Bounded Reachability Checking with Process Semantics *

Keijo Heljanko

Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O. Box 5400, FIN-02015 HUT, Finland
Keijo.Heljanko@hut.fi

Abstract Bounded model checking has been recently introduced as an efficient verification method for reactive systems. In this work we apply bounded model checking to asynchronous systems. More specifically, we translate the bounded reachability problem for 1-safe Petri nets into constrained Boolean circuit satisfiability. We consider three semantics: process, step, and interleaving semantics. We show that process semantics has often the best performance for bounded reachability checking.

1 Introduction

Bounded model checking [3] has been proposed as a verification method for reactive systems. The main idea is to look for counterexamples which are shorter than some fixed length for a given property. If a counterexample can be found, then the property does not hold for the system. If no counterexample can be found using this bound, usually the result is inconclusive.

The decision procedure most often used in bounded model checking is propositional satisfiability. Given the transition relation of the reactive system to be model checked, the property, and the bound n, the transition relation and property are "unrolled" n times to obtain a propositional formula which is satisfiable iff there is a counterexample with bound n. The implementation ideas are quite similar to procedures used in AI planning [11,15].

In this work we apply bounded model checking to asynchronous systems. More specifically, we translate the bounded reachability problem for 1-safe Petri nets into constrained Boolean circuit satisfiability. This work can be seen as a continuation of the work done in [9]. There we discuss using the step and interleaving semantics for bounded reachability, while the formalism into which the problem is translated being logic programs with stable model semantics. The main contribution of this paper is that we show that the so called process semantics of Petri nets [1,2] can be used to improve the efficiency of bounded model checking. Namely, also the process semantics can be efficiently encoded into constrained Boolean circuits.

* The financial support of the Academy of Finland (Projects 43963 and 47754), and Tekniikan Edistämissäätiö foundation are gratefully acknowledged.

© 2001/2002 Springer-Verlag Berlin Heidelberg. Reprinted, with permission, from
K.G. Larsen and M. Nielsen (Eds.): CONCUR 2001, LNCS 2154, pp. 218–232, 2001.

As an additional contribution we report on an implementation called punroll, which uses the BCSat constrained Boolean circuit satisfiability checker to check whether the generated constrained circuit is satisfiable, thus solving the bounded reachability problem.

The structure of the rest of the paper is the following. First we introduce Petri nets and the three different semantics in Sect. 2. Then we shortly introduce constrained Boolean circuits in Sect. 3, and in Sect. 4 show how the bounded reachability problem can be translated into them. After that we discuss our implementation and experiments in Sect. 5, and finish with conclusions in Sect. 6.

2 Petri nets

We will now introduce Petri nets. A *net* is a triple (P, T, F), where P and T are disjoint sets of *places* and *transitions*, respectively, and F is a function $(P \times T) \cup (T \times P) \rightarrow \{0, 1\}$. Places and transitions are generically called *nodes*. If $F(x, y) = 1$ then we say that there is an *arc* from x to y. The *preset* of a node x, denoted by ${}^\bullet x$, is the set $\{y \in P \cup T \mid F(y, x) = 1\}$. The *postset* of x, denoted by $x^\bullet$, is the set $\{y \in P \cup T \mid F(x, y) = 1\}$. In this paper we consider only finite nets in which every transition has a nonempty preset *and* a nonempty postset.

A *marking* of a net (P, T, F) is a mapping $P \rightarrow \mathbb{N}$ (where $\mathbb{N}$ denotes the natural numbers including 0). We identify a marking M with the multiset containing $M(p)$ copies of p for every $p \in P$. For instance, if $P = \{p_1, p_2\}$ and $M(p_1) = 1$, $M(p_2) = 2$, we write $M = \{p_1, p_2, p_2\}$. A 4-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) (called the *initial marking* of Σ). We will use as a running example the net system in Fig. 1.

2.1 Step Semantics

To save some space, we define the behavior of a net system through step semantics. The (usual) interleaving semantics will then be defined later based on this more general concept.

A step is a non-empty set of transitions $S \subseteq T$. [1] We denote a step by $[S\rangle$. A marking M *enables* a step S if for all $p \in P$ it holds that $M(p) \geq \sum_{t \in S} F(p, t)$. If the step S is enabled at M, then it can *fire* or *occur*, and its occurrence *leads to* a new marking M' defined as $M'(p) = M(p) + \sum_{t \in S}(F(t, p) - F(p, t))$ for every place $p \in P$. We denote this firing of a step by $M[S\rangle M'$.

A (possibly empty) sequence of steps $\sigma = [S_0\rangle[S_1\rangle \cdots [S_{n-1}\rangle$ is a *step execution* of the net system $\Sigma = (P, T, F, M_0)$ if there exist markings M_1, M_2, ... , M_n such that $M_0[S_0\rangle M_1[S_1\rangle \cdots M_{n-1}[S_{n-1}\rangle M_n$. The marking reached by the occurrence of σ is M_n. A marking M is a *reachable marking* if there exists a step execution σ such M is reached by the occurrence of σ. A marking M is *reachable with bound* n if there exists a step execution σ consisting of (exactly) n steps

[1] We only consider a class of nets where the transitions cannot be self-concurrent. Therefore a set suffices and multisets, i.e., bags are not needed.

