Behavioural Equivalences over Migrating Processes with Timers

Bogdan Aman, Gabriel Ciobanu, Maciej Koutny

Behavioural Equivalences over Migrating Processes with Timers

Bogdan Aman[1], Gabriel Ciobanu[1], and Maciej Koutny[2]

[1] Romanian Academy, Institute of Computer Science
and "A.I.Cuza" University, 700506 Iaşi, Romania
bogdan.aman@gmail.com, gabriel@info.uaic.ro
[2] School of Computing Science, Newcastle University
Newcastle upon Tyne, NE1 7RU, United Kingdom
maciej.koutny@newcastle.ac.uk

Abstract. The temporal evolution of mobile processes is governed by independently operating local clocks and their migration timeouts. We define a formalism modelling such distributed systems allowing (maximal) parallel execution at each location. Taking into account explicit timing constraints based on migration and interprocess communication, we introduce and study a number of timed behavioural equivalences, aiming to provide theoretical underpinnings of verification methods. We provide several relationships between such behavioural equivalences.
Keywords: mobility, timer, process algebra, bisimulation, behaviour, equivalence

1 Introduction

Process calculi are a family of formalisms used to model distributed systems. They provide algebraic laws allowing a high-level description and analysis of concurrent processes, behavioural equivalences (e.g., bisimulations) between processes, and automated tools for the verification of interaction, communication, and synchronization between processes. During the past couple of decades, a number of calculi supporting process mobility were defined and studied; in particular, π-calculus [17], fusion calculus [20], join calculus [13], and mobile ambients [5]. Various specific devices were introduced to obtain such formalisms, including explicit locations in distributed π-calculus [15], explicit migration and timers in timed distributed π-calculus [12], and timed mobile ambients [1]. Time is an important aspect of distributed computing systems, and can play a key role in their formal description. Since time is a complex subject, its introduction to the domain of process calculi has received a lot of attention in, e.g., [2, 3, 6, 16, 19, 25]. Papers like these assume the existence of a global clock which is usually required in the description of complex systems. However, there are several applications and systems for which considering a global clock would be inappropriate. This paper presents such an approach, in which local clocks operate

independently, and so processes at different locations evolve asynchronously. On the other hand, processes operating at the same location evolve synchronously. Overall, temporal evolution of mobile processes is governed by independent local clocks and their migration timeouts.

The ever increasing complexity of mobile processes calls for the development of effective techniques and tools for the automated analysis and verification of their properties including, in particular, behavioural equivalences between systems. Bisimulation is the most common mathematical concept used to capture behavioural equivalence between processes. The corresponding equivalence relation, called bisimilarity, is used to abstract from certain details of the systems, and is widely accepted as a standard behavioural equivalence over processes. Several kinds of bisimulations had been defined (e.g, strong or weak bisimulation for π-calculus [17]).

In the paper [9], we defined TiMo, a basic language for mobile systems in which it is possible to add timers to control process mobility and interaction. After that, in [11], a local clock was assigned to each location of a system modelled in TiMo. Each such clock determines the timing of actions executed at the corresponding location. Then, starting from TiMo, we created a flexible software platform supporting the specification of agents and their physical distribution, allowing also a timed migration in a distributed environment [8]. We obtained this implementation by using an advanced software technology, creating a platform for mobile agents with time constraints. In this paper, we consider the same specification language for mobile agents with timeouts, i.e., TiMo, and define various behavioural equivalences taking into account both timers and locations (for migration). We study both the strong bounded timed bisimulation and strong open bounded timed bisimulation. We then relax the bisimulations being considered, by applying the so-called "up-to" technique. In [10] we discussed the *TravelShop* running example, in which clients buy tickets to predefined destinations from travel agents. One can use the bisimilarities defined in this paper to differentiate, for example, between two travel agents using the same databases for prices, but having different delays for providing the answer (in an urgent situation, the faster one would be preferred). On the other hand, it is possible to define equivalence classes of agents offering similar services with respect to the waiting time (possibly up to an acceptable time difference). As *TravelShop* is a non-trivial example, in this paper we use a simpler one in order to illustrate how the newly defined bisimulations can be applied.

The paper is organised as follows. We start in Section 2 with a brief presentation of TiMo, including its syntax and operational semantics, and introduce an example to illustrate the basic features of TiMo. In Section 3, we formally define in the framework of TiMo a number timed bisimulations, and discuss some of their properties. Conclusion and references end the paper. Due to space limitations, proofs of formal results are omitted.

2 Syntax and Semantics of TiMo

Timing constraints for migration allow one to specify what is the time instant for a process to move from one location to another. A timer denoted by $\Delta 3$ associated to a migration action $go^{\Delta 3} work$ indicates that the process moves to location *work* after 3 time units. It is also possible to constrain the waiting for a communication on a channel; if a communication action does not happen before a deadline, the waiting process gives up and switches its operation to an alternative. A timer $\Delta 5$ associated to an output action $a^{\Delta 5}!\langle 10\rangle$ makes the channel available for communication only for the period of 5 time units. We assume suitable data sets including a set *Loc* of locations and a set *Chan* of communication channels. We use a set *Id* of process identifiers, and each $id \in Id$ has the arity m_{id}.

The syntax of TiMo, an extension of TiMo from [8], is given in Table 1, where P are processes and N networks. Moreover, for each $id \in Id$ there is a unique process definition of the form:

$$id(u_1, \ldots, u_{m_{id}}) \stackrel{df}{=} P_{id} \ ,$$

where the u_i's are distinct variables playing the role of parameters. The process 0 is called *primitive*. In Table 1, it is assumed that:

- $a \in Chan$ is a channel;
- $lt \in \mathbb{N} \cup \{\infty\}$ is a deadline, where lt stands for *local time*;
- v is a tuple of expressions built from values, variables and allowed operations;
- u is a tuple of variables;
- l is a location or a location variable;
- $\textcircled{S}P$ is a purely technical device which is used in the subsequent formalization of the structural operational semantics of TiMo; intuitively, $\textcircled{S}P$ specifies a process P which is temporarily (i.e., until a clock tick) stalled and so cannot execute any action.

