Minimizing Finite Complete Prefixes

Keijo Heljanko*

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

September 13, 1999

Abstract

Finite complete prefixes are used as a verification method for Petri nets and other formalisms where a similar notion of partial order behavior can be applied. They were introduced by McMillan, who also gave an algorithm to generate a finite complete prefix from a system description given as a Petri net. Later Esparza, Römer and Vogler improved McMillan's prefix generation algorithm, which could sometimes create exponentially larger prefixes than required. In this work we refine the approach of Esparza et.al. further, and define a refined cut-off criterion, which makes it sometimes possible to create much smaller finite complete prefixes. Experimental results from a prototype implementation, a prefix minimizer, are presented. The method can also be applied directly during prefix generation.

1 Introduction

Petri nets are a widely used model for analyzing concurrent and distributed systems. Finite complete prefixes are used as a verification method for Petri nets and other formalisms where a similar notion of partial order behavior can be applied. They were introduced by McMillan [11, 12], who also gave an algorithm to generate a finite complete prefix from a system description given as a Petri net. Later Esparza, Römer and Vogler improved McMillan's prefix generation (i.e. unfolding) algorithm [5], which could sometimes create exponentially larger prefixes than required. In this work we refine the approach of Esparza et.al. further, and define a refined cut-off criterion, which makes it sometimes possible to create much smaller finite complete prefixes.

*Currently visiting Technische Universität München, Fakultät für Informatik

The rest of the paper is divided as follows. First we present Petri net notations used in the paper. The Sect. 3 present the main result of this work, a refined cut-off criterion for finite complete prefixes. In Sect. 4 we will introduce the rule-based constraint programming framework. The next section deals with reachability checking in prefixes, which we need for the implementation. Section 6 discusses the implementation, and gives experimental results.

2 Petri Net Definitions

First we define basic Petri net notations. Next we introduce *occurrence nets*, which are Petri nets of a restricted form. Then *branching processes* are given as a way of describing partial order semantics for Petri nets. We define *finite complete prefixes* as a way of giving a finite representation of this partial order behavior. We follow mainly the notation of [5, 13].

2.1 Petri Nets

A triple $\langle S, T, F \rangle$ is a *net* if $S \cap T = \emptyset$ and $F \subseteq (S \times T) \cup (T \times S)$. The elements of S are called *places*, and the elements of T *transitions*. Places and transitions are also called *nodes*. We identify F with its characteristic function on the set $(S \times T) \cup (T \times S)$. The *preset* of a node x, denoted by ${}^\bullet x$, is the set $\{y \in S \cup T \mid F(y, x) = 1\}$. The *postset* of a node x, denoted by $x^\bullet$, is the set $\{y \in S \cup T \mid F(x, y) = 1\}$. Their generalizations on sets of nodes $X \subseteq S \cup T$ are defined as ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$, and $X^\bullet = \bigcup_{x \in X} x^\bullet$ respectively.

A *marking* of a net $\langle S, T, F \rangle$ is a mapping $S \mapsto \mathbb{N}$. A marking M is identified with the multi-set which contains $M(s)$ copies of s for every $s \in S$. A 4-tuple $\Sigma = \langle S, T, F, M_0 \rangle$ is a *net system* if $\langle S, T, F \rangle$ is a net and M_0 is a marking of $\langle S, T, F \rangle$. A marking M enables a transition t if $\forall s \in S : F(s, t) \leq M(s)$. If t is enabled, it can *occur* leading to a new marking (denoted $M \xrightarrow{t} M'$), where M' is defined by $\forall s \in S : M'(s) = M(s) - F(s, t) + F(t, s)$. A marking M_n is *reachable* in Σ iff there exist a sequence of transitions $t_1, t_2, \ldots, t_n$ and markings $M_1, M_2, \ldots, M_{n-1}$ such that: $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots M_{n-1} \xrightarrow{t_n} M_n$. A reachable marking is 1-safe if $\forall s \in S : M(s) \leq 1$. A net system Σ is 1-safe if all its reachable markings are 1-safe. In this work we will restrict ourselves to the set of net systems which are 1-safe, have a finite number of places and transitions, and also in which each transition $t \in T$ has both nonempty pre- and postsets.