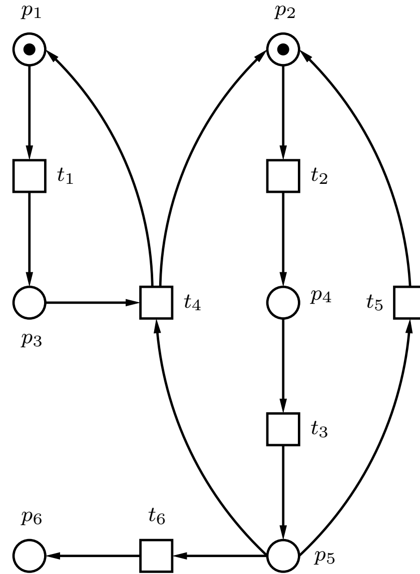

Figure 1. Running Example

such M is reached by the occurrence σ. Correspondingly we say that a marking M is *reachable within bound* n if there exists an integer $0 \leq i \leq n$ such that M is reachable with bound i.

In our running example the step $[t_1, t_2\rangle$ is enabled in the initial marking and thus $\{p_1, p_2\}[t_1, t_2\rangle\{p_3, p_4\}$. The marking $\{p_3, p_6\}$ is reachable with bound 3, as $\{p_1, p_2\}[t_2\rangle\{p_1, p_4\}[t_1, t_3\rangle\{p_3, p_5\}[t_6\rangle\{p_3, p_6\}$ is a step execution.

A marking M of a net is *n-safe* if $M(p) \leq n$ for every place p. A net system Σ is n-safe if all its reachable markings are n-safe. In this work we restrict ourselves to net systems which are 1-safe. They are quite an interesting class, as e.g., net systems arising from synchronization of state machines are 1-safe. Note that for 1-safe net systems all reachable markings are reachable within bound $n = (2^{|P|} - 1)$. Thus the set "marking reachable within bound n" can be seen as a lower approximation of the set of reachable markings which improves as the bound n increases. See discussion in [3] on how to check whether a bound is sufficient for completeness. Quite often a much smaller bound than the one discussed above suffices for completeness. For a general discussion of the computational complexity of verification problems for 1-safe Petri nets, see e.g., [6].

2.2 Interleaving Semantics

An *interleaving execution* is a step execution $M_0[S_0\rangle M_1[S_1\rangle \cdots M_{n-1}[S_{n-1}\rangle M_n$ such that for all $0 \leq i \leq n-1$ it holds that $|S_i| = 1$. A marking is *reachable in the interleaving semantics* if there exists an interleaving execution σ such that

M is reached by the occurrence of σ. The bounded versions of reachability are defined similarly to the step case.

Again in our example the marking $\{p_3, p_6\}$ is reachable in the interleaving semantics with a bound 4, as $\{p_1, p_2\}[t_1\rangle\{p_2, p_3\}[t_2\rangle\{p_3, p_4\}[t_3\rangle\{p_3, p_5\}[t_6\rangle\{p_3, p_6\}$ is an interleaving execution. Notice however, that the marking $\{p_3, p_6\}$ is *not* reachable in the interleaving semantics with bound 3.

It is well known, see e.g., [1] that for the net class used here the set of reachable markings in the step and interleaving semantics coincide. However, in bounded model checking using step semantics might be useful, as in many cases markings can be reached with a smaller bound than in the interleaving semantics.

2.3 Process Semantics

However, there is a problem with steps. Namely, there can be several step executions which intuitively represent the same "concurrent behavior". These can in bounded model checking introduce search space which can adversely effect the running time of the solver used. To avoid this we will use a well known semantics from the literature called the process semantics, see [1,2].

We will now recall from the literature a construction which constructs a process from a finite step execution. The following is a modified version (simpler because of 1-safeness) of the Construction 4.9 in [1].

For this definition we need some additional notation. For a net $N = (P, T, F)$ the function $Max(N) = \{x \in P \mid x^\bullet = \emptyset\}$. Let L be a finite set. A *labelled net* is a 4-tuple (P, T, F, l), where (P, T, F) is a net and $l\colon P \cup T \to L$ is a labelling.

Definition 1. *(Derivation of process from step execution.) Let* $\Sigma = (P, T, F, M_0)$ *be a net system and let* $\sigma = [S_0\rangle[S_1\rangle \cdots [S_{n-1}\rangle$ *be a sequence of steps such that* $M_0[S_0\rangle M_1[S_1\rangle \cdots [S_{n-1}\rangle M_n$ *is a step execution of* Σ. *We associate with* σ *a labelled net* $\Pi(\sigma)$ *by creating a sequence of labelled nets* $N_i = (B_i, E_i, G_i, l_i)$ *with labelling* $l_i : B_i \cup E_i \to P \cup T$ *by induction on* i, *where* $0 \le i \le n$.

> *(*$i = 0$*):* $E_0 = \emptyset$, $G_0 = \emptyset$, *and* B_0 *contains for each* $p \in P$ *such that* $M_0(p) = 1$ *a place* b *with* $l_0(b) = p$.
> *(*$i = i + 1$*): Suppose that* N_i *has been constructed.*
> *First we require that everything in* N_i *is also in* N_{i+1}. *For all* $x, y \in B_i \cup E_i$: $x \in B_i \Rightarrow x \in B_{i+1}$, $x \in E_i \Rightarrow x \in E_{i+1}$, $(x, y) \in G_i \Rightarrow (x, y) \in G_{i+1}$ *and* $l_{i+1}(x) = l_i(x)$.
> *Then for each* $t \in S_i$ *do the following:*
> - *for each* $p \in {}^\bullet t$ *find the place* $b(p) \in Max(N_i)$ *such that* $l_i(b(p)) = p$,
> - *add a new transition* e *to* E_{i+1} *with* $l_{i+1}(e) = t$ *and add* $(b(p), e)$ *to* G_{i+1} *for all* $p \in {}^\bullet t$,
> - *for each* $p \in t^\bullet$ *add a new place* $b'(p)$ *to* B_{i+1} *with* $l_{i+i}(b'(p)) = p$ *and* $(e, b'(p)) \in G_{i+1}$.

Finally take $\Pi(\sigma) = N_n = (B_n, E_n, G_n, l_n)$.

The construction above is fully deterministic (as this version is for 1-safe nets only) and thus the result is unique up to isomorphism. This fact is well known, see e.g., the discussion of a similar definition, Def. 3 in [12]. For simplicity, from now on we will identify all isomorphic processes as being equivalent.

Consider now our running example in Figure 1. It has a step execution $\{p_1, p_2\}[t_2\rangle\{p_1, p_4\}[t_1, t_3\rangle\{p_3, p_5\}[t_6\rangle\{p_3, p_6\}$. Now given $\sigma = [t_2\rangle[t_1, t_3\rangle[t_6\rangle$ we can construct the process $\Pi(\sigma)$ given in Figure 2, where the labelling l of nodes is given in parenthesis.

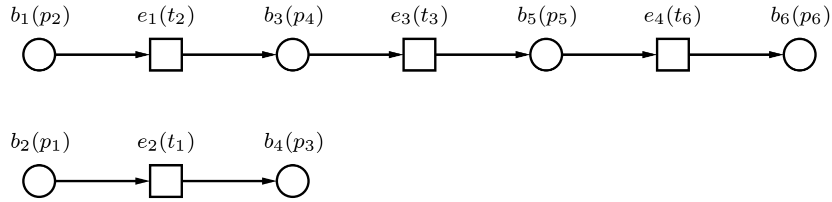

Figure 2. A process $\pi = (B, E, G, l)$

It is easy to see that for example also the sequences of steps $\sigma' = [t_1, t_2\rangle[t_3\rangle[t_6\rangle$, $\sigma'' = [t_2\rangle[t_3\rangle[t_1, t_6\rangle$, and $\sigma''' = [t_1\rangle[t_2\rangle[t_3\rangle[t_6\rangle$ will yield the same process, i.e., $\Pi(\sigma') = \Pi(\sigma'') = \Pi(\sigma''') = \Pi(\sigma)$. All of these step executions "solve the arising conflicts" in the same way and lead to the same final marking of the process π, i.e., $l(Max(\pi)) = \{p_3, p_6\}$. Thus if we are only interested in the final marking it should intuitively be sufficient to only generate one of them. We will now show how this can be done in bounded reachability checking.

We present an algorithm which given a process π gives a sequence of steps $FNF(\pi)$ (for *Foata normal form* of π) which together with Σ fully characterizes the process π. The Algorithm 1 computes the Foata normal form of a process. It is the algorithm presented on page 47 of [16] (with small notational changes). We define some notation for the algorithm. Given a set of transitions $C \subseteq E$ of the process $\pi = (B, E, G, l)$, let G^* be the transitive closure of the flow relation G, and define $MinE(C) = \{e \in C \mid \text{for all } e' \in (C \setminus \{e\}) \text{ it holds that } (e', e) \notin G^*\}$.

Assume that we are given a Foata normal form $FNF(\pi) = [S_0\rangle[S_1\rangle \cdots [S_{n-1}\rangle$ for a process π of a 1-safe net system Σ. It is easy to prove that there are markings $M_1, M_2, \ldots, M_n$ such that in the initial state M_0 of Σ the step execution $M_0[S_0\rangle M_1[S_1\rangle M_2 \cdots M_{n-1}[S_{n-1}\rangle M_n$ can occur.

This normal form is actually the Foata normal form from the theory of Mazurkiewicz traces, see e.g., [5]. It is only (quite trivially) adapted to processes of 1-safe net systems. To our knowledge it was first applied to processes of 1-safe net systems in the verification algorithm setting in [7]. (The fact that the technique used is a Foata normal form is discussed in more detail in an extended version [8], as well as in [16].)

Algorithm 1 *The Foata normal form of a process*

input: A process $\pi = (B, E, G, l)$ of a 1-safe net.
output: Foata normal form of π: A sequence of steps $FNF = [S_0\rangle[S_1\rangle \cdots [S_{n-1}\rangle$.

```
1  begin
2    C := E;
3    FNF := ϵ;
4    while C ≠ ∅ do
5      S := l(MinE(C));
6      FNF := FNF · [S⟩;
7      C := C \ MinE(C);
8    endwhile
9    return FNF;
10 end
```

When run on the process π of Figure 2, we will get the result $FNF(\pi) = [t_1, t_2\rangle[t_3\rangle[t_6\rangle$. This intuitively corresponds to a step execution which is "greedy", i.e., it always fires transitions at the earliest possible time moment, while still respecting the structure of the process π. Thus the step execution in Foata normal form is always among the shortest which yield the process π.

The Algorithm 1 gives an easy way of generating a Foata normal form of a process. We will in our implementation use a different definition, which is equivalent but more suitable for the implementation techniques we use. (We have not found this version in the literature. However, it is just a simple adaptation of the version for traces, see e.g., [5].)

Definition 2. *The sequence of steps $\sigma = [S_0\rangle[S_1\rangle \cdots [S_{n-1}\rangle$ is a step execution of a 1-safe net system Σ in Foata normal form if:*

(a) $\sigma = \epsilon$ (i.e., σ is the empty step sequence), or
(b) There are markings $M_1, M_2, \ldots, M_n$ such that in the initial state M_0 of Σ the step execution $M_0[S_0\rangle M_1[S_1\rangle M_2 \cdots M_{n-1}[S_{n-1}\rangle M_n$ can occur, and:
- *For each $1 \leq i \leq n-1$ and for each $t \in S_i$ there exists a transition t' in S_{i-1} such that $t'^{\bullet} \cap {}^{\bullet}t \neq \emptyset$. (Each transition t in step i with $i \geq 1$ has some transition t' in step $i-1$ which generates some part of its preset.)*

Now there is a bijection between processes and step executions in Foata normal form. Given a step execution σ one can construct the corresponding process $\pi = \Pi(\sigma)$, and given the process π we can construct the step execution $\sigma' = FNF(\pi)$ and in fact $\sigma' = \sigma$ iff σ was in Foata normal form (according to Def. 2). Thus they both describe the same concurrent behavior. It is therefore only a matter of taste whether one talks about processes or step executions in Foata normal form. We have chosen to talk about processes and process semantics, as that is the terminology most often used in Petri net literature [1,2]. Our actual implementation is, however, based on the definition of the Foata normal form for step executions, namely Def. 2.

We thus define the process semantics as follows. A marking M is a *reachable in the process semantics* if there exists a step execution σ in Foata normal

form, such that M is reached by the occurrence of σ. The bounded versions of reachability are again defined similarly to the step case.

To rephrase our discussion, here is the (not surprising) main result used in bounded model checking with process semantics.

Theorem 1. *Let Σ be a 1-safe net system. A marking M is reachable within bound n in Σ iff in the process semantics M is reachable within bound n in Σ.* [2]

3 Boolean Circuits

This section is largely based on the presentation of [10]. A *Boolean circuit* is an directed acyclic graph where the nodes are called *gates*. The gates with no outgoing edges are *output gates* and *input gates* are those gates which do not have incoming edges nor an associated Boolean function. Each non-input gate has a Boolean function associated with it and it "calculates" the output value from the values of its children. Boolean circuits can be expressed with *Boolean expression systems*. Given a finite set $\mathcal{V}$ of Boolean variables, a Boolean equation system $\mathcal{S}$ over $\mathcal{V}$ is a set of equations of the form $v = f(v_1, \ldots, v_k)$, where $v, v_1, \ldots, v_k \in \mathcal{V}$ and f is an arbitrary Boolean function. Boolean circuits can now be seen as Boolean equation systems with the following two properties. (i) Each variable has at most one equation. (ii) The equations are not recursive. (In the sense that the variable dependency graph [10] is acyclic.)

A *truth valuation* for $\mathcal{S}$ is a function $\tau : \mathcal{V} \to \{\text{true, false}\}$. A valuation is *consistent* if $\tau(v) = f(\tau(v_1), \ldots, \tau(v_k))$ for each equation in $\mathcal{S}$. The *constrained satisfiability problem* for Boolean circuits is the following: given that variables $c^+ \subseteq \mathcal{V}$ must be true and variables in $c^- \subseteq \mathcal{V}$ must be false, is there a consistent valuation that respects these constraints? We call such a truth assignment a *satisfying truth assignment*. The constrained Boolean circuit satisfiability problem is obviously an **NP**-complete problem under the plausible assumption that each Boolean function in the system can be evaluated in polynomial time.

In the rest of this paper we use Boolean circuits where the following Boolean functions are used as gates:

- $\top$ is always true.
- $\bot$ is always false.
- $\mathsf{not}(v) =$ true iff v is not true.
- $\mathsf{or}(v_1, \ldots, v_k) =$ true iff at least one of v_i, $1 \leq i \leq k$ is true.
- $\mathsf{and}(v_1, \ldots, v_k) =$ true iff all of v_i, $1 \leq i \leq k$ are true.
- $\mathsf{card}_L^U(v_1, \ldots, v_k) =$ true iff for the cardinality c of the set of variables v_i which are true it holds that $L \leq c \leq U$. (Where L and U are fixed constants $0 \leq L \leq U$.)

The function $\mathsf{card}_L^U(v_1, \ldots, v_k)$ is actually a family of functions. We use in this work only the special form $\mathsf{card}_0^1(v_1, \ldots, v_k)$, which is true if less than two of the variables in the set $\{v_1, \ldots, v_k\}$ are true. We will show that this function is quite useful for compactly encoding which transitions can not be fired concurrently.

[2] Note the use of *within* instead of *with*. A marking may be reachable with a bound n and only reachable with bound i in the process semantics, where $i < n$.

4 Translating Bounded Reachability into Boolean Circuits

We will now present how to translate the bounded reachability problem for 1-safe nets into constrained satisfiability problem for Boolean circuits. The Figures 3-5 give parts of the translation for our running example of Figure 1. We suggest the reader to consult them while reading the definition of the translation. Consider a 1-safe net system $\Sigma = (P, T, F, M_0)$ and a fixed bound n. We first construct (in (a)-(b) below) a constrained Boolean circuit which captures the possible step executions of Σ of length $\leq n$, where $n \geq 0$.

(a) To capture the initial marking, for each place $p_j \in P$ we create a gate $\mathsf{p_j}(0)$ and associate $\top$ as the function if $M_0(p_j) = 1$, and $\perp$ otherwise.