Processes	$P ::= a^{\Delta lt}!\langle v\rangle$ then P else P'	(output)
	$\mid\ a^{\Delta lt}?(u)$ then P else P'	(input)
	$\mid\ go^{\Delta lt}\ l$ then P else P'	(move)
	$\mid\ P \mid P'$	(parallel)
	$\mid\ 0$	(termination)
	$\mid\ id(v)$	(recursion)
	$\mid\ \textcircled{S}P$	(stalling)
Located processes	$L ::= l[\![P]\!]$	
Networks	$N ::= L \mid L \mid N$	

Table 1. TiMo Syntax

The only variable binding constructor is "$a^{\Delta lt}?(u)$ then P else P'" which binds the variable u within P (but *not* within P'). We use $fv(P)$ to denote the free variables of a process P (and similarly for networks). For a process definition, we assume that $fv(P_{id}) \subseteq \{u_1, \ldots, u_{m_{id}}\}$ and so the free variables of P_{id} are parameter bound. Processes are defined up to alpha-conversion, and $\{v/u, \ldots\}P$ is obtained from P by replacing all free occurrences of a variable u by v, possible after alpha-converting P in order to avoid clashes.

Intuitively, a process "$a^{\Delta lt}!\langle v\rangle$ then P else P'" attempts to send a tuple of values v over channel a for lt time units. If successful, it continues as process P; otherwise it continues as process P'. Similarly, "$a^{\Delta lt}?(u)$ then P else P'" is a process that attempts for lt time units to input a tuple of values and substitute them for the variables u. Mobility is implemented by a process "$go^{\Delta lt}l$ then P else P'" which moves from the current location to the location given by l after lt time units. If the movement does not take place in lt time units, because location l does not exist ($l \notin Loc$), the process continues at the current location with P'. Note that since l can be a variable, and so its value is assigned dynamically through the communication with other processes, migration actions support a flexible scheme for the movement of processes from one location to another. Processes are further constructed from the (terminated) process 0 and parallel composition $P|P'$. A located process $l[\![P]\!]$ specifies a process P running at location l, and a network is composed out of its components $N \mid N'$. A network N is well-formed if the following hold: there are no free variables in N, and there are no occurrences of the special symbol Ⓢ in N. The set of processes is denoted by $\mathcal{P}$, the set of located processes by $\mathcal{L}$, and the set of networks by $\mathcal{N}$.

By delaying the migration to another location, we can model in a simple way the movement time of processes along the network which is, in general, outside the control of a system designer.

The first component of the operational semantics of TiMo is the structural equivalence $\equiv$ on networks; it is the smallest congruence such that the first four equalities in Table 2 hold. Its role is to rearrange a network in order to apply the action rules which are also given in Table 2. Using the first four equalities in Table 2, one can always transform a given network N into a finite parallel composition of located processes of the form $l_1[\![P_1]\!] \mid \ldots \mid l_n[\![P_n]\!]$ such that no process P_i has the parallel composition operator at its topmost level. Each located process $l_i[\![P_i]\!]$ is called a component of N, and the parallel composition is called a *component decomposition* of the network N.

Table 2 introduces two kinds of rules, $N \xrightarrow{\lambda} N'$ and $N \xrightarrow{\surd_l} N'$. The former is an execution of an action λ, and the latter a time step at location l. In the rule (Move), if location $l' \in Loc$, the process from location l migrates to l'. In the rule (Time), $N \not\rightarrow_l$ means that no action at location l (i.e, no action λ of the form $l \triangleright l'$ or $a\langle v\rangle@l$ or $id@l$) can be applied to N. It can be noticed that in rule (Time) we use negative premises, i.e., an activity is performed in the absence of other actions. This is due to the fact that sequencing the evolution in time units can only be defined using negative premises, as done for sequencing

(NComm)	$N \mid N' \equiv N' \mid N$
(NAssoc)	$(N \mid N') \mid N'' \equiv N \mid (N' \mid N'')$
(NSplit)	$l[\![P \mid P']\!] \equiv l[\![P]\!] \mid l[\![P']\!]$
(LComm)	$l[\![P \mid Q]\!] \equiv l[\![Q \mid P]\!]$
(Move)	$\dfrac{l' \in Loc}{l[\![go^{\Delta 0} l' \text{ then } P \text{ else } Q]\!] \xrightarrow{l \triangleright l'} l'[\![\circledS P]\!]}$
(Com)	$l[\![a^{\Delta lt}!\langle v\rangle \text{ then } P \text{ else } Q \mid a^{\Delta lt'}?(u) \text{ then } P' \text{ else } Q']\!] \xrightarrow{a\langle v\rangle @ l} l[\![\circledS P \mid \circledS\{v/u\}P']\!]$
(Call)	$l[\![id(v)]\!] \xrightarrow{id@l} l[\![\circledS\{v/u\}P_{id}]\!]$
(Par)	$\dfrac{N \xrightarrow{\lambda} N'}{N \mid N'' \xrightarrow{\lambda} N' \mid N''}$
(Equiv)	$\dfrac{N \equiv N' \qquad N' \xrightarrow{\lambda} N'' \qquad N'' \equiv N'''}{N \xrightarrow{\lambda} N'''}$
(Time)	$\dfrac{N \not\xrightarrow{}_l}{N \xrightarrow{\surd_l} \phi_l(N)}$

Table 2. TiMo Operational Semantics

processes in [4, 14, 18]. Moreover, $\phi_l(N)$ is obtained by taking the component decomposition of N and simultaneously replacing all components:

$$l[\![a^{\Delta lt}\omega \text{ then } P \text{ else } Q]\!] \text{ by } \begin{cases} l[\![Q]\!] & \text{if } lt = 0; \\ l[\![a^{\Delta lt-1}\omega \text{ then } P \text{ else } Q]\!] & \text{otherwise.} \end{cases}$$

$$l[\![go^{\Delta lt} l' \text{ then } P \text{ else } Q]\!] \text{ by } \begin{cases} l[\![Q]\!] & \text{if } lt = 0; \\ l[\![go^{\Delta lt-1} l' \text{ then } P \text{ else } Q]\!] & \text{otherwise.} \end{cases}$$

where ω stands for $!\langle v\rangle$ or $?(u)$. After that, all the occurrences of the symbol Ⓢ in N are erased (this is done since processes that were unfolded or interacted with other processes or migrated need to be activated).

The rules of Table 2 express executions of individual actions. A complete computational step in distributed systems is captured by a derivation of the form $N \xRightarrow{\Lambda@l} N'$, where $\Lambda = \{\lambda_1, \ldots, \lambda_m\}$ $(m \geq 0)$ is a finite multiset of actions for some location l (i.e., actions λ_i of the form $l \triangleright l'$ or $a\langle v\rangle@l$ or $id@l$) such that

$$N \xrightarrow{\lambda_1} N_1 \ldots N_{m-1} \xrightarrow{\lambda_m} N_m \xrightarrow{\surd_l} N' \ .$$

That means that a derivation is a condensed representation of a sequence of individual actions followed by a clock tick, all happening at the same location. Intuitively, we capture the cumulative effect of the concurrent execution of the multiset of actions Λ at location l, and say that N' is directly reachable from N. Note that if there is only a time progression at a location l, we write $N \stackrel{\emptyset@l}{\Longrightarrow} N'$. We now give an example that shows the modelling features of our formalism. The example will be used in the next section to illustrate various timed bisimulations defined in this paper.

Example 1. To exemplify how TIMO works, we use an *UrgentTravel* running example in which a client process attempt to initiate a *travel* process as soon as possible it receives an offer. The scenario involves three locations with their local clocks and three processes. The role of each location is as follows: (i) *office* is a location where the *client* process starts and ends its journey, (ii) $agency_i$, for $1 \leq i \leq 2$ are the travel agencies from which the client can found out about the tickets prices. The role of each of the processes is:

- *client* is a process that initially resides the *office* location, and is determined to pay for a flight as soon as it receive one *offer* from the two existing agencies. After sending an email to each agency, it awaits for the quickest response to initiate the *travel*;
- $agent_i$ $(1 \leq i \leq 2)$ is a process that resides inside the $agency_i$ location, answers to the emails received from *clients* and sends them the *agency* offers.

In this formalism we use *timers* in order to impose deadlines on the execution of communications and migrations. Each location has its local clock which determines the timing of actions executed at that location. The timeout of a migration action indicates the network time limit for that action . We assume local clock for the three locations such that:

- *office* and $agency_1$ have the same time unit;
- $agency_2$ has a double time unit (i.e., equals with 2 time units of the local clock of the locations *office* and $agency_1$).

The specifications that capture the essential features of the scenario are:

$$
\begin{array}{lllll}
agent_i = a^{\Delta 5}!\langle offer_i \rangle & \text{then } agent_i & \text{else } agent_i & & \\
client = go^{\Delta 2} agency_1 & \text{then } a^{\Delta 1}?(x) & \text{then } go^{\Delta 2} office & \text{then } d^{\Delta 1}!\langle x \rangle & \\
 & & & \text{else } 0 & \\
 & & \text{else } 0 & & \\
 & \text{else } 0 & & & \\
\quad | \; go^{\Delta 2} agency_2 & \text{then } a^{\Delta 1}?(x) & \text{then } go^{\Delta 2} office & \text{then } d^{\Delta 1}!\langle x \rangle & \\
 & & & \text{else } 0 & \\
 & & \text{else } 0 & & \\
 & \text{else } 0 & & & \\
\quad | \; d^{\Delta 6}?(y) & \text{then } travel & \text{else } client & &
\end{array}
$$

In order to simplify the writing we have not added any alternatives for the communication action $d^{\Delta t}!\langle x \rangle$. Next we illustrate the execution of $UrgentTravel$.

The expired timers cause an immediate migration or the selection of an offer received in the communication action.

The initial network is

$$
\begin{array}{rlll}
\multicolumn{4}{l}{UrgentTravel = office[\![client]\!] \mid agency_1[\![agent_1]\!] \mid agency_2[\![agent_2]\!]} \\
\multicolumn{4}{l}{\xrightarrow{\emptyset@office}\xrightarrow{\emptyset@office}\xrightarrow{\emptyset@agency_1}\xrightarrow{\emptyset@agency_1}\xrightarrow{\emptyset@agency_2}} & 5\times \text{ (Time)} \\
office[\![go^{\Delta 0} agency_1 & \text{then } a^{\Delta 1}?(x) & \text{then } go^{\Delta 2} office & \text{then } d^{\Delta 1}!\langle x\rangle \\
 & & & \text{else } 0 \\
 & & \text{else } 0 & \\
 & \text{else } 0 & & \\
\mid go^{\Delta 0} agency_2 & \text{then } a^{\Delta 1}?(x) & \text{then } go^{\Delta 2} office & \text{then } d^{\Delta 1}!\langle x\rangle \\
 & & & \text{else } 0 \\
 & & \text{else } 0 & \\
 & \text{else } 0 & & \\
\mid d^{\Delta 4}?(y) & \text{then } travel & \text{else } client]\!] & \\
\mid agency_1[\![a^{\Delta 3}!\langle offer_1\rangle & \text{then } agent_1 & \text{else } agent_1]\!] & \\
\mid agency_2[\![a^{\Delta 4}!\langle offer_2\rangle & \text{then } agent_2 & \text{else } agent_2]\!] & \\
\multicolumn{4}{l}{\xrightarrow{\{office \triangleright agency_1,\ office \triangleright agency_2\}@office}} & 2\times \text{ (Move)} \\
\multicolumn{4}{l}{\xrightarrow{\{a\langle offer_1\rangle@agency_1\}@agency_1}\xrightarrow{\{a\langle offer_2\rangle@agency_2\}@agency_2}} & 2\times \text{ (Com)} \\
\multicolumn{4}{l}{office \quad [\![d^{\Delta 4}?(y) \text{ then } travel \text{ else } client]\!]} \\
\multicolumn{4}{l}{\mid agency_1\ [\![agent_1} \\
\multicolumn{4}{l}{\quad \mid go^{\Delta 2} office \text{ then } d^{\Delta 1}!\langle offer_1\rangle \text{ else } 0]\!]} \\
\multicolumn{4}{l}{\mid agency_2\ [\![agent_2} \\
\multicolumn{4}{l}{\quad \mid go^{\Delta 2} office \text{ then } d^{\Delta 1}!\langle offer_2\rangle \text{ else } 0]\!]} \\
\multicolumn{4}{l}{\xrightarrow{\emptyset@office}\xrightarrow{\emptyset@office}\xrightarrow{\emptyset@agency_1}\xrightarrow{\emptyset@agency_1}\xrightarrow{\emptyset@agency_2}} & 5\times \text{ (Time)} \\
\multicolumn{4}{l}{office \quad [\![d^{\Delta 1}?(y) \text{ then } travel \text{ else } client]\!]} \\
\multicolumn{4}{l}{\mid agency_1\ [\![a^{\Delta 2}!\langle offer_1\rangle \text{ then } agent_1 \text{ else } agent_1} \\
\multicolumn{4}{l}{\quad \mid go^{\Delta 0} office \text{ then } d^{\Delta 1}!\langle offer_1\rangle \text{ else } 0]\!]} \\
\multicolumn{4}{l}{\mid agency_2\ [\![a^{\Delta 4}!\langle offer_2\rangle \text{ then } agent_2 \text{ else } agent_2} \\
\multicolumn{4}{l}{\quad \mid go^{\Delta 1} office \text{ then } d^{\Delta 1}!\langle offer_2\rangle \text{ else } 0]\!]} \\
\multicolumn{4}{l}{\xrightarrow{\{agency_1 \triangleright office\}@agency_1}} & 1\times \text{ (Move)} \\
\multicolumn{4}{l}{\xrightarrow{\{d\langle offer_1\rangle@office\}@office}} & 1\times \text{ (Com)} \\
\multicolumn{4}{l}{office \quad [\![travel\{offer_1/y\}]\!]} \\
\multicolumn{4}{l}{\mid agency_1\ [\![a^{\Delta 2}!\langle offer_1\rangle \text{ then } agent_1 \text{ else } agent_1]\!]} \\
\multicolumn{4}{l}{\mid agency_2\ [\![a^{\Delta 4}!\langle offer_2\rangle \text{ then } agent_2 \text{ else } agent_2} \\
\multicolumn{4}{l}{\quad \mid go^{\Delta 1} office \text{ then } d^{\Delta 1}!\langle offer_2\rangle \text{ else } 0]\!]}
\end{array}
$$

