Coverability of Reset Petri Nets and other Well-Structured Transition Systems by Partial Deduction

Michael Leuschel and Helko Lehmann

Department of Electronics and Computer Science
University of Southampton
Highfield, Southampton, SO17 1BJ, UK
{mal,hel99r}@ecs.soton.ac.uk
www: http://www.ecs.soton.ac.uk/~mal

Abstract. In recent work it has been shown that infinite state model checking can be performed by a combination of partial deduction of logic programs and abstract interpretation. It has also been shown that partial deduction is powerful enough to mimic certain algorithms to decide coverability properties of Petri nets. These algorithms are *forward* algorithms and hard to scale up to deal with more complicated systems. Recently, it has been proposed to use a *backward* algorithm scheme instead. This scheme is applicable to so–called well–structured transition systems and was successfully used, e.g., to solve coverability problems for reset Petri nets. In this paper, we discuss how partial deduction can mimic many of these backward algorithms as well. We prove this link in particular for reset Petri nets and Petri nets with transfer and doubling arcs. We thus establish a surprising link between algorithms in Petri net theory and program specialisation, and also shed light on the power of using logic program specialisation for infinite state model checking.

1 Introduction

Recently there has been interest in applying logic programming techniques to model checking. Table-based logic programming and set-based analysis can be used as an efficient means of performing explicit model checking [29][4]. Despite the success of model checking, most systems must still be substantially simplified and considerable human ingenuity is required to arrive at the stage where the push button automation can be applied [31]. Furthermore, most *software* systems cannot be modelled directly by a *finite* state system. For these reasons, there has recently been considerable interest in *infinite model checking*. This, by its very undecidable nature, is a daunting task, for which *abstraction* is a key issue.

Now, an important question when attempting infinite model checking in practice is: How can one *automatically* obtain an abstraction which is finite, but still as precise as required? A solution to this problem can be obtained by using existing techniques for the *automatic* control of *logic program specialisation* [19]. More precisely, in program specialisation and partial evaluation, one faces a very

similar (and extensively studied) problem: To be able to produce efficient specialised programs, *infinite* computation trees have to be abstracted in a *finite* but also as *precise* as possible way. To be able to apply this existing technology we simply have to model the system to be verified as a logic program (by means of an interpreter). This obviously includes finite LTS, but also allows to express infinite state systems. This translation is often very straightforward, due to the built-in support of logic programming for non-determinism and unification. First successful steps in that direction have been taken in [12, 24]. [22] gave a first formal answer about the power of the approach and showed that when we encode ordinary Petri nets as logic programs and use existing program specialisation algorithms, we can decide the so-called "coverability problems" (which encompass quasi-liveness, boundedness, determinism, regularity,...). This was achieved by showing that the Petri net algorithms by Karp–Miller [15] and Finkel [8] can be exactly mimicked. Both algorithms are *forward* algorithms, i.e. they construct an abstracted representation of the whole reachability tree of a Petri net starting from the initial marking. However, to decide many coverability problems, such a complete abstraction is not necessary or even not precise enough for more complicated systems. To decide coverability problems for a wider class of transition systems, namely well structured transition systems, in [1, 9, 10] a backward algorithm scheme was proposed instead. This scheme has been successfully applied, e.g., to *reset Petri nets*.

In this paper we discuss how partial deduction can mimic these backward algorithms as well. We prove this correspondence in particular for reset Petri nets, since for many problems they lie on the "border between decidability and undecidability" [6]. Thus, in addition to establishing a link between algorithms in Petri net theory and program specialisation, our results also shed light on the power of using logic program analysis and specialisation techniques for infinite state model checking.

2 (Reset) Petri Nets and the Covering Problem

In this paper we want to study the power of partial deduction based approaches for model checking of infinite state systems. To arrive at precise results, it makes sense to focus on a particular class of infinite state systems and properties which are known to be decidable. One can then examine whether the partial deduction approach provides a decision procedure and how it compares to existing algorithms. In this section, we describe such a decidable class of properties and systems, namely covering problems for Petri nets, reset Petri nets, and well-structured transition systems. We start out by giving definitions of some important concepts in Petri net theory [30].

Definition 1. *A* Petri net Π *is a tuple* (S, T, F, M_0) *consisting of a finite set of places* S*, a finite set of transitions* T *with* $S \cap T = \emptyset$ *and a flow relation* F *which is a function from* $(S \times T) \cup (T \times S)$ *to* $\mathbb{N}$*. A* marking M *for* Π *is a mapping* $S \to \mathbb{N}$*.* M_0 *is a marking called* initial*.*

A transition $t \in T$ is enabled *in a marking M iff $\forall s \in S : M(s) \geq F(s,t)$. An enabled transition can be fired, resulting in a new marking M' defined by $\forall s \in S : M'(s) = M(s) - F(s,t) + F(t,s)$. We will denote this by $M[t\rangle M'$. By $M[t_1, \ldots, t_k\rangle M'$ we denote the fact that for some intermediate markings $M_1, \ldots, M_{k-1}$ we have $M[t_1\rangle M_1$, ..., $M_{k-1}[t_k\rangle M'$.*

We define the reachability tree $RT(\Pi)$ inductively as follows: Let M_0 be the label of the root node. For every node n of $RT(\Pi)$ labelled by some marking M and for every transition t which is enabled in M, add a node n' labelled M' such that $M[t\rangle M'$ and add an arc from n to n' labelled t. The set of all labels of $RT(\Pi)$ is called the reachability set *of Π, denoted $RS(\Pi)$. The set of words given by the labels of finite paths of $RT(\Pi)$ starting in the root node is called* language *of Π, written $L(\Pi)$.*

For convenience, we denote $M \geq M'$ iff $M(s) \geq M'(s)$ for all places $s \in S$. We also introduce *pseudo-markings*, which are functions from S to $\mathbb{N} \cup \{\omega\}$ where we also define $\forall n \in \mathbb{N} : \omega > n$ and $\omega + n = \omega - n = \omega + \omega = \omega$. Using this we also extend the notation $M_{k-1}[t_1, \ldots, t_k\rangle M'$ for such markings.

