpretation is decidable using classical finite–state model–checking techniques. Moreover, the symbolic interpretation of TL is closely related to the standard interpretation as stated in the following theorem:

Theorem 2.5 *Let $\overline{A}$ be a network agent, $\overline{v}$ a delay vector, and ϕ a TL formula. Then the following equivalence holds:*

$$[\overline{A}, [\overline{v}]] \models \phi \quad \Leftrightarrow \quad \langle \overline{A}, \overline{v} \rangle \models \phi$$

It follows that model–checking of network configurations with respect to TL formulae is decidable.

Example 2.6 Figure 3 indicates some symbolic states and transitions associated with the network $[0,2].a \mid [2,3].b$. Note that $A \xrightarrow{\chi^2} B$, $A \xrightarrow{\chi^3} C$ and $A \xrightarrow{\chi^4} D$, and that $\{2,3,4\}$ are precisely the successor numbers associated with the delay interval $[1,2]$ when considering state A. Using the symbolic interpretation of TL it can now easily be checked that $[0,2].a \mid [2,3].b$ does indeed satisfy $\forall[1,2]\langle a \rangle \forall[1,1]\langle b \rangle \mathrm{tt}$. □

3 Symbolic Contexts

We want to decompose logical properties required of a network agent into necessary and sufficient properties of one of its agents. More precisely, for any given regular agents $A_1, \ldots, A_n$ and any given TL formula ϕ, we want to find a formula ψ such that the following holds:

$$(A_1 \,|\, \ldots \,|\, A_n \,|\, A) \models \phi \quad \textit{if and only if} \quad A \models \psi \tag{3}$$

Clearly, the component property ψ will in general depend on the overall property ϕ as well as the agents $A_1, \ldots, A_n$. In the next section we shall define a *property transformer* $\mathcal{W}$, that — given the network agent $\overline{A} = (A_1 | \ldots | A_n)$ and the property ϕ — will construct a property $\psi = \mathcal{W}(\overline{A}, \phi)$ satisfying the requirement of (3).

In (3), the property ϕ expresses constraints on transitions of the complete network $(A_1 | \ldots | A_n | A)$, whereas ψ constrains transitions of the component agent A. Thus, in order to solve the above decomposition problem we must have a way of interrelating transitions of a network with transitions of one of its components. To achieve this we provide a symbolic operational semantics of the network contexts $C = (A_1 | \ldots | A_n | [\,])$ in terms of *action transducers*. That is, a network context is semantically viewed as an object which consumes actions from its component agent and produces actions for the external environment, thus acting as an interface between the two. Obviously, we expect the new operational semantics of network contexts to be consistent with the existing operational semantics of network agents. That is, if the component agent A has an a transition, and the context $(A_1 | \ldots | A_n | [\,])$ can consume this action while producing the action b, then we expect the combined network $(A_1 | \ldots | A_n | A)$ to have a b–transition.

The idea of modelling contexts as action transducers has already been pursued for finite state systems [LX90a, LX91]. In our real–time setting we need in addition to take into account the delay of the context agents $A_1, \ldots, A_n$ as well as the delay of the component to be placed in the hole $[\,]$. However, as our transductional semantics is intended to provide the basis of an effective transformation of properties, we deal with delays in a symbolic manner using regions. Thus, formally, a symbolic $n+1$–ary network context is a pair of the form:

$$\Big[(A_1 | \ldots | A_n | [\,]),\, [(v_1 \ldots v_n, v)]\Big]$$

Here $[(v_1 \ldots v_n, v)]$ is an $n+1$–ary region with $(v_1 \ldots v_n)$ giving delay information of $A_1, \ldots, A_n$ and v providing the delay information of the $[\,]$–component. The transductions between symbolic network contexts is given by the axioms and rule of Table 4. For γ being an $n+1$–ary region $[(v_1 \ldots v_n, v)]$, $\gamma^{\downarrow}$ denotes the unary region $[v]$ [4]. For $\overline{v} = (v_1 \ldots v_n)$ an n–ary delay vector and u a non–negative real, $\overline{v}u$ denotes the $n+1$–ary delay vector $(v_1 \ldots v_n, u)$.

Transductions may be inferred in two ways depending on whether the $[\,]$–component "participates" in the transduction or not. Thus for action trans-

[4] In the unary case regions are either integer points $[n, n]$, open intervals $]n, n+1[$ or open infinite intervals $]m, \infty[$, where m is the maximum delay bound.

DELAY TRANSDUCTIONS:

$$\left[\overline{A}|[\,],\gamma\right] \xrightarrow[\chi]{\chi} \left[\overline{A}|[\,],\mathrm{succ}(\gamma)\right] \text{ if } \mathrm{succ}(\gamma)^{\downarrow} = \mathrm{succ}(\gamma^{\downarrow})$$

$$\left[\overline{A}|[\,],\gamma\right] \xrightarrow[0]{\chi} \left[\overline{A}|[\,],\mathrm{succ}(\gamma)\right] \text{ if } \mathrm{succ}(\gamma)^{\downarrow} = \gamma^{\downarrow}$$

ACTION TRANSDUCTIONS:

$$\left[\overline{A}|[\,],[\overline{v}u]\right] \xrightarrow[a]{a} \left[\overline{A}|[\,],[\overline{v}0]\right]$$

$$\frac{\langle \overline{A},\overline{v}\rangle \xrightarrow{a} \langle \overline{B},\overline{w}\rangle}{\left[\overline{A}|[\,],[\overline{v}u]\right] \xrightarrow[0]{a} \left[\overline{B}|[\,],[\overline{w}u]\right]}$$

Table 4: Symbolic Transduction Semantics of Contexts

ductions the first axiom of Table 4 requires the $[\,]$–component to perform the a–action (after which the $[\,]$–delay is reset to 0). In the second action transduction rule, the a–action is performed entirely by the network context $\overline{A}$ without any involvement of the $[\,]$–component. This is modelled by a transduction using a unique 0–action (i.e. $0 \notin \mathcal{A}$). The symbolic semantics is extended in the obvious way to 0–actions by $[\overline{A},\gamma] \xrightarrow{0} [\overline{A}',\gamma']$ if and only if $\overline{A}' = \overline{A}$ and $\gamma' = \gamma$. Delay transductions model progression to a successor region. The two delay transduction axioms reflect that the projected $[\,]$–region may either remain unchanged or change to its (unary) successor region.

Example 3.1 In Figure 4 three types of transductions for the network $[0,2].a|[\,]$ are illustrated. As clearly $\gamma_B^{\downarrow} = \mathrm{succ}(\gamma_A^{\downarrow})$ and $\gamma_E^{\downarrow} = \gamma_D^{\downarrow}$, it follows from Table 4 that:

$$\left[([0,2].a|[\,]),\gamma_A\right] \xrightarrow[\chi]{\chi} \left[([0,2].a|[\,]),\gamma_B\right]$$

$$\left[(nil|[\,]),\gamma_D\right] \xrightarrow[0]{\chi} \left[(nil|[\,]),\gamma_E\right]$$

$$\left[([0,2].a|[\,]),\gamma_C\right] \xrightarrow[0]{a} \left[(nil|[\,]),\gamma_D\right]$$

□

The following Lemma demonstrates that the transductional semantics of contexts does indeed provide the key to relating symbolic transitions of a network and its component:

Lemma 3.2 *Let $\overline{A} = (A_1|\ldots|A_n)$ be an n–ary network agent, A a regular timed agent, γ an $n+1$–ary region, and let $\alpha \in \mathcal{A} \cup \{\chi\}$. Then the following equivalence holds:*

$$\left[\overline{A}|A,\gamma\right] \xrightarrow{\alpha} \left[\overline{A}'|A',\gamma'\right]$$

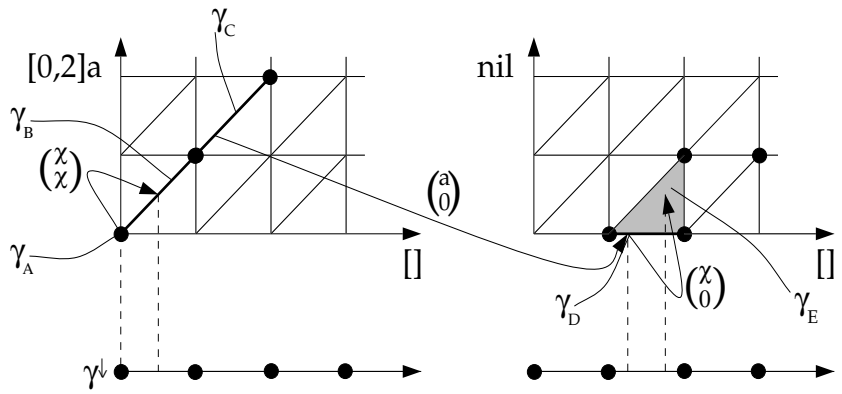


Figure 4: Context Transductions

if and only if

$$\left[\overline{A}|[\,],\gamma\right] \xrightarrow[\beta]{\alpha} \left[\overline{A}'|[\,],\gamma'\right] \text{ and } \left[A,\gamma^{\downarrow}\right] \xrightarrow{\beta} \left[A',\gamma'^{\downarrow}\right]$$

for $\beta = \alpha$ *or* $\beta = 0$.

4 Contexts as Property Transformers

As shown in the previous Lemma 3.2, contexts relate *symbolic* transitions of networks with *symbolic* transitions of their components. To facilitate the transformation of logical properties we extend our logic TL with a modality explicitly concerned with symbolic delay transitions ($\xrightarrow{\chi}$). Syntactically, we add the following production to the syntax for formulae:

$$\phi \;::=\; \odot\phi$$

We refer to the extended logic as $\mathrm{TL}^{\odot}$. Formulae with no occurrence of $\exists[l,u]$–modalities are called *pure* and the corresponding sublogic is refered to as $\mathrm{TL}^{\odot}_{p}$. Semantically, we interpret formulae $\odot\phi$ with respect to (standard) network configurations as well as symbolic network configurations thus extending the two existing interpretations of TL:

$$\langle\overline{A},\overline{u}\rangle \models \odot\phi \;\Leftrightarrow\; \langle\overline{A},\overline{v}\rangle \models \phi \text{ for some } \overline{v} \in \mathrm{succ}([\overline{u}])$$

$$[\overline{A},\gamma] \models \odot\phi \;\Leftrightarrow\; [\overline{A},\mathrm{succ}(\gamma)] \models \phi$$

It is straightforward to show that with these semantic definitions both Theorem 2.4 and Theorem 2.5 generalize to $\mathrm{TL}^{\odot}$. Furthermore, for any given network configuration, the original (interval) delay modalities of TL can be expressed using the new $\odot$–modality in the following way:

$$\langle\overline{A},\overline{v}\rangle \models \exists[l,u]\phi \;\Leftrightarrow\; \langle\overline{A},\overline{v}\rangle \models \bigvee_{k=l_{[\overline{v}]}}^{u_{[\overline{v}]}} \odot^{k}\phi$$

For $C = [\overline{A}|[\,],\gamma]$ an $n+1$–ary context and ϕ a TL$^{\odot}$–formula we now define the *transformed formula* $\mathcal{W}(C,\phi)$ as follows:

$$
\begin{array}{rl}
i) & \mathcal{W}(C,\mathrm{tt}) = \mathrm{tt} \\
ii) & \mathcal{W}(C,\phi_1 \wedge \phi_2) = \mathcal{W}(C,\phi_1) \wedge \mathcal{W}(C,\phi_2) \\
iii) & \mathcal{W}(C,\neg\phi) = \neg\mathcal{W}(C,\phi) \\
iv) & \mathcal{W}(C,\langle a\rangle\phi) = \bigvee_{C \xrightarrow[a]{a} C'} \langle a\rangle\mathcal{W}(C',\phi) \;\vee \bigvee_{C \xrightarrow[0]{a} C'} \mathcal{W}(C',\phi) \\
v) & \mathcal{W}(C,\exists[l,u]\phi) = \mathcal{W}(C, \bigvee_{k=l_\gamma}^{u_\gamma} \odot^k\phi) \\
vi) & \mathcal{W}(C,\odot\phi) = \begin{cases} \mathcal{W}(C',\phi) & \text{if } C \xrightarrow[0]{\chi} C' \\ \odot\mathcal{W}(C',\phi) & \text{if } C \xrightarrow[\chi]{\chi} C' \end{cases}
\end{array}
$$

Note that $\mathcal{W}(C,\phi)$ is always a pure TL$^{\odot}$formula. The following Theorem and Corollary shows that the transformer $\mathcal{W}$ does indeed yield the sufficient and necessary requirement to a networks component in order that the network itself satisfy a given property:

Theorem 4.1 *Let $C = [\overline{A}|[\,],\gamma]$ be an $n+1$–ary context. Then the following equivalence holds for any regular timed agent A:*

$$[\overline{A}|A,\gamma] \models \phi \quad \Leftrightarrow \quad [A,\gamma^{\downarrow}] \models \mathcal{W}(C,\phi)$$

Corollary 4.2 *Let $C = [\overline{A}|[\,],\gamma]$ be an $n+1$–ary context. Then the following equivalence holds for any regular timed agent A, whenever $\overline{v}u \in \gamma$:*

$$\langle\overline{A}|A,\overline{v}u\rangle \models \phi \quad \Leftrightarrow \quad \langle A,u\rangle \models \mathcal{W}(C,\phi)$$

Corollary 4.3 *Let $\overline{A} = (A_1|\ldots|A_n)$ be an n–ary network, and let A be a regular timed agent. Then the following equivalence holds:*

$$(A_1|\ldots|A_n|A) \models \phi \quad \Leftrightarrow \quad A \models \mathcal{W}([\overline{A}|[\,],[\overline{0}]],\phi)$$

Example 4.4 Using $\mathcal{W}$ we may now compute the necessary and sufficient requirement to the component A in order that $[0,2].a \mid A$ satisfies $\phi = \forall[1,2]\langle a\rangle\forall[1,1]\langle b\rangle\mathrm{tt}$. After some calculations based on the transductional semantics of $[0,2].a|[\,]$ (part of which is illustrated in Example 3.1) we get the following:

$$
\begin{array}{l}
\mathcal{W}\Big(\Big[\,[0,2].a|[\,]\,,\,[\overline{0}]\,\Big],\phi\Big) = \\
\quad \odot^2\Big(\odot^2\langle b\rangle\mathrm{tt} \vee \langle a\rangle\,\odot^2\,\langle b\rangle\mathrm{tt}\Big) \;\wedge\; \odot^3\Big(\odot^2\langle b\rangle\mathrm{tt} \vee \langle a\rangle\,\odot^2\,\langle b\rangle\mathrm{tt}\Big) \;\wedge \\
\quad \odot^4\Big(\odot^2\langle b\rangle\mathrm{tt} \vee \langle a\rangle\,\odot^2\,\langle b\rangle\mathrm{tt}\Big)
\end{array}
$$

□

Using the easily established fact that $(A_1|\ldots|A_n)$ and $(A_1|\ldots|A_n|nil)$ satisfy the same formulae the transformer $\mathcal{W}$ can also be used to obtain an alternative model–checking algorithm:

Corollary 4.5 *Let $\overline{A} = (A_1|\ldots|A_n)$ be an n–ary network, and let ϕ be a $TL^{\odot}$ formula. Then the following equivalence holds:*

$$(A_1|\ldots|A_n) \models \phi \quad \Leftrightarrow \quad nil \models \mathcal{W}([\overline{A}|[\,],[\overline{0}]],\phi)$$

To decide whether the agent nil satisfies a property is particularly easy as nil satisfies no $\langle a\rangle\phi$ formula and all $[a]\phi$ formulae; also nil satisfies formulae $\exists[l,u]\phi$, $\forall[l,u]\phi$ and $\odot\phi$ precisely if nil satisfies ϕ.

Example 4.6 To decide $[0,2].a|[2,3]b$ satisfies the property $\phi = \forall[1,2]\langle a\rangle\forall[1,1]\langle b\rangle$tt we should simply certify that nil satisfies the following transformed property (obtained after some calculations):

$$\begin{aligned}\mathcal{W}\Big(\Big[\,[0,2].a|[2,3].b|[\,]\,,\,[\overline{0}]\,\Big],\phi\Big) = \\ \odot^2\Big(\odot^2(\mathrm{tt}\vee\langle b\rangle\mathrm{tt})\vee\langle a\rangle\odot^2(\mathrm{tt}\vee\langle b\rangle\mathrm{tt})\Big)\ \wedge \\ \odot^3\Big(\odot^2(\mathrm{tt}\vee\langle b\rangle\mathrm{tt})\vee\langle a\rangle\odot^2(\mathrm{tt}\vee\langle b\rangle\mathrm{tt})\Big)\ \wedge \\ \odot^4\Big(\odot^2(\mathrm{tt}\vee\langle b\rangle\mathrm{tt})\vee\langle a\rangle\odot^2(\mathrm{tt}\vee\langle b\rangle\mathrm{tt})\Big)\end{aligned}$$

Now, using the simplification rules pointed out above (i.e. simplify all modalities) it is obvious that nil satisfies the above property. □

5 Direct Model Construction

In this section we provide an algorithm that given a pure $\mathrm{TL}^{\odot}$–formula ϕ will decide whether ϕ is satisfiable by some regular agent. Moreover if ϕ is satisfiable the algorithm will construct a satisfying agent. The technique applied is based on classical tableau methods applied for modal logic (see e.g. [HC68]). To simplify this part of the presentation we use an alternative version of $\mathrm{TL}^{\odot}_p$ with no negation but with all dual operators included (i.e. ff, $\vee$ and $[a]$).

Let Γ be the set of all unary regions of the form $[n,n]$ and $]n,n+1[$, where $n \in \mathbf{N}$. Then a *problem* Π is a finite subset of $\Gamma\times\mathrm{TL}^{\odot}_p$. We say that a problem Π is *satisfiable* if there exists a regular timed agent A such that $[A,\gamma] \models \phi$ whenever $(\gamma,\phi) \in \Pi$. In this case we call A a *solution* to Π. It follows from the results of the previous sections that if A is a solution to an *initial* problem of the form $\{(\mathcal{O},\phi)\}$ where $\mathcal{O} = \{0\}$ then $A \models \phi$.

A problem Π is called *simple* if whenever $(\gamma,\phi) \in \Pi$ then ϕ is of the form $\langle a\rangle\psi$ or $[a]\psi$; i.e. all conjunctions, disjunctions and $\odot$–modalities have been resolved. As we shall see in the following it is particularly easy to decide satisfiability of simple problems. However, we first provide a reduction mechanism for transforming problems into simple ones. The reduction relation $\rightsquigarrow$ between

problems is defined as the least relation satisfying the following axioms [5]:

$$
\begin{aligned}
&i) \quad \Pi \uplus \{(\gamma, \mathrm{tt})\} \rightsquigarrow \Pi \\
&ii) \quad \Pi \uplus \{(\gamma, \phi_1 \wedge \phi_2)\} \rightsquigarrow \Pi \cup \{(\gamma, \phi_1)\} \cup \{(\gamma, \phi_2)\} \\
&iii) \quad \Pi \uplus \{(\gamma, \phi_1 \vee \phi_2)\} \rightsquigarrow \Pi \cup \{(\gamma, \phi_1)\} \\
&iv) \quad \Pi \uplus \{(\gamma, \phi_1 \vee \phi_2)\} \rightsquigarrow \Pi \cup \{(\gamma, \phi_2)\} \\
&v) \quad \Pi \uplus \{(\gamma, \odot\phi)\} \rightsquigarrow \Pi \cup \{(\mathrm{succ}(\gamma), \phi)\}
\end{aligned}
$$

As the use of $\rightsquigarrow$ always strictly decreases the total size of the formulae in Π it is clear that any reduction sequence from Π must be finite. In fact any problem determines a finite reduction tree with the leaves being the irreducible reductions of Π; i.e. Π' is an irreducible reduction of Π if $\Pi \rightsquigarrow^\star \Pi'$ and $\Pi' \not\rightsquigarrow$. Now it follows directly from the semantic definition of the various operators of $\mathrm{TL}_p^{\odot}$ that there is a close connection between the satisfiability of a problem and its irreducible reductions:

Lemma 5.1 *A problem is satisfiable if and only if one of its irreducible reductions is satisfiable.*

Moreover, it is clear from the definition of $\rightsquigarrow$ that any irreducible problem is either simple or contains a pair of the form (γ, ff) in which case it is obviously not satisfiable. Thus, we are left with the problem of deciding satisfiability of simple problems.

First we define for $a \in \mathcal{A}$, $\gamma \in \Gamma$ and Π a problem the *projected* problem Π_a^γ as follows:

$$\Pi_a^\gamma = \{(\mathcal{O}, \phi) \mid (\gamma, [a]\phi) \in \Pi\}$$

From the symbolic interpretation of $[a]\phi$ it follows directly that whenever A is a solution to Π and $[A, \gamma] \xrightarrow{a} [A', \mathcal{O}]$, then A' is a solution to Π_a^γ. Moreover, when $\gamma =]n, n+1[$, A' is also a solution to $\Pi_a^{[n,n]}$ and $\Pi_a^{[n+1,n+1]}$ as regular agents enable actions within *closed* integer–bound intervals.

Theorem 5.2 *Let Π be a simple problem. Then Π is satisfiable if and only if whenever $([n,n], \langle a\rangle\phi) \in \Pi$ then*

$$\Big\{(\mathcal{O}, \phi)\Big\} \cup \Pi_a^{[n,n]} \tag{4}$$

is satisfiable, and whenever $(]n, n+1[, \langle a\rangle\phi) \in \Pi$ then

$$\Big\{(\mathcal{O}, \phi)\Big\} \cup \Pi_a^{[n,n]} \cup \Pi_a^{]n,n+1[} \cup \Pi_a^{[n+1,n+1]} \tag{5}$$

is satisfiable.

Proof: *Then*–direction: follows directly from the symbolic interpretation of formulae and the comments above.
If–direction: For $(\gamma, \langle a\rangle\phi) \in \Pi$ let $A_{(\gamma,\langle a\rangle\phi)}$ be a regular timed agent satisfying

[5] $\uplus$ denotes disjoint union of sets.

(4) if $\gamma = [n,n]$ and (5) if $\gamma =]n, n+1[$. Let $l_{[n,n]} = u_{[n,n]} = n$, $l_{]n,n+1[} = n$ and $u_{]n,n+1[} = n+1$. Then the agent A defined as:

$$A = \sum_{(\gamma,\langle a\rangle\phi)\in\Pi} [l_\gamma, u_\gamma].a.A_{(\gamma,\langle a\rangle\phi)}$$

satisfies Π. □

It now follows from the properties of $\rightsquigarrow$ and Theorem 5.2 that satisfiability of problems is decidable: to determine satisfiablity of a problem Π first (non–deterministically) reduce it to a simple problem Π' and then use the construction of Theorem 5.2. This leaves the satisfiability of the problems in (4) and (5) to be settled. However, as these problems all have strictly smaller maximum modal depth than Π' (and Π) we can apply the method recursively, with termination guaranteed.

Example 5.3 In order to synthesize the missing component A in Example 4.4 we should determine satisfiability of the problem:

$$\Pi = \left\{(\mathcal{O}, \odot^2\psi \wedge \odot^3\psi \wedge \odot^4\psi)\right\}$$

where $\psi = \odot^2\langle b\rangle \mathrm{tt} \vee \langle a\rangle \odot^2 \langle b\rangle \mathrm{tt}$. Now using the axioms for $\rightsquigarrow$ we obtain:

$$\begin{aligned}
&\Pi \rightsquigarrow^\star \\
&\left\{([1,1],\psi)\,,\,(]1,2[,\psi)\,,\,([2,2],\psi)\right\} \rightsquigarrow^\star \\
&\left\{([1,1],\odot^2\langle b\rangle\mathrm{tt})\,,\,(]1,2[,\odot^2\langle b\rangle\mathrm{tt})\,,\,([2,2],\odot^2\langle b\rangle\mathrm{tt})\right\} \rightsquigarrow^\star \\
&\left\{([2,2],\langle b\rangle\mathrm{tt})\,,\,(]2,3[,\langle b\rangle\mathrm{tt})\,,\,([3,3],\langle b\rangle\mathrm{tt})\right\} = \Pi'
\end{aligned}$$

Now — as *nil* obviously satisfies $\{(\mathcal{O},\mathrm{tt})\}$ — we obtain from the proof of Theorem 5.2 that the following agent:

$$A = [2,2].b + [2,3].b + [3,3].b$$

satisfies Π' and hence Π. □

Concluding Remarks

The presentation of this paper has been based on a somewhat simplified setting, and we want here to comment in slightly more detail on how our results extends.

The logic TL considered may be extended with constructs for defining properties recursively. The symbolic interpretation of TL extends easily to this recursive extension, thus providing the basis for decidability of model–checking. As for transforming recursive properties the techniques given in [LX90a, LX91] can be directly applied. Our direct model–construction method extends to maximal recursively defined properties using the techniques of [JLJL93].

The notion of parallel composition considered in this paper is simply that of interleaving of actions. However, our results extend to a variety of parallel compositions via parameterization on a *synchronization function* as studied in [HL89]. Thus, we may consider *parameterized* network of the form

$(A_1, \ldots, A_n)|_f$, where f is a synchronization (partial) function of type $(\mathcal{A} \cup \{0\})^n \hookrightarrow \mathcal{A}$. The use of the special no–action 0 enables the modelling of synchronizations where only some components participate. Also, the partiality of f enables synchronization of certain combinations of actions to be disallowed.

In this presentation we have not yet considered implicit process algebraic specifications; i.e. specifications of the form:

$$(A_1 \mid \ldots \mid A_n \mid X) \equiv B \tag{6}$$

where B is a regular timed agent and $\equiv$ is some abstracting equivalence (timed bisimilarity, say). However, (6) may easily be transformed into an equivalent logical implicit specification by using a *characteristic formula* ϕ_B for B; i.e. a formula such that $A \equiv B$ if and only if $A \models \phi_B$. Both for timed and time–abstracting bisimilarity such characteristic formulae can be effectively constructed.

Future work includes extension of our method to implicit specifications for arbitrary n–clock automata. However, it is already known that implicit specifications for general timed networks, i.e. specifications of the form:

$$(A_1 \mid \ldots \mid A_n \mid X_1 \mid \ldots \mid X_m) \text{ sat } S$$

where $X_1 \ldots X_m$ are regular timed agents (or 1–clock automata) are undecidable when $m > 1$ [Liu93].

References

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model–checking for Real–Timne Systems. *In Proceedings of Logic in Computer Science*, 1990.

[AD90] R. Alur and D. Dill. Automata for Modelling Real–Time Systems. *Lecture Notes in Computer Science*, 443, 1990. In Proceedings of ICALP.

[Cer92] K. Cerans. Decidability of Bisimulation Equivalences for Processes with Parallel Timers. *In Proceedings of CAV'92*, 1992.

[CGL93] K. Cerans, J.C. Godskesen, and K.G. Larsen. Timed modal specifications — theory and tools. *Lecture Notes in Computer Science*, 1993.

[GL94] J.C. Godskensen and K.G. Larsen. Synthesizing Distiniguishing Formulae for Real Time Systems. Technical report, Aalborg University, 1994.

[God94] J.C. Godskesen. *Timed Modal Specifications — A theory for verification of real–time concurrent systems.* PhD thesis, Aalborg University, 1994.

[HC68] G.E. Hughes and M.J. Cresswell. *An Introduction to Modal Logic.* Methuen and Co., 1968.

[HL89] Hans Hüttel and Kim G. Larsen. The Use of Static Constructs in a Modal Process Logic. *Lecture Notes in Computer Science*, (363), 1989. Logic at Botik'89.

[HLY92] U. Holmer, K.G. Larsen, and W. Yi. Decidability of bisimulation equivalence between regular timed processes. *Lecture Notes In Computer Science, Springer Verlag*, 575, 1992. In Proceedings of CAV'91.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the Association for Computing Machinery*, pages 137–161, 1985.

[HNSY92] T. A. Henzinger, Z. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Logic in Computer Science*, 1992.

[JLJL93] O.H. Jensen, J.T. Lang, C. Jeppesen, and K.G. Larsen. Model construction for implicit specifications in modal logic. *Lecture Notes in Computer Science*, 1993.

[Liu93] Xinxin Liu. On Small Model Property and Decidability of Process Equations. Unpublished note, Sussex University., 1993.

[LQ90] P. Lewis and H. Qin. Factorization of finite state machines under observational equivalence. *Lecture Notes In Computer Science, Springer Verlag*, 458, 1990.

[LS92] K.G. Larsen and A. Skou. Compositional verification of probabilistic processes. *In Proceedings of CONCUR'92. To appear in Lecture Notes in Computer Science.*, 1992.

[LW93] K.G. Larsen and Y. Wang. Time Abstracted Bisimulation: Implicit Specifications and Decidability. *In Proceedings of MFPS'93*, 1993.

[LX90a] K.G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. *Lecture Notes In Computer Science, Springer Verlag*, 443, 1990. In proceedings of International Colloquium on Algorithms, Languages and Programming 1990.

[LX90b] K.G. Larsen and L. Xinxinx. Equation Solving Using Modal Transition systems. *In Proceedings of Logic in Computer Science*, 1990.

[LX91] K.G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. *Journal of Logic and Computation*, 1(6):761–795, 1991.

[NRJV90] X. Nicollin, J.L. Richierand, J.Sifakis, and J. Voiron. ATP: an algebra for timed processes,. *In Proceedings of the IFIP TC 2 Working Conference on Programming Concepts and Methods*, 1990.

[NSY91] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to Timed Graphs and Hybrid Systems. *Lecture Notes in Computer Science*, 600, 1991. In Real–Time: Theory in Practice.

[Par89] J. Parrow. Submodule construction as equation solving in CCS. *Theoretical Computer Science*, 68, 1989.

[RT89] R.Alur and T.A.Henzinger. A Really Temporal Logic. *In Proceeding of IEEE Symp. on Foundations of Computer Science*, 1989.

[Shi] M.W. Shields. A note on the simple interface equation. Technical report, University of Kent at Canterbury.

[Wan90] Yi Wang. A Calculus of Real Time Systems. *Lecture Notes in Computer Science*, 458, 1990. In Proceedings of CONCUR.