COMPUTING SCIENCE

A Timed Mobility Semantics based on Rewriting Strategies

Gabriel Ciobanu, Maciej Koutny and Jason Steggles

TECHNICAL REPORT SERIES

No. CS-TR-1341 June 2012

TECHNICAL REPORT SERIES

No. CS-TR-1341 June, 2012

A Timed Mobility Semantics based on Rewriting Strategies

G. Ciobanu, M. Koutny and J. Steggles

Abstract

We consider TiMo (Timed Mobility) which is a process algebra for prototyping software engineering applications supporting mobility and timing constraints. We provide an alternative semantics of TiMo using rewriting logic; in particular, we develop a rewriting logic model based on strategies to describe a maximal parallel computational step of a TiMo specification. This new semantical model is proved to be sound and complete w.r.t. to the original operational semantics which was based on negative premises. We implement the rewriting model within the strategy-based rewriting system Elan, and provide an example illustrating how a TiMo specification is executed and how a range of (behavioural) properties are analysed.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

CIOBANU, G., KOUTNY, M., STEGGLES, J.

A Timed Mobility Semantics based on Rewriting Strategies
[By] G. Ciobanu, M. Koutny, J. Steggles

Newcastle upon Tyne: Newcastle University: Computing Science, 2012.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1341)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1341

Abstract

We consider TiMo (Timed Mobility) which is a process algebra for prototyping software engineering applications supporting mobility and timing constraints. We provide an alternative semantics of TiMo using rewriting logic; in particular, we develop a rewriting logic model based on strategies to describe a maximal parallel computational step of a TiMo specification. This new semantical model is proved to be sound and complete w.r.t. to the original operational semantics which was based on negative premises. We implement the rewriting model within the strategy-based rewriting system Elan, and provide an example illustrating how a TiMo specification is executed and how a range of (behavioural) properties are analysed.

About the authors

Gabriel Ciobanu is a senior researcher at the Institute of Computer Science, Romanian Academy. He is also a Professor at "Alexandru Ioan Cuza" University of Iasi, Romania.

Maciej Koutny obtained his MSc (1982) and PhD (1984) from the Warsaw University of Technology. In 1985 he joined the then Computing Laboratory of the University of Newcastle upon Tyne to work as a Research Associate. In 1986 he became a Lecturer in Computing Science at Newcastle, and in 1994 was promoted to an established Readership at Newcastle. In 2000 he became a Professor of Computing Science.

Dr L. Jason Steggles is a lecturer in the School of Computing Science, University of Newcastle. His research interests lie in the use of formal techniques to develop correct computing systems. In particular, he has worked extensively on using algebraic methods for specifying, prototyping and validating computing systems.

Suggested keywords

PROCESS ALGEBRA
MOBILITY
TIME
REWRITING LOGIC
STRATEGIES

A Timed Mobility Semantics based on Rewriting Strategies

Gabriel Ciobanu[1], Maciej Koutny[2], and Jason Steggles[2]
[1]Faculty of Computer Science, A. I. Cuza University of Iasi, 700483 Iasi, Romania.
email: gabriel@info.uaic.ro
[2]Department of Computing Science, University of Newcastle, U. K.
email: {maciej.koutny,jason.steggles}@ncl.ac.uk

June 28, 2012

Abstract

We consider TiMo (Timed Mobility) which is a process algebra for prototyping software engineering applications supporting mobility and timing constraints. We provide an alternative semantics of TiMo using rewriting logic; in particular, we develop a rewriting logic model based on strategies to describe a maximal parallel computational step of a TiMo specification. This new semantical model is proved to be sound and complete w.r.t. to the original operational semantics which was based on negative premises. We implement the rewriting model within the strategy-based rewriting system Elan, and provide an example illustrating how a TiMo specification is executed and how a range of (behavioural) properties are analysed.

1 Introduction

TiMo (Timed Mobility) is a process algebra proposed in [8] for prototyping software engineering applications in distributed system design. TiMo supports process mobility and interaction, and allows one to add timers to the basic migration and communication actions. Recently, the model has been extended to model security aspects such as access permissions [10]. The behaviour of TiMo specifications can be captured using a set of SOS rules or suitable Petri nets [9], both based on executing time actions with negative premises. In this paper, we provide an alternative semantics of TiMo using rewriting logic and strategies. Our aim is to obtain a semantical model of TiMo which can be used as the basis for developing efficient tool support and investigating different semantic choices.

Rewriting Logic (RL) [17] is an algebraic formalism for dynamic systems which uses equational specifications to define the states of a system, and rewrite rules to capture the dynamic state transitions. Strategies [5, 6] are an integral part of RL which provide control over the rewriting process, allowing important dynamic properties to be modelled. In our work, we develop a RL model for TiMo specifications. In particular, we formulate a strategy which captures the maximal parallel computational step of a TiMo specification, including its time rule based on negative premises. The resulting RL model is then formally validated, by showing that it is both *sound* and *complete* w.r.t. the original operational semantics of TiMo.

As a first attempt at developing tool support for TiMo based on the new semantics, we use the strategy-based rewrite system Elan [4, 6] to implement TiMo specifications. The simple example we discuss provides a useful insight into the proposed RL modelling approach, and illustrates the type of (behavioural) properties that can be analysed.

Processes	P ::=	$a^{\Delta t}\,!\,\langle \vec{v} \rangle$ `then` P `else` P' ∣	$(output)$
		$a^{\Delta t}\,?\,(\vec{u}{:}\vec{X})$ `then` P `else` P' ∣	$(input)$
		$\mathtt{go}^{\Delta t}\ l$ `then` P ∣	$(move)$
		$P \mid P'$ ∣	$(parallel)$
		$id(\vec{v})$ ∣	$(recursion)$
		stop ∣	$(termination)$
		Ⓢ P	$(stalling)$
Networks	N ::=	$l[\![\, P \,]\!]$ ∣ $N \mid N'$	
Definition		$id(u_1, \ldots, u_{m_{id}} : X_1^{id}, \ldots, X_{m_{id}}^{id}) \stackrel{\text{df}}{=} P_{id}$	(DEF)

Table 1: TIMO Syntax. Length of $\vec{u}$ is the same as $\vec{X}$, and length of $\vec{v}$ in $id(\vec{v})$ is m_{id}.

The paper is structured as follows. Section 2 describes the syntax and semantics of TIMO, and Section 3 briefly introduces RL and strategies. In Section 4, we develop an RL model of TIMO, and prove its correctness. In Section 5, we show how to implement our RL model within the ELAN, and discuss what properties can then be verified. Section 6 discusses related and future work.

2 TIMO (Timed Mobility Language)

TIMO (Timed Mobility) [8, 9, 10] is a process algebra for mobile systems where it is possible to add timers to the basic actions, and each location runs according to its own local clock which is invisible to processes. Processes have communication capabilities which are active up to a predefined time deadline. Other timing constraints specify the latest time for moving between locations.

We assume suitable data types together with associated operations, including a set *Loc* of *locations*, a set *Chan* of *communication channels*, and a set *Id* of process identifiers, where each $id \in Id$ has arity m_{id}. We use $\vec{x}$ to denote a finite tuple of elements $(x_1, \ldots, x_k)$ whenever it does not lead to a confusion.

The syntax of TIMO is given in Table 1, where P represents *processes* and N represents networks. Moreover, for each $id \in Id$, there is a unique process definition (DEF), where P_{id} is a process expression, the u_i's are distinct variables playing the role of parameters, and the X_i^{id}'s are data types. In Table 1, it is assumed that: (i) $a \in Chan$ is a channel, and $t \in \mathbb{N} \cup \{\infty\}$ represents a timeout; (ii) each v_i is an expression built from data values and variables; (iii) each u_i is a variable, and each X_i is a data type; (iv) l is a location or a location variable; and (v) Ⓢ is a special symbol used to state that a process is temporarily 'stalled'.

The only variable binding construct is $a^{\Delta t}\,?\,(\vec{u}{:}\vec{X})$ `then` P `else` P' which binds the variables $\vec{u}$ 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 as in (DEF), 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 the alpha-conversion, and $\{v/u, \ldots\}P$ is obtained from P by replacing all free occurrences of a variable u by v, etc, possibly after alpha-converting P in order to avoid clashes. Moreover, if $\vec{v}$ and $\vec{u}$ are tuples of the same length then $\{\vec{v}/\vec{u}\}P$ denotes $\{v_1/u_1, v_2/u_2, \ldots, v_k/u_k\}P$.

A process $a^{\Delta t}\,!\,\langle \vec{v} \rangle$ `then` P `else` P' attempts to send a tuple of values $\vec{v}$ over the channel a for t time units. If successful, it continues as process P; otherwise it continues as the alternative process P'. A process $a^{\Delta t}\,?\,(\vec{u}{:}\vec{X})$ `then` P `else` P' attempts for t time units to input a tuple of values of type $\vec{X}$ and substitute them for the variables $\vec{u}$. Mobility is implemented by a process $\mathtt{go}^{\Delta t}\ l$ `then` P which moves from the current location to the location l within t time units. Note that since l can be a variable, and so its value is assigned dynamically through communication with other

processes, migration actions support a flexible scheme for moving processes around a network. Processes are further constructed from the (terminated) process `stop` and parallel composition $P|P'$. Finally, process expressions of the form $\textcircled{S}P$ are a purely technical device which is used in the subsequent formalisation of structural operational semantics of TiMo; intuitively, $\textcircled{S}$ specifies that a process P is temporarily (i.e., until a clock tick) *stalled* and so cannot execute any action. A located process $l[\![P]\!]$ is a process running at location l, and a network is composed out of its components $N \mid N'$.

As an illustrative example, consider a simple workflow example in which a processing job moves from an initial location to a web service location and finally to a done location. If an error occurs with the web service then the job enters an error location. A pictorial representation of this example is given in Figure 1. The TiMo specification *WF* consists of four locations: *Init*;

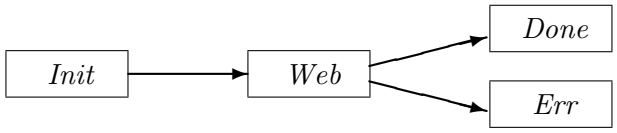

Figure 1: A pictorial representation of a simple TiMo work flow example.

Web; *Done*; and *Err*. The following process identifier definitions are used:

$$\begin{aligned} \mathit{job} &\stackrel{\text{df}}{=} a^{\Delta 1}\,\mathtt{?}\,(l{:}\mathit{Loc})\ \ \mathtt{then\ go}^{\Delta 1}\ l\ \mathtt{then}\ \mathit{job}\ \mathtt{else}\ \mathit{job} \\ \mathit{serv}(l{:}\mathit{Loc}) &\stackrel{\text{df}}{=} a^{\Delta 2}\,\mathtt{!}\,\langle l\rangle\ \ \mathtt{then}\ \mathit{serv}(l)\ \mathtt{else}\ \mathit{serv}(l) \\ \mathit{servErr}(l{:}\mathit{Loc}) &\stackrel{\text{df}}{=} a^{\Delta 2}\,\mathtt{!}\,\langle l\rangle\ \ \mathtt{then}\ \mathit{servErr}(l)\ \mathtt{else}\ \mathit{servErr}(\mathit{Err}) \end{aligned}$$

For instance, $\mathit{Init}\,[\![\,\mathit{job} \mid \mathit{serv}(\mathit{Web})\,]\!] \mid \mathit{Web}\,[\![\,\mathit{serv}(\mathit{Done})\,]\!]$ could be an initial TiMo network for this example.

A network N is *well-formed* if: (i) there are no free variables in N; (ii) there are no occurrences of the special symbol $\textcircled{S}$ in N; (iii) assuming that *id* is as in the recursive equation (Def), for every $\mathit{id}(\vec{v})$ occurring in N or on the right hand side of any recursive equation, the expression v_i is of type corresponding to X_i^{id}. We let $\mathit{Prs}(\mathit{TM})$ and $\mathit{Net}(\mathit{TM})$ represent the set of well-formed TiMo process and network terms respectively. The first component of the operational semantics of TiMo is the structural equivalence $\equiv$ on networks. It is the smallest congruence such that the equalities (Eq1–Eq3) in Table 2 hold. Using (Eq1–Eq3) one can always transform a given network N into a finite parallel composition of networks 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 subnetwork $l_i\,[\![\,P_i\,]\!]$ is called a *component* of N, the set of all components is denoted by $\mathit{comp}(N)$, and the parallel composition is called a *component decomposition* of the network N. Note that these notions are well defined since component decomposition is unique up to the permutation of the components. This follows from the rule (Call) which treats recursive definitions as function calls which take a unit of time. Another consequence of such a treatment is that it is impossible to execute an infinite sequence of action steps without executing any local clock ticks.

Table 2 introduces two kinds of operational semantics rules: $N \xrightarrow{\psi} N'$ and $N \xrightarrow{\surd_l} N'$. The former is an execution of an action ψ by some process, and the latter a unit time progression at location l. In the rule (Time), $N \not\rightarrow_l$ means that the rules (Call) and (Com) as well as (Move) with $\Delta t = \Delta 0$ cannot be applied to N for this particular location l. Moreover, $\phi_l(N)$ is obtained by taking the component decomposition of N and simultaneously replacing all the components of the form $l\,[\![\,\mathtt{go}^{\Delta t}\ l'\ \mathtt{then}\ P\,]\!]$ by $l\,[\![\,\mathtt{go}^{\Delta t-1}\ l'\ \mathtt{then}\ P\,]\!]$, and all components of the form $l\,[\![\,a^{\Delta t}\omega\ \mathtt{then}\ P\ \mathtt{else}\ Q\,]\!]$ (where ω stands for $\mathtt{!}\,\langle\vec{v}\rangle$ or $\mathtt{?}\,(\vec{u}{:}\vec{X})$) by $l\,[\![\,Q\,]\!]$ if $t = 0$, and

(Eq1-3) $N \mid N' \equiv N' \mid N \quad (N \mid N') \mid N'' \equiv N \mid (N' \mid N'') \quad l[\![P \mid P']\!] \equiv l[\![P]\!] \mid l[\![P']\!]$

(Call) $l[\![id(\vec{v})]\!] \xrightarrow{id@l} l[\![Ⓢ\{\vec{v}/\vec{u}\}P_{id}]\!]$ (Move) $l[\![\mathtt{go}^{\Delta t}\ l'\ \mathtt{then}\ P]\!] \xrightarrow{l'@l} l'[\![Ⓢ P]\!]$

(Com) $$\frac{v_1 \in X_1 \ \ldots \ v_k \in X_k}{l[\![a^{\Delta t}\,\mathtt{!}\,\langle\vec{v}\rangle\ \mathtt{then}\ P\ \mathtt{else}\ Q \mid a^{\Delta t'}\,\mathtt{?}\,(\vec{u}{:}\vec{X})\ \mathtt{then}\ P'\ \mathtt{else}\ Q']\!] \xrightarrow{a\langle\vec{v}\rangle@l} l[\![Ⓢ P \mid Ⓢ\{\vec{v}/\vec{u}\}P']\!]}$$

(Par) $$\frac{N \xrightarrow{\psi} N'}{N \mid N'' \xrightarrow{\psi} N' \mid N''}$$ (Time) $$\frac{N \not\longrightarrow_l}{N \xrightarrow{\surd_l} \phi_l(N)}$$

(Equiv) $$\frac{N \equiv N' \quad N' \xrightarrow{\psi} N'' \quad N'' \equiv N'''}{N \xrightarrow{\psi} N'''}$$

Table 2: Three rules of the structural equivalence (Eq1-Eq3), and six action rules (Call), (Move), (Com), (Par), (Equiv), (Time) of the operational semantics. In (Par) and (Equiv) ψ is an action, and in (Time) l is a location.

$l[\![a^{\Delta t-1}\omega\ \mathtt{then}\ P\ \mathtt{else}\ Q]\!]$ otherwise. After that, all the occurrences of the symbol Ⓢ in N are erased.

The above defines executions of individual actions. A complete computational step is captured by a *derivation* of the form $N \stackrel{\Psi}{\Longrightarrow} N'$, where $\Psi = \{\psi_1, \ldots, \psi_m\}$ $(m \geq 0)$ is a finite multiset of l-actions for some location l (i.e., actions of the form $id@l$ or $l'@l$ or $a\langle\vec{v}\rangle@l$) such that $N \xrightarrow{\psi_1} N_1 \cdots N_{m-1} \xrightarrow{\psi_m} N_m \xrightarrow{\surd_l} N'$. That is, 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 Ψ at location l. We say that N' is *directly reachable* from N. Note that whenever there is only a time progression at a location, we have $N \stackrel{\varnothing}{\Longrightarrow} N'$.

As an example, consider two derivation steps in the workflow network:

$$\begin{array}{l} \mathit{Init}[\![\mathit{job} \mid \mathit{serv}(\mathit{Web})]\!] \mid \mathit{Web}[\![\mathit{serv}(\mathit{Done})]\!] \\ \xRightarrow{\{\mathit{job}@\mathit{Init},\mathit{serv}@\mathit{Init}\}} \\ \mathit{Init}[\![a^{\Delta 1}\,\mathtt{?}\,(l{:}\mathit{Loc})\ \ \mathtt{then}\ \mathtt{go}^{\Delta 1}\ \ l\ \mathtt{then}\ \mathit{job}\ \mathtt{else}\ \mathit{job} \mid a^{\Delta 2}\,\mathtt{!}\,\langle \mathit{Web}\rangle\ \ \mathtt{then} \\ \mathit{serv}(\mathit{Web})\ \mathtt{else}\ \mathit{serv}(\mathit{Web})]\!] \mid \mathit{Web}[\![\mathit{serv}(\mathit{Done})]\!] \\ \xRightarrow{\{a\langle \mathit{Web}\rangle@\mathit{Init}\}} \\ \mathit{Init}[\![\mathtt{go}^{\Delta 1}\ \ \mathit{Web}\ \mathtt{then}\ \mathit{job}\ \mathtt{else}\ \mathit{job} \mid \mathit{serv}(\mathit{Web})]\!] \mid \mathit{Web}[\![\mathit{serv}(\mathit{Done})]\!] \\ \xRightarrow{\{\mathit{Web}@\mathit{Init},\mathit{serv}@\mathit{Init}\}} \\ \mathit{Init}[\![a^{\Delta 2}\,\mathtt{!}\,\langle \mathit{Web}\rangle\ \ \mathtt{then}\ \mathit{serv}(\mathit{Web})\ \mathtt{else}\ \mathit{serv}(\mathit{Web})]\!] \mid \mathit{Web}[\![\mathit{job} \mid \mathit{serv}(\mathit{Done})]\!] \end{array}$$

One can show that derivations are well defined as one cannot execute an unbounded sequence of action moves without time progress, and the execution Ψ is made up of independent (or concurrent) individual executions. Moreover, derivations preserve well-formedness of networks (see [8]).

3 Rewriting Logic and Strategies

Rewriting logic (RL) [17] is an algebraic specification approach which is able to model dynamic system behaviour. In RL the static properties of a system are described by a standard algebraic specification, whereas the dynamic behaviour of the system is modelled using rewrite rules. Rewrite strategies are then used to control the application of rewrite rules and allow a RL specification to capture subtle aspects of a dynamic system. A brief introduction to RL and rewriting strategies is presented below (for a more detailed introduction see [17, 6]).

An S–sorted signature Σ defines a collection of function symbols, where: $c : s \in \Sigma$ means c is a constant symbol of sort $s \in S$; and

$$f : s(1) \dots s(n) \rightarrow s \in \Sigma$$

means f is a function symbol in Σ of *domain type* $s(1) \dots s(n)$, *arity* n, and *codomain type* s. Let $X = \langle X_s \mid s \in S \rangle$ be a family of sets of variables. We let $T(\Sigma, X) = \langle T(\Sigma, X)_s \mid s \in S \rangle$ be the family of sets of all terms over Σ and X. For any term $t \in T(\Sigma, X)_s$, we let $Var(t) \subseteq \cup_{s \in S} X_s$ represent the set of variables used in t. We let $T(\Sigma, X)/E$ represent the free quotient algebra of terms with respect to a set of equations E over Σ and X. For for any term $t \in T(\Sigma, X)_s$, we let $\langle t \rangle_E$ represent the equivalence class of term t with respect to the equations E (see [16]).

In RL a specification (Σ, E) defines the states $\langle t \rangle_E$ of a system. The dynamic behaviour of the system is then specified by *rewrite rules* [17, 6]:

$$l \Longrightarrow r,$$

for terms $l, r \in T(\Sigma, X)_s$ and $s \in S$, where $Var(r) \subseteq Var(l)$. Such rules represent dynamic transitions between states $\langle l \rangle_E$ and $\langle r \rangle_E$. We also allow rules to be labelled and to contain conditions:

$$[lb]\ l \Longrightarrow r \quad if\ c,$$

where lb is a (not necessarily unique) label, $c \in T(\Sigma, X)_{bool}$ and $Var(c) \subseteq Var(l)$. Intuitively, the condition means that the rewrite rule can only be applied if term c rewrites to *true*. A *Rewriting logic specification* is therefore a triple $Spec = (\Sigma, E, R)$ consisting of an algebraic specification (Σ, E) and a set of (conditional) rewrite rules R over Σ and X.

As an example of an RL specification consider a model of a simple dynamic system in which states are multi–sets of symbols A, B, and C. The resulting RL specification $Spec(MS) = (\Sigma, E, R)$ is defined as follows. Let $S = \{ent, ms\}$ be a sort set and let Σ be an S-sorted signature which contains the following function symbols:

$$\begin{array}{l} A, B, C : ent \in \Sigma, \\ empty : ms \in \Sigma, , \\ @ : ent \rightarrow ms \in \Sigma, \\ @ \otimes @ : ms\ ms \rightarrow ms \in \Sigma, \end{array}$$

(where @ is used to indicate the position of an argument in a function symbol to allow for an infix notation). Note that the signature contains an implicit type coercion operator $@ : ent \rightarrow ms$.) The set of equations E contains the equations which axiomatize the associative/commutative properties of a multi–set. Note that the rewrite rules defined below will be applied modulo these equations. Finally, we define R to contain the following three rewrite rules:

$$[Rule1]\ A \otimes m1 \Longrightarrow B \otimes m1 \qquad [Rule2]\ B \otimes C \otimes m1 \Longrightarrow B \otimes A \otimes m1$$
$$[Rule3]\ B \otimes B \otimes m1 \Longrightarrow C \otimes m1.$$

where $m1 \in X_{ms}$. Let $A \otimes C$ be a multi–set representing the initial state of the system. Then the trace

$$A \otimes C \implies B \otimes C \implies B \otimes A \implies B \otimes B \implies C$$

represents one possible evolution of the system.

Rewriting Logic provides the notion of a *strategy* for controlling the application of rewrite rules [5, 6]. A strategy allows the user to specify the order in which rewrite rules are applied and the possible choices that can be made. The result of applying a strategy is the set of all possible terms that can be produced according to the strategy. A strategy is said to *fail* if it can not be applied (i.e. produces no results). The following is a brief overview of some *elementary strategies* (based on [5, 6]):

(i) **Basic strategy**: *lb* Any label used in a labelled rule $[lb] \quad t => t'$ is a strategy. The result of applying a basic strategy l is the set of all terms that could result from one application of any rule labelled *lb*.

(ii) **Concatenation strategy**: $s_1; s_2$ Allows strategies to be sequentially composed, i.e. s_2 is applied to the set of results from s_1.

(iii) **Don't know strategy**: $dk(s_1, \ldots, s_n)$ Returns the union of all the sets of terms that result from applying each strategy $s_1, \ldots, s_n$.

(iv) **Don't care strategy**: $dc(s_1, \ldots, s_n)$ Chooses nondeterministically to apply one of the strategies s_i which does not fail. The strategy $dc\ one(s_1, \ldots, s_n)$ works in a similar way but chooses a *single* result term to return, where as $first(s_1, \ldots, s_n)$ applies the first successful strategy in the sequence $s_1, \ldots, s_n$.

(v) **Iterative strategies**: $repeat*(s)$ Repeatedly applies s, zero or more times, until the s fails. It returns the last set of results produced before s failed.

As an example, $repeat*(first(Rule1, Rule2, Rule3)$ is a strategy for $Spec(MS)$ which prioritises the rules so that *Rule1* is always applied first if it can be, *Rule2* is applied only if the first rule cannot be applied and *Rule3* is applied only if the previous two rules cannot be applied.

The above elementary strategy language can be extended to a *defined strategy language* [5, 6] which allows recursive strategies to be defined. As an example, consider the simple recursive search strategy $search(i)$ defined below:

$$\begin{aligned} &doStep \implies dk(Rule1, Rule2, Rule3) \\ &search(i) \implies fail \quad \text{if } i <= 0 \\ &search(i) \implies first(found, doStep; search(i-1)) \quad \text{if } i > 0 \end{aligned}$$

The strategy $search(i)$ repeatedly applies the strategy *doStep* looking for a multi-set term that satisfies the strategy *found*. It fails if the given maximum number of iterations i is reached. So to search for a multi-set term containing $A \otimes B \otimes C$ we would define the strategy *found* by the following rewrite rule:

$$[found] \quad A \otimes B \otimes C \otimes m1 \implies A \otimes B \otimes C \otimes m1$$

A range of tools have been developed for supporting rewriting logic and strategies, including: Maude [12]; Elan [4, 6]; Stratego [20]; and Tom [2]. In this paper we have chosen to use Elan to implement our examples given its simple strategy language and the authors' experience with this tool.

4 Modelling TiMo using Rewriting Logic and Strategies

In this section we develop a semantic model of TiMo using rewriting logic and strategies, and provide a formal argument of correctness.

4.1 Developing an RL Model for TiMo

Given a TiMo specification TM we consider how to develop a corresponding RL model $RL(TM)$ that correctly captures the meaning of TM. Note for simplicity, the parameters used in communication between processes within TM are restricted to a single location parameter.

We begin by modelling the general concept of a process and network in RL. Let S be the set of sorts in $RL(TM)$ containing: *nat* for time; *Chan* for channels; *VLoc*, *ALoc*, and *Loc* for locations; *Prs* for processes. and *Nets* for networks. Coping with the parameter passing that occurs in communication requires careful consideration and for this reason the sort *Loc* is defined as the union of two subsorts: *VLoc* represents the input location variables; and *ALoc* represents the actual locations used in TM.

The S-sorted signature $\Sigma^{RL(TM)}$ for $RL(TM)$ contains the following function symbols to capture the syntax for processes given in Table 1:

$$stop : Prs, \qquad S(@) : Prs \to Prs, \qquad @ \mid @ : Prs\ Prs \to Prs$$

$$go(@, @)\ then\ @ : nat\ Loc\ Prs \to Prs$$

$$in(@, @)(@)\ then\ @\ else\ @ : Chan\ nat\ VLoc\ Prs\ Prs \to Prs$$

$$out(@, @) < @ >\ \ then\ @\ else\ @ : Chan\ nat\ Loc\ Prs\ Prs \to Prs.$$

The function symbol $@ \mid @$ is defined equationally to be associative and commutative as per the definition of TiMo. To model process definitions we add a function symbol $id : s_1\ \ldots\ s_n \to Prs$ for each process identifier $id(u_1, \ldots, u_n : s_1, \ldots, s_n)$, where s_i is assumed to be a well–defined data type in our model.

We then define the following function symbols to represent networks:

$$@[@] : ALoc\ Prs \to Nets; \qquad @ \mid @ : Nets\ Nets \to Nets;$$

where $@ \mid @$ is again defined to be associative and commutative.

We now need to formulate appropriate rewrite rules to begin to capture the intended semantics of TiMo. In the RL model developed here we choose the approach of forcing network components with the same location to merge (this turns out to be important since it simplifies the selection of a location to update). The above approach is realized using the rule $al[p1] \mid al[p2] \Rightarrow al[p1 \mid p2]$. Clearly, such a rule is compatible with Eq 3 from Table 2. Each network term will therefore have the form $at_1[pt_1] \mid \ldots \mid at_n[pt_n]$, where each location at_i is unique and where each pt_i will represent a set of parallel processes. Any process term which does not contain the parallel operator at its topmost level is referred to as an *atomic process term*. Each individual network location term will have the form $at_i[pt_i^1 \mid \ldots \mid pt_i^k]$, where each pt_i^j is an atomic process term.

Next we consider how to model the action rules given in Table 2 within our RL model. First, we define two labelled rules to model the action rule (Move):

$$\begin{array}{ll} [move] & al[go(t, al2)\ then\ p1 \mid p2] \implies al2[S(p1)] \mid al[p2] \\ [move] & al[go(t, al2)\ then\ p1 \mid p2] \implies \\ & \qquad al[S(go(t-1, al2)\ then\ p1) \mid p2] \quad if\ \ t > 0 \end{array}$$

The two rules can both be applied when $t > 0$ and this leads to a non–deterministic choice between moving location or allowing time to pass. Note that if $t = 0$ then only the rule that moves to a different location is applicable.

To model the synchronisation required for communication as defined by the action rule (COM) we have the following rule:

$$[com] \quad al[out(c,t1) < al1 > \ then\ p1\ else\ p2 \mid in(c,t2)(vl)\ then\ p3\ else\ p4 \mid p5] \\ \Longrightarrow\ al[S(p1) \mid S(p3[vl/al1]) \mid p5]$$

This rule makes use of a substitution function $@[@/@] : Prs\ VLoc\ ALoc \to Prs$, where $pt[vt/at]$ represents the process term that results by substituting all free occurrences (not bound by an input action symbol) of $VLoc$ term vt by the $ALoc$ term at within the process term pt. This function is straightforward to define algebraically using recursion on process terms.

In any TIMO specification TM there will be process definitions of the form $id(u_1, \ldots, u_n : s_1, \ldots, s_n) \stackrel{\text{df}}{=} P_{id}$ which allow each process identifier $id \in Id$ to be associated with a well–formed process expression P_{id} (see the action rule (CALL) in Table 2). In $RL(TM)$, for each $id \in Id$ we add a rewrite rule of the form:

$$[calls] \quad al[id(u_1, \ldots, u_n) \mid p] \Rightarrow al[RL(P_{id}) \mid p]$$

where $RL(P_{id})$ is the process term that results from translating P_{id} into $RL(TM)$ and each u_i is a variable of sort s_i in $RL(TM)$.

The above labelled rules are collectively referred to as *process transition rules* and are used to define a strategy *step* that represents an update step as follows:

$$step \Longrightarrow repeat*(dc(calls, move, com))$$

The strategy repeatedly applies the three process transition rules and makes use of the *dc* built–in strategy as the order the rules are applied in is irrelevant given that they act on disjoint sets of terms.

In TIMO the last step of any derivation involves applying the (TIME) action rule which allows time to progress and removes all stall symbols. We model this by using a function $tick(@) : Prs \to Prs$ which is applied to the terms resulting from *step*. We define *tick* recursively as illustrated by the sample rules below:

$$tick(stop) \Longrightarrow stop$$

$$tick(S(p1)) \Longrightarrow p1$$

$$tick(p1 \mid p2) \Longrightarrow tick(p1) \mid tick(p2)$$

$$tick(id(u_1, \ldots, u_n)) \Longrightarrow id(u_1, \ldots, u_n)$$

$$tick((out(a,t) < l > \ then\ p1\ else\ p2)) \Longrightarrow \\ (out(a, t-1) < l > then\ p1\ else\ p2) \quad if\ t > 0$$

$$tick((out(a,0) < l > then\ p1\ else\ p2)) \Longrightarrow p2$$

To make the application of this function straightforward we overload *tick* so that it can be applied to networks by defining $tick(@) : Nets \to Nets$ by

$$tick(al[p]) \Longrightarrow al[tick(p)], \qquad tick(n1 \mid n2) \Longrightarrow tick(n1) \mid tick(n2)$$

We can now formulate a rewrite rule *oneStep* in $RL(TM)$ using the strategy *step* and function *tick* that models a derivation step in TM:

$$[oneStep] \quad al[p] \mid n1 \Longrightarrow n3 \mid n1 \quad \textit{where } n2 := (step)\ al[p], \quad n3 := ()\ tick(n2)$$

The pattern $al[p] \mid n1$ is used to match non–deterministically with a collection of network components (due to the associative/commutative property of @ | @) and so chooses the next location to update.

It is interesting to note that different semantic choices can be considered for TiMo by appropriately updating the *oneStep* strategy. For example, we could straightforwardly consider a synchronous semantics, introduce priorities to locations or add fairness assumptions. This provides further motivation for developing our RL model.

4.2 Correctness of RL Model

Having developed an RL model for TiMo we now validate that it correctly captures the semantics of TiMo. We do this by showing that our model is *sound* (each step in our RL model represents a derivation step in TiMo) and *complete* (every derivation step possible in TiMo is represented in our RL model). In the sequel let TM be a TiMo specification and let $RL(TM)$ be the corresponding RL model as defined in Section 4.1.

Not all the terms of sort Prs in $RL(TM)$ represent valid processes in TM since they may contain the stall symbol S. Another problem can arise with the improper use of location variables, that is terms of sort $VLoc$, since all uses other than those in an input command need to be bound by an outer input command. We formalise what we mean by a valid process term by defining a function VP.

Definition 1 The function $VP : T(\Sigma^{RL(TM)})_{Prs} \times \mathcal{P}(T(\Sigma^{RL(TM)})_{VLoc}) \rightarrow \mathbf{B}$ is defined recursively other the structure of process terms as follows:

$$VP(stop, VS) = true$$

$$VP(S(pt), VS) = false$$

$$VP(id(v_1, \ldots, v_n), VS) = \begin{cases} true & \text{if } v_i \in VS, \text{ for all } v_i \text{ of sort } VLoc \\ false & \text{otherwise} \end{cases}$$

$$VP(go(nt, at)\ then\ pt, VS) = VP(pt, VS)$$

$$VP(go(nt, vt)\ then\ pt, VS) = vt \in VS \wedge VP(pt, VS)$$

$$VP(in(ct, nt)(vt)\ then\ pt_1\ else\ pt_2, VS) = VP(pt_1, VS \cup \{vt\}) \wedge VP(pt_2, VS)$$

$$VP(out(ct, nt) < vt >\ then\ pt_1\ else\ pt_2, VS) = VP(pt_1, VS) \wedge vt \in VS \wedge VP(pt_2, VS)$$

$$VP(out(ct, nt) < at >\ then\ pt_1\ else\ pt_2, VS) = VP(pt_1, VS) \wedge VP(pt_2, VS)$$

$$VP(pt_1 \mid pt_2, VS) = VP(pt_1, VS) \wedge VP(pt_2, VS).$$

We define $valPrs(TM) = \{pt \mid pt \in T(\Sigma^{RL(TM)})_{Prs}\ and\ VP(pt, \{\})\ \}$ to be the set of all *valid process terms* and define the set $valNet(TM)$ of *valid network terms* recursively by: (1) $at[pt] \in valNet(TM)$ if $pt \in valPrs(TM)$; and (2) $net_1 \mid net_2 \in valNet(TM)$, if $net_1, net_2 \in valNet(TM)$.

It can be shown that *oneStep* preserves valid network terms.

Theorem 2 The strategy *oneStep* is well–defined with respect to valid network terms, i.e. for any $net_1 \in valNet(TM)$, if $net_1 \Longrightarrow net_2$ using *oneStep* then $net_2 \in valNet(TM)$.

Proof. By the definition of *oneStep* it suffices to consider a valid network location term of the form

$$at[pt_1 \mid \ldots \mid pt_n] \in valNet(TM),$$

where $n > 0$ and each pt_i is an atomic process term. It can be seen that each process term pt_i is involved in at most one process transition rule application when *oneStep* is applied. This gives use four possible cases to consider.

Case 1) Suppose pt_i is not involved in the application of any process transition rules. Then by the definition of *oneStep* and $valNet(TM)$ (Definition 1) we need to show that $tick(pt_i)$ results in a valid process term. There are three possible subcases to consider:
i) Suppose pt_i is the process term *stop*. Then by definition of *tick* we know $tick(stop)$ rewrites to *stop* which is clearly a valid process term.
ii) Suppose pt_i is the process term $tick(out(ct, nt) < vt > \ then\ pt_i^1\ else\ pt_i^2)$. By the definition of *tick* there are two possibilities to consider. First, if $nt > 0$ holds then time is allowed to progress and the resulting process term $out(ct, nt-1) < vt > \ then\ pt_i^1\ else\ pt_i^2$ must be valid given that the original output term was. Secondly, we could have $nt = 0$ in which case the timer has expired and the resulting process term is pt_i^2. Clearly pt_i^2 must be valid given that the original output term was.
iii) Suppose pt_i is the process term $tick(in(ct, nt)(vt)\ then\ pt_i^1\ else\ pt_i^2)$. Then a similar argument to ii) above can be used, noting that the input operator does not bind vt in pt_i^2.

Case 2) Suppose pt_i has the form $id(v_1, \ldots, v_n)$ and that a [*calls*] rule is applied to it, i.e.

$$id(v_1, \ldots, v_n) \Rightarrow S(pt)$$

where pt is the process term $RL(P_{id})$ with variables $u_1, \ldots, u_n$ replaced by terms $v_1, \ldots, v_n$. Clearly the stall symbol S will be removed by the application of the *tick* function at the end of *oneStep*. Since P_{id} was a well–formed TIMO process expression it is straightforward to show that $pt \in valPrs(TM)$ as required.

Case 3) Suppose pt_i has the form $go(nt, at_2)\ then\ pt$ and that a [*move*] rule is applied to it. Then we have two possible cases:
i) Suppose that $nt > 0$ holds and that applying the [*move*] rule simply allows time to pass, i.e.

$$go(nt, at_2)\ then\ pt \Rightarrow S(go(nt-1, at_2)\ then\ pt)$$

Since the stall symbol S will be removed by the application of the *tick* function, it can be seen that $go(nt-1, at_2)\ then\ pt$ must be a valid process term given that the original process term was.
ii) Suppose that applying a [*move*] rule results in the process moving locations, i.e. produces the network term $at_2[pt]$. Since the original atomic process term was valid it follows that pt must be valid and so the resulting new network term must also be valid as required.

Case 4) Suppose that the [*com*] rule has been applied to two process terms pt_i and pt_j, for $i \neq j$. That is, suppose

$$\begin{aligned} &out(ct, nt_1) < at_2 > \ then\ pt_i^1\ else\ pt_i^2 \mid in(ct, nt_2)(vt)\ then\ pt_j^1\ else\ pt_j^2 \\ &\quad \Rightarrow S(pt_i^1) \mid S(pt_j^1[vt/at_2]) \end{aligned}$$

Then the stall symbol S will be removed by the application of the *tick* function at the end of *oneStep*. It follows that pt_i^1 must be a valid process term since the original output term was valid. Also $pt_j^1[vt/at_2]$ must be valid since $VP(pt_j^1, \{vt\})$ must be true as the original input term was valid and as $pt_j^1[vt/at_2]$ is simply pt_j^1 with all unbounded occurrences of location variable term vt replaced by the actual location term at_2. □

We can define an interpretation mapping between TIMO terms in TM and terms in the corresponding RL model $RL(TM)$ as follows.

Definition 3 The process term mapping $\sigma_{Prs} : Prs(TM) \rightarrow valPrs(TM)$ is defined recursively by:

$$\sigma_{Prs}(\texttt{ stop }) = stop, \qquad \sigma_{Prs}(id(v_1, \ldots, v_n)) = id(v_1, \ldots, v_n),$$

$$\sigma_{Prs}(\texttt{go}^{\Delta t}\ l\ \texttt{then}\ P) = go(t, l)\ then\ \sigma_{Prs}(P),$$

$$\sigma_{Prs}(a^{\Delta t}\,\texttt{?}\,(vl : Loc)\ \texttt{then}\ P\ \texttt{else}\ P') = in(a, t)(vl)\ then\ \sigma_{Prs}(P)\ else\ \sigma_{Prs}(P'),$$

$$\sigma_{Prs}(a^{\Delta t}\,\texttt{!}\,\langle l\rangle\ \texttt{then}\ P\ \texttt{else}\ P') = out(a, t) < l >\ then\ \sigma_{Prs}(P)\ else\ \sigma_{Prs}(P'),$$

$$\sigma_{Prs}(P \mid P') = \sigma_{Prs}(P) \mid \sigma_{Prs}(P').$$

The network term mapping $\sigma_{Net} : Net(TM) \rightarrow valNet(TM)$ is defined using σ_{Prs} by $\sigma_{Net}(l\,[\![\,P\,]\!]) = l[\sigma_{Prs}(P)]$ and $\sigma_{Net}(N \mid N') = \sigma_{Net}(N) \mid \sigma_{Net}(N')$.

It is straightforward to show that σ_{Prs} and σ_{Net} are bijective mappings and thus have inverses. In order to show the correctness of the RL model we need to prove it is *sound* and *complete* with respect to TIMO (see Figure 2).

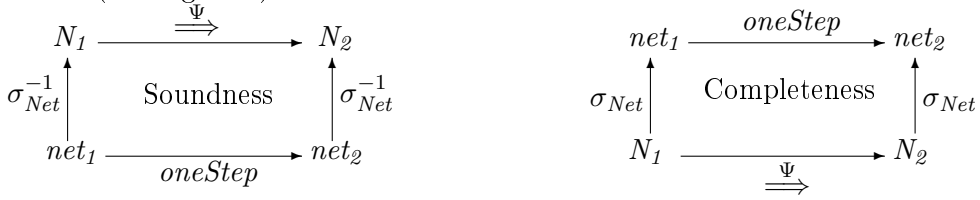

Figure 2: The properties of soundness and completeness required for $RL(TM)$ to be a correct model of TM.