2.2 Occurrence Nets

We use $\leq_F$ to denote the reflexive transitive closure of F. Let $\langle S, T, F \rangle$ be a net and let $x_1, x_2 \in S \cup T$. The nodes x_1 and x_2 are in *conflict*, denoted by $x_1 \# x_2$, if there exist $t_1, t_2 \in T$ such that $t_1 \neq t_2$, ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, $t_1 \leq_F x_1$, and $t_2 \leq_F x_2$. An occurrence net is a net $N = \langle B, E, F \rangle$ such that:

- $\forall b \in B : |{}^\bullet b| \leq 1$,

- F is acyclic, i.e. the irreflexive transitive closure of F is a partial order,
- N is finitely preceded, i.e. for any node x of the net, the set of nodes y such that $y \leq_F x$ is finite, and
- $\forall x \in S \cup T : \neg(x \# x)$.

The elements of B and E are called *conditions* and *events*, respectively. The set $Min(N)$ denotes the set of minimal elements of the transitive closure of F. A *configuration* C of an occurrence net is a set of events satisfying:

- If $e \in C$ then $\forall e' \in E : e' \leq_F e$ implies $e' \in C$ (C is causally closed),
- $\forall e, e' \in C : \neg(e \# e')$ (C is conflict-free).

A local configuration $[e]$ of an event e is the set of events e', such that $e' \leq_F e$.

2.3 Branching Processes

Branching processes are "unfoldings" of net systems and were introduced by Engelfriet [3]. Let $N_1 = \langle S_1, T_1, F_1 \rangle$ and $N_2 = \langle S_2, T_2, F_2 \rangle$ be two nets. A *homomorphism* is a mapping $S_1 \cup T_1 \mapsto S_2 \cup T_2$ such that: $h(S_1) \subseteq S_2 \wedge h(T_1) \subseteq T_2$, and for all $t \in T_1$, the restriction of h to ${}^\bullet t$ is a bijection between ${}^\bullet t$ and ${}^\bullet h(t)$, and similarly for $t^\bullet$ and $h(t)^\bullet$. A *branching process* of a net system Σ is a tuple $\beta = \langle N', h \rangle$, where N' is a occurrence net, and h is a homomorphism from N' to $\langle S, T, F \rangle$ such that: the restriction of h to $Min(N')$ is a bijection between $Min(N')$ and M_0, and $\forall e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2 \wedge h(e_1) = h(e_2)$ then $e_1 = e_2$. The set of places associated with a configuration C of β is denoted by $Mark(C) = h((Min(N) \cup C^\bullet) \setminus {}^\bullet C)$.

It is shown in [3] that a net has a maximal branching process up to isomophism, called the *unfolding* of the system. Given a configuration C of the unfolding, we denote by $C \oplus E$ the set $C \cup E$ (the *extension* of C by E), under the condition that $C \cup E$ is a configuration of the unfolding satisfying $C \cap E = \emptyset$. We also use the same notation $\beta \oplus E$ on braching processes to denote the extension of a branching process β by the set of events E and their postset conditions.

2.4 Finite Complete Prefixes

A finite branching process β is a *finite complete prefix* of a net system Σ iff for each reachable marking M of Σ there exists a configuration C of β such that:

- $Mark(C) = M$, and
- for every transition t enabled in M there exists a configuration $C \cup \{e\}$ such that $e \notin C$ and $h(e) = t$.

Algorithms to obtain a finite complete prefix β given a 1-safe net system Σ are presented in e.g. [5, 11, 12]. The algorithms will mark some events of the prefix β as special *cut-off events*, which we denote by the set $CutOffs(\beta) \subseteq E$. These algorithms have as a parameter a special partial order on the configurations of the unfolding called the *adequate order*. Due to space limitations we direct the interested reader to [5, 11, 12, 13].

3 Refining the Cut-off Criterion

The section represents the main contribution of this paper, a refined cut-off criterion for finite complete prefixes. For finite complete prefix generation (i.e. unfolding) we use exactly the approach as Esparza et.al. in [5]. The only change we do to their approach is refining the following definition of a cut-off event:

Definition 3.1 *Let $\prec$ be an adequate order on the configurations of the unfolding of a net system. Let β be prefix of the unfolding containing an event e. The event e is a* cut-off event *of β iff β contains a local configuration $[e']$ such that:*

(a) $\mathit{Mark}([e]) = \mathit{Mark}([e'])$, and

(b) $[e'] \prec [e]$.

The intuition behind cutoff events is that for each cut-off event e there already exists another corresponding event e' in the prefix. The markings reachable after executing e can also be reached after executing e', and thus the markings after e need not to be considered any further.

We suggest that the corresponding configuration doesn't need to be a local configuration of another event, but any configuration which contains no cut-off events, and fills the other requirements, will do. Unfortunately this leads to an inductive, and not as intuitive notion of a cut-off events:

Definition 3.2 *Let $\prec$ be an adequate order on the configurations of the unfolding of a net system. We define the cut-off events of a prefix β inductively on the number of events in the prefix. Let $e_1, e_2, \ldots, e_k$ be a sequence of all the events of β, sorted in a non-decreasing $\prec$ order of their local configurations.*

Let β_0 be the empty prefix with $\mathit{Cutoffs}(\beta_0) = \emptyset$. For the induction step, let $\beta_i = \beta_{i-1} \oplus \{e_i\}$, and let $\mathit{Cutoffs}(\beta_i) = \mathit{Cutoffs}(\beta_{i-1})$. The event e_i is added into $\mathit{Cutoffs}(\beta_i)$ iff β_{i-1} contains a configuration C_i such that:

(a) $\mathit{Mark}([e_i]) = \mathit{Mark}(C_i)$,

(b) $C_i \prec [e_i]$, and

(c) $C_i \cap \mathit{Cutoffs}(\beta_{i-1}) = \emptyset$.

Now the set of cut-off events of β is defined as $\mathit{Cutoffs}(\beta) = \mathit{Cutoffs}(\beta_k)$.

The proofs of prefix finiteness and completeness with this refined cut-off criterion are almost identical to similar proofs in [5], the only change is changing the local configuration $[e']$ into the configuration C. (The fact that the corresponding configuration is a local configuration is not needed by the proofs.)

Note that we need totality of the order $\prec$ between local configurations of a prefix for the definition above to be non-ambiguous. Thankfully this can be done by using a total order such as the one presented in [5].

4 Rule-Based Constraint Programming

We will use normal logic programs with stable model semantics [6] as the underlying formalism to represent combinatorial problems. This section is to a large extent based on [17]. We employ logic programs as a *constraint programming framework* [14], where stable models are the solutions of the program rules seen as constraints. We consider normal logic programs that consist of rules of the form

$$\mathtt{h} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}, \mathit{not}\,(\mathtt{b_1}), \ldots, \mathit{not}\,(\mathtt{b_m}) \tag{1}$$

where $\mathtt{a_1}, \ldots, \mathtt{a_n}, \mathtt{b_1}, \ldots, \mathtt{b_m}$ and $\mathtt{h}$ are propositional atoms. Such a rule can be seen as a constraint saying that if atoms $\mathtt{a_1}, \ldots, \mathtt{a_n}$ are in a model and atoms $\mathtt{b_1}, \ldots, \mathtt{b_m}$ are not in a model, then the atom $\mathtt{h}$ is in a model. The stable model semantics also enforces minimality and groundedness of models. This makes many combinatorial problems easily and succinctly describable using logic programming with stable model semantics.

The stable model semantics for a normal logic program P is defined as follows [6]. The reduct P^A of P with respect to the set of atoms A is obtained (i) by deleting each rule in P that has a not-atom $\mathit{not}\,(\mathtt{x})$ in its body such that $\mathtt{x} \in A$ and (ii) by deleting all not-atoms in the remaining rules. A set of atoms A is a stable model of P if and only if A is the deductive closure of P^A when the rules in P^A are seen as inference rules.

A non-deterministic way of constructing stable models is to guess which assumptions (not-atoms of the program) to use, and then check using the deductive closure (in linear time) whether the resulting model agrees with the assumptions. The problem of determining the existence of a stable model is in fact NP-complete [10].

4.1 The tool `smodels`

There is a tool, the `smodels` system [15, 17], which provides an implementation of logic programs as a rule-based constraint programming framework. It finds (some or all) stable models of a logic program. It can also tell when the program has no stable models. It contains strong pruning techniques to make the problem tractable for a large class of programs. The `smodels` implementation needs space linear in the size of the input program [17].

The stable model semantics is defined using rules of the form (1). The `smodels` 2 handles extended rule types, which can be seen as succinct encodings of sets of basic rules. One of the rule types is a rule of the form: $\mathtt{h} \leftarrow n\{\mathtt{a_1}, \ldots, \mathtt{a_k}\}$. The semantics of this rule is that if n or more atoms from the set $\mathtt{a_1}, \ldots, \mathtt{a_k}$ belong to the model, then also the atom $\mathtt{h}$ will be in the model.

We also use the so called *integrity rules* in the programs. They are rules with no head, i.e. of the form: $\leftarrow \mathtt{a}_1, \ldots, \mathtt{a}_n, \mathit{not}\,(\mathtt{b}_1), \ldots, \mathit{not}\,(\mathtt{b}_m)$. The semantics are defined in such a way that if atoms $\mathtt{a}_1, \ldots, \mathtt{a}_n$ are in a model and atoms $\mathtt{b}_1, \ldots, \mathtt{b}_m$ are not in a model, then this model is not a stable model. For more information about the extended rules, see [16].

5 Translating Reachability into Logic Programs

In this section we briefly review the method used in our previous work to do reachability and deadlock checking with prefixes [8, 9, 7]. We need this for the basis of our minimization algorithm, which is based on solving reachability problems on prefixes.

First we define some additional notation. We assume a unique numbering of the events (and conditions) of the finite complete prefix. We use the notation e_i (b_i) to refer to the event (condition) number i. In the logic programs $\mathtt{e_i}$, ($\mathtt{b_i}$) is an atom of the logic program corresponding to the event e_i (condition b_i). In the logic program definitions of this paper we use the convention that a part of a rule will be omitted, if the corresponding set evaluates to the empty set.

We define a mapping from the atoms of a logic program to the events of a prefix.

Definition 5.1 *The set of events corresponding to a stable model Δ of a logic program P is Events$(\Delta) = \{e_i \in E \mid \mathtt{e_i} \in \Delta\}$.*

With assertions we can easily formulate the reachability problem.

Definition 5.2 *An* assertion *on a marking of a 1-safe net system $\Sigma = \langle S, T, F, M_0 \rangle$ is a tuple $\langle S^+, S^- \rangle$, where $S^+, S^- \subseteq S$, and $S^+ \cap S^- = \emptyset$. The assertion $\langle S^+, S^- \rangle$* agrees *with a marking M of Σ iff:*

$$S^+ \subseteq \{s \in S \mid M(s) = 1\} \wedge S^- \subseteq \{s \in S \mid M(s) = 0\}.$$

Next we define a logic program whose stable models are exactly those configurations of the prefix, whose marking agrees with the given assertion.

Definition 5.3 *Let $\beta = \langle N, h \rangle$ with $N = \langle B, E, F \rangle$ be a finite complete prefix of a given 1-safe net system $\Sigma = \langle S, T, F, M_0 \rangle$, and let $\phi = \langle S^+, S^- \rangle$ be an assertion on the places of Σ. Let $P_R(\beta, \phi)$ be a logic program containing the following rules:*

1. *For all $e_i \in E \setminus \mathit{CutOffs}(\beta)$ a rule:*
   $\mathtt{e_i} \leftarrow \mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}, \mathit{not}\,(\mathtt{be_i})$,
   such that $\{e_{p_1}, \ldots, e_{p_n}\} = {}^\bullet({}^\bullet e_i)$.

2. *For all $e_i \in E \setminus \mathit{CutOffs}(\beta)$ a rule:*
   $\mathtt{be_i} \leftarrow \mathit{not}\,(\mathtt{e_i})$.

3. *For all $b_i \in B$ such that $|b_i^\bullet \setminus CutOffs(\beta)| \geq 2$ a rule:*
   $\leftarrow 2\{\mathtt{e_{p_1}}, \ldots, \mathtt{e_{p_n}}\}$,
   such that $\{e_{p_1}, \ldots, e_{p_n}\} = b_i^\bullet \setminus CutOffs(\beta)$.

4. *For all $b_i \in \{b_j \in B \mid h(b_j) \in S^+ \cup S^- \wedge {}^\bullet b_j \in E \setminus Cutoffs(\beta)\}$ a rule:*
   $\mathtt{b_i} \leftarrow \mathtt{e_l}$, *not* $(\mathtt{e_{p_1}})$, ... , *not* $(\mathtt{e_{p_n}})$,
   such that $\{e_l\} = {}^\bullet b_i$, and $\{e_{p_1}, \ldots, e_{p_n}\} = b_i^\bullet \setminus CutOffs(\beta)$.

5. *For all $b_i \in \{b_j \in B \mid h(b_j) \in S^+ \cup S^- \wedge {}^\bullet b_j \in E \setminus Cutoffs(\beta)\}$ a rule:*
   $\mathtt{s_i} \leftarrow \mathtt{b_i}$,
   such that $s_i = h(b_i)$.

6. *For all $s_i \in S^+$ a rule:*
   $\leftarrow$ *not* $(\mathtt{s_i})$.

7. *For all $s_i \in S^-$ a rule:*
   $\leftarrow \mathtt{s_i}$.

Note that cut-off postset conditions are not translated, because cut-offs will not be fired by the translation. Proof of the following theorem can be found from [8].

Theorem 5.1 *The logic program $P_R(\beta, \phi)$ has a stable model iff there exists a reachable marking of Σ which agrees with ϕ. Additionally, for any stable model Δ of $P_R(\beta, \phi)$, the configuration $C = