Rewriting Logic and Elan:
Prototyping Tools for Petri Nets with Time

L. J. Steggles
Department of Computer Science, University of Newcastle, UK.
email L.J.Steggles@newcastle.ac.uk

November 9, 2000

Abstract

Rewriting logic (RL) is an extension of standard algebraic specification techniques which uses rewrite rules to model the dynamic behaviour of a system. In this paper we consider using RL and an associated support tool Elan as an environment for rapidly prototyping and analysing Petri nets with time. We link these algebraic tools to PEP, an existing Petri net tool which provides a user-friendly front end to our framework. Our flexible approach allows the wide range of possible time extensions presented in the literature to be investigated and thus overcomes one of the major drawbacks of the current hardwired tools. We demonstrate our ideas by considering time Petri nets in which transitions are associated with a firing interval. The flexibility of our approach is illustrated by modelling a range of semantic alternatives for time Petri nets taken from the literature.

1 Introduction.

The theory of Petri nets (see for example Peterson [1977], Reisig [1985] and Murata [1989]) provides a graphical notation with a formal mathematical semantics for modelling and reasoning about concurrent, distributed systems. One shortcoming of basic Petri nets is that they do not provide any insight into the time behaviour of systems. For real-time systems such as protocols with timeouts such timing information is extremely important (see for example Walter [1983]). To address this a variety of Petri net extensions with time have been proposed in the literature (see the surveys Bestuzheva and Rudnev [1990] and Starke [1995]). One problem however is the sheer number of different semantic interpretations that can be made: timing information can be assigned either to places, transitions, arcs or tokens; time durations or intervals can be used; specified time can represent a period of inhibition or a period when an activity can occur. The tools currently available are unable to cope with this wide range of choices and tend to be hardwired to one specific time approach. This makes investigating different time extensions extremely difficult.

In this paper we consider using rewriting logic (RL) (see Meseguer [1992]) and an associated support tool Elan (see Borovansky et al [1998a]) to rapidly prototype and analyse Petri nets with time. RL is an algebraic formalism that extends the standard algebraic specification techniques by allowing the dynamic behaviour of systems to be modelled using rewrite rules. The idea in RL is to define the static and functional aspects of a system using a standard algebraic specification and to then view terms over this specification as system states. Rewrite rules are then used to specify the dynamic transitions between these states.

As a case study we consider prototyping and analysing time Petri nets in which an interval for firing is associated with each transition (see Merlin and Faber [1976]). We present an RL
model for this time extension and consider how the support tool Elan can be used to simulate and analyse the RL model. We consider what it means for our model to be correct and provide a formal argument to show that the model we have given correctly simulates time Petri nets. Even for this standard approach to extending Petri nets with time there are a range of possible semantic interpretations that can be considered (see Starke [1995]). We illustrate the flexibility of our approach by considering how to adapt our model to represent these semantic alternatives.

In order to make our approach practical and take advantage of existing Petri net tool support we have linked our approach with the widely used PEP tool (see Best and Grahlmann [1996]). We use PEP as a front end to our framework, using it to create the initial Petri net graphs and to animate the firing sequences that result from our RL simulations using Elan. An overview of how we integrate the two tools Elan and PEP is given in figure 1. This work illustrates how existing modelling tools can be combined to address new problems.

![Figure 1: Integration of Elan and PEP.](image)

The paper is organized as follows. In Section 2 we introduce the essential background definitions and results concerning RL and the support tool Elan. In Section 3 we introduce *time Petri nets* and consider how to model and analyse such nets using RL and Elan. This case study demonstrates how different semantic choices can be explored and we present a correctness argument to show that our model correctly simulates a time Petri net. Finally in Section 4 we present some concluding remarks.

We note that we assume the reader is familiar with the basic notation and definitions of Petri nets (see for example Peterson [1977], Reisig [1985] and Murata [1989]).

2 Background: Petri Nets, Rewriting Logic and Elan.

In this section we briefly present the background material on Rewriting Logic (RL) and its associated support tool Elan needed for this paper. We present a small illustrative example of how RL can be used to model simple P/T nets (see Meseguer [1992]).

2.1 Rewriting Logic.

*Rewriting Logic* (RL) is an extension of standard algebraic specification techniques which is able to model dynamic system behaviour. In RL the functional and static properties of a system
are described by a standard algebraic specification, whereas the dynamic behaviour of the system is modelled using rewrite rules. Terms over a given signature \( \Sigma \) represent the global states (or configurations) of a system and rewrite rules model the dynamic transitions between these states. We now present a brief introduction to RL; for a more detailed introduction to RL see Meseguer and Winkler [1992] and Meseguer [1992].

A standard algebraic specification \((\Sigma, E)\) is a pair consisting of a signature \( \Sigma \) and a set of equations \( E \) over \( \Sigma \) and a set of variables \( X \) (see for example Ehrlig and Mahr [1995] and Loeckx et al [1996]). In RL such a specification is seen as defining the states of a system with each equivalence class (with respect to the equations \( E \)) of terms \([t]_E\) being a particular state. We can then define rewrite rules \( t \rightarrow t' \), for terms \( t, t' \) over \( \Sigma \) and variables \( X \), which define the dynamic transitions that can occur between states.

2.1.1 Definition. A Rewriting Logic specification \( Spec = (\Sigma, E, R) \) is a triple consisting of:
- an algebraic signature \( \Sigma \) which defines a set of sorts \( S \) and a set of function symbols \( \Sigma \);
- a set of equations \( E \) over \( \Sigma \) and a set of variables \( X \);
- and a set of rewrite rules \( R \) over \( \Sigma \) and \( X \).

As an example of an RL specification let us consider how we might model the simple Petri net depicted in figure 2 (see Meseguer [1992] for a more detailed discussion). The basic idea will be to model a token being present on a place \( p_i \) by a constant \( p(i) \). A marking can then be modelled as a multi-set of these constants, for example the marking which contains two tokens on place \( p_1 \), one token on place \( p_3 \) and three tokens on place \( p_4 \) could be represented by the term

\[ p(1) \otimes p(1) \otimes p(3) \otimes p(4) \otimes p(4) \otimes p(4), \]

where \( \otimes \) is the symbol used to denote multi-set union. Note that since places \( p_2 \) and \( p_5 \) don't contain any tokens they do not appear in the multi-set. Each transition will be represented by a rewrite rule which consumes tokens and produces new tokens. For example, transition \( t_3 \) would be modelled by the following rule:

\[ p(3) \otimes p(4) \rightarrow p(1) \otimes p(2). \]

The complete RL specification \( SpecPN = (\Sigma, E, R) \) for the Petri net depicted in figure 2 is defined below.

![Figure 2: A simple example of a Petri net.](image-url)
(i) Signature $\Sigma$: Let $S = \{\text{place}, \text{pnet}\}$ be a sort set and let $\Sigma$ be an $S$-sorted signature which contains the following function symbols:

$$\text{empty} : \text{pnet}, \quad \otimes : \text{pnet} \otimes \text{pnet} \to \text{pnet}.$$  

For simplicity we assume the signature contains an implicit type coercion operator $\_ : \text{place} \to \text{pnet}$.

(ii) Equations $E$: Define the set of equations $E$ to contain the following two equations which axiomatize the commutativity and associativity properties of $\otimes$:

$$m_1 \otimes m_2 = m_2 \otimes m_1, \quad m_1 \otimes (m_2 \otimes m_3) = (m_1 \otimes m_2) \otimes m_3.$$  

Note that these equations allow the elements within a multi-set to move around and that the rewrite rules defined below will be applied modulo these equations.

(iii) Rewrite rules $R$: Finally, define the set of rewrite rules $R$ to contain the following rules which axiomatize the transitions in the Petri net:

$$p(1) \longrightarrow p(3), \quad p(2) \longrightarrow p(4),$$

$$p(3) \otimes p(4) \longrightarrow p(1) \otimes p(2), \quad p(3) \longrightarrow p(5).$$

Let $p(1) \otimes p(2)$ be the multi-set representing the initial marking in figure 2. Then the firing sequence $t_1 \ t_2 \ t_3$ in the Petri net can be simulated by the following sequence of rewrites:

$$p(1) \otimes p(2) \longrightarrow p(3) \otimes p(2) \longrightarrow p(3) \otimes p(4) \longrightarrow p(1) \otimes p(2).$$

2.2 The Support Tool Elan.

A number of advanced support tools have been developed to allow RL specifications to be simulated and analysed including Maude (Clavel et al [1998]), Elan (Borovanský et al [1998a]) and CafeOBJ (Diaconescu et al [1998]). These tools provide the means of performing fast AC rewriting, modular structuring mechanisms, and powerful user definable rewrite strategies. We have chosen to use the Elan system here (see Borovanský et al [1996], Borovanský et al [1998a] and the tool's web site\(^1\)). This choice was motivated mainly by the author's experience with the tool and the fact that Elan has a simple built-in strategy language.

As an example of the syntax of Elan we formulate the RL specification presented in the previous section in Elan. Note that we use the ASCII symbol $\#$ to represent multi-set union $\otimes$, $E$ the empty multi-set and for generality replace the constants $p(1), \ldots, p(5)$ by a function symbol $p : \text{int} \to \text{place}$.

```plaintext
module PNet
  import global int;   end
  sort place pnet;     end
operators global
  $\emptyset$           : (place) pnet;
  $\otimes$             : pnet;
  $\emptyset \ # \ \emptyset$ : (pnet pnet) pnet \ (AC);
```

\(^1\)http://www.loria.fr/equipes/protheo/PROJECTS/ELAN/elan.html
p(Ø) : (int) place;
end
rules for pnet
global
  □ p(1) => p(3)  end
  □ p(2) => p(4)  end
  □ p(3) # p(4) => p(1) # p(2)  end
  □ p(4) => p(5)  end
end

The symbol Ø is used to denote the position of an argument to a function symbol allowing a
mix-fix notation. Each rule can be given an optional label by including text within the square
brackets at the start of the rule (all the rules above are unlabelled). The equations of the RL
specification have not been explicitly given in the Elan specification. Instead for reasons of
efficiency the built-in associativity and commutativity facility of Elan has been used by flagging
# as an (AC) operator.

One key feature of Elan is that it provides a built in strategy language for controlling the
application of rewrite rules. It allows the user to specify a general order in which rewrite rules
are to be applied and the possible choices that can be made. The result of applying a strategy to
a term is the set of all possible terms that can be produced according to the strategy. A strategy
is said to fail if, and only if, it can not be applied (i.e. produces no results). The following is a
brief overview of Elan’s elementary strategy language:

(i) **Basic strategy:** 1 Any labelled rule [1] t => t’ is a strategy. The result of applying
this basic strategy is the set of all terms that could result from one application of the labelled
rule. The strategy is said to fail if, and only if, the labelled rule cannot be applied.

(ii) **Concatenation strategy:** s1; s2 The concatenation strategy allows two strategies s1 and
s2 to be sequentially composed, i.e. s2 is applied to the results from s1. The strategy fails if,
and only if, either s1 or s2 fails.

(iii) **Don’t know strategy:** dk(s1, . . . , sn) The don’t know strategy takes a list of strategies
s1, . . . , sn and returns the union of all possible sets of terms that can result from these strategies.
This strategy fails if, and only if, all the strategies s1, . . . , sn fail.

(iv) **Don’t care strategy:** dc(s1, . . . , sn) The don’t care strategy takes a list of strategies
s1, . . . , sn and chooses nondeterministically to apply one of these strategies si which does not fail.
Thus the strategy can only fail if all of s1, . . . , sn fail. The strategy dc one(s1, . . . , sn) works in
a similar way but chooses a single result term to return. One final variation is the strategy
first(s1, . . . , sn) which applies the first successful strategy in the sequence s1, . . . , sn.

(v) **Iterative strategies:** repeat*(s) The repeat*(s) strategy repeatedly applies s, zero or
more times, until the strategy s fails. It returns the last set of results produced before the strategy
s failed. The repeat+(s) version works in a similar way but insists that s must be successfully
applied at least once.

Elan also provides the so called defined strategy language which extends the above elementary
language by allowing recursive strategies. For a detailed discussion of Elan’s strategy language see
Borovansky [1988a]. Examples of the application of Elan’s strategy languages will be presented
in the case study that follows.
3 Modelling Time Petri Nets using RL.

In this section we consider modelling and analysing time Petri nets, a time extension in which transitions are associated with firing intervals. We begin by introducing the general ideas and basic definitions of time Petri nets. We then consider how to model time Petri nets using RL and their analysis using the support tool Elan. We conclude by presenting a correctness argument for our RL model and by demonstrating the flexibility of our approach by modelling a range of semantic alternatives for time Petri nets.

3.1 Time Petri Nets.

Time Petri nets were introduced in Merlin and Faber [1976] and have since become one of the most popular Petri net time extensions (see for example Berthomieu and Diaz [1991] and Aura and Lilius [1997]). Time Petri nets are based on associating a firing interval $[e, l]$ with each transition, where $e$ is referred to as the earliest firing time and $l$ is referred to as the latest firing time. The idea is that a transition is only allowed to fire if it has been continuously enabled for at least $e$ units of time and is forced to fire once it has been enabled for $l$ units of time (unless a conflicting transition fires first). Firing a transition (i.e. consuming enabling tokens and producing output tokens) is assumed to be instantaneous.

As an example consider the time Petri net depicted in figure 3. In this example both $t_1$ and $t_2$ are enabled but only transition $t_2$ can fire since its earliest firing time is zero. Transition $t_1$ needs to be enabled for at least 1 clock cycle before it can fire. Both transitions will be forced to fire after they have been enabled for 2 clock cycles.

For simplicity, we assume we are dealing with discrete intervals (see Aura and Lilius [1997]) and let $I = \{[e, l] \mid e \in \mathbb{N}, l \in \mathbb{N} \cup \{\infty\}, e \leq l\}$. Note that a latest firing time of $\infty$ indicates that a transition will never be forced to fire. We can formally define a time Petri net as follows.

3.1.1 Definition. A Time Petri Net $TPN = (P, T, F, m_0, SI)$ is a 4-tuple where:

$P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places;
$T = \{t_1, t_2, \ldots, t_k\}$ is a finite set of transitions, such that $P \cap T = \emptyset$;
$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (called a flow relation);
$m_0 : P \rightarrow \mathbb{N}$ is the initial marking of the Petri net; and
$SI : T \rightarrow I$ is a static interval function that assigns a firing interval to each transition.

Let $TPN = (P, T, F, m_0, SI)$ be a time Petri net. When $TPN$ is clear from the context we let $Eft(t)$ and $Lft(t)$ denote the earliest and latest firing times respectively for any transition $t \in T$. A state $(m, c)$ in $TPN$ is a pair consisting of a marking $m : P \rightarrow \mathbb{N}$ and a clock function $c : T \rightarrow \mathbb{N}$ indicating the state of each transitions local clock. For each transition $t \in T$ its local clock records the amount of time $0 \leq c(t) \leq Lft(t)$ that $t$ has been continuously enabled. We let $States(TPN)$ denote the set of all possible states in $TPN$. For any transition $t \in T$, we let $\bullet t = \{ p \mid (p, t) \in F \}$ denote the set of input places to $t$ and $t\bullet = \{ p \mid (t, p) \in F \}$ denote the set of output places to $t$. A transition $t \in T$ is said to be enabled in a state $(m, c)$ if, and only if, $m(p) > 0$, for each $p \in \bullet t$. We let $Enabled(m)$ denote the set of all enabled transitions in a state $(m, c)$.

Next we define the conditions necessary for a transition to be able to fire.

### 3.1.2 Definition
A transition $t \in T$ is fireable in state $(m, c)$ after delay $d$ if, and only if,
(i) $t$ is enabled in $(m, c)$;
(ii) $Eft(t) \leq c(t) + d \leq Lft(t)$; and
(iii) for all other enabled transitions $t' \in Enabled(m)$ we have $c(t') + d \leq Lft(t')$.

We denote by $Fireable((m, c), d)$ the set of all transitions that may be fired in a state $(m, c)$ with delay $d$. We can define what happens to a state when a transition fires as follows.

### 3.1.3 Definition
Given a transition $t \in Fireable((m, c), d)$ we can fire $t$ after a delay $d$ to produce a new state $(m', c')$, denoted $(m, c)[t, d > (m', c')]$, which is defined as follows:

$m' = m'' \cup t \bullet$ and $m'' = m \setminus t\bullet$;

$c'(t_i) =\begin{cases} c(t_i) + d, & \text{if } t_i \in Enabled(m'') \text{ and } t_i \neq t; \\ 0, & \text{otherwise}; \end{cases}$

for all $t_i \in T$.

Note that in the above definition the new tokens produced by a transition are discounted when considering whether or not to reset a transitions local clock (this is the reason for defining the intermediate marking $m''$). In other words, we are able to distinguish newly produced tokens (see Aura and Liljes [1997]). Other semantic approaches exist for resetting local clocks (see Starke [1995]) and we will consider modelling some of these in Section 3.4.

A firing schedule for a time Petri net is a sequence of firing steps (i.e. pairs of transitions and delay values):

$$\sigma = (t_1, d_1), (t_2, d_2), \ldots, (t_k, d_k),$$

where $t_i \in T$ and $d_i$ is some delay. A firing sequence $\sigma$ is said to be fireable from a state $s_1$ if, and only if, there exists states $s_2, \ldots, s_{k+1}$ such that $s_i[t_{f_i}, d_i > s_i+1$, for $i = 1, \ldots, k$.

### 3.2 Modelling Time Petri Nets using RL

We now consider how to construct an RL model of a time Petri net that correctly simulates its behaviour. We build on the multi-set approach introduced in Section 2 and introduce new terms $t(i)[n, e, l]$ to represent transitions, where $e$ is the earliest firing time, $l$ the latest firing time and $n$ is the amount of time a transition has been continuously enabled. For example, the initial state of the time Petri net depicted in figure 3 would be represented by the following RL term:

$$p(1) \otimes p(2) \otimes t(1)[0, 1, 2] \otimes t(2)[0, 0, 2] \otimes t(3)[0, 0, 1] \otimes t(4)[0, 1, 1].$$
In order to allow an in-depth analysis of a time Petri net we enhance this term structure for a state to include information about what action resulted in the state:

\[
\langle \text{pn} \rangle[t]s,
\]

where \( \text{pn} \) is a multi-set representing the current state (see above example) and \( ts \) is a multi-set recording the action that produced the state.

Next we define the rewrite rules that will be used to model the dynamic behaviour of a time Petri net. In our RE model we choose to simulate time progression by single clock ticks. We show in Section 3.3 that this approach is equivalent to allowing arbitrary time progression as defined in Definition 3.1.3. Let \( n, e, l : \text{nat} \) and \( \text{pn}, ts : \text{pn} \) be variables. For each transition \( t_i \in T \) with input places \( p_{in_1}, \ldots, p_{in_j} \) and output places \( p_{o_1}, \ldots, p_{o_k} \) we have four distinct types of labelled rules:

1. **A must fire rule** that forces a transition to fire when it has reached its latest firing time:

   \[\text{[mr]} \quad \langle p(in_1) \otimes \ldots \otimes p(in_j) \otimes t(i)[l, e, l] \otimes \text{pn} \rangle[t]s \rightarrow \langle t(i)[0, e, l] \otimes N(p(o_1)) \otimes \ldots \otimes N(p(o_k)) \otimes \text{pn} \rangle[t(i) \otimes ts] \]

2. **A firing rule** that allows a transition to choose to fire if it is within its firing interval:

   \[\text{[fr]} \quad \langle p(in_1) \otimes \ldots \otimes p(in_j) \otimes t(i)[n, e, l] \otimes \text{pn} \rangle[t]s \rightarrow \langle t(i)[0, e, l] \otimes N(p(o_1)) \otimes \ldots \otimes N(p(o_k)) \otimes \text{pn} \rangle[t(i) \otimes ts] \quad \text{if } n \geq e \land n < l \]

3. **A time progression rule** to allow an enabled transition to progress in time:

   \[\text{[tr]} \quad \langle p(in_1) \otimes \ldots \otimes p(in_j) \otimes t(i)[n, e, l] \otimes \text{pn} \rangle[t]s \rightarrow \langle p(in_1) \otimes \ldots \otimes p(in_j) \otimes D(t(i)[n + 1, e, l]) \otimes \text{pn} \rangle[D(t(i)) \otimes ts] \]

4. **Finally we have an enabling rule** which distinguishes all enabled transitions (ignoring newly produced tokens):

   \[\text{[sr]} \quad \langle p(in_1) \otimes \ldots \otimes p(in_j) \otimes t(i)[n, e, l] \otimes \text{pn} \rangle[t]s \rightarrow \langle p(in_1) \otimes \ldots \otimes p(in_j) \otimes D(t(i)[n, e, l]) \otimes \text{pn} \rangle[t]s \]

The [fr] and [mr] rules allow a transition to fire if it is within its firing interval. The new tokens produced are distinguished by a marker \( N \) which allows them to be temporarily ignored when considering whether or not a transition is enabled (as required by the semantics defined in Definition 3.1.3). The distinction between a can fire [fr] and a must fire [mr] rule is needed to allow an appropriate rewrite strategy to be formulated (see below) to capture the semantics of a time Petri net. As long as no [mr] rule can be applied the [tr] rule can be used to allow time to progress by one unit. These rules use markers \( D \) to synchronize time progression and prevent multiple clock ticks. The enabling rules [sr] are used to distinguish those transitions which are still enabled after a transition has fired (ignoring newly produced tokens). All transition terms which are not surrounded by a \( D \) marker will have their local clocks reset to zero after a state step by a reset function.

As an example, consider the following partial Elan specification for the time Petri net in figure 3 (for brevity we have only included the rules for transitions \( t_1 \) and \( t_3 \)). The specification is built on top of a module basic which specifies the basic components needed such as multi-sets and place/transition terms.
module timePN
import global basic; end
rules for state
  pn, pn2, ts : pnet;
  m, e, l : int;
global
/** Rules for Transition 1 **/
[mr] <p(1)#t(1)[l,e,1]#pn>[ts] => <t(1)[0,e,1]#N(p(3))#pn>[t(1)#ts] end
[fr] <p(1)#t(1)[m,e,1]#pn>[ts] => <t(1)[0,e,1]#N(p(3))#pn>[t(1)#ts]
  if m >= e and m < 1 end
[tr] <p(1)#t(1)[m,e,1]#pn>[ts] => <D(t(1)[m+1,e,1])#p(1)#pn>[D(t(1))#ts] end
[sr] <p(1)#t(1)[m,e,1]#pn>[ts] => <D(t(1)[m,e,1])#p(1)#pn>[ts] end
end

/** Rules for Transition 3 **/
[mr] <p(3)#p(4)#t(3)[l,e,1]#pn>[ts] =>
  <t(3)[0,e,1]#N(p(1))#N(p(2))#pn>[t(3)#ts] end
[fr] <p(3)#p(4)#t(3)[m,e,1]#pn>[ts] =>
  <t(3)[0,e,1]#N(p(1))#N(p(2))#pn>[t(3)#ts] if m=e and m<1 end
[tr] <p(3)#p(4)#t(3)[m,e,1]#pn>[ts] =>
  <D(t(3)[m+1,e,1])#p(3)#p(4)#pn>[D(t(3))#ts] end
[sr] <p(3)#p(4)#t(3)[m,e,1]#pn>[ts] => <D(t(3)[m,e,1])#p(3)#p(4)#pn>[ts] end
end

Given a term representing a state in a time Petri net we can either choose to fire a transition or allow time to progress by one unit. However, time progression is not allowed if there exists a transition which has reached its latest firing time; in this case we are forced to fire such a transition. In order to correctly model the semantics outlined above we need to define a rewrite strategy step which will control the application of the rewrite rules. This strategy will be used to represent a single state step in a time Petri net. We begin by defining the following four intermediate rewrite strategies:

strategies for state
  implicit
    [] fire => fr;repeat*(dc one(sr)) end
    [] must => mr;repeat*(dc one(sr)) end
    [] time => repeat*(dc one(tr)) end
    [] step => dk(fire, first(must, time) ) end

The rewrite strategy fire is used to choose non-deterministically a transition to fire which is enabled and within its firing interval. It applies the [fr] rule to produce a set of of possible terms and then applies the strategy repeat*(dc one(sr)) to these results to mark all those transitions which are still enabled (the other transitions will have their local clocks reset). The strategy must is similar to fire but chooses transitions to fire which have reached their latest firing time. The time progression strategy time performs one complete time step, incrementing the local clocks of all enabled transitions by one unit. These three strategies are combined into a strategy step which calculates the set of possible next states. It uses the strategy first(must, time) to ensure that time can only progress if no transition is at it’s latest firing time (i.e. it tries to apply must but if this fails then it applies time).
After applying step we “clean up” the resulting state term by applying a reset function which removes all N and D markers, and resets local clocks to zero:

- \( \text{reset}(D(t(n)[m,e,1])) \Rightarrow t(n)[m,e,1] \)
- \( \text{reset}(t(n)[m,e,1]) \Rightarrow t(n)[0,e,1] \)
- \( \text{reset}(N(p(n))) \Rightarrow p(n) \)
- \( \text{reset}(p(n)) \Rightarrow p(n) \)
- \( \text{reset}(E) \Rightarrow E \)
- \( \text{reset}(pn \# pn2) \Rightarrow \text{reset}(pn) \# \text{reset}(pn2) \)

where we have the variables \( pn, \ pn2 : \ \text{pnet and m,e,1 : int.} \)

Thus a state step in our RL model, denoted \( s \rightarrow s' \) using step, involves applying the strategy step and then the function reset. Given we can now define a state step it is interesting to consider how to explore the resulting state space. We begin by defining the exit states we wish to find by specifying exit conditions as rewrite rules, e.g.

\[
\begin{align*}
\text{[exit]} & \quad <p(4) \# pn>[t_\text{s}] \Rightarrow <p(4) \# pn>[t_\text{s}] \\
& \text{Place } P4 \text{ contains at least one token;}
\end{align*}
\]

\[
\begin{align*}
\text{[exit]} & \quad <pn>[t_\text{s} \# t_\text{s}] \Rightarrow <pn>[t_\text{s}] \\
& \text{Transition } T4 \text{ has fired;}
\end{align*}
\]

\[
\begin{align*}
\text{[exit]} & \quad <pn>[t_\text{s}] \Rightarrow <pn>[t_\text{s}] \text{ if } \text{length}(pn) > 10 \\
& \text{The size of the state (tokens plus number of transitions) has exceeded } 10.
\end{align*}
\]

A state is said to be an exit state if, and only if, one of the exit rules can be successfully applied to it. Using the defined strategy language of Elan (see Borovansky et al [1998b]) we can then define various search strategies that look for exit states. For example, we could define a strategy search which given an initial state performs a depth first search (possibly bounded) until it finds an exit state. This strategy can be generalized to a strategy searchall which finds all exit states. (For a detailed discussion of search strategies see see Borovansky [1998] and Steggles [2000]).

As an example, consider using Elan and the strategy search to find a firing sequence for the time Petri net in figure 3 which results in a state with a token on place \( p_5 \), i.e. we have the exit rule:

\[
\begin{align*}
\text{[exit]} & \quad <p(5) \# pn>[t_\text{s}] \Rightarrow <p(5) \# pn>[t_\text{s}] \quad \text{end}
\end{align*}
\]

The following is an excerpt from the Elan tool:

\[
\begin{align*}
\text{[start with term :} \\
& \text{[search]} <p(1)\#p(2)\#t(1)[0,1,2]\#t(2)[0,0,2]\#t(3)[0,0,1]\#t(4)[0,1,1]>[E] \\
& \text{[result term :} \\
& (<p(4)\#p(5)\#t(3)[0,0,1]\#t(4)[0,1,1]\#t(2)[0,0,2]\#t(1)[0,1,2]>[t(4)]). \\
& (D(t(4))\#D(t(3)))([t(1)]). (D(t(1)))([t(2)])([t(3)])([t(4)]). \\
& (D(t(1)))([t(2)])([E])) \\
& \text{[end]}
\end{align*}
\]

The above result term indicates one possible firing sequence that produces an exit state from the initial marking. It has been displayed using a display strategy (see Steggles [2000]) that
outputs only the final state reached, and the steps involved in reaching that state (i.e. \( [t(i)] \) indicates transition \( t_i \) has fired and \( D(t(i)) \neq D(t(k)) \) indicates that transitions \( t_i \) and \( t_k \) have progressed by one unit in time). The above term represents a firing sequence involving nine state steps (the initial \([E]\) represents the initial state). It corresponds to the following time Petri net firing sequence:

\[
(t_2, 0), (t_1, 1), (t_3, 0), (t_2, 0), (t_1, 1), (t_4, 1).
\]

### 3.3 Correctness Argument.

In this section we consider the correctness of our RL model for time Petri nets. We show that our model is both sound (each step in our RL model has a corresponding state step in the time Petri net) and complete (every state step possible in a time Petri net has a corresponding step in our RL model).

In the sequel let \( TPN = (P,T,F,m_0,SI) \) be an arbitrary time Petri net and let \( RL(TPN) \) be the corresponding RL model as defined in Section 3.2.

It turns out that not all terms of type \( puet \) in \( RL(TPN) \) represent valid states in \( TPN \). Thus we define \( ValidRL(TPN) \), the set of all state terms \( s \) in \( RL(TPN) \) such that: (1) if \( p(i) \) in \( s \) then \( p_i \in P \); and (2) if \( t[i][n,e,l] \) in \( s \) then \( t_i \in T, [n,e] = SI(t_i), \) and \( 0 \leq n \leq l \). If \( s \in ValidRL(TPN) \) then we say \( s \) is a valid state term for \( TPN \).

#### 3.3.1 Proposition. The rewrite strategy step is well-defined with respect to valid state terms, i.e. for any \( s \in ValidRL(TPN) \), if \( s \rightarrow s' \) using step then \( s' \in ValidRL(TPN) \).

**Proof.** Suppose \( s \in ValidRL(TPN) \) and \( s \rightarrow s' \) using step. By definition of the strategy step it follows there are three cases to consider:

- **Case (1):** The strategy \( \text{fire} \) was used (followed by the reset function). The application of a \([fr]\) rule simply consumes some token terms and produces some new token terms which must by definition correspond to places in \( TPN \). The application of the \([sr]\) rules taken together with the reset function will just reset the local clock of some transition terms to zero. Thus if the original state term \( s \) was valid then the resulting state term \( s' \) after applying \( \text{fire} \) will also be valid.

- **Case (2):** The strategy \( \text{must} \) was used (followed by the reset function). Similar argument to above based on the \([sr]\) rules.

- **Case (3):** The strategy \( \text{must} \) failed and so the strategy \( \text{time} \) was used (followed by the reset function). If the \( \text{must} \) strategy fails then we know there are no enabled transition terms in \( s \) which have reached their latest firing time. This means that performing a time step, that is incrementing by one the local clock of all enabled transitions (which is exactly what the strategy \( \text{time} \) does), will not result in any local clock going past its transitions latest firing time. Thus the resulting state term \( s' \) must be valid.

Recall that our model is said to be sound if, and only if, each step in our RL model has a corresponding state step in the time Petri net; and complete if, and only if, every state step possible in a time Petri net has a corresponding (sequence of) step(s) in our RL model. We can now define what we mean by correctness: \( RL \) model is correct with respect to a time Petri net if, and only if, it is both sound and complete with respect to the time Petri net.

We show that for any time Petri net \( TPN \), the corresponding RL model \( RL(TPN) \) is correct. We begin our correctness proof by defining a mapping between states of a time Petri net and valid state terms in the corresponding RL model as follows.


\[\]
3.3.2 Definition. The term mapping \( \sigma : \text{States}(TPN) \rightarrow \text{ValidRL}(TPN) \) is defined on each state \((m,c)\) to return the multi-set term \(\sigma(m,c)\) which contains only the following:
(i) for each place \(p_i \in P\), \(\sigma(p_i)\) will contain \(m(p_i)\) occurrences of the place term \(p(i)\);
(ii) for each transition \(t_i \in T\), \(\sigma(t_i)\) will contain the transition term \(t(t_i)[c(t_i), Eft(t_i), Lft(t_i)]\).

By the definition of \(\text{ValidRL}(TPN)\) it is straightforward to show that \(\sigma\) is a well-defined, bijective mapping with an inverse \(\sigma^{-1} : \text{ValidRL}(TPN) \rightarrow \text{States}(TPN)\).

Suppose a state step \((m,c)[t,d > (m',c')]\) can occur in a time Petri net. Then observe that we can break such a state step down into a series of intermediate steps consisting of a series of time ticks which allow the delay \(d\) to pass, followed by the transition \(t\) firing. In other words we can represent \((m,c)[t,d > (m',c')]\) by the following sequence of events:

\[(m,c)[\text{tick} > (m,c_1)[\text{tick} > (m,c_2)[\text{tick} > \cdots [\text{tick} > (m,c_d)[t > (m',c')],\]

where \((m,c_1)[\text{tick} > (m,c_{i+1})\) represents a clock tick (i.e. increments the local clocks of all enabled transitions by one unit, resetting all other local clocks to zero) and \((m,c_d)[t > (m',c_d)\) fires transition \(t\) and resets to zero the local clocks of all transitions which are not enabled (i.e. corresponds to \((m,c_d)[t, 0 > (m',c')]\).

![Figure 4: (a) Soundness; (b) Completeness.](image)

Using the above observation we can show that for any timed Petri \(TPN\), the corresponding RL model \(RL(TPN)\) is sound and complete with respect to \(TPN\).

3.3.3 Theorem. (Soundness) Let \(s \in \text{ValidRL}(TPN)\) be any state term. If \(s \to s'\) using the strategy step then either \(\sigma^{-1}(s)[\text{tick} > \sigma^{-1}(s')\) or there must exist a transition \(t \in T\) such that \(\sigma^{-1}(s)[t > \sigma^{-1}(s')\), i.e. the diagram in figure 4.(a) commutes.

Proof. Suppose \(s \in \text{ValidRL}(TPN)\) and \(s \to s'\) using step. By definition of the strategies step it follows that there are three cases to consider:

Case (1): The strategy fire was used (followed by the reset function). This strategy begins by applying a \([fr]\) rule which corresponds to some transition, say \(t \in T\). We will show that the result of this strategy \(s'\) corresponds to the result of \(\sigma^{-1}(s)[t > \sigma^{-1}(s')\). Clearly if this rule can be applied then it follows that the local clock for \(t\) must be within the \(Eft(t)\) and \(Lft(t)\) firing times and that for each \(p_i \in \bullet t\) there must exist a token term \(p(i) \in s\). Thus it follows that the transition \(t\) must be fireable in state \(\sigma^{-1}(s)\). Applying the \([fr]\) rule will result in the token terms corresponding to the input places \(\bullet t\) being removed and token terms corresponding to the output places \(\bullet s\) being produced. But this is exactly what will happen to the marking in state.
\( \sigma^{-1}(s) \) when \( t \) is fired. Finally, the strategy fire will use the \([sr]\) rules and reset function to set all unenabled transition clocks (discounting the newly produced token terms) to zero. Again, by definition of firing a transition (see Definition 3.1.3) this is what happens when \( t \) is fired. **Case (2):** The strategy must was used (followed by the reset function). Similar argument to above using \([sr]\) rules. **Case (3):** The strategy must failed and so the strategy time was used (followed by the reset function). If the must strategy fails then we know there are no enabled transition terms in \( s \) which have reached their latest firing time. This means that the strategy time will be applied and a time step will be performed, i.e. the local clocks of all enabled transitions will be incremented by one and all other local clocks will be reset. Note that allowing time to progress is a valid action since no transition has reached its latest firing time. In this case it will be valid to apply the tick action to the state \( \sigma^{-1}(s) \) which performs exactly the time update detailed above. Thus we will have \( \sigma^{-1}(s') \langle \text{tick} \rangle > \sigma^{-1}(s) \) as required. \( \square \)

3.3.4 Theorem. (Completeness) Let \((m, c) \in \text{States}(TPN)\) be any state in TPN. If \((m, c) \langle t, d \rangle > (m', c') \) for some transition \( t \in T \) and some duration \( d \), then it follows that \( \sigma(m, c) \rightarrow \sigma(m', c') \) by a series of applications of the strategy step, i.e. the diagram in figure 4. (b) commutes.

**Proof.** Suppose that \((m, c) \langle t, d \rangle > (m', c') \) for some transition \( t \in T \) and some duration \( d \). Then by the observation above we can represent this by the following sequence of events:

\[
(m, c) \langle \text{tick} \rangle > (m, c_1) \langle \text{tick} \rangle > (m, c_2) \langle \text{tick} \rangle \cdots \langle \text{tick} \rangle > (m, c_d) \langle t \rangle > (m', c').
\]

Thus it suffices to show that the following two facts hold.

**Fact 1:** If \((m, c) \langle \text{tick} \rangle > (m', c') \) then it follows that \( \sigma(m, c) \rightarrow \sigma(m', c') \) using the strategy step.

**Proof.** Suppose that \((m, c) \langle \text{tick} \rangle > (m', c') \). Then it follows that for all \( t_i \in T \) we have \( c(t_i) < L\text{ft}(t_i) \) (otherwise the tick event would not be valid). Therefore we know by definition of \( \sigma \) that no \([sr]\) rule can be applied to the state term \( \sigma(m, c) \) and so the strategy must will fail. This means the time strategy can be applied which (along with reset function) will increment by one the local clocks of all enabled transition terms in \( \sigma(m, c) \), resetting all unenabled transition clocks to zero. But this is exactly what the tick event will do. Thus we have \( \sigma(m, c) \rightarrow \sigma(m', c') \) as required.

**Fact 2:** If \((m, c) \langle t \rangle > (m', c') \), for some \( t \in T \), then \( \sigma(m, c) \rightarrow \sigma(m', c') \) using step.

**Proof.** Suppose \((m, c) \langle t \rangle > (m', c') \). Then we have two cases to consider.

(i) Suppose \( c(t) < L\text{ft}(t) \). By definition of RL(TPN) there is a \([fr]\) rule for transition \( t \). Given that \( t \) is fireable (i.e. enabled in \( m \) and \( c(t) \geq E\text{ft}(t) \)) it follows that its corresponding \([fr]\) rule can be applied to the state term \( \sigma(m, c) \) as part of the strategy fire and that this will remove the enabling token terms \( p(i) \), for each \( p_i \in \cdot t \) and add the new token terms \( p(i) \) to \( \sigma(m, c) \), for each \( p_i \in \cdot t \). Applying the \([sr]\) rules and reset function will then reset the local clocks of all unenabled transitions (ignoring the newly produced token terms). But this corresponds to the event of firing \( t \) to produce \( (m', c') \) and thus we have \( \sigma(m, c) \rightarrow \sigma(m', c') \).

(ii) Suppose \( c(t) = L\text{ft}(t) \). The proof follows along similar lines to (i) above but uses the \([sr]\) rules. \( \square \)

The above two theorems prove that for any timed Petri TPN, the corresponding RL model RL(TPN) is correct with respect to TPN.

3.3.5 Theorem. (Correctness) Given any time Petri net TPN we have that RL(TPN) is correct RL model with respect to TPN.
3.4 Modelling Alternative Semantic Choices.

In the preceding sections we have constructed an RL model for the standard semantic interpretation of time Petri nets (see Berthomieu and Diaz [1991] and Aura and Liljeberg [1997]). In this section we demonstrate the flexibility of our approach by considering some alternative semantic choices and showing how our RL model can be easily adapted to represent these alternatives. We note that for each of these new semantics a corresponding correctness proof along the lines of that given in Section 3.3 would be needed to ensure the new RL model is correct. For brevity we omit these proofs here and leave them as an instructive exercise for the reader.

(1) **Giving priority to latest firing transitions.**
One possible change to the standard semantics is to give priority to those transitions which have reached their latest firing times. This would mean that a transition which is within its firing interval but not yet at its latest firing time would only be allowed to fire if no transition had reached its latest firing time. Such a change in semantics is easy to represent in our model; we simply change the step strategy to reflect this change in priority as follows:

\[
\begin{align*}
\text{step} &\Rightarrow \text{first(must, dk(fire, time))} \\
\text{end}
\end{align*}
\]

The strategy step now reflects the fact that we start by considering only the must fire transitions. If the strategy must fails then clearly no transition has reached its latest firing time; in such a case we can either choose to fire a transition or perform a time step.

(2) **Resetting local clocks of conflicting transitions.**
In Starke [1995] an alternative semantic approach is proposed which involves resetting the local clocks of conflicting transitions. The idea is that when a transition \( t \) fires any transition which shares an input place with \( t \) has its local clock reset to zero. For example, in the time Petri net depicted in figure 3 if transition \( t_4 \) fires then the local clock for transition \( t_3 \) will be reset. This alternative semantics is straightforward to incorporate into our RL model for time Petri nets; we simply change the firing rules ([fr] and [mr]) so that they include all transition terms for conflicting transitions and then reset their clocks when the transition fires. As an illustrative example the following would be the new firing rules for transition \( t_4 \) in the time Petri net depicted in figure 3:

\[
\begin{align*}
\text{[mr]} &\quad <p(3) \# t(4)[1,e,1] \# t(3)[n,e',l'] \# pn[t] \Rightarrow \\
&\quad <t(4)[0,e,1] \# t(3)[0,e',l'] \# N(p(5)) \# pn[t(t4)ts] \quad \text{end} \\
\text{[fr]} &\quad <p(3) \# t(4)[m,e,1] \# t(3)[n,e',l'] \# pn[t] \Rightarrow \\
&\quad <t(4)[0,e,1] \# t(3)[0,e',l'] \# N(p(5)) \# pn[t(t4)ts] \\
&\quad \text{if } m \geq e \text{ and } m < l \quad \text{end}
\end{align*}
\]

(3) **Maximal step semantics.**
The final alternative we consider is imposing a maximal step semantics on time Petri nets: at each step a maximal set of concurrent fireable transitions are allowed to fire (see Starke [1995]). In practice, the implications of such a semantics would represent a major restriction; transitions would be forced to fire whenever they reach their earliest firing time, which contradicts assigning a firing interval to a transition. However, we consider the maximal step semantics as an illustrative example of the flexibility of our framework. To model the maximal step semantics we need only
redefine the fire and step strategies; the time strategy remains the same and of course the must strategy is no longer required.

\[
\begin{align*}
\text{fire} & \Rightarrow \text{repeat}^* (\text{dk}(\text{fr}, \text{mr})); \text{repeat}^* (\text{dc one}(\text{sr})) & \text{end} \\
\text{step} & \Rightarrow \text{first} (\text{fire}, \text{time}) & \text{end}
\end{align*}
\]

4 Conclusions.

In this paper we have considered using RL and the support tool Elan to model and analyse time Petri nets. We discussed the important issue of correctness and showed that our RL model correctly simulates a time Petri net. We demonstrated the flexibility of our approach by considering several alternative semantics for time Petri nets. We showed that these alternatives could be straightforwardly represented in our model by making small adjustments either to the step strategy or to the basic RL rules. This case study illustrates how RL and Elan can be used to prototype and analyse Petri net extensions with time. Furthermore, by coupling our approach with an existing Petri net tool such as PEP we have shown how several different tools can be combined to produce a practical and useable new approach.

We have performed a similar analysis to the one presented here using timed Petri nets (see Ramchandani [1974] and Zuberek [1991]), in which a duration is associated with transitions. This work is presented in technical report Steggs [2000].

The aim of this work has been to: (i) provide a flexible formal framework for defining semantic models of Petri net extensions with time which are succinct and easily communicated; (ii) provide tools to allow a range of different Petri net extensions with time to be simulated and practically investigated, thus overcoming the problems associated with the current hardwired tools. We note that we are not proposing that our approach should replace efficient hardwired tools for large scale verification tasks. We see our approach as allowing a tool developer to formally specify the semantics they have chosen and to expediently prototype their ideas before committing themselves to the development of a practical tool. Our framework could also be seen as a design aid, allowing a developer to test out their ideas before committing to a particular Petri net time model.

In future work we intend to investigate extending our approach to prototyping verification algorithms, such as the finite prefix construction. We also intend to perform a variety of verification case studies to illustrate the application of our methods and investigate its limitations.

Acknowledgements. It is a pleasure to thank Maciej Koutny for many valuable discussions during the preparation of this paper. We are also very grateful to Hans Fleischhack and Christain Stehno who provided helpful advice and technical assistance. Finally, we would like to gratefully acknowledge the financial support provided by the British Council and DAAD during this work.

5 References.