Reset Petri Nets and WSTS's One can extend the power of Petri nets by adding a set of *reset arcs* $R \subseteq (S \times T)$ from places to transitions: when the associated transition fires the number of tokens in the originating place is reset to zero. Such nets were first introduced in [2], and we adapt all of the above concepts and notations in the obvious way.

Well-structured transition systems (WSTS) [9, 10] are a further generalisation of Petri nets. They cover reset Petri nets but also Petri nets with transfer arcs, post self-modifying nets, as well as many formalisms not directly related to Petri nets (Basic Process Algebras, Context-free grammars, Timed Automata,...). To define WSTS we first need the concept of a well-quasi order:

Definition 2. *A sequence $s_1, s_2, \ldots$ of elements of S is called* admissible wrt a binary relation $\leq_S$ *on $S \times S$ iff there are no $i < j$ such that $s_i \leq_S s_j$. We say that $\leq_S$ is a* well-quasi relation *(wqr) iff there are no infinite admissible sequences wrt $\leq_S$. A* well quasi order *(wqo) is a reflexive and transitive wqr.*

A well-structured transition system (WSTS) [9, 10] is a structure $\langle S, \rightarrow, \leq\rangle$ where S is a (possibly infinite) set of states, $\rightarrow \subseteq S \times S$ a set of transitions, and:
(1) $\leq \subseteq S \times S$ is a wqo and
(2) $\leq$ is (upward) compatible wrt $\rightarrow$: for all $s_1 \leq t_1$ and $s_1 \rightarrow s_2$ there exists a sequence $t_1 \rightarrow^* t_2$ such that $s_2 \leq t_2$.

Reset Petri nets can be modelled as a WSTS $\langle S, \rightarrow, \leq\rangle$ with S being the set of markings, $M \rightarrow M'$ if for some t we have $M[t\rangle M'$ and using the corresponding $\leq$ order on markings seen as vectors of numbers (this order is a wqo).

Coverability Analysis The *covering problem* is a classical problem in Petri net theory and is also sometimes referred to as the *control-state reachability problem*. The question is: given a marking M is there a marking M' in $RS(\Pi)$

which covers M, i.e., $M' \geq M$. This problem can be analysed using the so-called Karp-Miller-tree $KM(\Pi)$ [15], which is computed as follows:
1.start out from a tree with a single node labelled by the initial marking M_0; **2.** repeatedly pick an unprocessed leaf labelled by some M; for every transition t such that $M[t\rangle M'$ and such that there is no ancestor $M'' = M'$ do: **a.** generalise M' by replacing all $M'(p)$ by ω such that there is an ancestor $M'' < M'$ and $M''(p) < M'(p)$ **b.** create a child of M labelled by M'.

The intuition behind step 2a. is that if from M'' we can reach the strictly larger marking M' we then extrapolate the growth by inserting ω's. For example for $M'' = \langle 0, 1, 1\rangle$ and $M' = \langle 1, 2, 1\rangle$ we will produce $\langle \omega, \omega, 1\rangle$. This is sufficient to ensure termination of the procedure and thus finiteness of $KM(\Pi)$.

Some of the properties of ordinary Petri nets decidable by examining $KM(\Pi)$ are:[1] boundedness, place-boundedness, quasi-liveness of a transition t (i.e. is there a marking in $RT(\Pi)$ where t is enabled), and regularity of $L(\Pi)$ (cf. [8],[15], [35]).

The quasi-liveness question is a particular instance of the covering problem, which can be decided using the Karp-Miller tree simply by checking whether there is a pseudo-marking M' in $KM(\Pi)$ such that $M' \geq M$. For example if there is a marking $\langle \omega, \omega, 1\rangle$ in $KM(\Pi)$ then we know that we can, e.g., reach a marking greater or equal to $\langle 10, 55, 1\rangle$.

The reason why this approach is correct is the monotonicity of ordinary Petri nets: if $M[t_1, \ldots, t_k\rangle M'$ and $M'' > M$ (the condition to introduce ω) then $M''[t_1, \ldots, t_k\rangle M'''$ for some $M''' > M'$ (i.e., we can repeat the process and produce ever larger markings and when an ω is generated within $KM(\Pi)$ for a particular place s we can generate an arbitrarily large number of tokens in s[2]).

Unfortunately, this monotonicity criterion is no longer satisfied for Petri nets with reset arcs! More precisely, when we have $M[t_1, \ldots, t_k\rangle M'$ with $M' > M$ (the condition to introduce ω) we still have that $M'[t_1, \ldots, t_k\rangle M''$ for some M'' but we no longer have $M'' > M'$ (we just have $M'' \geq M'$). This means that, when computing the Karp-Miller tree, the generation of ω places is sound but no longer "precise," i.e., when we generate an ω we are no longer guaranteed that an unbounded number of tokens can actually be produced. The Karp-Miller tree can thus no longer be used to decide boundedness or coverability.

