Net Reductions for LTL Model-Checking

Javier Esparza and Claus Schröter

Institut für Informatik, Technische Universität München
{esparza,schroete}@in.tum.de

Abstract. We present a set of reduction rules for LTL model-checking of 1-safe Petri nets. Our reduction techniques are of two kinds: (1) Linear programming techniques which are based on well-known Petri net techniques like invariants and implicit places, and (2) local net reductions. We show that the conditions for the application of some local net reductions can be weakened if one is interested in LTL model-checking using the approach of [EH00,EH01]. Finally, we present a number of experimental results and show that the model-checking time of a net system can be significantly decreased if it has been preprocessed with our reduction techniques.

1 Introduction

In two recent papers [EH00,EH01], Esparza and Heljanko have developed an unfolding technique to the problem of model-checking LTL for concurrent systems. Using the automata-theoretic approach, the model-checking problem is reduced to two simpler problems concerning the infinite executions of a combined system obtained from the original system and from a Büchi automaton for the negation of the property to be checked. Loosely speaking, the unfolding technique exploits the concurrency present in the system to obtain a compact representation of the state space.

Esparza and Heljanko's approach is part of the Model Checking Kit, a collection of programs which allow to model a finite-state system using a variety of modelling languages, and verify it using a variety of checkers, including deadlock-checkers, reachability-checkers, and model-checkers for the temporal logics CTL and LTL. The most interesting feature of the Kit is that, independently of the description language chosen by the user, different checkers can be applied to the same model. The Kit is held together by a program called the Glue. Given a system and a property modelled in one of the description languages supported by the Kit, the Glue (1) translates them into a 1-safe Petri net and a property of the net, described both in a common internal representation language;[1] (2) translates the net and the property into input for the target model checker, and (3) collects the output of the checker and translates it back into the original description language.

[1] 1-safe Petri nets are chosen because they are unstructured and have a simple notion of independent actions, which makes them a suitable "assembler language" for concurrent systems.

Since step (1) is automatic, it may introduce many redundancies, leading to large 1-safe Petri nets. In this paper we introduce some techniques that allow to remove a good number of these redundancies. For that, we exploit well-known Petri net techniques: (sub)invariants [DE95,Rei85], implicit places [CS90], and local net reductions [Ber85,PPP00]. Invariants and implicit places are used to remove places and transitions which do not affect the behaviour of the Petri net. Local net reductions are used to reduce the Petri net while guaranteeing that the result of the analysis on the reduced net will be the same as the result in the original net. Notice that, since we work at the level of the common internal representation language, redundancies can be eliminated independently of the language in which the system was described.

The paper contains two main contributions. First, we show that if we are interested in LTL model-checking using the approach of [EH00,EH01], the conditions for the applications of some net reductions can be *weakened*; therefore, the rules can be applied more often, leading to larger reductions. The second contribution is a number of experimental results on a set of examples modelled using several of the Kit's system description languages.

The paper is organised as follows. Section 2 contains basic definitions about 1-safe Petri nets. Section 3 gives some information on the approach of [EH00, EH01]. Section 4 introduces the reduction rules. (Sub)invariants and implicit places are just taken from the literature, while some local net reductions are improved. Section 5 describes the routine for the application of the rules, and discusses implementation issues. Section 6 presents experimental results on systems modelled using B(PN)^2, a high-level programming language integrated in the Kit, and communicating automata.

2 1-Safe Petri Nets

A triple (P, T, F) is a *net* if P and T are disjoint sets and F is a subset of $(P \times T) \cup (T \times P)$. The elements of P are called *places* and the elements of T *transitions*. Places and transitions are generally called *nodes*. We identify F with its characteristic function on the set $(P \times T) \cup (T \times P)$. The *preset* ${}^\bullet x$ of a node x is the set $\{y \in P \cup T \mid F(y, x) = 1\}$. The *postset* $x^\bullet$ of a node x is the set $\{y \in P \cup T \mid F(x, y) = 1\}$.

A *marking* M of a net (P, T, F) is a mapping $M: P \mapsto \{0, 1\}$. We identify a marking M with the set $P' \subseteq P$ such that $\forall p \in P: p \in P' \Leftrightarrow M(p) = 1$ holds.

A four-tuple $\Sigma = (P, T, F, M_0)$ is a *net system* if (P, T, F) is a net and M_0 is a marking of (P, T, F). M_0 is called the *initial marking* of the net system Σ. A marking M *enables* a transition t if $\forall p \in {}^\bullet t: M(p) = 1$ holds. If t is enabled at M, then t can *occur*, and its occurrence leads to a new marking M' (denoted $M \xrightarrow{t} M'$), defined by $M'(p) = M(p) - F(p, t) + F(t, p)$ for every place p. A sequence of transitions $\sigma = t_1 t_2 \dots t_n$ is an *occurrence sequence* if there exist markings $M_1, M_2, \dots, M_n$ such that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots M_{n-1} \xrightarrow{t_n} M_n$. M_n is the marking reached by the occurrence of σ, also denoted by $M_0 \xrightarrow{\sigma} M_n$. M is a *reachable marking* if there exists an occurrence sequence σ such that $M_0 \xrightarrow{\sigma} M$.

3 An Unfolding Approach to LTL Model-Checking

The unfolding technique, originally introduced by McMillan [McM92], has been very successfully applied to several verification tasks, e.g. deadlock detection, reachability analysis and LTL model-checking. The 1-safe Petri net is unfolded into an acyclic net until a finite complete prefix is generated. This is a finite acyclic net having exactly the same reachable markings as the original one. Esparza and Heljanko have introduced an unfolding approach to LTL model-checking in [EH00]. This approach makes use of the automata-theoretic approach to model-checking [Var96]. A *synchronized net system* is constructed as the product of the original net system and a Büchi automaton accepting the negation of the property to be checked. Then the model-checking problem is reduced to the problem of detecting illegal ω-traces and illegal livelocks in the synchronized net system. Both problems are solved by constructing finite prefixes of the unfolding of the synchronized net system. The main advantage of this approach with respect to Wallner's approach [Wal98] is its simplicity. Wallner first calculates a complete prefix and then he constructs a graph, but the definition of the graph is non-trivial, and the graph can be exponential in the size of the prefix. The approach of [EH00] avoids the construction of the graph, but unfortunately constructs a larger prefix.
Now, we briefly review the main definitions and results of [EH00]. Given an LTL property φ and a net system $\Sigma = (N, M_0)$ the transitions of the net $N = (P, T, F)$ are divided into two sets V and $T \setminus V$ of *visible* and *invisible* transitions. Let $AP(\varphi) \subseteq P$ be a set of places corresponding to the atomic propositions of φ. Then it holds that $\forall t \in T\colon t \in V \Leftrightarrow t \in ({}^\bullet p \cup p^\bullet)$ for a $p \in AP(\varphi)$. Then a Büchi automaton $A_{\neg\varphi}$ is constructed accepting the negation of the property φ. The Büchi automaton is a tuple $A = (\Gamma, Q, q_0, F, \rho)$ where Γ is an *alphabet*, Q is a finite non-empty set of *states*, $q_0 \in Q$ is an *initial state*, $F \subseteq Q$ is the set of *accepting states*, and $\rho \subseteq Q \times \Gamma \times Q$ is the *transition relation.* A *accepts* an infinite word $w \in \Gamma^\omega$ if some run of A on w visits some state in F infinitely often.

In a next step a *synchronized net system* $\Sigma_{\neg\varphi}$ is constructed as the product of the original net system Σ and the Büchi automaton $A_{\neg\varphi}$. The system Σ and the Büchi automaton are synchronized only by the visible transitions of Σ. For more details we refer the reader to [EH00]. In the following we call the places and transitions of $\Sigma_{\neg\varphi}$ corresponding to the Büchi automaton *Büchi places* and *Büchi transitions*. Büchi transitions and visible transitions have a characteristic feature as described in the following Lemma 1.

Lemma 1. *Property of Büchi transitions and visible transitions*

Let $\Sigma_{\neg\varphi}$ be a synchronized net system. Then for all Büchi transitions and visible transitions t it holds that ${}^\bullet t \geq 2$ and $t^\bullet \geq 2$.

Proof: Clear from the construction of the synchronized system $\Sigma_{\neg\varphi}$ (see [EH00]).

Once $\Sigma_{\neg\varphi}$ is constructed, two subsets of Büchi transitions, called *infinite trace monitors* and *livelock monitors*, are defined as follows.

Definition 1. *Infinite Trace Monitors and Livelock Monitors*

- The set I of *infinite trace monitors* contains the transitions t such that there exists in $t^\bullet$ a final state q of $A_{\neg\varphi}$. Loosely speaking, these are the transitions which put a token into a final state of the Büchi automaton.
- The set L of *livelock monitors* contains the transitions t such that there exists in $t^\bullet$ a state q of $A_{\neg\varphi}$ satisfying the following condition: with q as initial state, the automaton $A_{\neg\varphi}$ accepts an infinite sequence of transitions.

■ 1

With this knowledge we can define a notion of illegal ω-traces and illegal livelocks.

Definition 2. *Illegal ω-Traces and Illegal Livelocks*

Let Σ be a net system where T is divided into two sets of V and $T \setminus V$ of visible and invisible transitions, and T contains the two subsets I and L of transitions as mentioned above.

- An *illegal ω-trace* of Σ is an infinite sequence $M_0 \xrightarrow{\sigma}$ such that σ contains infinitely many I-transitions.
- An *illegal livelock* of Σ is an infinite sequence $M_0 \xrightarrow{\sigma t} M \xrightarrow{\sigma_1}$ such that $t \in L$ and σ_1 contains only invisible transitions.

■ 2

The main result of [EH00] is Theorem 1.

Theorem 1. *LTL Model-Checking*

Let Σ be a labelled net system and φ an LTL formula. Σ satisfies φ if and only if $\Sigma_{\neg\varphi}$ has no illegal ω-traces and no illegal livelocks.

4 Reduction Rules for LTL Model-Checking

The LTL model-checking approach described in the previous section uses unfolding techniques for detecting illegal ω-traces and illegal livelocks. Since the unfolding grows exponentially in the size of the net system, our aim is to reduce the synchronized net system before unfolding it without changing the conditions of Theorem 2 for the LTL model-checking task.

Theorem 2. *Conditions for LTL Model-Checking*

Let Σ be a labelled net system and φ an LTL formula. Σ violates φ if and only if at least one of the following conditions hold for $\Sigma_{\neg\varphi}$:

(i) there exists an infinite sequence $M_0 \xrightarrow{\sigma}$ containing infinitely many I-transitions, or

(ii) there exists a sequence $M_0 \xrightarrow{\sigma} M$ such that

(a) there exists an infinite sequence $M|_{A_{\neg\varphi}} \xrightarrow{\sigma_1}$ which goes infinitely often through a final state, and

(b) there exists an infinite sequence $M|_\Sigma \xrightarrow{\sigma_2}$ containing only invisible transitions

Now, with respect to the LTL model-checking approach of [EH00] we only have to guarantee that the two conditions of Theorem 2 hold for a net system if and only if they hold for the reduced net system obtained by applying our reduction rules. Additionally, we restrict ourselves to the fact that we will never remove atomic propositions, Büchi places and Büchi transitions from the net. In the following sections we present some reduction rules which are practicable for the LTL model-checking approach of [EH00].

4.1 Linear Programming Rules

In this section we present two reduction rules which are based on linear programming techniques. First, we briefly introduce some basic notions. When applying linear programming techniques for verification tasks of Petri nets, one of the basic concepts is the so-called marking equation that can be used as an algebraic representation of the set of reachable markings of an acyclic net. Given a marking M reachable from the initial marking M_0 and a place p, the number of tokens of p in M can be calculated as the number of tokens p carries in M_0 plus the difference of tokens added by the input places and removed by the output places. This leads to the following equation: $M(p) = M_0(p) + \Sigma_{t \in {}^\bullet p} \#t - \Sigma_{t \in p^\bullet} \#t$ where $\#t$ denotes the number of occurrences of t in an occurrence sequence $\sigma = t_1 \dots t_m$. Usually this equation is written in the form $M = M_0 + \mathbf{N} \cdot \boldsymbol{\sigma}$, where $\boldsymbol{\sigma} = {}^t(\#t_1, \dots, \#t_n)$ is called the *Parikh vector* of σ and $\mathbf{N}$ denotes the *incidence matrix* of a net N, a $P \times T$ matrix given by $\mathbf{N}(p,t) = F(t,p) - F(p,t)$. Let l_p be the *incidence vector* of a place p. Then we have $M(p) = M_0(p) + l_p \cdot \boldsymbol{\sigma}$.

Dead Transition Rule. In many cases systems that should be checked for LTL properties are specified in a high level programming language, for instance B(PN)^2 which is well-known from the PEP-tool and also an input language of the Kit. The systems have been translated automatically into 1-safe Place/Transition nets. During these translations many redundancies might have been introduced, for instance places which never carry a token. It is clear that also their output transitions never fire. The aim of the dead transition rule is to detect such places and transitions and to remove them from the net system. Applying this rule does not affect the size of the unfolded net prefix which will be calculated during the model checking process of [EH00,EH01] but it surely decreases the prefix construction time. A solution vector $Y \geq 0$ and $Y \neq 0$ for both equations $Y^T \cdot M_0 = 0$ and $Y^T \cdot \mathbf{N} \leq 0$ yields a set of places which never carry tokens. Since the solution is not unique this solution might not give us all such places. Therefore we repeat the application of the dead transition rule. However, a solution of $Y^T \cdot M_0 = 0$ determines a set of places which are not initially marked, and a solution of $Y^T \cdot \mathbf{N} \leq 0$ yields a subinvariant of places for which firing a transition does not increase the number of tokens in these places.

Definition 3. *Dead Transition Rule*

Let $N = (P, T, F)$ be a net. A set $P_d \subseteq P$ of places and a set $T_d \subseteq T$ of transitions satisfy the dead transition rule if and only if

- the following equation system has a solution for Y:
  Variables: Y
  $Y^T \cdot \mathbf{N} \leq 0$
  $Y^T \cdot M_0 = 0$
  $Y \geq 0$
- $\forall p \in P: p \in P_d \Leftrightarrow Y_p > 0$
- $\forall t \in T: t \in T_d \Leftrightarrow \exists p \in P_d: t \in p^\bullet$

The reduced net $N_{red} = (P_{red}, T_{red}, F_{red})$ obtained by applying the dead transition rule on N is defined by

- $P_{red} = P \setminus P_d$
- $T_{red} = T \setminus T_d$
- $F_{red} = F \cap ((P_{red} \times T_{red}) \cup (T_{red} \times P_{red}))$
- $M_0^{red} = M_0|_{P_{red}}$

■ 3

Implicit Place Rule. A place is called *implicit* if it never restricts the firing of its output transitions. Loosely speaking, an implicit place is never the only reason that a transition of its postset cannot fire. Colom and Silva have published a method for detecting implicit places of bounded Place/Transition nets with linear programming techniques [CS90]. Our rule is based on this approach. Figure 1 shows an example for the implicit place rule. Place p can be seen as a linear combination of the places p_1 and p_2 since $M(p) = M(p_1) + M(p_2)$ holds. It is clear that p never restricts the firing of transition t and therefore place p can be removed without changing the firing sequences of the net.

Definition 4. *Implicit Place Rule*

Let $N = (P, T, F)$ be a net. A place $p \in P$ satisfies the implicit place rule if and only if the following linear program has a solution for Y and μ:
Variables: Y, μ
Minimize $Y^T \cdot M_0 + \mu$ such that
$Y^T \cdot \mathbf{N} \leq l_p$
$Y^T \cdot pre(t_i) + \mu \geq 1, \forall t_i \in p^\bullet$
$Y^T \cdot M_0 + \mu \leq M_0(p)$
$Y \geq 0$
where the vector $pre(t)$ is defined as follows:
$\forall\, 1 \leq j \leq |P|: pre_j(t) = 1$, if $p_j \in {}^\bullet t$, and 0 otherwise.

The reduced net $N_{red} = (P_{red}, T_{red}, F_{red})$ obtained by applying the implicit place rule on N is defined by

- $P_{red} = P \setminus \{p\}$
- $T_{red} = T$
- $F_{red} = F \cap ((P_{red} \times T_{red}) \cup (T_{red} \times P_{red}))$
- $M_0^{red} = M_0|_{P_{red}}$

■ 4

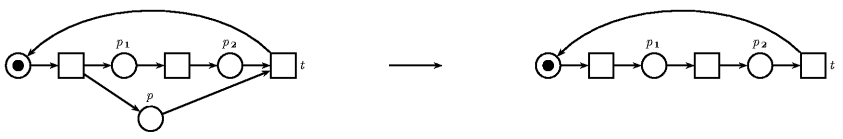

Fig. 1. Implicit Place Rule

Theorem 3. *Linear Programming Rules preserve LTL conditions*

Let $\Sigma_{\neg\varphi}$ be a synchronized net system and $\Sigma_{\neg\varphi}^{red}$ be the net system obtained by applying the dead transition rule and the implicit place rule. Conditions (i) and (ii) of Theorem 2 hold for $\Sigma_{\neg\varphi}$ if and only if they hold for $\Sigma_{\neg\varphi}^{red}$.

Proof: Applying the dead transition rule preserves the firing sequences of the net system. In [CS90] it has been proven that removing implicit places from the net system also preserves the firing sequences. Therefore the rules do not affect the conditions (i) and (ii) of Theorem 2.

4.2 Local Rules

In this section we present some local reduction rules. They differ from the linear programming rules in the sense that the linear programming rules globally investigate the whole net whereas the local rules only inspect small sub-nets. The local rules are taken from the literature, but we have improved some of them by weakening their conditions. This allows us to apply the rules more often and to obtain larger reductions.

Abstraction Rule. In this section we introduce a new abstraction rule that to the best of our knowledge has not been mentioned before in the literature. The abstraction rule is graphically described in Fig. 2. The main idea of the abstraction rule is that any occurrence sequence of the net can be reordered into an occurrence sequence where an occurrence of t has to be immediately preceded by an occurrence of an input transition of p. The reduction hides the occurrence of t by merging t with the input transitions of p. But we have to ensure that the input transitions of p are enabled in the reduced net if and only if the transition t is enabled in the original net. Therefore the presets of the input transitions of p are extended by the input places of t. Our rule differs from the abstraction rule in [DE95] in such way that in our rule the transition t may have more than one place in its preset. So our rule describes a generalization of [DE95].

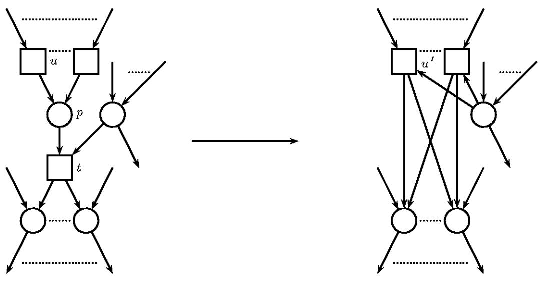

Fig. 2. Abstraction Rule

Definition 5. *Abstraction Rule*

Let $N = (P, T, F)$ be a net. A place $p \in P$ and a transition $t \in T$ satisfy the abstraction rule if and only if
- $^\bullet p \neq \emptyset$, $p^\bullet = \{t\}$
- $\forall u \in {}^\bullet p\colon u^\bullet = \{p\}$
- $t^\bullet \neq \emptyset$
- $M_0(p) = 0$

The reduced net $N_{red} = (P_{red}, T_{red}, F_{red})$ obtained by applying the abstraction rule on N is defined by:
- $P_{red} = P \setminus \{p\}$
- $T_{red} = T \setminus (\{t\} \cup \{u \in {}^\bullet p \mid {}^\bullet u \cap {}^\bullet t \neq \emptyset\})$
- $F_{red} = (F \cap ((P_{red} \times T_{red}) \cup (T_{red} \times P_{red}))) \cup (({}^\bullet p \cap T_{red}) \times t^\bullet) \cup (({}^\bullet t \cap P_{red}) \times ({}^\bullet p \cap T_{red}))$
- $M_0^{red} = M_0|_{P_{red}}$

■ 5

Theorem 4. *Abstraction Rule preserves LTL conditions*

Let $\Sigma_{\neg\varphi}$ be a synchronized net system and $\Sigma_{\neg\varphi}^{red}$ be the net system obtained by applying the abstraction rule. Conditions (i) and (ii) of Theorem 2 hold for $\Sigma_{\neg\varphi}$ if and only if they hold for $\Sigma_{\neg\varphi}^{red}$.

Pre-Agglomeration Rule. The pre-agglomeration rule is graphically described in Fig. 3. The main idea of this rule is that any occurrence sequence of a net can be reordered into an occurrence sequence where the occurrence of transition t is immediately followed by an occurrence of an output transition of p. The reduction hides the occurrence of t by merging t with the output transitions of p. This structural condition implies that one can delay the firing of the transition t. The pre-agglomeration rule has been introduced by Berthelot [Ber85] and has been used also in [PPP00] but our rule differs from their rule in the following way: In [Ber85,PPP00] the restriction holds that $\forall q \in {}^\bullet t\colon q^\bullet = \{t\}$. This restriction is not neccessary in our approach.

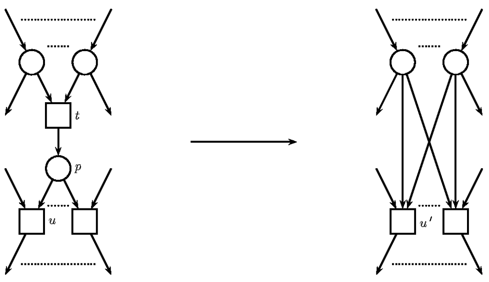

Fig. 3. Pre-Agglomeration Rule

Definition 6. *Pre-Agglomeration Rule*

Let $N = (P, T, F)$ be a net. A place $p \in P$ and a transition $t \in T$ satisfy the pre-agglomeration rule if and only if

- $^\bullet p = \{t\}$, $p^\bullet \neq \emptyset$
- $^\bullet t \neq \emptyset$, $t^\bullet = \{p\}$
- $M_0(p) = 0$

The reduced net $N_{red} = (P_{red}, T_{red}, F_{red})$ obtained by applying the pre-agglomeration rule on N is defined by:

- $P_{red} = P \setminus \{p\}$
- $T_{red} = T \setminus (\{t\} \cup \{u \in p^\bullet \mid {}^\bullet u \cap {}^\bullet t \neq \emptyset\})$
- $F_{red} = (F \cap ((P_{red} \times T_{red}) \cup (T_{red} \times P_{red}))) \cup ({}^\bullet t \times (p^\bullet \cap T_{red}))$
- $M_0^{red} = M_0|_{P_{red}}$

■ 6

Theorem 5. *Pre-Agglomeration Rule preserves LTL conditions*

Let $\Sigma_{\neg\varphi}$ be a synchronized net system and $\Sigma_{\neg\varphi}^{red}$ be the net system obtained by applying the pre-agglomeration rule. Conditions (i) and (ii) of Theorem 2 hold for $\Sigma_{\neg\varphi}$ if and only if they hold for $\Sigma_{\neg\varphi}^{red}$.

Post-Agglomeration Rule. The post-agglomeration rule is graphically described in Fig. 4. Let us consider the sets $^\bullet p$ and $p^\bullet$ of transitions. The main idea of the post-agglomeration rule is that any occurrence sequence of the net containing a transition $h \in {}^\bullet p$ and a transition $u \in p^\bullet$ can be reordered into an occurrence sequence such that the occurrence of h is immediately followed by the occurrence of u. This structural condition implies that one can anticipate the firing of u. The post-agglomeration rule has been introduced in [Ber85] and has been used also in [PPP00].

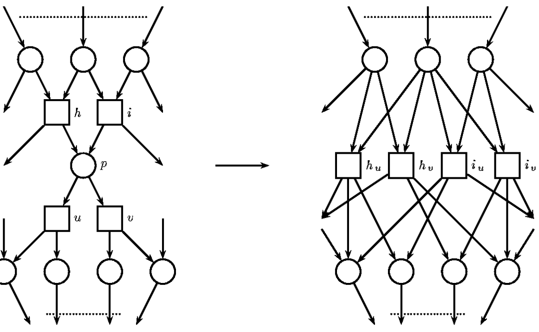

Fig. 4. Post-Agglomeration Rule

Definition 7. *Post-Agglomeration Rule*

Let $N = (P, T, F)$ be a net. A place $p \in P$ satisfies the post-agglomeration rule if and only if
- $^{\bullet}p \neq \emptyset$, $p^{\bullet} \neq \emptyset$
- $\forall t \in p^{\bullet} \colon {}^{\bullet}t = \{p\}$
- $M_0(p) = 0$

The reduced net $N_{red} = (P_{red}, T_{red}, F_{red})$ obtained by applying the post-agglomeration rule on N is defined by:
- $P_{red} = P \setminus \{p\}$
- $T_{red} = (T \setminus ({}^{\bullet}p \cup p^{\bullet})) \cup ({}^{\bullet}p \times p^{\bullet})$
- $\forall q \in P_{red}, \forall u \in T_{red} \setminus ({}^{\bullet}p \times p^{\bullet})$:
  $F_{red}(q, u) = F(q, u), F_{red}(u, q) = F(u, q)$
- $\forall q \in P_{red}, \forall t_1t_2 \in ({}^{\bullet}p \times p^{\bullet})$:
  $F_{red}(q, t_1t_2) = F(q, t_1), F_{red}(t_1t_2, q) = F(t_1, q) + F(t_2, q)$
- $M_0^{red} = M_0|_{P_{red}}$

■ 7

Theorem 6. *Post-Agglomeration Rule preserves LTL conditions*

Let $\Sigma_{\neg\varphi}$ be a synchronized net system and $\Sigma_{\neg\varphi}^{red}$ be the net system obtained by applying the post-agglomeration rule. Conditions (i) and (ii) of Theorem 2 hold for $\Sigma_{\neg\varphi}$ if and only if they hold for $\Sigma_{\neg\varphi}^{red}$.

5 Implementation Issues

Special care has been taken to speed-up the net reductions. Therefore we mention some facets of our implementation in this section. First of all, we apply the dead transition rule because it investigates globally the whole net and yields sets of places and transitions which can be removed from the net. As mentioned in chapter 4.1 the dead transition rule detects places which never carry tokens. The solution is not unique, and therefore this solution might not give us all such places. We have to repeat the application of the dead transition rule to detect maybe more such places. $Y = 0$ is always a solution of the equation system. Therefore we use the objective function *Maximize* $Y^T \cdot 1$, $0 \leq Y \leq 1$, to get other possible solutions.

The following rules are part of a loop which will be repeated until no changes can be made. First, we detect redundant places and remove them from the net. A place p is called *redundant* if and only if there exists a place q such that p and q have the same initial markings and equal sets of input/output transitions. Redundant places would also be detected by applying the implicit place rule but it takes more time to solve a linear equation system than to pass through a list of places and to compare their pre- and postsets. We use efficient data structures for storing nets which allow fast comparisons of pre- and postsets of places and transitions. This universal data structure provides fast access to single nodes [Röm00]. Places, transitions and arcs are represented by nodes of doubly linked adjacent lists. Then we apply the abstraction, pre-agglomeration and post-agglomeration rules. These rules are based on simple operations conducted on our efficient data structures. They yield fast transformations compared to the linear programming techniques. We apply the implicit place rule at last for the following two reasons: The number and size of the equation systems to be solved depend on the number of places and transitions of the nets. Therefore it is reasonable to make them as small as possible before applying the implicit place rule. The creation of the equation system for each place can be done without significant loss of time because the objective function *Minimize* $Y^T \cdot M_0 + \mu$ and the part $Y^T \cdot \mathbf{N}$ are equal for all places. We have to build them only once and can reuse them for all equation systems. We only have to make minor modifications on the right side, and have to add few rows for each place.

6 Experimental Results

In this section we will present our experimental results. Our aim is to show that we obtain good reduction ratios in practice and that the reduction times are very small compared with the verification times for the LTL model-checking task.

The systems we have used are as follows:

- buf(100): Buffer with capacity 100 generated by Römer.
- fifo(20): 1-bit-FIFO with depth 20 [Mar86,RCP95].
- plate(5): Production cell which handles 5 plates [LL95,HD95].
- dph(7): Variant of the dining philosophers with a butler [Cor94].
- furnace(3): Manages the temperature data for 3 furnaces [Cor94].
- key(4): Manages keyboard/screen interaction in a window manager for 4 customer tasks [Cor94].
- rw(3r1w), rw(1r2w): Address a scalable and bottleneck-free readers/writers synchronization algorithm for shared memory parallel machines [Hel93].

	$\Sigma_{\neg\varphi}$		$\mathrm{Unf}(\Sigma_{\neg\varphi})$			$\Sigma_{\neg\varphi}^{red}$			$\mathrm{Unf}(\Sigma_{\neg\varphi}^{red})$		
	$\|S\|$	$\|T\|$	$\|B\|$	$\|E\|$	t_{LTL}	$\|S\|$	$\|T\|$	t_{red}	$\|B\|$	$\|E\|$	t_{LTL}
buf(100)	205	107	10111	5054	1274.9	7	8	41.6	13	5	< 0.1
fifo(20)	171	132	64167	42107	47229.8	39	31	3.9	736	364	2.5
plate(5)	239	214	1803	810	35.8	117	128	2.9	998	416	8.4
dph(7)	71	127	72472	35021	11280.9	31	101	1.0	2971	1406	4.5
furnace(3)	58	105	33363	19322	1541.7	36	92	0.6	12819	7640	132.4
key(4)	169	180	138052	68585	49516.4	121	153	2.8	95416	57062	31196.4
rw(3r1w)	111	276	29717	15862	3828.5	80	202	0.9	25882	12078	1796.2
rw(1r2w)	214	1488	19874	9770	2384.8	159	981	8.2	17644	7550	1395.4
slotring(8)	85	86	24734	17195	5379.5	54	55	0.5	15283	7956	930.4
slotring(10)	105	106	67266	48145	44407.4	67	68	0.7	44562	23901	8842.9

Fig. 5. Experimental results

– slotring(n): Slotted ring protocol with n nodes [PRCB94].

The LTL properties we checked have the form:

– $\neg F(P_1 \wedge P_2)$ ($F \hat{=}$ *eventually*)
– $G((P_1 \wedge \neg P_2 \wedge \neg P_3) \vee (\neg P_1 \wedge P_2 \wedge \neg P_3) \vee (\neg P_1 \wedge \neg P_2 \wedge P_3))$ ($G \hat{=}$ *always*)

We have applied the first property on all systems except the production cell, and have checked the invariant (second property) for the production cell.

All experiments were performed on a SUN Ultra 60 with 1.5 GByte of RAM and a 295 MHz UltraSPARC-II CPU. The rules which are based on linear programming techniques use CPLEX^{TM} (version 6.5.1) as its underlying LP-solver. The LTL model-checker of [EH00] (*unfsmodels 0.9*) is implemented by Heljanko [Hel01].

Figure 5 shows the results. The columns $|S|$ ($|B|$) and $|T|$ ($|E|$) denote the numbers of places (conditions) and transitions (events) of the net (unfolding). The columns t_{LTL} and t_{red} denote the times for the LTL model-checking (*unfsmodels*) and for our reduction procedure. The times are measured in seconds.

The results show that we obtain very considerable reduction ratios for the systems buf(100), fifo(20), plate(5), dph(7), and also for the slotted ring protocols slotring(n). For these systems the unfoldings of the reduced nets are much smaller than the unfoldings of the original nets. This strongly affects the model-checking algorithm of [EH00] because it uses unfolding-based verification techniques. In fact, our practical results confirm this assumption. The LTL model-checking times for the reduced nets are much smaller than the verification times for the original systems. For instance, the model-checking time for the fifo(20) system can be decreased from 13 hours to only 3 seconds. Also the verification time for the slotring(10) protocol may be reduced from 12 hours to only 2 and a half by applying our reduction algorithm before the LTL verification. As one can see our optimized reduction algorithm takes only a few seconds, a negligible time compared to the actual verification times.

	Weakened Rules					Original Rules from literature				
	$\Sigma_{\neg\varphi}^{red}$		Unf($\Sigma_{\neg\varphi}^{red}$)			$\Sigma_{\neg\varphi}^{red}$		Unf($\Sigma_{\neg\varphi}^{red}$)		
	$\|S\|$	$\|T\|$	$\|B\|$	$\|E\|$	t_{LTL}	$\|S\|$	$\|T\|$	$\|B\|$	$\|E\|$	t_{LTL}
dph(7)	31	101	2971	1406	4.5	50	114	27922	13281	709.5
slotring(10)	67	68	44562	23901	8842.9	67	77	125351	62067	31674.9

Fig. 6. Experimental results with original rules

As mentioned in chapter 4.2 we have improved some of the local reduction rules compared to the rules known in the literature by weakening their conditions. This allows us to apply the rules more often and to obtain larger reductions. To confirm this assumption we have implemented the original rules just as taken from the literature and have conducted some experiments.

The results which are shown in Fig. 6 meet our expectations. Applying the original rules without weakening their conditions yields much larger unfolding sizes and LTL verification times. For instance, the dining philosophers system (dph(7)) can only be reduced to 50 places and 114 transitions (in contrast to 31 places and 101 transitions by applying our weakened rules). This leads to an unfolding which is ten times larger, and also the verification time for the LTL property grows from 5 seconds to 12 minutes. Applying the original and our weakened rules on the slotted ring protocol the reduced nets differ from 9 transitions. On the first look, this seems to be a negligible difference but indeed it has an considerable effect with respect to the unfolding size and the verification time. Preprocessing the net with our weakened rules causes that the unfolding is three times smaller, and that the verification time can be decreased from 9 hours to only 2 and a half. The unfolding of the reduced slotring(10) net obtained by applying the original rules is even larger than the unfolding of the original net. This is caused by the post-agglomeration rule which use can lead to larger unfoldings in some cases.

The reduction ratio for a system depends on the LTL property to be checked. All places corresponding to atomic propositions of the formula and all their input and output transitions remain untouched by our reduction rules. This entails that the number of places and transitions that must not be removed from the system depends on the number of places corresponding to the atomic propositions in the LTL property.

Altogether, our results have shown that our reduction algorithm yields a very efficient and suitable preprocessing technique for LTL model-checking.

7 Conclusions

We have presented techniques that allow to remove redundancies from systems modelled as 1-safe Petri nets. These techniques are of two kinds: (1) Linear programming techniques for detecting subinvariants and implicit places, and (2) local net reductions. We have shown that the conditions for some local net reductions known from the literature can be weakened if one is interested in model-checking LTL using the approach of Esparza and Heljanko [EH00,EH01]. Moreover, we have presented a number of experimental results. These results have confirmed that many redundancies may be detected and removed with our reduction techniques. Furthermore the results have shown that our reduction algorithm runs very fast, and that it has an considerable effect on the model-checking algorithm of [EH00,EH01]. Altogether, our reductions seem to be a good preprocessing technique for model-checking LTL with the approach of [EH00, EH01]. Due to space limitations we have omitted the proofs of the Theorems. The full version of this paper (including all proofs) is available in [ES01].

Acknowledgements. We would like to thank Keijo Heljanko for valuable comments and sending us an implementation of the LTL model-checker.

References

[Ber85] G. Berthelot. Checking properties of nets using transformations. In *Advances in Petri Nets*, LNCS 222, pages 19 – 40. Springer-Verlag, 1985.

[Cor94] J. C. Corbett. Evaluating Deadlock Detection Methods, 1994.

[CS90] J. M. Colom and M. Silva. Improving the Linearly Based Characterization of P/T Nets. In *Advances in Petri Nets*, LNCS 483, pages 113 – 145. Springer-Verlag, 1990.

[DE95] J. Desel and J. Esparza. Free Choice Petri Nets. Cambridge University Press, 1995.

[EH00] J. Esparza and K. Heljanko. A new Unfolding Approach to LTL Model Checking. In *ICALP'00*, LNCS 1853, pages 475 – 486. Springer-Verlag, 2000.

[EH01] J. Esparza and K. Heljanko. Implementing LTL Model Checking with Net Unfoldings. Accepted paper for SPIN'01, 2001.

[ES01] J. Esparza and C. Schröter. Net Reductions for LTL Model-Checking. Full version, available at `ftp://131.159.22.8/pub/theory/schroete/reduct.ps.gz`, 2001.

[HD95] M. Heiner and P. Deusen. Petri net based qualitative analysis - A case study. Technical report I-08/1995. Brandenburg Technische Universität Cottbus, 1995, 1995.

[Hel93] H. Hellwagner. Scalable Readers/Writers Synchronization on Shared-Memory Machines. Esprit P5404 (GP MIMD), Working Paper, 1993.

[Hel01] K. Heljanko. Unfsmodels 0.9. Available at `http://www.tcs.hut.fi/~kepa/experiments/spin2001/`, 2001.

[LL95] C. Lewerentz and T. Lindner. Formal Development of Reactive Systems: Case Study Production Cell. LNCS 891. Springer-Verlag, 1995.

[Mar86] A. J. Martin. Self-timed FIFO: An exercise in compiling programs into VLSI circuits. In *From HDL Descriptions to Guruanteed Correct Circuit Designs*, pages 133 – 153. Elsevier Science Publishers, 1986.

[McM92] K. L. McMillan. Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits. In *CAV'92*, LNCS 663, pages 164 – 174. Springer-Verlag, 1992.

[PPP00] D. Poitrenaud and J. F. Pradat-Peyre. Pre- and Post-agglomerations for LTL Model Checking. In *ICATPN'00*, LNCS 1825, pages 387 – 408. Springer-Verlag, 2000.

[PRCB94] E. Pastor, O. Roig, J. Cortadella, and R. M. Badia. Petri Net Analysis Using Boolean Manipulation. In *ATPN'94*, LNCS 815, pages 416 – 435. Springer-Verlag, 1994.

[RCP95] O. Roig, J. Cortadella, and E. Pastor. Verification of Asynchronous Circuits by BDD-based Model Checking of Petri Nets. In *ATPN'95*, LNCS 935, pages 374 – 391. Springer-Verlag, 1995.

[Rei85] W. Reisig. Petri Nets. Volume 4 of the EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[Röm00] S. Römer. *Theorie und Praxis der Netzentfaltungen als Grundlage für die Verifikation nebenläufiger Systeme*. PhD thesis, Tech. Univ. München, 2000.

[Var96] M. Y. Vardi. An automata theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, LNCS 1043, pages 238 – 265. Springer-Verlag, 1996.

[Wal98] F. Wallner. Model checking LTL using net unfoldings. In *CAV'98*, LNCS 1427, pages 207 – 218. Springer-Verlag, 1998.