3 Timed Bisimulation in TiMo

In what follows, we define various equivalences for processes and networks by considering their temporal behaviour. In a similar way as in timed distributed π-calculus of [7], we start by extending the standard notion of strong bisimilarity to take into account timed transitions.

Definition 1 (strong timed simulation and bisimulation).
Let $\mathcal{R} \subseteq \mathcal{N} \times \mathcal{N}$ be a binary relation on networks of processes.

1. *$\mathcal{R}$ is a* strong timed simulation *(ST simulation) if*

$$(N_1, N_2) \in \mathcal{R} \ \wedge \ N_1 \xrightarrow{\lambda} N_1' \implies \exists N_2' \in \mathcal{N} : \ N_2 \xrightarrow{\lambda} N_2' \ \wedge \ (N_1', N_2') \in \mathcal{R} \ .$$

2. *$\mathcal{R}$ is a* strong timed bisimulation *(ST bisimulation) if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are strong timed simulations.*
3. *The* strong timed bisimilarity *is the union $\sim$ of all ST bisimulations.*

Essentially, the above definition treats timed transitions just as any other transitions, and therefore coincides with the original notion of bisimilarity for labelled transition systems. It is easy to check that $\sim$ is an equivalence relation, and also the largest strong timed bisimulation. From the point of view of the evolutionary behaviour of TiMo networks, a crucial result is that strong timed bisimulation can be used to compare the complete computational steps of two systems.

Theorem 1. *Let N_1, N_2 be two networks such that $N_1 \sim N_2$.*
If $N_1 \stackrel{\Lambda@l}{\Longrightarrow} N_1'$ then there exists $N_2' \in \mathcal{N}$ such that

$$N_2 \stackrel{\Lambda@l}{\Longrightarrow} N_2' \ \ \textit{and} \ \ N_1' \sim N_2' \ .$$

Example 2. Inspired by the *UrgentTravel* of Example 1, consider the following two located processes:

$$\begin{array}{rlll}
L_1 = agency_1[\![a^{\Delta 1}?(x) & \text{then } go^{\Delta 4} office & \text{then } d^{\Delta 1}!\langle x\rangle & \\
 & & \text{else } 0 & \\
 & \text{else } 0 & & \\
\mid a^{\Delta 3}!\langle offer_1\rangle & \text{then } agent_1 & \text{else } agent_1]\!] & \\
L_2 = agency_1[\![a^{\Delta 2}?(x) & \text{then } go^{\Delta 3} office & \text{then } d^{\Delta 1}!\langle x\rangle & \\
 & & \text{else } 0 & \\
 & \text{else } 0 & & \\
\mid a^{\Delta 4}!\langle offer_1\rangle & \text{then } agent_1 & \text{else } agent_1]\!] &
\end{array}$$

This means that all the processes having the same timer for go are equivalent, even the timers of the communication channels differ. However, if the *client* reaches *office* after different number of time units, the two located processes are not strong timed bisimilar, i.e., ($L_1 \not\sim L_2$), since they have different evolutions in time after 3 units of time.

$$\begin{array}{ll}
L_1 \stackrel{\{a\langle offer_1\rangle@agency_1\}@agency_1}{\Longrightarrow} & \text{(Com)} \\
\quad agency_1[\![go^{\Delta 4} office \text{ then } d^{\Delta 1}!\langle offer_1\rangle \text{ else } 0 \mid agent_1]\!] & \\
\quad \stackrel{\emptyset@agency_1}{\Longrightarrow}\stackrel{\emptyset@agency_1}{\Longrightarrow}\stackrel{\emptyset@agency_1}{\Longrightarrow} & 3\times \text{ (Time)} \\
\quad agency_1[\![go^{\Delta 1} office \text{ then } d^{\Delta 1}!\langle offer_1\rangle \text{ else } 0 & \\
\qquad \mid a^{\Delta 2}!\langle offer_1\rangle \text{ then } agent_1 \text{ else } agent_1]\!] & \\
\quad \stackrel{\emptyset@agency_1}{\Longrightarrow} & \text{(Time)} \\
\quad agency_1[\![go^{\Delta 0} office \text{ then } d^{\Delta 1}!\langle offer_1\rangle \text{ else } 0 &
\end{array}$$

$$\mid a^{\Delta 1}!\langle \mathit{offer}_1\rangle \text{ then } \mathit{agent}_1 \text{ else } \mathit{agent}_1[\![$$

$$\dots$$

$$L_2 \xrightarrow{\{a\langle \mathit{offer}_1\rangle @ \mathit{agency}_1\}@\mathit{agency}_1} \quad \text{(Com)}$$

$$\mathit{agency}_1[\![\mathit{go}^{\Delta 3}\mathit{office} \text{ then } d^{\Delta 1}!\langle \mathit{offer}_1\rangle \text{ else } 0 \mid \mathit{agent}_1[\![$$

$$\xrightarrow{\emptyset@\mathit{agency}_1}\xrightarrow{\emptyset@\mathit{agency}_1}\xrightarrow{\emptyset@\mathit{agency}_1} \quad 3\times \text{ (Time)}$$

$$\mathit{agency}_1[\![\mathit{go}^{\Delta 0}\mathit{office} \text{ then } d^{\Delta 1}!\langle \mathit{offer}_1\rangle \text{ else } 0 \quad \text{(HPar)}$$

$$\mid a^{\Delta 2}!\langle \mathit{offer}_1\rangle \text{ then } \mathit{agent}_1 \text{ else } \mathit{agent}_1[\![$$

$$\xrightarrow{\{\mathit{agency}_1 \triangleright \mathit{office}\}@\mathit{agency}_1} \quad \text{(Move)}$$

$$\mathit{office}[\![d^{\Delta 1}!\langle \mathit{offer}_1\rangle[\![$$

$$\dots$$

The above definition of equivalence compares the evolution of whole networks, but does not provide means for reasoning about equivalence of compositionally defined networks. To be able to reason in this way, one may augment (only for the purpose of dealing with equivalences) the operational semantics of processes with two additional rules relating to communication, which intuitively represent individual evolutions of interacting processes:

$$\text{(Snd)} \qquad l[\![a^{\Delta lt}!\langle v\rangle \text{ then } P \text{ else } Q]\!] \xrightarrow{a!\langle v\rangle@l} l[\![\mathbb{S}P]\!]$$

$$\text{(Rcv)} \quad l[\![a^{\Delta lt}?(u) \text{ then } P \text{ else } Q]\!] \xrightarrow{a?\langle v\rangle@l} l[\![\mathbb{S}\{v/u\}P]\!]$$

Even when these two new rules are added, rule (Com) remains unchanged. However, in this paper, we concentrate on the network-level evolutions, leaving the detailed treatment of compositionally defined network to a future work.

3.1 Strong Bounded Timed Equivalences

We first introduce some additional terminology:

- If $|Loc| = n$, then by $t = (t_1@l_1, \dots, t_n@l_n) \in (\mathbb{N}@Loc)^n$ we denote a tuple in which each location $l_k \in Loc$ has attached a natural number t_k.
- for $t, u \in (\mathbb{N}@Loc)^n$, we say that $t \leq u$ if $\forall l \in Loc$, $t_l@l$ and $u_l@l$ we have that $t_l \leq u_l$.
- σ is an function used to substitute free names, with other names.
- A *timed relation* over $\mathcal{N}$ is any relation $\mathcal{R} \subseteq \mathcal{N} \times (\mathbb{N}@Loc)^n \times \mathcal{N}$.
- The *identity timed relation* is $ident_t \stackrel{df}{=} \{(N, t, N) \mid N \in \mathcal{N},\ t \in (\mathbb{N}@Loc)^n\}$.
- The *inverse of a timed relation* $\mathcal{R}$ is

$$\mathcal{R}^{-1} \stackrel{df}{=} \{(N_2, t, N_1) \mid (N_1, t, N_2) \in \mathcal{R}\}\ .$$

- The *composition of timed relations* $\mathcal{R}_1$ and $\mathcal{R}_2$ is

$$\mathcal{R}_1\mathcal{R}_2 \stackrel{df}{=} \{(N, t, N'') \mid \exists N' \in \mathcal{N} :\ (N, t, N') \in \mathcal{R}_1 \wedge (N', t, N'') \in \mathcal{R}_2\}\ .$$

- If $\mathcal{R}$ is a timed relation and $t \in \mathbb{N}$ then

$$\mathcal{R}_t \stackrel{df}{=} \{(N_1, N_2) \mid (N_1, t, N_2) \in \mathcal{R}\}$$

  is $\mathcal{R}$'s *t-projection*. We also denote $\mathcal{R}_\infty \stackrel{df}{=} \bigcup_{t \in (\mathbb{N}@Loc)^n} \mathcal{R}_t$.
- $\mathcal{R}$ is a *timed equivalence* if $\mathcal{R}_\infty$ is an equivalence relation, and an *equivalence up-to time* $u \in \mathbb{N}$ if $\bigcup_{0 \le t < u} \mathcal{R}_t$ is an equivalence relation.

The first notion of equivalence takes into account the timed behaviour requiring an exact match of transitions of two networks, for their entire evolution. Sometimes these requirements are too strong. According to [21] where a similar approach is presented, real-time distributed systems usually require a certain behaviour within a given amount t of time units. That is why we restrict our equivalences up-to a time t, namely to a local time t_l for each location $l \in Loc$, defining *bounded* timed equivalences.

Definition 2 (strong bounded timed simulation and bisimulation). *Let $\mathcal{R} \subseteq \mathcal{N} \times (\mathbb{N}@Loc)^n \times \mathcal{N}$ be a timed relation over $\mathcal{N}$.*

1. *$\mathcal{R}$ is a* strong timed simulation *(SBT simulation) if*

$$\begin{cases} (N_1, (\ldots, t_l@l, \ldots), N_2) \in \mathcal{R} \\ N_1 \xrightarrow{\surd_l} N_1' \end{cases} \Longrightarrow \exists N_2' \in \mathcal{N} : \begin{cases} N_2 \xrightarrow{\surd_l} N_2' \\ (N_1', (\ldots, (t_l - 1)@l, \ldots), N_2') \in \mathcal{R} \end{cases}$$

   and

$$\begin{cases} (N_1, t, N_2) \in \mathcal{R} \\ N_1 \xrightarrow{\lambda} N_1' \end{cases} \Longrightarrow \exists N_2' \in \mathcal{N} : \begin{cases} N_2 \xrightarrow{\lambda} N_2' \\ (N_1', t, N_2') \in \mathcal{R}\ , \end{cases}$$

   where λ is not of the form $\surd_l$.
2. *$\mathcal{R}$ is a* strong timed bisimulation *(SBT bisimulation) if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are strong bounded timed simulations.*
3. *The* strong bounded timed bisimilarity *is the union $\simeq$ of all SBT bisimulations.*