Example 1. Take for example a simple reset Petri net with two transitions t_1, t_2 and two places s_1, s_2 depicted in Fig. 1. Transition t_1 takes one token in s_1 and putting one token in s_2 and resetting s_1. Transition t_2 takes one token from s_2 and producing 2 tokens in s_1. Then we have $\langle 1, 0\rangle$ $[t_1\rangle$ $\langle 0, 1\rangle$ $[t_2\rangle$ $\langle 2, 0\rangle$ and the Karp-Miller procedure generates a node labelled with $\langle \omega, 0\rangle$ even though the net is bounded!

[1] It was shown in [8] that these problems can also be decided using minimal coverability graphs, which are often significantly smaller.

[2] However, it does not guarantee that we can generate *any* number of tokens. To decide whether we can, e.g., reach *exactly* the marking $\langle 10, 55, 1\rangle$ the Karp-Miller tree is not enough.

Fig. 1. Reset Petri from Ex. 1

It turns out that boundedness (as well as reachability) is actually undecidable for Petri nets with reset arcs [6]. However, the covering problem (and thus, e.g., quasi-liveness) is still decidable using a backwards algorithm [1, 9, 10], which works for any WSTS for which $\leq$ and $pb(.)$ (see below) can be computed.

Given a WSTS $\langle S, \rightarrow, \leq \rangle$ and a set of states $I \in S$ we define:

- the upwards-closure $\uparrow I = \{y \mid y \geq x \wedge x \in I\}$
- the immediate predecessor states of I: $Pred(I) = \{y \mid y \rightarrow x \ \wedge \ x \in I\}$
- all predecessor states of I, $Pred^*(I) = \{y \mid y \rightarrow^* x \wedge x \in I\}$
- $pb(I) = \bigcup_{x \in I} pb(x)$ where $pb(x)$ is a finite basis of $\uparrow Pred(\uparrow \{x\})$ (i.e., $pb(x)$ is a finite set such that $\uparrow pb(x) = \uparrow Pred(\uparrow \{x\})$).

The *covering problem* for WSTS is as follows: given two states s and t can we reach $t' \geq t$ starting from s. Provided that $\leq$ is decidable and $pb(x)$ exists and can be effectively computed, the following algorithm [1, 9, 10] can be used to decide the covering problem:

1. Set $K_0 = \{t\}$ and $j = 0$
2. $K_{j+1} = K_j \cup pb(K_j)$
3. if $\uparrow K_{j+1} \neq \uparrow K_j$ then increment j and goto 2.
4. return true if $\exists s' \in K_j$ with $s' \leq s$ and false otherwise

This procedure terminates and we also have the property that $\uparrow \bigcup_i K_i = Pred^*(\uparrow \{t\})$ [10]. At step 4. we test whether $s \in \uparrow K_j$, which thus corresponds to $s \in Pred^*(\uparrow \{t\})$ (because we have reached the fixpoint), i.e., we indeed check whether $s \rightarrow^* t'$ for some $t' \geq t$.

$pb(M)$ can be effectively computed for Petri nets and reset Petri nets by simply executing the transitions backwards and setting a place to the minimum number of tokens required to fire the transition if it caused a reset on this place:

$$pb(M) = \{\ P' \mid \exists t \in T :\ (\ P[t\rangle M \wedge \\ \forall s \in S :\ P'(s) = (F(s,t) \text{ if } (s,t) \in R \text{ else } P(s))\ )\ \}$$

We can thus use the above algorithm to decide the covering problem. In the remainder of the paper we will show that, surprisingly, the exact same result can be obtained by encoding the (reset) Petri net as a logic program and applying a well-established program specialisation technique!

3 Partial Evaluation and Partial Deduction

We will now present the essential ingredients of the logic program specialisation techniques that were used for infinite model checking in [12, 24].

Throughout this article, we suppose familiarity with basic notions in logic programming. Notational conventions are standard. In particular, we denote variables through (strings starting with) an uppercase symbol, while constants, functions, and predicates begin with a lowercase character.

In logic programming full input to a program P consists of a goal $\leftarrow Q$ and evaluation corresponds to constructing a complete SLD-tree for $P \cup \{\leftarrow Q\}$, i.e., a tree whose root is labeled by $\leftarrow Q$ and where children of nodes are obtained by first selecting a literal of the node and then resolving it with the clauses of P. For partial evaluation, static input takes the form of a *partially instantiated* goal $\leftarrow Q'$ and the specialised program should be correct for all runtime instances $\leftarrow Q'\theta$ of $\leftarrow Q'$. A technique which achieves this is known under the name of *partial deduction*, which we present below.

3.1 Generic Algorithm for Partial Deduction

The general idea of partial deduction is to construct a finite number of finite but possibly incomplete[3] trees which "cover" the possibly infinite SLD-tree for $P \cup \{\leftarrow Q'\}$ (and thus also all SLD-trees for all instances of $\leftarrow Q'$). The derivation steps in these SLD-trees are the computations which have been pre-evaluated and the clauses of the specialised program are then extracted by constructing one specialised clause (called a *resultant*) per branch. These incomplete SLD-trees are obtained by applying an unfolding rule:

Definition 3. *An* unfolding rule *is a function which, given a program P and a goal $\leftarrow Q$, returns a non-trivial[4] and possibly incomplete SLD-tree τ for $P \cup \{\leftarrow Q\}$. We also define the set of leaves, leaves(τ), to be the leaf goals of τ.*

Given *closedness* (all leaves are an instance of a specialised atom) and *independence* (no two specialised atoms have a common instance), correctness of the specialised program is guaranteed [25]. Independence is usually (e.g. [23, 5]) ensured by a *renaming* transformation. Closedness is more difficult to ensure, but can be satisfied using the following generic algorithm based upon [27, 23]. This algorithm structures the atoms to be specialised in a *global tree*: i.e., a tree whose nodes are labeled by atoms and where A is a descendant of B if specialising B lead to the specialisation of A. Apart from the missing treatment of conjunctions [5] the following is basically the algorithm implemented in the ECCE system [23, 5] which we will employ later on.

