PROJECT FOR CSC 2108 - AUTOMATED VERIFICATION

Prof. Marsha Checkic

Studying the behaviour of Place/Transition and Coloured Petri Nets through a formalization as Term Rewriting Systems - an implementation using LP.

Cathy Jansen, Alberto Paccanaro

Abstract

Some elementary concepts on Equational Theories and Term Rewriting Systems are introduced, followed by a brief review of the main ideas of the Knuth-Bendix Completion procedure. Then a formalization of Place/Transition and Coloured Petri Nets as Term Rewriting Systems is proposed, which allows the use of Rewriting Techniques for studying the behaviour of such nets. We verify the relationships existing between Completion, Theorem Proving and Term Simplification and some properties of Petri Nets, and show how such operations can be used to prove facts about the nets. Particularly, the formalization proposed in this paper leads to an efficient method for determining the movements of the tokens in the net, and it allows a natural way of simplifying nets in order to express more efficiently the movements of tokens when transitions are fired. The theory has been implemented using LP, the Larch Prover which has been used to derive the results of the examples presented in this paper.

1. An informal introduction to Term Rewriting Systems

In this paragraph an informal introduction to Term Rewriting Systems is presented, together with a description of the Knuth-Bendix Completion Procedure (Knuth et al., 1970), which was suggested as a means of taking an axiomatization of an equational theory and generating a Term Rewriting System that can be used to decide questions of validity of identities in the given theory.

The ability to reason with equations is important in many computer science applications, including algebraic specifications and automatic deductions. Reasoning about equations may, for example, involve deciding if an equation "follows" as a consequence of a given set of equations. For example, the Group Theory can be expressed by the following set of equations:

$$f(e, x) = x \qquad \text{(left identity)}$$
$$f(i(x), x) = e \qquad \text{(left inverse)}$$
$$f(f(x, y), z) = f(x, f(y, z)) \qquad \text{(associativity)}$$

A theorem of an equational theory is also expressed by an equation:

$$l = r$$

and the usual way of proving it by hand, is through a chain of equalities:

$$l = n1 = n2 = ... nm = r$$

where the first and last term of the chain are respectively the left hand side (lhs) and right hand side (rhs) of the theorem, and every term nk is obtained from the last one by substituting a subterm with one equivalent according to the three equations of the Group Theory. This method of deducing a term from another one is called Replacement of Equals by Equals.

Let us introduce a few definitions in order to formalize the above ideas.

Term:let F be a (finite) set of function symbols (we assume constants to be 0-ary function symbols), and X a (countable) set of variables; the set of terms T(F, X) is defined as the smallest set verifying the properties:

a) $X \subseteq T(F, X)$

b) $f \in F$, $t1, \ldots, tn \in T(F, X) \Rightarrow f(t1, \ldots, tn) \in T(F, X)$

A term not containing any variable is called a *pure term*.

Subterm:a term s is a subterm of a term t if s is contained in t; we will write t[s] to denote this.

Substitution:a substitution σ is a finite set of pairs (variable, term)

$$\sigma \equiv\text{def } \{(x_i, ti) \mid xi \in X, ti \in T(F, X), \text{for all } i\}$$

where every ti is a term different from vi , and no two pairs in the set have the same variable as first element of the pair.

We can apply a substitution σ to a term s, obtaining a term sσ defined in the following way:

$x\sigma \equiv\text{def } t \quad \text{if } (x, t) \in \sigma$

$x\sigma \equiv\text{def } x \quad \text{otherwise}$

$f(t1, \ldots, tn)\sigma \equiv\text{def } f(t1\sigma, \ldots, tn\sigma)$

Equation: an equation is an unordered pair of terms:

$$(s, t) \qquad s, t \in T(F, X)$$

In the following, the notation s = t will be used to denote an equation.

Equational Theory: an equational theory is given by a set of equations $E = \{l_i = r_i\}_{i=1}^{n}$

An equational theorem is an equation; proving a theorem is the problem of deciding if the two terms on the right-hand-side (rhs) and left-hand-side (lhs) of the theorem are equal in the given equational theory.

Now we can formalize the concept of:

Replacement (of equals by equals): given an equational theory E, a term t is replaced by a term t', iff $\exists l = r \in E, \exists s$, subterm of t, $\exists \sigma$, substitution, such that $t[s] = l\sigma \wedge t' \equiv t[s \leftarrow r\sigma]$ or $t[s] = r\sigma \wedge t' \equiv t[s \leftarrow l\sigma]$. The term t' will result to be the same as term t except for the subterm $t[s]$.

The following fundamental result ensures that replacing equals by equals is a sound and complete method for equational theories:

***Birkhoff's Theorem*:** an equation s = t is a logical consequence of the set of equational assumptions E (i.e. s = t is a theorem for the theory E) iff it is possible to get s from t or vice versa by repeated applications of Replacement of Equals by Equals (Birkhoff).

Unfortunately proofs by pure replacements are not easily mechanizable since it is clear that they could very easily get into loop situations where the same equation is applied in both directions, or the same equation is applied indefinitely many times. To overcome this problem, Knuth and Bendix (1970) proposed a method in which an equational theory is rephrased as a Term Rewriting System (TRS). Equations of the form:

$$l = r$$

are substituted by oriented rewrite rules:

$$l \rightarrow r$$

which can be applied for replacement only in the left to right direction. A set of Rewrite Rules is called Term Rewriting System. Formally:

Term Rewriting System:a Term Rewriting System (TRS) R is a set of ordered pairs:

$$R \equiv def \{(l, r) \mid l, r \in T(F, X)\}$$

Throughout this paper, a pair (l, r) ∈ R will be called a *rewrite rule*, and we shall write $l \to r$ to denote it; the terms l and r are called the lhs and the rhs of the rule, respectively.

Replacement of equals by equals of equational theories now becomes:

Reduction:a term t is said to reduce in one step to the term t' under a TRS R, iff $\exists l \to r \in R, \exists s$, subterm of t, $\exists \sigma$, substitution, such that $t[s] = l\sigma \wedge t' \equiv t[s \leftarrow r\sigma]$. The term t' will result to be the same as term t except for the subterm $t[s]$.

Finally, we shall define the:

Simplification of a term with respect to a TRS: a term t is said to simplify to a term t' with respect to a TRS R, if there is a sequence of terms t1, t2, ... tn such that t = t1, t' = tn , every term tk is obtained from the previous one by applying one reduction step, and t' is such that no further rule in the TRS can be applied to it. The term t' is called the *Canonical form* of the term t with respect to the TRS.

Clearly, using reduction with a TRS we can not prove all the theorems which we could previously prove using Replacement of Equals by Equals with the original Equational Theory, since we would not be able to prove those theorems for which every possible proof contains at least one use of an equation in the right to left direction. In other words leaving out the symmetry of equality solves the problem of nondeterminism in the rewriting direction and lets mechanization of rewriting easier, but Birkhoff theorem does not apply anymore: it is impossible to prove by replacement using a set of patterns as rewrite rules all that is provable by replacement using the same set of patterns as equations. It follows that we need to transform the TRS, possibly adding some Rewrite Rules, in order to re-capture the lost semantics.

But, how should we transform the TRS in order to be able to prove theorems? We recall that we are dealing with theories in which the only available predicate is equality; and the theorems we want to prove are questions if either the term on the lhs of the theorem is equal to the one on the rhs, or not. Equality is an equivalence relation that therefore divides the set of all possible terms into equivalence classes. Proving a theorem means verifying if the two terms of a theorem belongs to the same equivalence class.

Therefore, if our TRS could guarantee that, starting from any term, the simplification process will always terminate (*Noetherian Property*) (Dershowitz, 1990), and that for each equivalency class there always exists one and only one Canonical form such that all the simplifications of any term belonging to that class end up giving that term (*Confluence Property*) (Dershowitz, 1990), then we could use the following method to prove a theorem using TRS:

1. simplify the lhs and rhs of the theorem to their respective canonical form
2. check if these coincide

A Term Rewriting System for which the two properties above hold, is called a Canonical Term Rewriting System. A canonical system for an equational theory E, if it exists, will have the same theoretical power of E and will eliminate the unmanageable search space often encountered in Automated Theorem Proving.

Clearly, a TRS obtained by arbitrarily orienting a set of equations, will be in general non-canonical. The Knuth and Bendix algorithm is a procedure that, given an equational theory, will try to construct a canonical TRS, when it exists.

To obtain a Noetherian system the Knuth-Bendix method uses an ordering relation over terms, closed under substitution, that is, $t > s \Rightarrow \forall\sigma\ t\sigma > s\sigma$, and chooses to transform the equation s = t into the rewrite rule s -> t only if s > t. In this way, it is easy to see that a sequence of reduction steps generates a decreasing sequence of terms, and therefore its termination is granted. Unfortunately, all the ordering relations over non-pure terms found so far are partial (Bachmair et al., 1986; Dershowitz, 1982); thus, there exists noncomparable pairs of terms and therefore non-orientable equations. This constituted the biggest limitation of the original method, which has been partially solved by later refinements (Buchberger et al., 1983; Dershowitz, 1989; Hsiang et al., 1987).

Knuth and Bendix gave a condition which is necessary and sufficient for a Noetherian TRS to be confluent. First let us introduce the following definition:

Critical Pair: given a TRS R and two rules $l_1 \rightarrow r_1$ and $l_2[u] \rightarrow r_2$ in R, where u is not a variable, if there is a (most general) unifier σ such that $l_1\sigma = u\sigma$ then $\langle l_2[r_1]\sigma, r_2\sigma \rangle$ is called a critical pair of $l_1 \rightarrow r_1$ and $l_2 \rightarrow r_2$. A critical pair is divergent if $l_2[r_1]\sigma \neq r_2\sigma$.

Knuth and Bendix proved the following "local test" theorem:

Theorem: a Noetherian TRS is confluent iff there is no divergent Critical Pair among rules.

Finally, to obtain a confluent system, Knuth and Bendix proposed to increment the system by converting divergent critical pairs into new rules. This method is called the *Knuth-Bendix Completion Procedure*.
If the procedure terminates successfully we obtain a complete TRS which can decide for the word problem for the theory, i.e. the problem of deciding if two terms are equal in a given equational theory. The procedure can also loop forever, due to the fact that the TRS might have an infinite number of rewrite rules. Notice that even if the algorithm does not terminate we still have a semi-decision procedure for the word problem of the theory. This happens because if the problem s=t is true in the theory, we will eventually generate enough rules that can reduce both terms to an identical form. Examples of canonical systems of interesting theories can be found in Hullot (1980).

2. FORMALIZING PLACE/TRANSITION AND COLOURED PETRI NETS AS TERM REWRITING SYSTEMS

In this section we formalize Place/Transition (P/T) and Coloured (CPN) Petri Nets as TRSs.
Petri Nets were developed to model concepts of asynchronous and concurrent processes. They are widely used: their formalism is simple and allows a number of properties to be modelled. We suppose the reader well acquainted with the basic concepts of Petri Nets. Refer to Peterson (1981) or Reising (1985) for an introduction on the topic; for a detailed description of Coloured Petri Nets refer to Jensen (1994). In the next sections we shall refer to deterministic Petri Nets.

A Petri net is a diagram, comprised of places (circles), transitions (rectangles), tokens (dots) and directed arcs. Conditions are represented by places, and events are represented by transitions. A condition holds when it has a token. A condition b is a precondition of an event t, if b ceases to hold when the event t occurs. In the Petri net this is graphically represented by an directed arc from b to t. A condition b is a postcondition of an event t, if b starts to hold after the event t has occurred. This is represented by a directed arc from t to b. Furthermore, if b is not affected by the occurrence of t then there is no arc between b and t.

In Place Transition nets (P/T nets), each place may hold more than one token. Also, a natural number called a weight may be attaced to the arc. A transition t is allowed to fire if all of its preconditions are satisfied. A precondition is satisfied if it has at least n tokens, where n is the weight of the directed arc that connects it to its transition. When a transition t fires a certain number of tokens is removed from each precondition, which is equal to the weight of the arc that connects such a precondition to t. Also, a number of tokens is added to each postcondition equal to the weight of the arc connecting t to such a postcondition.

Colour Petri nets (CPN nets) offer a compact representation of a net by equipping each token with an attached data value - called the token colour. Each place has an associated type, and all tokens must have colours that belong to that type. The colour type is called the colour set of the place. Attaching a colour to each token and a colour set to each place, allows us to use fewer places and fewer transitions then would

would be needed in a P/T net. The firing of a transition is now dependent not only on the availability of an appropriate number of tokens in each precondition, but also on the availability of an appropriately coloured token. Similarily, the output of a transition is not just a token but a token with a specific colour.

All the examples presented in the following were implemented using LP, the Larch Prover. LP is an interactive theorem proving system for multisorted first-order logic. It is currently used to reason about designs for circuits, concurrent algorithms, hardware, and software. For more information on Larch see Guttag, J.V. et al. (1993) and (http://larch-www.lcs.mit.edu:8001/larch/LP/overview.html).

The implementation in LP was quite straightforward. LP provides multisorted first order logic, and the possibility of defining equational theories. A manual method was then used to orient these equations into TRSs, in order to obtain the desired ordering. In this paper we shall focus on the formalization of the nets as TRS, and on the meaning of some basic operations on such TRS. We shall present several examples which have been implemented in LP: the details of the LP implementation and a summary of the actual commands used can be found in the appendix.

The state of a Petri Net at a certain moment, is defined as the distribution of tokens in the net at that moment; this is often referred to as the current marking. *The main idea of this work is to formalize Petri Nets in such a way that the set of possible markings of the net turns out to be the set of terms of a TRS.*

Naturally extending the previous discussion on TRS to multi-sorted logics, we formalize places and transitions in a net as follows:

- <u>Places:</u> we choose to describe each place with a unary function symbol, having an associated sort T, whose argument can be either a constant or a variable symbol of sort T (or a sub-sort of T). If its argument is a constant, say 'a', it will mean that a token of colour T (or a sub-sort of T) and value 'a' is in that place; if its argument is a variable symbol, it will mean that any token of colour T (or a sub-sort of T) can be found in that place.

So, for example, p1(x), where p1 has an associated sort T and x is a variable of type Q (with Q identical to T or sub-sort of T), means that a generic element of type Q is in place p1. Also p1(c), where c is a constant of type Q, means an element of type Q and value c is in place p1.

We also introduce a binary function AND, to represent the case in which the state of the net is constituted of more than one token. Nesting of AND functions allows us to represent any number of tokens. The AND function will have to be treated as an associative-commutative function. In the examples we shall sometimes use the character "^" in infix notation to represent the AND function just described.

So, for example, p1(b) ^ p1(c) , where p1 has an associated sort T and b and c are constant symbols of type Q (with Q identical to T or sub-sort of T), means that p1 contains 2 elements of type Q, namely b and c.

In this way, we obtain that the terms in our TRS turn out to be the set of possible markings of the Petri Net.

- <u>Transitions</u>: for every transition in the net we write one or more rewrite rules. Transitions will not be described explicitly. The meaning of a transition - the passage of tokens from pre-conditions to post-conditions is captured by the structure of the rewrite rules. The lhs of the rewrite rule will be constituted by the pre-conditions of a transition, the rhs by its post-conditions, and we write one rewrite rule for every allowed colour of the tokens that can be found in the pre-conditions and post-conditions: this describes the "passage" of tokens of any allowed colour in the pre-conditions, to tokens of any allowed colour in the post-conditions. Notice how the rewrite rules expresses the fact that tokens are "moved" from pre-condition to post-condition, thus capturing the meaning of the transition.

As explained in section 1, a rewrite rule transforms a term into another one of lower weight; by analogy this means that any transition firing changes a net into a state of lower weight. It is clear that the orientation of the rules is important: the type of ordering over terms must be chosen in a way that the proper orientation of the rules is always maintained.

Notice how, by using the above formalization, the value of **arc expressions**, often found in P/T nets, are also easily represented, by means of the AND function applied to same predicate symbol in the lhs or rhs of the rules. Moreover, it is easy to see that, if only variable symbols are present, and all variables are of the same sort, we are representing deterministic Place/Transition Nets (P/T nets).

Let us see two simple examples. The P/T net in figure 1:

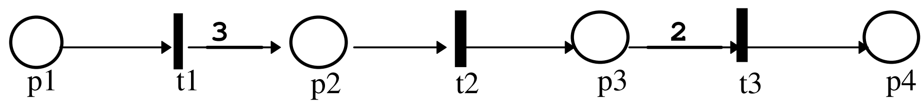

Figure 1

is formalized as follows:

```
p1(x) → AND(p2(x), AND(p2(x), p2(x)))
p2(x) → p3(x)
AND(p3(x), p3(x)) → p4(x)
```

This means that: any token can be put into place p1, and it will generate 3 tokens in place p2; and so on.
The net in figure 2 is also a P/T net:

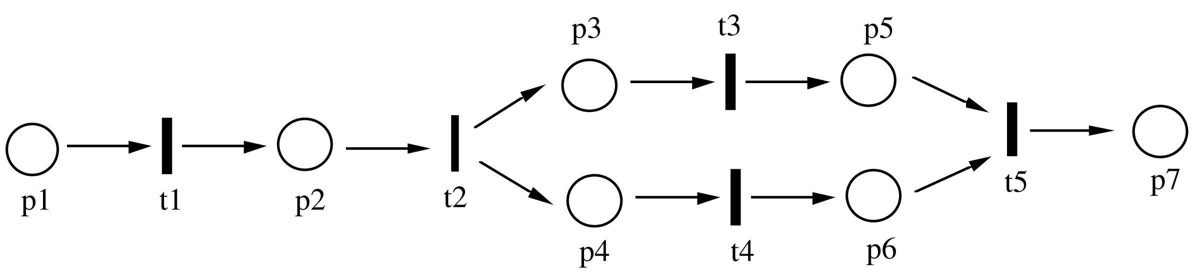

Figure 2

and will be formalized as:

```
p1(X) → p2(X)
p2(X) → AND(p3(X), p4(X))
p3(X) → p5(X)
p4(X) → p6(X)
AND(p5(X), p6(X)) → p7(X)
```

Notice how this method allows us to formalize P/T and CPN Nets in a uniform way. If for example, the net in figure 2 was a CPN, where the colour set of places p3, p4, p5 and p6 was different from the colour set of the rest of the places, then its formalization would be very similar:

```
p1(x) → p2(x)
p2(x) → AND(p3(y), p4(y))
p3(y) → p5(y)
p4(y) → p6(y)
AND(p5(y), p6(y)) → p7(x)
```

where the variable x is of a different sort than variable y.

As another example of CPN we shall consider a problem which is well known in literature as "The Dining Philosophers Problem". Five Chinese philosophers are sitting around a circular table. Each philosopher has a plate in front of him. Between any two neighbouring plates lies a single chopstick. Each philosopher alternates between thinking and eating. To eat a philosopher needs both chopsticks, the one to the right and the other to the left of his plate. It is obvious that this restriction (lack of resources) prevents two neighbours from eating at the same time. When a philosopher has finished eating, he replaces both of his forks on the table and starts thinking. The following CPN models the problem:

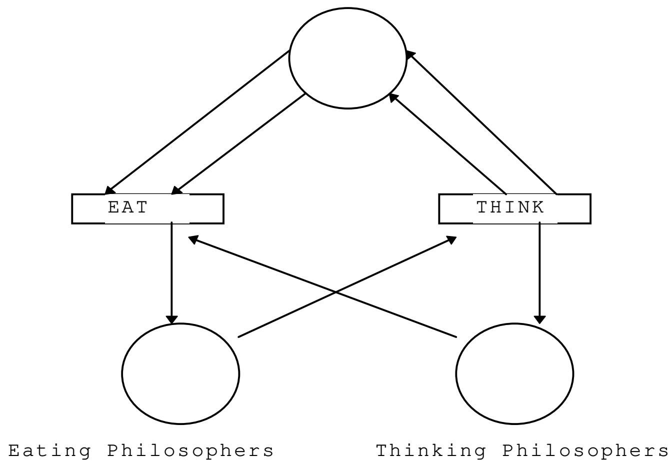

Figure 3

If philosopher i want to eat:
(a) Philosopher i must be thinking.
(b) The *i* and *i+1* chopstick must be available for the philosopher to pick up.

When the philosopher i starts eating (transition 'eat' fires) he is removed from the 'thinking' state and is added to the 'eating' state; moreover, chopstick *i* and *i+1* are removed from the set of available chopsticks.

In the same way, when philosopher *i* starts thinking (transition 'think' fires) he is removed from the 'eating' state and is added to the 'thinking' state; moreoever, chopstick *i* and *i+1* are added to the set of available chopsticks.

According to the formalization detailed above, this net can be expressed by the following TRS:

```
think(0) ^ chopstick(0) ^ chopstick(1) → eat(0)
think(1) ^ chopstick(1) ^ chopstick(2) → eat(1)
think(2) ^ chopstick(2) ^ chopstick(3) → eat(2)
think(3) ^ chopstick(3) ^ chopstick(4) → eat(3)
think(4) ^ chopstick(4) ^ chopstick(0) → eat(4)
eat(0) → think(0) ^ chopstick(0) ^ chopstick(1)
eat(1) → think(1) ^ chopstick(1) ^ chopstick(2)
eat(2) → think(2) ^ chopstick(2) ^ chopstick(3)
eat(3) → think(3) ^ chopstick(3) ^ chopstick(4)
eat(4) → think(4) ^ chopstick(4) ^ chopstick(0)
```

In the following we shall analyse how TRSs obtained in this way can be used to study the behaviour of Petri Nets. We start by looking at nets without cycles.

3. PETRI NETS WITHOUT CYCLES

In this section we verify the relations existing between completion, theorem proving and term simplification and some properties of Petri nets without cycles. For such nets, we prove the following:

Theorem: let T be a TRS which is the formalization of a Petri Net as described above. Then if the net does not contain cycles, T is canonical.

Proof**:** Let T be a TRS obtained by formalizing (as discussed above) a Petri Net which does not contain cycles. We need to prove that T is terminating and that there is no critical pair among its rules. It will follow from the (Knuth Bendix) theorem presented above that T is canonical.

The fact that there is no divergent critical pair is straightforward, following from the way we constructed T: with the exception of trivial cases, no unification of the lhs of any pair of rules is possible, and thus there can not be any divergent critical pair.

To show that T is Noetherian we must show that there exists a weight function over terms that induces a partial ordering over terms such that the terms on the lhs of each of the rewrite rules of T is greater than the one on the rhs.

Let us define a finite sequence of non-negative integers, called weights, $w_1, w_2, \ldots, w_m$ assigned to each unary function symbol $f_1, f_2, \ldots, f_m$ in our TRS.
A function $w: T(X,F) \to \mathrm{N}$ that computes the weight of a term t can be defined as follows:

$$w(t) = \sum_{j=1}^{m} w_j \cdot n(f_j, t)$$

where n is a function that computes the number of occurrences of the unary function symbol f_j in a term t and w_j is the weight assigned to f_j.
Then $\forall t_1, t_2 \in T(X,F)$ we say that $t_1 > t_2$ if $w(t_1) > w(t_2)$.

Notice that variables do not take part in the definition of the weight function, and the > relation is closed under substitution (Knuth et al., 1970).

Now we need to prove that there always exists a weight assignment to the unary function symbols such that the order of the rewrite rules is preserved.

Since the net does not contain any cycles, there must be a set of places which are postconditions of some transitions but are not preconditions of any transition. Let us call this set of places P_0.

Let $l_{i,t}$ be the number of transitions that we encounter traversing the net backward with respect to the direction of the arrows, starting from the element i of P_0 up to the transition t including it. Let $l_{i,t} = 0$ when there is not such a path from i to t.
For each transition t of our net let us assign a positive integer I_t such that $I_t = \max(l_{i,t})$ where i ranges over the elements of P_0 .
For each place p of our net let us assign a positive integer I_p such that $I_p = \max(I_t)$ where t ranges over the set of the transitions of which I_p is a precondition.

Let us call P_i the set of places having index $I_p = i$.

Then let us assign a weight to the unary function symbols of our TRS as follows: we assign a positive integer W_0 as the weight of the function symbols representing elements of P_0, and then we assign to the function symbols representing elements of P_j a weight W_{P_j} such that: $W_{P_j} = \left(W_{P_{j-1}} \cdot N_{P_{j-1}}\right) + k$ where $N_{P_{j-1}}$ is the number of elements of the set P_{j-1}, and k is a positive constant.

With this assignment the places which are pre-conditions of any transition are assigned a weight which is greater than the one assigned to the places which are the post-conditions of that transition; therefore each rewrite rule of our TRS will be properly oriented and the TRS is terminating.
□

As we have seen, a complete TRS has the property that it can simplify any term into its canonical form. The equational theory induces an equivalency relation over terms which divides them into equivalency classes, and a canonical TRS is sufficient to transform any term into the canonical representative of its class.

In our case, the fact that the TRS we generated is complete, means that it can simplify every possible initial state into a state in which there are no more fireable transition. Because of the choice over the weights on terms, such states are the canonical representatives of every class, being the states of the net with minimum weight. In other words it means that from any initial state, with a finite number of transition firings, we obtain a unique state from which it is impossible to proceed further - i.e. no more transition are fireable. This means that the net goes into a deadlock state for any initial state. Notice that this makes sense, since a Petri Net not containing any cycles goes into a deadlock state for every initial marking. This happens with the nets in figures 1 and 2.

Let us now see what are the meanings of the operations of term simplification and theorem proving when applied to a TRS which is the formalization of a Petri Net without cycles, from the Petri Net "point of view". Then we shall analyse the meaning of *mostly simplifying* the TRS.

Term Simplification

Simplifying a term with respect to our TRS is equivalent to verifying how the net evolves from that particular initial marking. The termination is assured by the property of Noetherianity of the system.

For example, reducing the term p1(x) with respect to the TRS of the net in fig.1 until no rewrite rule further applies, yields:

```
p3(x) ^ p4(x)
```

In this simple example it is easy to check, simulating the net "by hand", that if we put a token in p1 and let all the possible transitions fire, the net will get to the marking `p3(x)^p4(x)`, and then no other transition will be allowed to fire.

For the net in fig.2, given different initial states - we obtain the values shown in table 1:

TERM TO SIMPLIFY	RESULT OF TERM SIMPLIFICATION
`p1(x)`	`p7(x)`
`p2(x)`	`p7(x)`
`p3(x)`	`p5(x)`
`AND(p2(x), p3(x))`	`AND(p5(x), p7(x))`
`AND(p5(x), p4(x))`	`p7(x)`

Table 1

Notice how, strictly speaking, for P/T nets it only makes sense to simplify terms containing a unique variable and no constant symbols - in such nets only the presence or absence of a token is of importance, the tokens have no individual values, they are "generic".

Also notice, looking at each reduction step in the simplification process allows us to monitor the evolution of the net as the tokens "travel" through it (this was easy to implement in LP, see the appendix). For example, for the net in fig.2, we can see that by putting a token into place p1, the net goes through the following states:

```
p1(x)
p2(x)
AND(p3(x), p4(x))
AND(p5(x), p4(x))
AND(p5(x), p6(x))
p7(x)
```

The great advantage of this method lies in the efficiency of computing the evolution of states of a Petri Net: the TRS can reduce any initial state into a final state by a finite number of applications of the term reduction.

Theorem Proving

When proving a theorem, our implementation in LP simplifies both the lhs and rhs of the theorem to their canonical form for the TRS and checks if these coincide. *Using our formalization, proving theorems means verifying if from two different initial markings we can obtain the same deadlock state. The two terms which constitute a theorem will be the two markings we want to verify.*

In the net in fig. 1, let us consider two different initial markings: the first constituted by a single token in p1, and the second consisting of 3 tokens, two of which in p2 and the third in p3. To check the conjecture that both markings lead to the same deadlock state, we insert the theorem:

```
p1(x) = p2(x) ^ p2(x) ^ p3(x)
```

LP proves the theorem, therefore our conjecture was true. In the same way, we can see that

```
p3(x) = p1(x)
```

is false.
In the net of fig.2, LP proves the theorem:

```
p1(x) = p2(x)
```

while the theorem:

```
p3(x) = p1(x)
```

is false.

Simplifying the TRS

One of the most attractive feature of this formalization is that it allows a natural way of simplifying Petri Nets. In other words, there is an easy way to transform any given Petri Net into another one which is simpler, but for which the operations of term simplification and theorem proving will yield the same results as before.

Let us say that a *TRS is mostly simplified* when the terms on the lhs and rhs of each of the rewrite rules are in normal form with respect to the rest of the TRS.

The TRS we obtain from formalizing a Petri Net as discussed above is complete, but it is not necessarily mostly simplified. To mostly simplify a TRS T, for every rewrite rule $l \rightarrow r$ we need to simplify l and r with respect to (T-{$l \to r$}). Let us call l_0, r_0 the terms resulting from this process. Then, denoting with $w(t)$ the weight of a term t according to the definition of weight function discussed above:

if $l_0 = r_0$, remove $l \to r$ from T
if $w(l_0) > w(r_0)$ substitute $l \to r$ with $l_0 \to r_0$
if $w(l_0) < w(r_0)$ substitute $l \to r$ with $r_0 \to l_0$

We can use this procedure to simplify the TRS of the net in fig.1; we obtain:

```
p1(x) → AND(p4(x), p3(x))
p2(x) → p3(x)
AND(p3(x), p3(x)) → p4(x)
```

The simplified TRS clearly expresses the fact that for this net starting from any initial state we shall always obtain a token in p4 after all possible transitions are fired. Simplifying the TRS of fig.2 we obtain:

```
p3(x) → p5(x)
p4(x) → p6(x)
AND(p5(x), p6(x)) → p7(x)
p2(x) → p7(x)
p1(x) → p7(x)
```

Using these simplified TRS, the operations of term simplification and theorem proving will yield the same results as with the original TRS, but it will obviously take fewer reduction steps. Therefore, the results previously shown for the two examples above still hold true.

As we can see from the above examples, the importance of the operation just described lies in the fact that it causes some knowledge about the structure of the net that is implicitly contained in the input TRS, to become explicit. *The simplification of the TRS gives us a new TRS which most efficiently expresses the movements of tokens in the net when transitions are fired. As a matter of fact the net is transformed into a simpler one where, allowing all fireable transition to fire, from any initial state we get to the same final state we would reach in the original net.*

4. PETRI NETS WITH CYCLES

Up to now, our examples have only dealt with "linear" nets, which did not contain cycles. The problem in formalizing nets with cycles lies in the fact that we might need to use rewrite rules where the state on the lhs would have lower weight than the ones on the rhs of the rule. In other words, there may be no adequate ordering over terms that preserves the termination property of the TRS. Therefore, the process of Term Reduction and Theorem Proving is not assured to end for every initial state anymore. Notice,however, that this is exactly what happens in a net that contains cycles: it may possibly enter into an infinite loop i.e. the tokens in the net might be allowed to "loop" indefinitely. Therefore, in those cases in which term simplification with respect to the TRS terminates, it will give us the final state of the theory, while in those cases for which the simplification procedure does not terminate, it will mean that the net went into an infinite loop (our LP implementations will detect this fact and break out of the loop, see the appendix).

Clearly, this does not happen every time that a net has cycles. Figure 4 contains a modification of the net in figure 2, such that it contains a cycle, but for which an adequate ordering over terms that preserves the termination property of the TRS, does exist; in other words, no token is allowed to "loop" indefinitely.

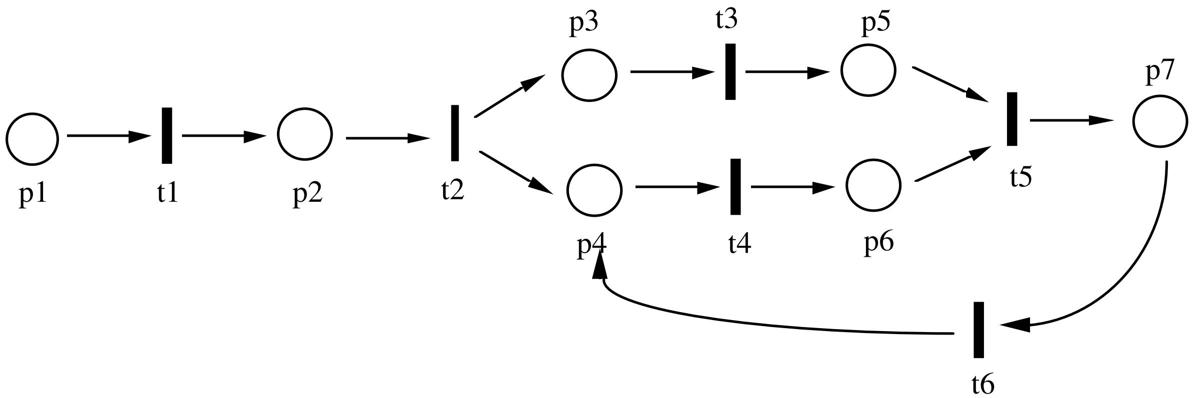

Figure 4

The term reduction of some initial markings is presented in table 2:

TERM TO SIMPLIFY	RESULT OF TERM SIMPLIFICATION
`p1(x)`	`p6(x)`
`p3(x)`	`p5(x)`
`AND(p5(x),p4(x))`	`p6(x)`
`AND(p1(x),p2(x))`	`AND(p6(x), p6(x))`

Table 2

Mostly simplifying the TRS we obtain:

```
p1(x) → p6(x)
p2(x) → p6(x)
p3(x) → p5(x)
p4(x) → p6(x)
AND(p5(x), p6(x)) → p6(x)
p7(x) → p6(x)
```

It is easy to modify the net of figure 4 so that no adequate ordering over terms will exist and therefore the TRS will be non terminating; now tokens are allowed to "loop" indefinitely.

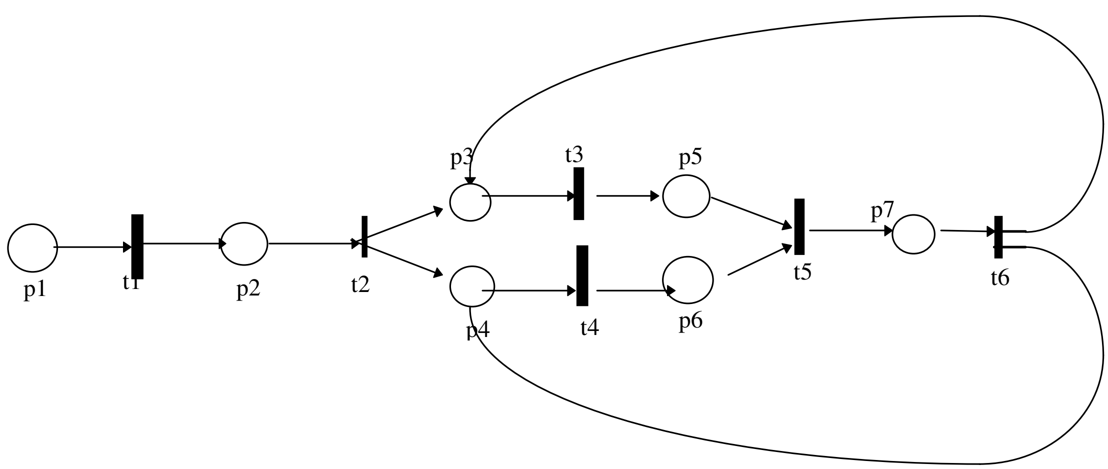

Figure 5

Applying term simplification to some initial markings for the net in fig.5, we obtain (table 3):

TERM TO SIMPLIFY	RESULT OF TERM SIMPLIFICATION
`p3(x)`	`p5(x)`
`p4(x)`	`p6(x)`
`p2(x)`	`does not terminate`
`and(p4(x),p5(x))`	`does not terminate`

Table 3

For nets containing infinite loops, the simplification of the TRS is the most problematic operation: we would be interested in simplifying the net, since we could obtain the TRS that most efficiently expresses the movements of tokens in the net when transitions are fired; on the other hand, problems arise when trying to obtain a mostly simplified TRS for such nets, since the simplification process for the TRS shall not end. The problem can be solved by introducing some extra temporary places and having some transitions pointing to them, in such a way as to disrupt the loops. For example, in the net in fig. 5, we can introduce a temporary place `peight` and have transition t6 pointing to it rather than to the higher weight places p3 and p4.

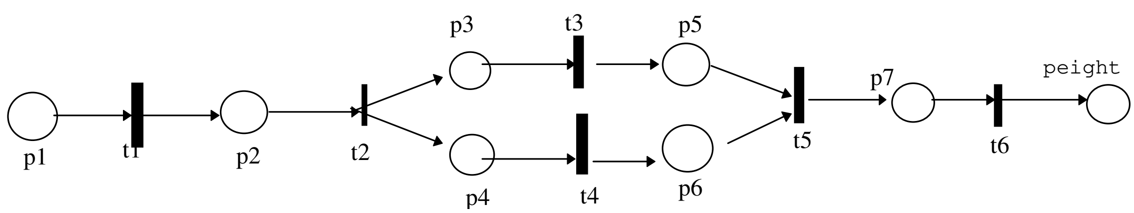

Figure 6

The equivalent TRS would now be:

```
p1(x) → p2(x)
p2(x) → and(p3(x), p4(x))
p3(x) → p5(x)
p4(x) → p6(x)
and(p5(x), p6(x)) → p7(x)
p7(x) → peight(x)
```

After having eliminated loops from the net, we can use the simplification procedure, which will now terminate, since the above TRS is canonical. Thus we obtain:

```
p1(x) → peight(x)
p2(x) → peight(x)
```

```
p3(x) → p5(x)
p4(x) → p6(x)
AND(p5(x), p6(x) → peight(x)
p7(x) → peight(x)
```

Now that the theory has been simplified we can modify the TRS, substituting our temporary places with the places where our transitions were actually pointing to in our original net. In the example of figure 5, `peight(x)` must be substituted with `AND(p3(x), p4(x))`, thus obtaining the following TRS:

```
p1(x) → and(p3(x), p4(x))
p2(x) → and(p3(x), p4(x))
p3(x) → p5(x)
p4(x) → p6(x)
AND(p5(x), p6(x) → and(p3(x), p4(x))
p7(x) → and(p3(x), p4(x))
```

This TRS represents the original net of figure 5. As before, simplification has been very useful in order to obtain a more efficient TRS. Again, using these simplified TRS any operation of Term Reduction and Theorem Proving will yield the same results as with the original TRS. Therefore, Tables 3 still hold true.

Now let us go back to the dining philosophers problem. As expected, starting from any legal initial marking, the TRS presented in section 2 will go into an infinite loop. Let us eliminate the loop, by introducing a temporary place called thinkTemp, thus obtaining the new TRS:

```
think(0) ^ chopstick(0) ^ chopstick(1) → eat(0)
think(1) ^ chopstick(1) ^ chopstick(2) → eat(1)
think(2) ^ chopstick(2) ^ chopstick(3) → eat(2)
think(3) ^ chopstick(3) ^ chopstick(4) → eat(3)
think(4) ^ chopstick(4) ^ chopstick(0) → eat(4)
eat(0) → thinkTemp(0) ^ chopstick(0) ^ chopstick(1)
eat(1) → thinkTemp(1) ^ chopstick(1) ^ chopstick(2)
eat(2) → thinkTemp(2) ^ chopstick(2) ^ chopstick(3)
eat(3) → thinkTemp(3) ^ chopstick(3) ^ chopstick(4)
eat(4) → thinkTemp(4) ^ chopstick(4) ^ chopstick(0)
```

Now the simplification procedure will clearly terminate for any initial marking. For example, the term:

```
eat(0)^think(1)^think(1)^think(1)^think(1)^chopstick(2)^chopstick(2)^
^chopstick(2)
```

will simplify to:

```
chopstick(0)^chopstick(1)^chopstick(2)^chopstick(3)^chopstick(4)^thinkTe
mp(0) ^thinkTemp(1)^thinkTemp(2)^thinkTemp(3)^thinkTemp(4)
```

and the term:

```
eat(0)^eat(3)
```

will simplify to:

```
chopstick(0)^chopstick(1)^chopstick(3)^chopstick(4)^thinkTemp(0)^thinkTe
mp(3)
```

(these and other examples can be found in greater detail in the appendix).

Now we can proceed, and simplify the TRS. We obtain:

```
chopstick(0)^chopstick(1)^think(0) → chopstick(0)^chopstick(1)^thinkTemp(0)
chopstick(1)^chopstick(2)^think(1) → chopstick(1)^chopstick(2)^thinkTemp(1)
chopstick(2)^chopstick(3)^think(2) → chopstick(2)^chopstick(3)^thinkTemp(2)
chopstick(3)^chopstick(4)^think(3) → chopstick(3)^chopstick(4)^thinkTemp(3)
chopstick(0)^chopstick(4)^think(4) → chopstick(0)^chopstick(4)^thinkTemp(4)
eat(0) → chopstick(0)^chopstick(1)^thinkTemp(0)
eat(1) → chopstick(1)^chopstick(2)^thinkTemp(1)
eat(2) → chopstick(2)^chopstick(3)^thinkTemp(2)
eat(3) → chopstick(3)^chopstick(4)^thinkTemp(3)
eat(4) → chopstick(0)^chopstick(4)^thinkTemp(4)
```

And then, finally replacing our temporary place `thinkTemp` with `think`, as in the original net, we obtain:

```
chopstick(0)^chopstick(1)^think(0) → chopstick(0)^chopstick(1)^think(0)
chopstick(1)^chopstick(2)^think(1) → chopstick(1)^chopstick(2)^think(1)
chopstick(2)^chopstick(3)^think(2) → chopstick(2)^chopstick(3)^think(2)
chopstick(3)^chopstick(4)^think(3) → chopstick(3)^chopstick(4)^think(3)
chopstick(0)^chopstick(4)^think(4) → chopstick(0)^chopstick(4)^think(4)
eat(0) → chopstick(0)^chopstick(1)^think(0)
eat(1) → chopstick(1)^chopstick(2)^think(1)
eat(2) → chopstick(2)^chopstick(3)^think(2)
eat(3) → chopstick(3)^chopstick(4)^think(3)
eat(4) → chopstick(0)^chopstick(4)^think(4)
```

Notice how this process has extracted knowledge from the network, highlighting the fact that there exists a loop (see the first 5 rewrite rules). This net is completely different from the one we started with - here the philosophers, maybe after an iteration in which they are eating, will be thinking forever - but this system best expresses the fact that, for any initial marking, the tokens will enter an infinite loop.

5. Discussion

This paper proposes a way to formalize deterministic Petri Nets as Term Rewriting Systems. This allows the use of Rewriting Techniques to study the behaviour of Petri Nets.

Here we did not indulge in all the possible and slightly different definitions and nuances of Petri Nets found in the literature. Rather we presented quite general ideas, that can be easily extended to specific kinds of nets, as needed. For example, the implementation of concepts such as the capacity of the places, though not treated in this paper, would be very easy to implement, since it follows directly form the ideas explained here.

This formalization leads to the straightforward implementation of rewriting based tools to simulate the behaviour of Petri Nets. Heuristically driven ad hoc procedures could be added to detect properties of the net such as loops, liveness, and the ability to graphically represent the evolution of the net.

The nets could be simulated using both the original TRS that comes out from the formalization described, or its simplified version. The first one could be used for simulating the net step by step, when there is a need for monitoring the firing of the transitions through time. Although having the same power of the original TRS, the simplified version could be more efficient in finding the deadlock state reached from a particular initial state, or to detect a loop in the net.

References

L.Bachmair, N.Dershowitz, J.Hsiang, 1986, Orderings for equational proofs, in Proceedings 1st Annual IEEE Symp. on Logic in Computer Science, Cambridge, MA, pp. 346-357.

G. Birkhoff. On the Structure of Abstract Algebras, *Proc. Cambridge Philos. Soc.*, 31 (1935), 433-454.

B.Buchberger, R. Loos, 1983, Algebraic simplification, in Computer Algebra: Symbolic and Algebraic Computation, Springer Verlag, Wien.

N.Dershowitz, 1982, Orderings for term rewriting systems, *J. of Theoretical Computer Science* 17, pp. 279-301.

N.Dershowitz, 1989, Completion and its applications, in Resolution of Equations in Algebraic Structures, pp. 31-48.

N. Dershowitz and Jean-Pierre Jouannad, Rewrite Systems in Handbook of Theoretical Computer Science, Ed.by J. vanLeeuwen, Elsevier Science Publishers B.V., 1990.

Guttag, J.V. and J.J. Horning. Larch: Languages and Tools for Formal Specification, January, 1993

J.Hsiang, M. Rusinovitch, 1987, On word problems in equational theories, Proceedings 14th Int. Conf. on Automata Languages and Programming, Karlsrhue.

J.M.Hullot, 1980, A catalogue of canonical term rewriting systems, Rept. CSL-113, SRI International, Menlo Park, CA.

D.E.Knuth, P.Bendix, 1970, Simple word problems in universal algebras", Proceedings of the Conf. on Computational Problems in Abstract Algebras, Pergamon Press, pp. 263-298.

K. Jensen, 1994, Coloured Petri Nets, Second Edition, Springer-Verlag, Berlin

J.L.Peterson, 1981, Petri Net Theory and the Modeling of Systems, Prentice Hall, Englewood Cliffs, N.J.

W.Reising, 1985, Petri Nets: An Introduction, Springer-Verlag, Berlin-New York

http://larch-www.lcs.mit.edu:8001/larch/LP/overview.html, "LP, the Larch Prover"