Theorem 2. *Let N_1, N_2 be two networks s.t. $N_1 \simeq_t N_2$, with $t = (\ldots, t_l@l, \ldots)$. If $N_1 \stackrel{\Lambda@l}{\Longrightarrow} N_1'$ then there exists $N_2' \in \mathcal{N}$ such that*

$$N_2 \stackrel{\Lambda@l}{\Longrightarrow} N_2' \quad \textit{and} \quad N_1' \simeq_{t'} N_2', \textit{with } t' = (\ldots, (t_l - 1)@l, \ldots).$$

Corollary 1. *Let N_1, N_2 be two networks such that $N_1 \simeq N_2$. If $N_1 \stackrel{\Lambda@l}{\Longrightarrow} N_1'$ then there exists $N_2' \in \mathcal{N}$ such that $N_2 \stackrel{\Lambda@l}{\Longrightarrow} N_2'$ and $N_1' \simeq N_2'$.*

Example 3. Consider the two networks of Example 2. Even if those networks have different definitions, they are strong bounded timed bisimilar before the $3rd$ time unit at location $agency_1$ ($N_1 \simeq_{(3@agency_1, 0@office)} N_2$) since they have the same evolutions, at location $agency_1$, during this period. Hence N_1 and N_2 cannot be identified by strong timed bisimulation, but this is possible by using strong bounded timed bisimulation up-to time 3 at location $agency_1$. However if $t > (3@agency_1, 0@office)$ we have that $N_1 \not\simeq_t N_2$.

Proposition 1. *1. Identity, inverse, composition and union of SBT bisimulations are SBT bisimulations.*
2. $\simeq$ is a timed equivalence.
3. $\simeq$ is an SBT bisimulation.
4. $\simeq$ is the largest SBT bisimulation.

Strong bounded timed bisimulation satisfies the following properties showing that the equivalence up-to a certain deadline includes the equivalence up-to any bound below that deadline.

Proposition 2. *If $N \simeq_t N'$ then $N \simeq_u N'$, for every $u \leq t$.*

3.2 Strong Open Bounded Timed Equivalences

In this section we are looking to define a timed equivalence which is closed under arbitrary substitutions. According to the following example, the SBT equivalence is not closed under substitution.

Example 4. Consider the client from Example 1 decides to *query* someone from the office to which agency to go. We have two situations: the *client* decides to use the receives information (process P_1) or not (process P_2). In order to simplify the notations we use some generic processes P' and Q' to illustrate the evolution of the *client* after moving to a $agency_1$ or $agency_2$.

$N_1 = l[\![P_1]\!] \mid agency_1[\![R_1]\!] \mid agency_2[\![R_2]\!]$
$N_2 = l[\![P_2]\!] \mid agency_1[\![R_1]\!] \mid agency_2[\![R_2]\!]$
$P_1 = query^{\Delta 1}!\langle agency_1\rangle$ then P else Q
$\quad \mid query^{\Delta 5}?(u)$ then $(go^{\Delta 3}u$ then P' else $Q')$ else 0,
$P_2 = query^{\Delta 1}!\langle agency_1\rangle$ then P else Q
$\quad \mid query^{\Delta 5}?(u)$ then $(go^{\Delta 3}agency_1$ then P' else $Q')$ else 0.

We have $N_1 \simeq_t N_2$, for any $t = (t_1@l, t_2@agency_1, t_3@agency_2)$, since applying the rules for communication on channel *query*:

$l[\![P_1]\!] \xrightarrow{\{query\langle agency_1\rangle@l\}@l} l[\![go^{\Delta 3}agency_1$ then P' else $Q']\!]$ (Com), and
$l[\![P_2]\!] \xrightarrow{\{query\langle agency_1\rangle@l\}@l} l[\![go^{\Delta 3}agency_1$ then P' else $Q']\!]$ (Com),

we obtain the same network N that has the same evolution ($N \simeq_t N$). However, if for these two processes P_1 and P_2 we consider the substitution $\sigma = \{agency_2/agency_1\}$ (the *client* receives the answer $agency_2$), after applying

$l[\![P_1\sigma]\!] \xrightarrow{\{query\langle agency_2\rangle@l\}@l} l[\![go^{\Delta t}agency_2$ then P' else $Q']\!]$ (Com), and
$l[\![P_2\sigma]\!] \xrightarrow{\{query\langle agency_1\rangle@l\}@l} l[\![go^{\Delta t}agency_1$ then P' else $Q']\!]$ (Com).

we obtain that the networks evolve in a different way, and thus $N_1 \not\simeq_t N_2$.

Thus we define the following equivalence.

Definition 3 (strong open bounded timed simulation and bisimulation). *Let $\mathcal{R} \subseteq \mathcal{N} \times (\mathbb{N}@Loc)^n \times \mathcal{N}$ be a timed relation over $\mathcal{N}$.*

1. $\mathcal{R}$ *is a* strong open bounded timed simulation *(SOBT simulation) if, for every substitution* σ*,*

$$\begin{cases} (N_1, (\ldots, t_l@l, \ldots), N_2) \in \mathcal{R} \\ N_1\sigma \xrightarrow{\surd_{l\sigma}} N_1' \end{cases} \Longrightarrow \exists N_2' \in \mathcal{N} : \begin{cases} N_2\sigma \xrightarrow{\surd_{l\sigma}} N_2' \\ (N_1', (\ldots, (t_l - 1)@l, \ldots)\sigma, N_2') \in \mathcal{R} \end{cases}$$

and

$$\begin{cases} (N_1, t', N_2) \in \mathcal{R} \\ N_1\sigma \xrightarrow{\lambda} N_1' \end{cases} \Longrightarrow \exists N_2' \in \mathcal{N} : \begin{cases} N_2\sigma \xrightarrow{\lambda} N_2' \\ (N_1', t'\sigma, N_2') \in \mathcal{R}\ , \end{cases}$$

where λ *is not of the form* $\surd_l$*.*

2. $\mathcal{R}$ *is a* strong open bounded timed bisimulation *(SBT bisimulation) if both* $\mathcal{R}t$ *and* $\mathcal{R}^{-1}$ *are strong open bounded timed simulations.*
3. *The* strong open bounded timed bisimilarity *is the union* $\simeq^o$ *of all SOBT bisimulations.*

The intuition is that two located processes are equivalent whenever all possible instantiations (substitutions of their free names) have matching transitions.

Proposition 3.

1. *Identity, inverse, composition and union of SOBT bisimulations are SOBT bisimulations;*
2. $\simeq^o$ *is a timed equivalence;*
3. $\simeq^o$ *is a SOBT bisimulation;*
4. $\simeq^o$ *is the largest SOBT bisimulation.*

Proposition 4. *If* $N \simeq^o_t N'$ *then* $N \simeq^o_u N'$*, for every* $u \leq t$*.*

Definition 4 (closure under substitutions). *A timed relation* $\mathcal{R}$ *is said to be* closed under substitutions *if whenever* $(N, t, N') \in \mathcal{R}$*, then* $(N\sigma, t\sigma, N'\sigma) \in \mathcal{R}$ *for any substitution* σ*.*

Proposition 5.

1. *If* $\mathcal{R}$ *is a SBT bisimulation closed under substitution, then it is a SOBT bisimulation.*
2. *If* $\mathcal{R}$ *is a SOBT bisimulation, then it is also a SBT.*
3. $\simeq^o \subsetneq \simeq$*.*
4. $\simeq^o$ *is closed under substitutions.*

Theorem 3. *Let* N_1, N_2 *be two networks s.t.* $N_1 \simeq^o_t N_2$*, with* $t = (\ldots, t_l@l, \ldots)$*. If* $N_1 \stackrel{\Lambda@l}{\Longrightarrow} N_1'$ *then there exists* $N_2' \in \mathcal{N}$ *such that*

$$N_2 \stackrel{\Lambda@l}{\Longrightarrow} N_2' \quad \textit{and} \quad N_1' \simeq^o_{t'} N_2',, \textit{with } t' = (\ldots, (t_l - 1)@l, \ldots)\sigma.$$

Corollary 2. *Let* N_1, N_2 *be two networks such that* $N_1 \simeq^o N_2$*. If* $N_1 \stackrel{\Lambda@l}{\Longrightarrow} N_1'$ *then there exists* $N_2' \in \mathcal{N}$ *such that* $N_2 \stackrel{\Lambda@l}{\Longrightarrow} N_2'$ *and* $N_1' \simeq^o N_2'$*.*

3.3 Relaxing Timed Bisimulation by Using "Up-To" Technique

In what follows we use the "up-to" technique presented in [23, 24] in the context of bounded timed bisimulations. The standard proof technique to establish that N_1 and N_2 are bisimilar is to find a bisimulation $\mathcal{R}$ s.t. $(N_1, N_2) \in \mathcal{R}$ and $\mathcal{R}$ is closed under transitions, namely the derivatives (N_1', N_2') of (N_1, N_2) are also in $\mathcal{R}$. Sometimes it is difficult to find directly such a relation $\mathcal{R}$. Instead, there is an useful alternative technique, the so-called bisimulation "up-to" some relation $\mathcal{R}'$: for a relation $\mathcal{R}$, which is not a bisimulation, if $(N_1, N_2) \in \mathcal{R}$, then the derivatives (N_1', N_2') are in $\mathcal{R}'$. Under certain conditions we can establish that N_1 and N_2 are bisimilar. For this technique, in [22] is presented a general framework that works on untimed labelled transition systems. We cannot make direct use of that framework, but we can extend it in a straightforward manner to timed labelled transition systems. We begin by introducing a notion of "progressing" a timed relation towards another timed relation:

Definition 5. *Let $\mathcal{R}$, $\mathcal{R}'$ be any timed relations. We say that $\mathcal{R}$* **strongly progresses** *to $\mathcal{R}'$, written $\mathcal{R} \rightsquigarrow \mathcal{R}'$, if*

$$\begin{cases} (N_1, (\ldots, t_l@l, \ldots), N_2) \in \mathcal{R} \\ N_1 \xrightarrow{\surd_l} N_1' \end{cases} \Longrightarrow \exists N_2' \in \mathcal{N} : \begin{cases} N_2 \xrightarrow{\surd_l} N_2' \\ (N_1', (\ldots, (t_l - 1)@l, \ldots), N_2') \in \mathcal{R}' \end{cases}$$

and

$$\begin{cases} (N_1, t, N_2) \in \mathcal{R} \\ N_1 \xrightarrow{\lambda} N_1' \end{cases} \Longrightarrow \exists N_2' \in \mathcal{N} : \begin{cases} N_2 \xrightarrow{\lambda} N_2' \\ (N_1', t, N_2') \in \mathcal{R}' \end{cases}.$$

where λ is not of the form $\surd_l$.

This definition is similar to that of SBT bisimulation, except that the derivatives (N_1', t, N_2') must be in $\mathcal{R}'$ rather than in $\mathcal{R}$.

Remark 1. $\mathcal{R}$ is a SBT bisimulation if and only if $\mathcal{R} \rightsquigarrow \mathcal{R}$.

Proposition 6. *If $\mathcal{R} \rightsquigarrow \mathcal{R}'$ and $\mathcal{R}'$ is a SBT bisimulation, then $\mathcal{R}$ is also a SBT bisimulation.*

Therefore, to establish that $N_1 \simeq_t N_2$ it is enough to find a relation $\mathcal{R}$ with $(N_1, t, N_2) \in \mathcal{R}$ which strongly progresses to a known SBT bisimulation $\mathcal{R}'$. The choice of $\mathcal{R}'$ depends on the particular equivalence we are trying to establish. One of the most common cases is when $\mathcal{R}' = \simeq$. However, in general we may not have a relation $\mathcal{R}'$ known to be a bisimulation. Nevertheless we may find that $\mathcal{R}$ progresses to a relation $\mathcal{R}' = \mathcal{F}(\mathcal{R})$ for some function $\mathcal{F}$ over relations. The idea is that if $\mathcal{R}$ progresses to $\mathcal{F}(\mathcal{R})$ and $\mathcal{F}$ satisfies certain conditions, then $\mathcal{R}$ is included in $\simeq$. Thus, to establish $N_1 \simeq_t N_2$ we need to find such an $\mathcal{F}$ whenever $\mathcal{R}$ contains (N_1, t, N_2).

These functions $\mathcal{F}$ over relations are characterized in [24] as follows:

Definition 6. *A function $\mathcal{F}$ is* **strongly safe** *if for any timed relations $\mathcal{R}, \mathcal{R}'$, if $\mathcal{R} \subseteq \mathcal{R}'$ and $\mathcal{R} \rightsquigarrow \mathcal{R}'$, then $\mathcal{F}(\mathcal{R}) \subseteq \mathcal{F}(\mathcal{R}')$ and $\mathcal{F}(\mathcal{R}) \rightsquigarrow \mathcal{F}(\mathcal{R}')$.*

More details about the "up-to" techniques and safe functions can be found in [24].

4 Conclusion

This paper presents an approach in which local clocks operate independently, and so processes at different locations evolve asynchronously; on the other hand, processes operating at the same location evolve synchronously, and temporal evolution of mobile processes is governed by independent local clocks and their migration timeouts. A computational step captures the cumulative effect of the concurrent execution of a multiset of actions executed at a certain location.

In process calculi like distributed π-calculus, timed distributed π-calculus and other formalisms with explicit migration operators, bisimulations are used to compare behaviours of mobile processes evolving in distributed systems with explicit locations. Bisimulations are behavioural equivalences used to study the properties of a concurrent system by verifying its bisimilarity with a system known to enjoy those properties. Moreover, given the model of a system, bisimulations can be used to consider equivalent simplified models.

In this paper we define three behavioural equivalences between migrating process in distributed systems in terms of local time and locations. We prove that strong timed bisimulation can be used to compare the complete computational steps of two networks. We also prove that two networks that are strong bounded timed bisimilar up to a certain deadline t remain equivalent at any time u before t. Finally, we define a strong open bounded timed equivalence that is closed under arbitrary substitutions; to establish that two networks are strong bounded timed equivalent, it is enough to find a relation that strongly progresses to a known SBT relation.

Among the next steps to follow, we intend to work with these behavioural equivalences over mobile agents evolving in distributed systems by using a software platform presented extensively in [8].

Acknowledgement

The work of Bogdan Aman and Gabriel Ciobanu was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0919, while the work of Bogdan Aman was also supported by POSDRU/89/1.5/S/49944.

References

1. Aman, B., Ciobanu, G.: Timed mobile ambients for network protocols. In: Suzuki, K., Higashino, T., Yasumoto, K., El-Fakih, K. (eds.) FORTE. Lecture Notes in Computer Science, vol. 5048, pp. 234–250. Springer (2008)
2. Baeten, J.C.M., Bergstra, J.A.: Discrete time process algebra: Absolute time, relative time and parametric time. Fundam. Inform. 29(1-2), 51–76 (1997)
3. Berger, M.: Towards Abstractions for Distributed Systems. Ph.D. thesis, Department of Computing, Imperial College (2002)
4. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can't be traced. In: POPL. pp. 229–239 (1988)

5. Cardelli, L., Gordon, A.D.: Mobile ambients. In: Nivat, M. (ed.) FoSSaCS. Lecture Notes in Computer Science, vol. 1378, pp. 140–155. Springer (1998)
6. Chen, L.: Timed Processes: Models, Axioms and Decidability. Ph.D. thesis, School of Informatics, University of Edinburgh (1993)
7. Ciobanu, G.: Behaviour equivalences in timed distributed π-calculus. In: Wirsing, M., Banâtre, J.P., Hölzl, M.M., Rauschmayer, A. (eds.) Software-Intensive Systems and New Computing Paradigms, Lecture Notes in Computer Science, vol. 5380, pp. 190–208. Springer (2008)
8. Ciobanu, G., Juravle, C.: Flexible software architecture and language for mobile agents. Concurrency and Computation: Practice and Experience 24(6), 559571 (2012)
9. Ciobanu, G., Koutny, M.: Modelling and verification of timed interaction and migration. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE. Lecture Notes in Computer Science, vol. 4961, pp. 215–229. Springer (2008)
10. Ciobanu, G., Koutny, M.: Timed migration and interaction with access permissions. In: Butler, M., Schulte, W. (eds.) FM. Lecture Notes in Computer Science, vol. 6664, pp. 293–307. Springer (2011)
11. Ciobanu, G., Koutny, M.: Timed mobility in process algebra and petri nets. J. Log. Algebr. Program. 80(7), 377–391 (2011)
12. Ciobanu, G., Prisacariu, C.: Timers for distributed systems. Electr. Notes Theor. Comput. Sci. 164(3), 81–99 (2006)
13. Fournet, C., Gonthier, G.: The join calculus: A language for distributed mobile programming. In: Barthe, G., Dybjer, P., Pinto, L., Saraiva, J. (eds.) APPSEM. Lecture Notes in Computer Science, vol. 2395, pp. 268–332. Springer (2000)
14. Groote, J.F.: Transition system specifications with negative premises. Theoretical Computer Science 118, 263–299 (1993)
15. Hennessy, M.: A distributed π-calculus. Cambridge University Press (2007)
16. Hennessy, M., Regan, T.: A process algebra for timed systems. Inf. Comput. 117(2), 221–239 (1995)
17. Milner, R.: Communicating and mobile systems - the π-calculus. Cambridge University Press (1999)
18. Moller, F.: Axioms for Concurrency. Ph.D. thesis, Department of Computer Science, University of Edinburgh (1989)
19. Nicollin, X., Sifakis, J.: The algebra of timed processes, atp: Theory and application. Inf. Comput. 114(1), 131–178 (1994)
20. Parrow, J., Victor, B.: The fusion calculus: Expressiveness and symmetry in mobile processes. In: LICS. pp. 176–185. IEEE Computer Society (1998)
21. Posse, E., Dingel, J.: Theory and implementation of a real-time extension to the *pi*-calculus. In: Hatcliff, J., Zucca, E. (eds.) FMOODS/FORTE. Lecture Notes in Computer Science, vol. 6117, pp. 125–139. Springer (2010)
22. Sangiorgi, D.: On the proof method for bisimulation (extended abstract). In: Wiedermann, J., Hájek, P. (eds.) MFCS. Lecture Notes in Computer Science, vol. 969, pp. 479–488. Springer (1995)
23. Sangiorgi, D.: A theory of bisimulation for the π-calculus. Acta Inf. 33(1), 69–97 (1996)
24. Sangiorgi, D., Walker, D.: The π-Calculus - a theory of mobile processes. Cambridge University Press (2001)
25. Yi, W.: A Calculus of Real-Time Systems. Ph.D. thesis, Department of Computer Science, Chalmers University of Technology (1991)