Algorithm 3.1 (*generic partial deduction algorithm*)

[3] An *incomplete* SLD-tree is a SLD-tree which, in addition to success and failure leaves, also contains leaves where no literal has been selected for a further derivation step.

[4] A trivial SLD-tree has a single node where no literal has been selected for resolution.

Input: a program P and a goal $\leftarrow A$
Output: a set of atoms or conjunctions $\mathcal{A}$ and a global tree γ
Initialisation: $\gamma :=$ a "global" tree with a single unmarked node, labelled by A
repeat
  pick an unmarked leaf node L in γ
  if $covered(L, \gamma)$ **then** mark L as processed
  else
    $W = whistle(L, \gamma)$
    if $W \neq fail$ **then**
      $label(L) := abstract(L, W, \gamma)$[5]
    else
      mark L as processed
      for all atoms $A \in leaves(U(P, label(L)))$ **do**
        add a new unmarked child C of L to γ
        $label(C) := A$
until all nodes are processed
output $\mathcal{A} := \{label(A) \mid A \in \gamma\}$ and γ

The above algorithm is parametrised by an unfolding rule U, a predicate $covered(L, \gamma)$, a whistle function $whistle(L, \gamma)$ and an abstraction function $abstract(L, W, \gamma)$. Intuitively, $covered(L, \gamma)$ is a way of checking whether L or a generalisation of L has already been treated in the global tree γ. Formally, $covered(L, \gamma) = true$ must imply that $\exists M \in \gamma$ such that M is processed or abstracted and for some substitution θ: $label(M)\theta = label(L)$. A particular implementation could be more demanding and, e.g., return true only if there is another node in γ labelled by a variant of L.

The other two parameters are used to ensure termination. Intuitively, the $whistle(L, \gamma)$ is used to detect whether the branch of γ ending in L is "dangerous", in which case it returns a value different from $fail$ (i.e., it "blows"). This value should be an ancestor W of L compared to which L looked dangerous (e.g., L is bigger than W in some sense). The abstraction operation will then compute a generalisation of L and W, less likely to lead to non-termination. Formally, $abstract(L, W, \gamma)$ must be an atom which is more general than both L and W. This generalisation will replace the label of W in the global tree γ.

If the Algorithm 3.1 terminates then the closedness condition of [25] is satisfied, i.e., it is ensured that *together* the SLD-trees $\tau_1, \ldots, \tau_n$ form a *complete description* of all possible computations that can occur for all concrete instances $\leftarrow A\theta$ of the goal of interest [20, 18]. We can then produce a *totally correct* specialised program. On its own, Algorithm 3.1 does not ensure termination (so its strictly speaking not an algorithm but a procedure). To ensure termination, we have to use an unfolding rule that builds finite SLD-trees only. We also have to guarantee that infinite branches in the global tree γ will be spotted by the *whistle* and that the abstraction can not be repeated infinitely often.

[5] Alternatively one could remove all descendants of W and change the label of W. This is controlled by the *parent_abstraction* switch in ECCE.

3.2 Concrete Algorithm

We now present a concrete partial deduction algorithm, which is *online* (as opposed to *offline*) in the sense that control decisions are taken *during* the construction of γ and not beforehand. It is also rather naïve (e.g., it does not use characteristic trees [23]; also the generic Algorithm 3.1 does not include recent improvements such as conjunctions [5], constraints [21, 16] or abstract interpretation [18]). However, it is easier to comprehend (and analyse) and will actually be sufficiently powerful for our purposes (i.e., decide covering problems of reset Petri nets and other WSTS's).

Unfolding Rule In this paper we will use a very simple method for ensuring that each individual SLD-tree constructed by U is finite: we ensure that we unfold every predicate at most once in any given tree!

Whistle To ensure that no infinite global tree γ is being built-up, we will use a more refined approach based upon well-quasi orders: In our context we will use a wqo to ensure that no infinite tree γ is built up in Algorithm 3.1 by setting *whistle* to true whenever the sequence of labels on the current branch is not admissible. A particularly useful wqo (for a finite alphabet) is the pure homeomorphic embedding [33, 23]:

Definition 4. *The* (pure) homeomorphic embedding *relation* $\trianglelefteq$ *on expressions is inductively defined as follows (i.e.* $\trianglelefteq$ *is the least relation satisfying the rules):*

1. $X \trianglelefteq Y$ *for all variables* X, Y
2. $s \trianglelefteq f(t_1, \ldots, t_n)$ *if* $s \trianglelefteq t_i$ *for some* i
3. $f(s_1, \ldots, s_n) \trianglelefteq f(t_1, \ldots, t_n)$ *if* $\forall i \in \{1, \ldots, n\} : s_i \trianglelefteq t_i$.