We now show that for any TIMO specification TM the RL model $RL(TM)$ defined in Section 4.1 is a sound and complete model of TM.

Theorem 4 (Soundness) Let $net_1, net_2 \in valNet(TM)$ be any valid network terms. Then if $net_1 \Longrightarrow net_2$ using the strategy *oneStep* then $\sigma_{Net}^{-1}(net_1) \overset{\Psi}{\Longrightarrow} \sigma_{Net}^{-1}(net_2)$ for some finite multiset $\Psi = \{\psi_1, \ldots, \psi_m\}$ of l-actions and some location l. In other words, the diagram for soundness in Figure 2 must commute.

Proof. By the definition of *oneStep* and the notion of a derivation in TIMO it suffices to consider a valid network location term of the form

$$at[pt_1 \mid \ldots \mid pt_n] \in valNet(TM),$$

where $n > 0$ and each pt_i is an atomic process term. It can be seen that each process term pt_i is involved in at most one process transition rule application when *oneStep* is applied. This gives use four possible cases to consider.

Case 1) Suppose pt_i is not involved in the application of any process transition rules during a derivation using *oneStep*. Then we need to show that applying a time step in TIMO to the process expression $\sigma_{Prs}^{-1}(pt_i)$ results in the process $\sigma_{Prs}^{-1}(tick(pt_i))$. There are three possible subcases to consider:
i) Suppose pt_i is the process term *stop*. Then by definition of *tick* and σ_{Prs}^{-1} we have $\sigma_{Prs}^{-1}(stop) =$ stop and $\sigma_{Prs}^{-1}(tick(stop)) =$ stop. Then by time progression in TIMO we have $at[\![\ \mathsf{stop}\ ]\!] \xrightarrow{\surd_{at}} at[\![\ \mathsf{stop}\ ]\!]$ as required.
ii) Suppose pt_i is the process term $out(ct, nt) < at_2 >$ *then* pt_i^1 *else* pt_i^2. By the definition of *tick* there are two possibilities to consider. First, if $nt > 0$ holds then time is allowed to progress and the resulting process term is $out(ct, nt-1) < at_2 >$ *then* pt_i^1 *else* pt_i^2. By definition of time progression in TIMO, if $nt > 0$ we know

$$at[\![\ ct^{\Delta nt}\ \mathbf{!}\ \langle at_2\rangle\ \mathtt{then}\ pt_i^1\ \mathtt{else}\ pt_i^2\ ]\!] \xrightarrow{\surd_{at}} at[\![\ ct^{\Delta nt-1}\ \mathbf{!}\ \langle at_2\rangle\ \mathtt{then}\ pt_i^1\ \mathtt{else}\ pt_i^2\ ]\!]$$

as required. Secondly, we could have $nt = 0$ in which case the timer has expired and the resulting process term is pt_i^2. Again, by definition of time progression we know

$$at[\![\ ct^{\Delta 0}\ \mathbf{!}\ \langle at_2\rangle\ \mathtt{then}\ pt_i^1\ \mathtt{else}\ pt_i^2\ ]\!] \xrightarrow{\surd_{at}} at[\![\ pt_i^2\ ]\!]$$

as required.
iii) Suppose pt_i is the process term $in(ct, nt)(vt)$ *then* pt_i^1 *else* pt_i^2. Then the result follows by a similar argument to ii) above.

Case 2) Suppose pt_i has the form $id(v_1, \ldots, v_n)$ and that a [*calls*] rule is applied to it, i.e.

$$id(v_1, \ldots, v_n) \Rightarrow S(pt)$$

where pt is the process term $RL(P_{id})$ with variables $u_1, \ldots, u_n$ replaced by terms $v_1, \ldots, v_n$. Note that *tick* will remove the stall symbol and so the final atomic process term will be pt. Then by the action rule (CALL) in Table 2 we have

$$at[\![\ id(v_1, \ldots, v_n)\ ]\!] \xrightarrow{id@at} at[\![\ Ⓢ\sigma_{Prs}^{-1}(pt)\ ]\!]$$

The result follows since the stall symbol Ⓢ will be removed by the time progression step in TIMO.

Case 3) Suppose pt_i has the form $go(nt, at_2)$ *then* pt and that a [*move*] rule is applied to it. Then we have two possible cases:
i) Suppose that $nt > 0$ holds and that applying the [*move*] rule simply allows time to pass, i.e.

$$go(nt, at_2)\ then\ pt \Rightarrow S(go(nt-1, at_2)\ then\ pt)$$

Clearly, the stall symbol S will be removed by the application of the *tick* function. In TIMO by the definition of time progression and the assumption $nt > 0$ we have

$$at[\![\ \mathtt{go}^{\Delta nt}\ at_2\ \mathtt{then}\ \sigma_{Prs}^{-1}(pt)\ ]\!] \xrightarrow{\surd_{at}} at[\![\ \mathtt{go}^{\Delta nt-1}\ at_2\ \mathtt{then}\ \sigma_{Prs}^{-1}(pt)\ ]\!]$$

as required.
ii) Suppose that applying a [*move*] rule results in the process moving locations, i.e. produces the network term $at_2[S(pt)]$. Clearly, the stall symbol S will be removed by the application of the *tick* function. Then by the action rule (MOVE) in Table 2 we have

$$at\,[\![\,\mathtt{go}^{\Delta nt}\ at_2\ \mathtt{then}\ \sigma^{-1}_{Prs}(pt)\,]\!] \xrightarrow{at_2@at} at_2\,[\![\,Ⓢ\sigma^{-1}_{Prs}(pt)\,]\!]$$

The results follows since the stall symbol Ⓢ will be removed by the time progression step in TIMO.

Case 4) Suppose that the [*com*] rule have been applied to two process terms pt_i and pt_j, for $i \neq j$. That is, suppose

$$\begin{array}{l} out(ct, nt_1) < at_2 >\ \ then\ pt_i^1\ \ else\ pt_i^2 \mid in(ct, nt_2)(vt)\ \ then\ pt_j^1\ \ else\ pt_j^2 \\ \quad \Rightarrow\ \ S(pt_i^1) \mid S(pt_j^1[vt/at_2]) \end{array}$$

The instances of the stall symbol S will be removed by the application of the *tick* function at the end of *oneStep*. Then by the action rule (COM) in Table 2 we have

$$at\,[\![\,ct^{\Delta nt_1}\,!\,\langle at_2\rangle\ \mathtt{then}\ \sigma^{-1}_{Prs}(pt_i^1)\ \mathtt{else}\ \sigma^{-1}_{Prs}(pt_i^2)\ \mid\ ct^{\Delta nt_2}\,?\,(vt : Loc)$$

$$\mathtt{then}\ \sigma^{-1}_{Prs}(pt_j^1)\ \mathtt{else}\ pt_j^2)\,]\!] \xrightarrow{ct<at_2>@at} at\,[\![\,Ⓢ\sigma^{-1}_{Prs}(pt_i^1)\ \mid\ Ⓢ\{at_2/vt\}\sigma^{-1}_{Prs}(pt_j^1)\,]\!]$$

The result follows since the stall symbol Ⓢ will be removed by the time progression step in TIMO and since we can show $\sigma^{-1}_{Prs}(pt_j^1[vt/at_2]) = \{at_2/vt\}\sigma^{-1}_{Prs}(pt_j^1)$. □

Theorem 5 (Completeness) Let $N_1, N_2 \in Net(TM)$ be any well–formed network terms in TM. Then, if $N_1 \stackrel{\Psi}{\Longrightarrow} N_2$, for some location l and some multi-set $\Psi = \{\psi_1, \ldots, \psi_m\}$ of l-actions, then $\sigma_{Net}(N_1) \Longrightarrow \sigma_{Net}(N_2)$ using *oneStep*. In other words, the diagram for completeness in Figure 2 commutes.

Proof. By the definition of a derivation in TIMO and the strategy *oneStep* it suffices to consider a well–formed network of the form

$$at\,[\![\,P_1\ \mid\ \ldots\ \mid\ P_n\,]\!] \equiv at\,[\![\,P_1\,]\!]\ \mid\ \ldots\ \mid\ at\,[\![\,P_n\,]\!],$$

where $n > 0$ and each P_i is an atomic process. Suppose $at\,[\![\,P_1\ \mid\ \ldots\ \mid\ P_n\,]\!] \stackrel{\Psi}{\Longrightarrow} N'$, for some finite set of at–actions $\Psi = \{\psi_1, \ldots, \psi_m\}$, $m \geq 0$. Then it can be seen that each atomic process P_i is involved in at most one at–action ψ_i. We show that the derivation applied to each process P_i is correctly captured by the *oneStep* strategy in the RL model. We have four possible cases to consider.

Case 1): Suppose P_i is not involved in any of the action rules (CALL), (MOVE) and (COM) during the derivation step. That is $at\,[\![\,P_i\,]\!] \xrightarrow{\surd_l} at\,[\![\,P_i'\,]\!]$, where $at\,[\![\,P_i'\,]\!] = \phi_l(at\,[\![\,P_i\,]\!])$. Then by definition of *oneStep* we need to show that $tick(\sigma_{Prs}(P_i))$ results in the process $\sigma_{Prs}(P_i')$. Considering the possible form of P_i gives us three subcases to consider:
i) Suppose P_i is the process stop . Then $\phi_l(at\,[\![\ \mathsf{stop}\ ]\!]) = at\,[\![\ \mathsf{stop}\ ]\!]$ by definition of ϕ_l and we have $\sigma_{Prs}(\ \mathsf{stop}\ ) = stop$ and $\sigma_{Prs}(\phi_l(\ \mathsf{stop}\ )) = stop$. It follows by the definition of *tick* that $tick(stop) = stop$ as required.
ii) Suppose P_i is the process $ct^{\Delta nt}\,!\,\langle at_2\rangle\ \mathtt{then}\ P_i^1\ \mathtt{else}\ P_i^2$. Then $\sigma_{Prs}(P_i) = out(ct, nt) <$

$at_2 >$ *then* $\sigma_{Prs}(P_i^1)$ *else* $\sigma_{Prs}(P_i^2)$. By the definition of time progression in TiMo there are two possibilities to consider. First, if $nt > 0$ then time is allowed to progress resulting in the process

$$ct^{\Delta nt-1}\,!\,\langle at_2\rangle\ \texttt{then}\ P_i^1\ \texttt{else}\ P_i^2$$

Applying σ_{Prs} to this process gives $out(ct, nt-1) < at_2 >$ *then* pt_i^1 *else* pt_i^2) and so by the definition of *tick* the result follows. Secondly, we could have $nt = 0$ in which case the timer has expired and the resulting process is P_i^2. Again, by definition of *tick* it can be seen that the result follows.
iii) Suppose P_i is the process $ct^{\Delta nt}\,?\,(vt : Loc)\ \texttt{then}\ P_j^1\ \texttt{else}\ P_j^2$. Then the result follows by a similar argument to ii) above.

Case 2) Suppose P_i is the process $id(v_1, \ldots, v_n)$ and that the action rule (Call) is applied, i.e.

$$at\,[\![\, id(v_1, \ldots, v_n)\,]\!] \xrightarrow{id@at} at\,[\![\, \textcircled{S}\{\vec{v}/\vec{u}\}P_{id}\,]\!]$$

where the stall symbol Ⓢ is removed by the final time progression step. We have $\sigma_{Prs}(id(v_1, \ldots, v_n)) = id(v_1, \ldots, v_n)$ and so applying the [*calls*] rule to this term gives

$$id(v_1, \ldots, v_n) \Rightarrow S(pt)$$

where pt is the process term $\sigma_{Prs}(P_{id})$ with variables $u_1, \ldots, u_n$ replaced by terms $v_1, \ldots, v_n$. Note that *tick* will remove the stall symbol and so the final atomic process term will be pt. It is straightforward to see that $\sigma_{Prs}(\{\vec{v}/\vec{u}\}P_{id}) = pt$ as required.

Case 3) Suppose P_i has the form $\texttt{go}^{\Delta nt}\ at_2\ \texttt{then}\ P'$. Then we have

$$\sigma_{Prs}(\texttt{go}^{\Delta nt}\ at_2\ \texttt{then}\ P') = go(nt, at_2)\ then\ \sigma_{Prs}(P')$$

By the definition of a derivation in TiMo we have two possible cases:
i) Suppose that $nt > 0$ holds and that the action rule (Move) has *not* been applied and instead time has been allowed to progress, i.e.

$$at\,[\![\, \texttt{go}^{\Delta nt}\ at_2\ \texttt{then}\ P'\,]\!] \xrightarrow{\surd_{at}} at\,[\![\, \texttt{go}^{\Delta nt-1}\ at_2\ \texttt{then}\ P'\,]\!]$$

This can be modelled by applying the appropriate [*move*] rule

$$go(nt, at_2)\ then\ \sigma_{Prs}(P') \Rightarrow S(go(nt-1, at_2)\ then\ \sigma_{Prs}(P'))$$

where the stall symbol S will be removed by the application of the *tick* function. It is straightforward to see that

$$\sigma_{Prs}(\texttt{go}^{\Delta nt-1}\ at_2\ \texttt{then}\ P') = go(nt-1, at_2)\ then\ \sigma_{Prs}(P')$$

as required.
ii) Suppose the action rule (Move) was applied to P_i

$$at\,[\![\, \texttt{go}^{\Delta nt}\ at_2\ \texttt{then}\ P'\,]\!] \xrightarrow{at_2@at} at_2\,[\![\, \textcircled{S}P'\,]\!]$$

where the stall symbol Ⓢ is removed by the final time step. This can be copied in the RL model by applying the appropriate [*move*] rule

$$at[go(nt, at_2)\ then\ \sigma_{Prs}(P')] \Rightarrow at_2[S(\sigma_{Prs}(P'))]$$

Clearly, the stall symbol S will be removed by the application of the *tick* function. It is then straightforward to see that

$$\sigma_{Net}(at_2 \,[\![\, P' \,]\!]) = at_2[\sigma_{Prs}(P')]$$

by the definition of σ_{Prs}.

Case 4) Suppose the action rule (COM) has been applied to two processes P_i and P_j, for $i \neq j$, i.e.

$$at \,[\![\, ct^{\Delta nt_1} \,!\, \langle at_2 \rangle \texttt{ then } P_i^1 \texttt{ else } P_i^2 \;\mid\; ct^{\Delta nt_2} \,?\, (vt : Loc) \texttt{ then } P_j^1 \texttt{ else } P_j^2 \,]\!]$$

$$\xrightarrow{ct<at_2>@at} at \,[\![\, \textcircled{S} P_i^1 \;\mid\; \textcircled{S}\{at_2/vt\}P_j^1 \,]\!]$$

where the stall symbols Ⓢ have been removed by the final time step. Then we have

$$\begin{array}{l}\sigma_{Prs}(ct^{\Delta nt_1} \,!\, \langle at_2 \rangle \texttt{ then } P_i^1 \texttt{ else } P_i^2 \;\mid\; ct^{\Delta nt_2} \,?\, (vt : Loc) \texttt{ then } P_j^1 \texttt{ else } P_j^2)) \\ = out(ct, nt_1) < at_2 > \; then\ \sigma_{Prs}(P_i^1)\ else\ \sigma_{Prs}(P_i^2) \mid \\ \quad in(ct, nt_2)(vt)\ then\ \sigma_{Prs}(P_j^1)\ else\ \sigma_{Prs}(P_j^2)\end{array}$$

By applying the [*calls*] rule we have

$$\begin{array}{c} out(ct, nt_1) < at_2 > \; then\ \sigma_{Prs}(P_i^1)\ else\ \sigma_{Prs}(P_i^2) \mid in(ct, nt_2)(vt)\ then \\ \sigma_{Prs}(P_j^1)\ else\ \sigma_{Prs}(P_j^2) \;\Rightarrow\; \sigma_{Prs}(P_i^1) \mid \sigma_{Prs}(P_j^1)[vt/at_2]\end{array}$$

where all occurrences of the stall symbol S will be removed by the *tick* function. It is then straightforward to see that

$$\sigma_{Prs}(P_i^1 \;\mid\; \{at_2/vt\}P_j^1) = \sigma_{Prs}(P_i^1) \mid \sigma_{Prs}(P_j^1)[vt/at_2]$$

by definition of σ_{Prs} and since we can show $\sigma_{Prs}(\{at_2/vt\}P_j^1) = \sigma_{Prs}(P_j^1)[vt/at_2]$. □

5 An Illustrative Example

In this section we investigate using ELAN [4, 6], a strategy-based rewrite system, to implement a TIMO specification based on our RL model. We consider a small example which provides useful insight into the RL modelling approach used and illustrates the type of (behavioural) properties that can be analysed.

Recall the simple TIMO workflow example introduced in Section 2. The specification *WF* can be mapped into an RL model $RL(WF)$ as described in Section 4.1 and then investigated using ELAN to provide insight into the behaviour of the original TIMO specification. A range of (behavioural) properties can be analysed including time constraints, use of locations, and causality between actions. For example, consider the following initial TIMO network:

$$Init \,[\![\, job \,|\, serv(Web) \,]\!] \;\mid\; Web \,[\![\, serv(Done) \,]\!]$$

After translating this into $RL(WF)$ we can use ELAN to derive the following rewriting trace which shows how a processing job can reach the *Done* location:

$Init[job \mid serv(Web)] \mid Web[serv(Done)]$
$\Longrightarrow$
$Init[in(a,1)(WL)\ then\ go(1, WL)\ then\ job\ else\ job \mid out(a,2) < Web > \; then$
$serv(Web)\ else\ serv(Web)] \mid Web[serv(Done)]$

$\Longrightarrow$
$Init[go(1, Web)\ then\ job \mid serv(Web)] \mid Web[serv(Done)]$
$\Longrightarrow$
$Init[out(a, 2) < Web >\ then\ serv(Web)\ else\ serv(Web)] \mid Web[job \mid serv(Done)]$
$\Longrightarrow$
$Init[out(a, 2) < Web >\ then\ serv(Web)\ else\ serv(Web)] \mid Web[in(a, 1)(WL)\ then$
$go(1, WL)\ then\ job\ else\ job \mid out(a, 2) < Done >\ then\ serv(Done)$
$else\ serv(Done)]$
$\Longrightarrow$
$Init[out(a, 2) < Web >\ then\ serv(Web)\ else\ serv(Web)] \mid$
$Web[go(1, Done)\ then\ job \mid serv(Done)]$
$\Longrightarrow$
$Init[out(a, 2) < Web >\ then\ serv(Web)\ else\ serv(Web)] \mid$
$Web[out(a, 2) < Done >\ then\ serv(Done)\ else\ serv(Done)] \mid Done[job]$

The example trace contains six derivation steps and indeed it is easy to verify using ELAN that this is the smallest number of steps needed in order for a processing job starting at *Init* to reach the *Done* location. Next we consider what happens if we change our network so that it contains a faulty service process:

$$Init\ [\![\ job \mid serv(Web)\ ]\!]\ \mid\ Web\ [\![\ servErr(Done)\ ]\!]$$

Again, using ELAN and a simple search strategy we are able to confirm that that it is still possible for a processing job to reach the *Done* location. Furthermore, we can show that it is now possible for a processing job to end up in the *Err* location as the following term derived using ELAN shows:

$$Init[out(a, 2) < Web >\ then\ serv(Web)\ else\ serv(Web)] \mid$$
$$Web[out(a, 2) < Err >\ then\ servErr(Err)\ else\ servErr(Err)] \mid Err[job]$$

6 Conclusions

In this paper we have considered using RL to develop a model and implementation of TIMO. The RL model was based on developing a strategy which can capture a maximal parallel computational step of a TIMO specification, including its time rule based previously on negative premises. We have also formally shown the correctness of the resulting semantics by proving it is both sound and complete. We illustrated how the ELAN tool and, in particular, its user defined strategies can be used to model and analyse a TIMO specification. While the example used is intentionally simple for brevity, it still provides an interesting first insight into the range of properties that can be investigated.

TIMO [8] is an appealing process algebra proposed for prototyping software engineering applications where time and mobility are combined. Related models can be found in the literature, such as the timed π-calculus [3], timed distributed π-calculus [11], and timed mobile ambients [1]. RL provides an ideal logical framework for modelling concurrent systems and has been used to model a range of process algebras, such as CCS [15]. In particular, [19] provides a high-level discussion of the use of ELAN for prototyping Π-calculus specifications but while the use of strategies is mentioned no specific details are provided. The RL model of TIMO presented here appears to be novel in its use of strategies to cope with maximal parallel computational steps.

In future work we intend to investigate extending our approach to handle security related aspects of software engineering designs, such as the access permissions defined for TIMO specifications in [10]. Interestingly, the RL model allows a range of semantic choices for TIMO to be

considered by changing the derivation step strategy (e.g. adding priorities or fairness assumptions) and we are currently investigating these different semantic choices. We also intend to perform a variety of verification case studies to illustrate the practical application of our methods and investigate its limitations. Finally, we note that at present the analysis of TiMo specifications is limited by the search capabilities and efficiency of Elan. Work is now underway to develop Maude [12] and Tom [2] implementations of the RL model presented here with the aim of improving both the range and efficiency of model analysis.

Acknowledgements We gratefully acknowledge the financial support of the School of Computing Science, Newcastle University.

References

[1] B.Aman and G.Ciobanu: Mobile Ambients with Timers and Types. Proc. of *ICTAC 2007*, Springer, LNCS 4711 (2007) 50–63.

[2] E.Balland, P.Brauner, R.Kopetz, P.-E.Moreau, and A.Reilles: Tom: Piggybacking rewriting on java. Proc. of *RTA'07*, Springer, LNCS 4533 (2007) 36–47.

[3] M.Berger: Basic Theory of Reduction Congruence for Two Timed Asynchronous Pi-Calculi. Proc. of *CONCUR'04*, Springer, LNCS 3170 (2004) 115–130.

[4] P.Borovanský, C.Kirchner, H.Kirchner, P.–E. Moreau and C.Ringeissen: An overview of ELAN. In: C. Kirchner and H. Kirchner (eds), Proc. of *WRLA '98*, *Electronic Notes in Theoretical Computer Science* 15 (1998).

[5] P.Borovanský, C.Kirchner, H.Kirchner, and C.Ringeissen: Rewriting with Strategies in ELAN: A Functional Semantics. *International Journal of Foundations of Computer Science* 12(1) (2001) 69–95.

[6] P.Borovanský, C.Kirchner, H.Kirchner, and P.–E.Moreau: Elan from a rewriting logic point of view. *Theoretical Computer Science* 285(2) (2002) 155–185.

[7] L.Cardelli and A.Gordon: Mobile Ambients. *Theoretical Computer Science* 240 (2000) 170–213.

[8] G.Ciobanu and M.Koutny: Modelling and Verification of Timed Interaction and Migration. Proc. of *FASE'08*, Springer, LNCS 4961 (2008) 215–229.

[9] G.Ciobanu and M.Koutny: Timed Mobility in Process Algebra and Petri Nets. *Journal of Algebraic and Logic Programming* 80(7) (2011) 377–391.

[10] G.Ciobanu and M.Koutny: Timed Migration and Interaction with Access Permissions. Proc. of *FM'11*, Springer, LNCS 6664 (2011) 293–307.

[11] G.Ciobanu and C.Prisacariu: Timers for Distributed Systems. *Electronic Notes in Theoretical Computer Science* 164 (2006) 81–99.

[12] M.Clavel, *et al*: Maude: Specification and Programming in Rewriting Logic. *Theoretical Computer Science* 285(2) (2002) 187–243.

[13] F.Corradini: Absolute Versus Relative Time in Process Algebras. *Information and Computation* 156 (2000) 122–172.

[14] M.Hennessy: *A Distributed π-Calculus*. Cambridge University Press (2007).

[15] N.Martí-Oliet and J.Meseguer: Rewriting logic as a logical and semantic framework. *Electronic Notes in Theoretical Computer Science* 4 (1996) 190–225.

[16] K.Meinke and J.V.Tucker: Universal Algebra. In: S.Abramsky, D.Gabbay and T.Maibaum (eds), *Handbook of Logic in Computer Science*, *Volume I: Mathematical Structures, Oxford University* (1992) 189–411.

[17] J.Meseguer: Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science* 96(2) (1992) 73–155.

[18] R.Milner: *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press (1999).

[19] P.Viry: A rewriting implementation of pi-calculus. Technical Report TR-96-30, Departimento di Informatica, Universit'a di Pisa, 26 (1996).

[20] E.Visser: Stratego: A language for program transformation based on rewriting strategies. Proc. of *RTA'04*, Springer, LNCS 2051 (2001) 357–361.