(b) For each step $0 \leq i \leq n-1$ we add the following gates:

1. For each transition $t_j \in T$ we create an input gate $\mathsf{t_j}(i)$. If this gate is true, it intuitively means that the transition t_j fires in step i.
2. For each place $p_j \in P$ we create an or gate $\mathsf{gp_j}(i+1)$ with the children $\{\mathsf{t_1}(i), \ldots, \mathsf{t_k}(i)\}$, where $\{t_1, \ldots, t_k\}$ is the preset of p_j. The gate $\mathsf{gp_j}(i+1)$ will be true if some transition in step i generates a token to the place p_j.
3. For each place $p_j \in P$ we create an or gate $\mathsf{rp_j}(i+1)$ with the children $\{\mathsf{t_1}(i), \ldots, \mathsf{t_k}(i)\}$, where $\{t_1, \ldots, t_k\}$ is the postset of p_j. The gate $\mathsf{rp_j}(i+1)$ will be true if some transition in step i removes a token from p_j.
4. For each place $p_j \in P$ we create a not gate $\mathsf{nrp_j}(i+1)$ with the child $\mathsf{rp_j}(i+1)$.
5. For each place $p_j \in P$ we create an and gate $\mathsf{fp_j}(i+1)$ with the children $\mathsf{p_j}(i)$ and $\mathsf{nrp_j}(i+1)$. The gate $\mathsf{fp_j}(i+1)$ is true when a place p_j contains a token before step i, and no transition removing tokens from it appears in step i.
6. For each place $p_j \in P$ we create an or gate $\mathsf{p_j}(i+1)$ with the children $\mathsf{gp_j}(i+1)$ and $\mathsf{fp_j}(i+1)$. The gate $\mathsf{p_j}(i+1)$ is true when after step i the place p_j contains a token. (Either a token was generated in step i or a token residing on the place p_j before step i still remains on the place p_j after the step i.)
7. For each transition $t_j \in T$ we create an and gate $\mathsf{pt_j}(i)$ with the children $\{\mathsf{p_1}(i), \ldots, \mathsf{p_k}(i)\}$, where $\{p_1, \ldots, p_k\}$ is the preset of t_j. The gate $\mathsf{pt_j}(i)$ will be true if all the preset places of transition t_j in step i contain a token.
8. For each transition $t_j \in T$ we create a not gate $\mathsf{nt_j}(i)$ with the child $\mathsf{t_j}(i)$.
9. For each transition $t_j \in T$ we create an or gate $\mathsf{tt_j}(i)$ and constrain it to be true. It has two children $\mathsf{nt_j}(i)$ and $\mathsf{pt_j}(i)$. The constrained gate $\mathsf{tt_j}(i)$ ensures that either the transition t_j is not fired in step i or all of its preset tokens are available.
10. For each place $p_j \in P$ such that $|p^\bullet| \geq 2$ we create a card_0^1 gate $\mathsf{ncp_j}(i)$ and constrain it to true. It has children $\{\mathsf{t_1}(i), \ldots, \mathsf{t_k}(i)\}$, where $\{t_1, \ldots, t_k\}$ is the postset of p_j. The constrained gate $\mathsf{ncp_j}(i)$ ensures

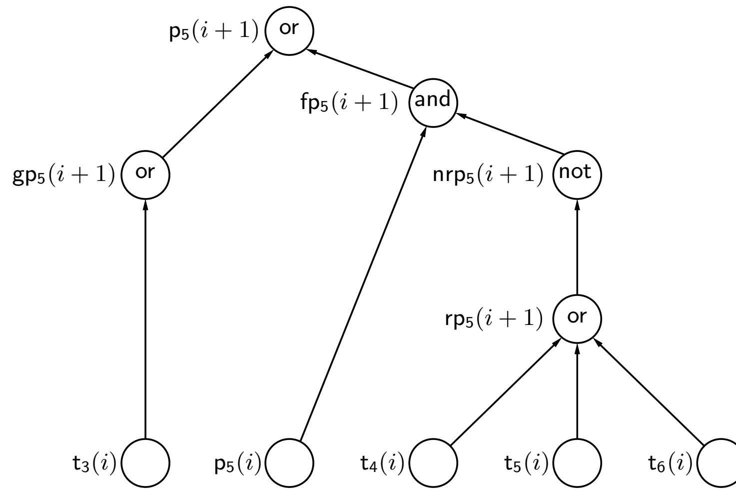

Figure 3. Example: translation for the place p_5

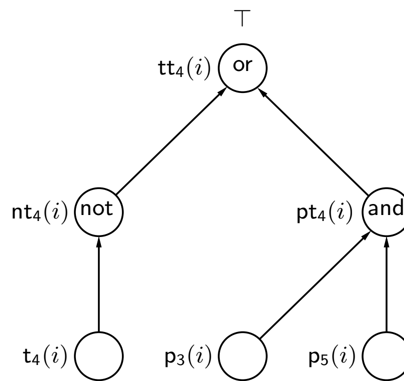

Figure 4. Example: translation for the transition t_4

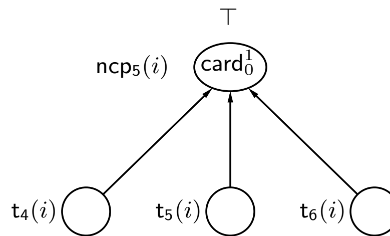

Figure 5. Example: translation of the conflicts with respect to place p_5

that at most one of the transitions which have the place p_j in preset can appear in step i. We say that this set of transitions is *in conflict* with respect to the place p_j.

The translation (a)-(b) as given above allows for "idle steps" in which no transition occurs. Thus the program encodes all the step executions of length n or less. We have chosen to remove the possibility of idling steps in our implementation.[3] Thus we always add the following gates to the system:

(c) For each step $0 \leq i \leq n-1$ add an or gate $\mathsf{ni}(i)$ (for non-idle) and constrain it to true. It has the children $\{\mathsf{t_1}(i), \ldots, \mathsf{t_k}(i)\}$, where $\{t_1, \ldots, t_k\} = T$. Thus the gate $\mathsf{ni}(i)$ will be true if at least one transition fires in step i.

We denote by $SC(\Sigma, n)$ (for step circuit) the translation given by (a)-(c). Given a valuation τ of the circuit $SC(\Sigma, n)$, we can obtain the corresponding sequence of markings and steps $M_0, [S_0\rangle, M_1, [S_1\rangle, \ldots, M_{n-1}, [S_{n-1}\rangle, M_n$ by having transition $t_j \in S_i$ iff $\mathsf{t_j}(i)$ is true, and $p_j \in M_i$ iff $\mathsf{p_j}(i)$ is true. Because gates of form $\mathsf{t_j}(i)$ are the only input gates, the mapping from sequences of steps to consistent truth valuations is in fact a bijection.

Lemma 1. *The constrained Boolean circuit $SC(\Sigma, n)$ has a satisfying truth assignment τ iff $M_0[S_0\rangle M_1[S_1\rangle \cdots M_{n-1}[S_{n-1}\rangle M_n$ is a step execution of Σ, where $M_0, [S_0\rangle, M_1, [S_1\rangle, \ldots, M_{n-1}, [S_{n-1}\rangle, M_n$ is the sequence of markings and steps corresponding to τ.*

Thus we get our main result.

Theorem 2. *The constrained Boolean circuit $SC(\Sigma, n)$ encodes step executions of length n.*

4.1 The Interleaving Semantics

Sometimes we would also like to consider the interleaving semantics. It is easy to add a set of constrained gates to the circuit which disallow non-singleton steps.

(i) For each step $0 \leq i \leq n-1$ add an card_0^1 gate $\mathsf{nc}(i)$ (for non-concurrent) and constrain it to true. It has the children $\{\mathsf{t_1}(i), \ldots, \mathsf{t_k}(i)\}$, where $\{t_1, \ldots, t_k\} = T$. Thus the gate $\mathsf{nc}(i)$ will be true if at most one transition fires in step i.

We call the translation given by (a)-(c),(i) the interleaving circuit $IC(\Sigma, n)$.

Theorem 3. *The constrained Boolean circuit $IC(\Sigma, n)$ encodes interleaving executions of length n.*

[3] Here the semantics of the translation differs from the one presented in [9].

4.2 The Process Semantics

The translation for the process semantics is the main contribution of this paper. The main idea behind it is to modify the translation for step semantics in such a way that all step executions which are not in Foata normal form are disallowed.

If one looks at Def. 2 it is easy to see that each transition t in step S_i (not including the initial step S_0) has to have at least one transition t' in step S_{i-1} which generates at least one token to the preset of t. It is now straightforward to enforce this in a local way.

We change the preset of a transition in the following way. The part (b) of the translation is replaced by (b'), which is identical to (b) except that 7 is replaced by the 7' and 7" (see Figure 6 for an example):

(b') For each step $0 \leq i \leq n-1$ we add the following gates (1-6,8-10 omitted):

7'. For each transition $t_j \in T$ we create an or gate $\mathsf{dpt_j}(i)$ (for disjunctive preset) with the children $\{\mathsf{gp_1}(i), \ldots, \mathsf{gp_k}(i)\}$, where $\{p_1, \ldots, p_k\}$ is the preset of t_j. The gate $\mathsf{dpt_j}(i)$ will be true if a token was generated to some preset place of transition t_j in step $i-1$. (The previous step!)

7". For each transition $t_j \in T$ we create an and gate $\mathsf{pt_j}(i)$ with the children $\{\mathsf{p_1}(i), \ldots, \mathsf{p_k}(i), \mathsf{dpt_j}(i)\}$, where $\{p_1, \ldots, p_k\}$ is the preset of t_j. The gate $\mathsf{pt_j}(i)$ will be true if all the preset places of transition t_j in step i contain a token and the transition is locally in Foata normal form.

Note that the child gates of gates added by 7' already existed in the step translation as they are generated by 2. The 7" is almost identical to 7 except that the gate created in 7' has been added to the list of children. The gate generated by 7" now assures that both the preset of the transition is available *and* the transition is locally in Foata normal form. These local constraints on transition enabledness together imply that the step execution will as a whole be in Foata normal form (again according to Def. 2).

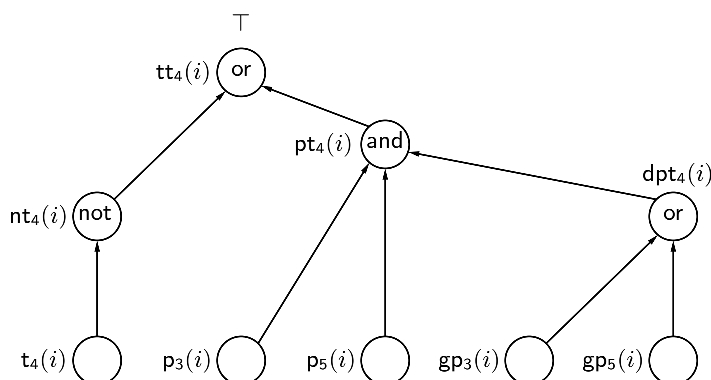

Figure 6. Example: process semantics translation of t_4

As in Def. 2, the initial step is special.

(p) For each place $p_j \in P$ we create a gate $\mathsf{gp_j}(0)$ and associate $\top$ with it.

We call the translation given by (a),(p),(b'),(c) the process circuit $PC(\Sigma, n)$. We say that a process π has depth n if the corresponding Foata normal form step execution $FNF(\pi)$ has length n. We have the following result.

Theorem 4. *The constrained Boolean circuit $PC(\Sigma, n)$ encodes processes of depth n.*

4.3 Checking Reachability

We have presented three translations which encode executions with bound n in different semantics. We can now add any Boolean constraint on the final marking M, as given by the syntax $f ::= p \in P \mid \neg f_1 \mid f_1 \vee f_2 \mid f_1 \wedge f_2$. Given a parse tree of the formula f, we convert it to a Boolean circuit $FC(f, n)$ of same size by replacing each atomic proposition $p \in P$ by the gate $\mathsf{p}(n)$, and all other formula types with the corresponding gates having the same children as in the parse tree. Finally the top-level gate f is constrained to true.

Theorem 5. *Let $C(\Sigma, n)$ be one of $PC(\Sigma, n)$, $SC(\Sigma, n)$, $IC(\Sigma, n)$. The constrained Boolean circuit $RC(\Sigma, f, n) = C(\Sigma, n) \cup FC(f, n)$ has a satisfying truth assignment iff there exists a marking M which satisfies f and is reachable in Σ with bound n in (process, step, interleaving) semantics.*

The size of each translation $RC(\Sigma, f, n)$ as the sum of number of gates and connections between them is linear, i.e., $\mathcal{O}((n \cdot (|P| + |T| + |F|)) + |f|)$.[4]

5 Experimental Results

We have implemented the reachability translations described in the previous section in a tool called punroll (for process unroller). We have implemented the following optimization which simplifies away places (transitions) which can never have a token (can never fire). For each step $0 \leq i \leq n - 1$:

(i) For each transition $t_j \in T$: If for some place $p \in {}^\bullet t_j$ the gate $\mathsf{p}(i)$ has function $\bot$ associated with it (or alternatively in the process semantics: for all places $p \in {}^\bullet t_j$ the gate $\mathsf{gp}(i)$ has function $\bot$ associated with it), then associate gate $\mathsf{t_j}(i)$ with function $\bot$.

(ii) For each place $p_j \in T$: If for all transitions $t \in {}^\bullet p_j$ the gate $\mathsf{t}(i)$ is associated with $\bot$, then associate the gate $\mathsf{gp_j}(i + 1)$ with $\bot$.

(iii) For each place $p_j \in T$: If both gates $\mathsf{p_j}(i)$ and $\mathsf{gp_j}(i + 1)$ are associated with $\bot$, then associate $\mathsf{p_j}(i + 1)$ with $\bot$.

[4] This bound also holds if we restrict ourselves to Boolean circuits without card_0^1 gates, because in principle each card_0^1 gate with k children can be simulated with (a simple ripple-carry adder style) circuit of size $\mathcal{O}(k)$ which contains only and and or gates.

(iv) Simplify the circuits of step i by substituting $\perp$ when associated by (i)-(iii).

The punroll tool can also add a constraint which requires that the marking reached is a deadlock, as given by the property $f = dead = \neg \bigvee_{t \in T} \bigwedge_{p \in {}^\bullet t} p$.

As a constrained satisfiability checker for Boolean circuits we use BCSat [10]. It operates internally on Boolean circuits, and also directly supports card_0^1 gates. BCSat is available from: ⟨`http://www.tcs.hut.fi/~tjunttil/bcsat/`⟩.

We use a set of deadlock checking benchmarks collected by Corbett [4]. They have been converted from communicating state machines to nets by Melzer and Römer [13]. The BYZA4_2A example is an exception to this rule, it is from [14]. The models were picked by choosing the nontrivial ones which have a deadlock.

For each model and all three semantics we incremented the used bound n until a deadlock was found. After that we stored the translation using that bound, and report the time for BCSat 0.3 to find the first satisfying truth assignment. In some cases a satisfying truth assignment could not be found within a reasonable time in which case we report the time used to prove that there are no satisfying truth assignments for the circuit with bound n.

The experimental results can be found in Fig. 7. The columns are:

- Problem: The problem name with the size of the instance in parenthesis.
- $|P|$: Number of places in the net.
- $|T|$: Number of transitions in the net.
- Pr. n: The smallest integer n such that a deadlock could be found using the process semantics / in case of $> n$ the largest integer n for which we could prove that there is no deadlock with that bound using the process semantics.
- Pr. s: The time in seconds to find the first satisfying truth assignment / to prove that there is no satisfying truth assignment. (See Pr. n above.)
- St. n and St. s: same as Pr. n and Pr. s but for the step semantics.
- Int. n and Int. s: same as Pr. n and Pr. s but for the interleaving semantics.
- States: Number of reachable states of the net system, or a lower bound $> n$.[5]

The times reported are the average of 5 runs as reported by the `/usr/bin/time` command on a Linux PC with an AMD Athlon 1GHz processor, 512MB RAM.

The set of experiments we used is too small to say anything conclusive about the performance of the method. There are, however, still some interesting observations to be made. In the experiments the process and step semantics often allow to use a smaller bound to find a deadlock. This partly explains their better performance when compared to the interleaving semantics. The process semantics has better performance than step semantics on e.g., BYZA4_2A, KEY(2), and MMGT(4). Several of the benchmarks (14 out of the 54 circuits used) were solved "with preprocessing" by BCSat, for example DARTES(1) in all semantics. The KEY(x) examples do not have a large number of reachable states, but seem to be still hard for bounded model checking, the results also indicate the reverse to be sometimes true, see e.g., BYZA4_2A with process semantics.

The punroll tool, the net systems, and the circuits used are available from: ⟨`http://www.tcs.hut.fi/~kepa/experiments/Concur2001/`⟩.

[5] These differ from the ones reported in [9], where there unfortunately are some errors.

Problem	$\|P\|$	$\|T\|$	Pr. n	Pr. s	St. n	St. s	Int. n	Int. s	States
BYZA4_2A	579	473	8	5.6	8	179.8	>7	6.8	>2500000
DARTES(1)	331	257	32	0.1	32	1.5	32	1.5	>1500000
DP(12)	72	48	1	0.0	1	0.0	12	1.5	531440
ELEV(1)	63	99	4	0.0	4	0.0	9	0.6	163
ELEV(2)	146	299	6	0.0	6	0.2	12	12.7	1092
ELEV(3)	327	783	8	0.4	8	2.7	15	126.5	7276
ELEV(4)	736	1939	10	5.4	10	67.7	>13	560.5	48217
HART(25)	127	77	1	0.0	1	0.0	>5	0.3	>1000000
HART(50)	252	152	1	0.0	1	0.0	>5	1.3	>1000000
HART(75)	377	227	1	0.0	1	0.0	>5	3.2	>1000000
HART(100)	502	302	1	0.0	1	0.0	>5	5.9	>1000000
KEY(2)	94	92	36	22.7	>27	76.0	>27	30.1	536
KEY(3)	129	133	>30	179.0	>27	198.6	>27	47.3	4923
KEY(4)	164	174	>27	32.9	>27	221.0	>27	58.7	44819
MMGT(2)	86	114	6	0.1	6	0.2	8	1.3	816
MMGT(3)	122	172	7	0.4	7	1.0	10	40.4	7702
MMGT(4)	158	232	8	2.9	8	253.6	>11	476.0	66308
Q(1)	163	194	9	0.1	9	0.2	>17	660.5	123596

Figure 7. Experiments

6 Conclusions

We have presented how bounded reachability checking for 1-safe Petri nets can be done using constrained Boolean circuits. For step and interleaving semantics these translations can be seen as circuit versions of the logic program translations in [9]. The process semantics translation is new and is based on the notion of Foata normal form for step executions. We have created on an implementation called punroll. We report on a set of benchmarks, where the BCSat tool is used to find whether the constrained circuit is satisfiable or not. The experiments seem to indicate that the process semantics translation is often the most competitive one.

It should be quite straightforward to also use other forms of concurrency than 1-safe net systems with process semantics. The crucial point is to be able to encode the constraints needed for a step execution to be in a Foata normal form in a local manner.

The close connection of bounded reachability checking to AI planning techniques [11,15] needs to be investigated further. It might be useful to use stochastic methods [11] in the verification setting. Also applying process semantics for AI planning needs to be investigated. (Step semantics has been used in [15].)

There are interesting topics for further research. We would like to extend the tool to handle bounded LTL model checking [3]. For interleaving semantics this is quite straightforward, but there are some subtle issues with step and process semantics which need to be solved.

Acknowledgements The author would like to warmly thank T. A. Junttila and I. Niemelä for creating BCSat, and for fruitful discussions.

References

1. E. Best and R. Devillers. Sequential and concurrent behaviour in Petri net theory. *Theoretical Computer Science*, 55(1):87–136, 1987.
2. E. Best and C. Fernández. *Nonsequential Processes: A Petri Net View*, volume 13 of *EATCS monographs on Theoretical Computer Science*. Springer-Verlag, 1988.
3. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, pages 193–207. Springer, 1999. LNCS 1579.
4. J. C. Corbett. Evaluating deadlock detection methods for concurrent software. Technical report, Department of Information and Computer Science, University of Hawaii at Manoa, 1995.
5. V. Diekert and Y. Métivier. Partial commutation and traces. In *Handbook of formal languages, Vol. 3*, pages 457–534. Springer, Berlin, 1997.
6. J. Esparza. Decidability and complexity of Petri net problems – An introduction. In *Lectures on Petri Nets I: Basic Models*, pages 374–428. Springer-Verlag, 1998. LNCS 1491.
7. J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Proceedings of 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 87–106, 1996. LNCS 1055.
8. J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm, 2001. Accepted for publication in Formal Methods for System Design.
9. K. Heljanko and I. Niemelä. Answer set programming and bounded model checking. In *Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, pages 90–96, Stanford, USA, March 2001. AAAI Press, Technical Report SS-01-01.
10. T. A. Junttila and I. Niemelä. Towards an efficient tableau method for Boolean circuit satisfiability checking. In *Computational Logic – CL 2000; First International Conference*, pages 553–567, London, UK, 2000. LNCS 1861.
11. H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201. AAAI Press / MIT Press, 1996.
12. H. C. M. Kleijn and M. Koutny. Process semantics of P/T-nets with inhibitor arcs. In *Proceedings of the 21st International Conference on Application and Theory of Petri Nets*, pages 261–281, 2000. LNCS 1825.
13. S. Melzer and S. Römer. Deadlock checking using net unfoldings. In *Proceedings of 9th International Conference on Computer-Aided Verification (CAV '97)*, pages 352–363, 1997. LNCS 1254.
14. S. Merkel. Verification of fault tolerant algorithms using PEP. Technical Report TUM-19734, SFB-Bericht Nr. 342/23/97 A, Technische Universität München, München, Germany, 1997.
15. I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.
16. S. Römer. *Theorie und Praxis der Netzentfaltungen als Basis für die Verifikation nebenläufiger Systeme*. PhD thesis, Technische Universität München, Fakultät für Informatik, München, Germany, 2000.