Notice that n is allowed to be 0 and we thus have $c \trianglelefteq c$ for all constant and proposition symbols. The intuition behind the above definition is that $A \trianglelefteq B$ iff A can be obtained from B by "striking out" certain parts, or said another way, the structure of A reappears within B. We have $f(a, b) \trianglelefteq p(f(g(a), b))$.

Abstraction Once the *whistle* has identified a potential non-termination one will usually compute generalisations which are as precise as possible (for partial deduction): The *most specific generalisation* of a finite set of expressions S, denoted by $msg(S)$, is the most specific expression M such that all expressions in S are instances of M. E.g., $msg(\{p(0, s(0)), p(0, s(s(0)))\}) = p(0, s(X))$. The *msg* can be computed [17].

Algorithm 3.2 We define an instance of Algorithm 3.1 as follows:

- U unfolds every predicate just once
- $covered(L, \gamma) = true$ if there exists a processed node in γ whose label is *more general than* L
- $whistle(L, \gamma) = M$ iff M is an ancestor of L such that $label(M) \trianglelefteq label(L)$ and $whistle_{\trianglelefteq}(L, \gamma) = fail$ if there is no such ancestor.
- $abstract(L, W, \gamma) = msg(L, W)$.

Algorithm 3.2 terminates for any program P and goal $\leftarrow Q$ (this can be proven by simplified versions of the proofs in [23] or [32]).

4 Encoding (Reset) Petri Nets as Logic Programs

It is very easy to implement (reset) Petri nets as (non-deterministic) logic programs (see also [11]). Figure 2 contains a particular encoding of the reset Petri Net from Ex. 1 and a simple predicate `reachable` searching for reachable markings in $RS(\Pi)$. To model a reset arc (as opposed to an ordinary arc), one simply allows the `trans/3` facts to carry a 0 within the post-marking. Other nets can be encoded by changing the `trans/3` facts and the `initial_marking` fact.

Based upon such a translation, [12, 24] pursued the idea that model checking of safety properties amounts to showing that there exists no trace which leads to an invalid state, i.e., exploiting the fact that $\forall\Box safe \equiv \neg\exists\Diamond(\neg safe)$. Proving that no trace leads to a state where $\neg safe$ holds is then achieved by a *semantics-preserving program specialisation and analysis technique*. For this, an instance of Algorithm 3.1 was applied to several systems, followed by an abstract interpretation based upon [26] (we will return to [26] later in the paper).

```
reachable(R) :- initial_marking(M), reachable(Tr,R,M).
reachable([],State,State).
reachable([Action|As],Reach,InState) :-
   trans(Action,InState,NewState),reachable(As,Reach,NewState).
trans(t1, [s(S1),S2],[0,s(S2)]).
trans(t2, [S1,s(S2)],[s(s(S1)),S2]).
initial_marking([s(0),0]).
```

Fig. 2. Encoding a Reset Petri net as a logic program

As was shown in [22], this approach actually gives a decision procedure for coverability, (place-)boundedness, regularity of ordinary Petri nets. One can even establish a one-to-one correspondence between the Karp-Miller tree $KM(\Pi)$ and the global tree produced by (an instance of) partial deduction [22].

As we have seen, boundedness is undecidable for Petri nets with reset arcs [6] so the partial deduction approach, although guaranteed to terminate, will no longer give a decision procedure. (However, using default settings, ECCE can actually prove that the particular Reset net of Fig. 1 is bounded.)

But let us turn towards the covering problem which is decidable, but using the backwards algorithm we presented in Section 2. To be able to use partial deduction on this problem it seems sensible to write an "inverse" interpreter for reset Petri nets. This is not very difficult, as shown in Fig. 3, exploiting the fact that logic programs can be run backwards.

We can use this program in Prolog to check whether a particular marking such as $\langle 2, 0\rangle$ can be reached from the initial marking:

```
| ?- search_initial(T,[s(s(0)),0]).
T = [t2,t1]
```

Unfortunately, we cannot in general solve covering problems using Prolog or even XSB-Prolog due to their inability of detecting infinite failures. For example,

```
search_initial([],State) :- initial_marking(State).
search_initial([Action|As],InState) :-
   trans(Action,PredState,InState),search_initial(As,PredState).
trans(t1,[s(P1),P2],[0,s(P2)]).
trans(t2,[P1,s(P2)],[s(s(P1)),P2]).
initial_marking([s(0),0]).
```

Fig. 3. Backwards Interpreter for Reset Petri nets

the query `?-search_initial(T,[s(s(s(_X1))),_X2])`, checking whether $\langle 3, 0\rangle$ can be covered from the initial situation will loop in both Prolog or XSB-Prolog. However, the logic program and query is still a correct encoding of the covering problem: indeed, no instance of `search_initial(T,[s(s(s(_X1))),_X2])` is in the least Herbrand model. Below, we will show how this information can be extracted from the logic program using partial deduction, even to the point of giving us a decision procedure.

We will denote by $C(\Pi, M_0)$ the variation of the logic program in Fig. 3 encoding the particular (reset) Petri net Π with the initial marking M_0.[6]

5 Coverability of Reset Petri Nets by Partial Deduction

We will now apply our partial deduction algorithm to decide the covering problem for reset Petri nets. For this we need to establish a link between markings and atoms produced by partial deduction.

First, recall that an atom for partial deduction denotes all its instances. So, if during the partial deduction we encounter `search_initial(T,[M1,...,Mk])` this represents all markings $\langle m_1, ..., m_k\rangle$ such that for some substitution θ we have $\forall i$: `Mi`$\theta = \lceil m_i \rceil$. For example the term $s(s(s(X)))$ corresponds to the set represented by the number 3 in Section 2 (where a number n represents all numbers $m \geq n$). The following is a formalisation of the encoding of natural numbers as terms that will occur when specialising $C(\Pi, M_0)$:

- $\lceil i \rceil = X$ if $i = 0$ and where X is a fresh variable
- $\lceil i \rceil = s(\lceil i - 1 \rceil)$ otherwise

From now on, we also suppose that the order of the places in M_0 is the same as in the encoding in Figures 2, 3 and define $\lceil \langle m_1, \ldots, m_k \rangle \rceil = [\lceil m_1 \rceil, \ldots, \lceil m_k \rceil]$.

Lemma 1. *Let M and M' be two markings. Then:*

1. *$M \leq M'$ iff $\lceil M \rceil$ is more general than $\lceil M' \rceil$.*
2. *$M \leq M'$ iff $\lceil M \rceil \trianglelefteq \lceil M' \rceil$.*
3. *$M < M'$ iff $\lceil M \rceil$ is strictly more general than $\lceil M' \rceil$.*
4. *$\uparrow \{M\} = \{M' \mid \exists \theta$ with $\lceil M' \rceil = \lceil M \rceil \theta\}$.*

[6] To keep the presentation as simple as possible, contrary to [22], we do not perform a preliminary compilation. We compensate this by using a slightly more involved unfolding rule (in [22] only a single unfolding step is performed).

We can now establish a precise relationship between the computation of $pb(.)$ and SLD-derivations of the above logic program translation:

Lemma 2. *Let Π be Petri Net with reset arcs, and M be a marking for Π. Then $M_i \in pb(M)$ iff there exists an incomplete SLD-derivation of length 2 for $C(\Pi, M_0) \cup \{\leftarrow search\_initial(T, \lceil M \rceil)\}$ leading to $\leftarrow search\_initial(T', \lceil M_i \rceil)$. Also, $M_0 \in \uparrow \{M\}$ iff there exists an SLD-refutation of length 2 for $C(\Pi, M_0) \cup \{\leftarrow search\_initial(T, \lceil M \rceil)\}$.*

However, partial deduction atoms are more expressive than markings: e.g., we can represent all $\langle m_1, m_2, m_3 \rangle$ such that $m_1 > 0$, $m_2 = m_1 + 1$, and $m_3 = 1$ by: `search_initial(T,[s(X),s(s(X)),s(0)])`. In other words, we can establish a link between the number of tokens in several places via shared variables and we can represent exact values for places.[7] However, such information will never appear, because a) we start out with a term which corresponds exactly to a marking, b) we only deal with (reset) Petri nets and working backwards will yield a new term which corresponds exactly to a marking (as proven in Lemma 2) c) the generalisation will never be needed, because if our whistle based upon $\trianglelefteq$ blows then, by Lemma 1, the dangerous atom is an instance of an already processed one.[8]

Theorem 1. *Let Π be Petri Net with reset arcs with initial marking M_0 and let P be the residual program obtained by Algorithm 3.2 applied to $C(\Pi, M_0)$ and $\leftarrow search\_initial(T', \lceil M_c \rceil)$. Then P contains facts iff there exists a marking M' in $RT(\Pi)$ which covers M_c, i.e., $M' \geq M_c$*

The above theorem implies that when we perform a bottom-up abstract interpretation after partial deduction using, e.g., [26] (as done in [24]), we will be able to deduce failure of $\leftarrow search\_initial(T', \lceil M_c \rceil)$ *if and only if* $RT(\Pi)$ does not cover M_c! The following example illustrates this. When applying Algorithm 3.2 (using the ECCE system) to specialise the program in Fig. 3 for $\leftarrow search\_initial(T', \lceil \langle 3, 0 \rangle \rceil)$ we get:

```
/* Specialised Predicates:
search_initial__1(A,B,C) :- search_initial(C,[s(s(s(B))),A]).
search_initial__2(A,B) :- search_initial(B,[s(A),s(C1)]).
search_initial__3(A,B) :- search_initial(B,[A,s(s(C1))]).     */

search_initial(A,[s(s(s(B))),C]) :- search_initial__1(C,B,A).
search_initial__1(A,B,[t2|C]) :- search_initial__2(B,C).
search_initial__2(s(A),[t2|B]) :- search_initial__3(A,B).
search_initial__3(0,[t1|A]) :- search_initial__2(B,A).
search_initial__3(s(s(A)),[t2|B]) :- search_initial__3(A,B).
```

[7] More precisely, each atom represents a linear set $L \subseteq \mathbb{N}^k$ of markings $L = \{b + \sum_{i=1}^{r} n_i p^i \mid n_i \in \mathbb{N}\}$ with $b, p^i \in \mathbb{N}^k$ and the restriction that $\sum_{i=1}^{r} p^i \leq \langle 1, \ldots, 1 \rangle$.

[8] This also implies that a similar result to Theorem 1, for reset Petri nets, might be obtained by using OLDT abstract interpretation in place of partial deduction.

After which the most specific version abstract interpretation [26] implemented in ECCE will produce:

```
search_initial(A,[s(s(s(B))),C]) :-  fail.
```

It turns out that Algorithm 3.2, when expanding the global tree γ in a breadth-first manner, can actually mimic an improved version of the backwards algorithm from [10] (see Section 2): provided that we improve the backwards algorithm to not compute $pb(.)$ of markings which are already covered, we obtain that the set of labels in $\gamma = \{search\_initial(T, \lceil M \rceil) \mid M \in K_j\}$, where K_j is the set obtained by the improved backwards algorithm.

6 Other Well-Structured Transition Systems

Let us now turn to another well-known Petri net extension which does not violate the WSTS character [10]: *transfer arcs* [13, 6] which enable transitions to transfer tokens from one place to another, *doubling arcs* which double the number of tokens in a given place, or any mixture thereof. Take for example the following simple fact:

```
trans(t3,[P1,s(P2)],[P2,P2]).
```

This transition employs a combination of a reset arc (removing the number of tokens $P1$ present in place 1) and a kind of transfer arc (transferring all but one token from place 2 to place 1). Transitions like these will not pose a problem to our partial deduction algorithm: it can be used as is as a decision procedure for the covering problem and Theorem 1 holds for this extended class of Petri nets as well. In fact, a similar theorem should hold for any *post self-modifying net* [34, 10] and even *Reset Post G-nets* [6]. (However, the theorem is not true for Petri nets with inhibitor arcs.)

Another class of WSTS are basic process algebras (BPP's) [7], a subset of CCS without synchronisation. Below, we try to analyse them using our partial deduction approach. Plugging the following definitions into the code of Fig. 3 we encode a process algebra with action prefix ., choice +, and parallel composition ||, as well as a process starting from $(a.stop + b.stop)||c.stop)$:

```
trans(A,pre(A,P),P).
trans(A,or(X,_Y),XA) :- trans(A,X,XA).
trans(A,or(_X,Y),YA) :- trans(A,Y,YA).
trans(A,par(X,Y),par(XA,Y)) :- trans(A,X,XA).
trans(A,par(X,Y),par(X,YA)) :- trans(A,Y,YA).
initial_marking(par(or(pre(a,stop),pre(b,stop)),pre(c,stop))).
```

Compared to the WSTS's we have studied so far, the term representation of states gets much more complex (we no longer have lists of fixed length of natural numbers but unbounded process expressions). Our $\trianglelefteq$ relation is of course still a wqo on processes in this algebra, and it is also upwards compatible. Unfortunately we no longer have the nice correspondence between $\trianglelefteq$ and the instance-of

relation (as in Lemma 1 for reset nets). For instance, we have $pre(a, stop) \trianglelefteq stop$, but $pre(a, stop)$ is not more general than *stop* and partial deduction will not realise that it does not have to analyse *stop* if has already analysed $pre(a, stop)$ (indeed, in general it would be unsound not to examine *stop*). This means that the partial deduction approach will contain some "redundant" nodes. It also means that we cannot in general formulate covering problems as queries; although we can formulate reachability questions (for reset Petri nets we could do both). Nonetheless, specialising the above code using Algorithm 3.2 and [26], e.g., for the reachability query `search_initial(A,stop)`, we get:

```
search_initial(A,stop) :- fail.
```

This is correct: we can reach $stop||stop$ but not *stop* itself. However, despite the success on this particular example, we believe that to arrive at a full solution we will need to move to an abstract partial deduction [18] algorithm: this will enable us to re-instate the correspondence between the wqo of the WSTS and the instance-of relation of the program specialisation technique, thus arriving at a full-fledged decision procedure.

7 Future Work and Conclusion

One big advantage of the partial deduction approach to model checking is it scales up to any formalism expressible as a logic program. More precisely, proper instantiations of Algorithm 3.1 will terminate for any system and will provide safe approximations of properties under consideration. However, as is to be expected, we might no longer have a decision procedure.

[24] discusses how to extend the model checking approach to liveness properties and full CTL. Some simple examples are solved. E.g., the approach was applied to the manufacturing system used in [3] and it was able to prove absence of deadlocks for parameter values of, e.g., 1,2,3. When leaving the parameter unspecified, the system was unable to prove the absence of deadlocks and produced a residual program with facts. And indeed, for parameter value ≥ 9 the system can actually deadlock. The timings compare favourably with HyTech [14].

Reachability can be decided in some but not all cases using the present partial deduction algorithm. In future we want to examine the relationship to Mayr's algorithm [28] for ordinary Petri nets and whether it can be mimicked by abstract partial deduction [18].

Finally, an important aspect of model checking of finite state systems is the complexity of the underlying algorithms. We have not touched upon this issue in the present paper, but plan to do so in future work.

Conclusion We have examined the power of partial deduction (and abstract interpretation) for a particular class of infinite state model checking tasks, namely covering problems for reset Petri nets. The latter are particularly interesting as they lie on the "border between decidability and undecidabilty" [6]. We have proven that a well-established partial deduction algorithm based upon $\trianglelefteq$ can be

used as a decision procedure for these problems and we have unveiled a surprising correspondence with an existing algorithm from the Petri net area.

We have also shown that this property of partial deduction holds for other Petri net extensions which can be viewed as WSTS's. We have also studied other WSTS's from the process algebra arena. For these we have shown that, to arrive at a full-fledged decision procedure, we will need to move to the more powerful abstract partial deduction [18].

References

1. P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proceedings LICS'96*, pages 313–321, July 1996. IEEE Computer Society Press.
2. T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3:85–104, 1977.
3. B. Bérard and L. Fribourg. Reachability analysis of (timed) petri nets using real arithmetic. In *Proceedings Concur'99*, LNCS 1664, pages 178–193. Springer-Verlag, 1999.
4. W. Charatonik and A. Podelski. Set-based analysis of reactive infinite-state systems. In B. Steffen, editor, *Proceedings TACAS'98*, LNCS 1384, pages 358–375. Springer-Verlag, March 1998.
5. D. De Schreye, R. Glück, J. Jørgensen, M. Leuschel, B. Martens, and M. H. Sørensen. Conjunctive partial deduction: Foundations, control, algorithms and experiments. *J. Logic Progam.*, 41(2 & 3):231–277, November 1999.
6. C. Dufourd, A. Finkel, and P. Schnoebelen. Reset nets between decidability and undecidability. In *Proceedings ICALP'98*, LNCS 1443, pages 103–115. Springer-Verlag, 1998.
7. J. Ezparza. Decidability of model-checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.
8. A. Finkel. The minimal coverability graph for Petri nets. *Advances in Petri Nets 1993*, LNCS 674, pages 210–243, 1993.
9. A. Finkel and P. Schnoebelen. Fundamental structures in well-structured infinite transition systems. In *Proceedings LATIN'98*, LNCS 1380, pages 102–118. Springer-Verlag, 1998.
10. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere ! *Theoretical Computer Science*, 2000. To appear.
11. L. Fribourg and H. Olsen. Proving Safety Properties of Infinite State Systems by Compilation into Presburger Arithmtic. In *Proceedings Concur'97*, LNCS 1243, pages 213–227. Springer-Verlag, 1997.
12. R. Glück and M. Leuschel. Abstraction-based partial deduction for solving inverse problems – a transformational approach to software verification. In *Proceedings PSI'99*, LNCS 1755, pages 93–100, 1999. Springer-Verlag.
13. B. Heinemann. Subclasses of self-modifying nets. In *Applications and Theory of Petri Nets*, pages 187–192. Springer-Verlag, 1982.
14. T. A. Henzinger and P.-H. Ho. HYTECH: The Cornell HYbrid TECHnology tool. *Hybrid Systems II*, LNCS 999:265–293, 1995.
15. R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3:147–195, 1969.

16. L. Lafave and J. Gallagher. Constraint-based partial evaluation of rewriting-based functional logic programs. In N. Fuchs, editor, *Proceedings LOPSTR'97*, LNCS 1463, pages 168–188, July 1997.
17. J.-L. Lassez, M. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan-Kaufmann, 1988.
18. M. Leuschel. Program specialisation and abstract interpretation reconciled. In J. Jaffar, editor, *Proceedings JICSLP'98*, pages 220–234, Manchester, UK, June 1998. MIT Press.
19. M. Leuschel. Logic program specialisation. In J. Hatcliff, T. Æ. Mogensen, and P. Thiemann, editors, *Partial Evaluation: Practice and Theory*, LNCS 1706, pages 155–188 and 271–292, 1999. Springer-Verlag.
20. M. Leuschel and D. De Schreye. Logic program specialisation: How to be more specific. In H. Kuchen and S. Swierstra, editors, *Proceedings PLILP'96*, LNCS 1140, pages 137–151, September 1996. Springer-Verlag.
21. M. Leuschel and D. De Schreye. Constrained partial deduction and the preservation of characteristic trees. *New Gen. Comput.* , 16:283–342, 1998.
22. M. Leuschel and H. Lehmann. Solving Coverability Problems of Petri Nets by Partial Deduction. Submitted.
23. M. Leuschel, B. Martens, and D. De Schreye. Controlling generalisation and polyvariance in partial deduction of normal logic programs. *ACM Transactions on Programming Languages and Systems*, 20(1):208–258, January 1998.
24. M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialisation. In A. Bossi, editor, *Proceedings LOPSTR'99*, LNCS 1817, pages 63–82, Venice, Italy, September 1999.
25. J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *J. Logic Progam.*, 11(3& 4):217–242, 1991.
26. K. Marriott, L. Naish, and J.-L. Lassez. Most specific logic programs. *Annals of Mathematics and Artificial Intelligence*, 1:303–338, 1990.
27. B. Martens and J. Gallagher. Ensuring global termination of partial deduction while allowing flexible polyvariance. In L. Sterling, editor, *Proceedings ICLP'95*, pages 597–613, June 1995. MIT Press.
28. E. W. Mayr. An algorithm for the general Petri net reachability problem. *Siam Journal on Computing*, 13:441–460, 1984.
29. Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In *Proceedings CAV'97*, LNCS 1254, pages 143–154. Springer-Verlag, 1997.
30. W. Reisig. *Petri Nets - An Introduction*. Springer Verlag, 1982.
31. J. Rushby. Mechanized formal methods: Where next? In *Proceedings of FM'99*, LNCS 1708, pages 48–51, Sept. 1999. Springer-Verlag.
32. M. H. Sørensen. Convergence of program transformers in the metric space of trees. In *Proceedings MPC'98*, LNCS 1422, pages 315–337. Springer-Verlag, 1998.
33. M. H. Sørensen and R. Glück. An algorithm of generalization in positive supercompilation. In J. W. Lloyd, editor, *Proceedings ILPS'95*, pages 465–479, December 1995. MIT Press.
34. R. Valk. Self-modifying nets, a natural extension of Petri neets. In *Proceedings ICALP'78*, LNCS 62, pages 464–476. Springer-Verlag, 1978.
35. R. Valk and G. Vidal-Naquet. Petri nets and regular languages. *Journal of Computer and System Sciences*, 23(3):299–325, Dec. 1981.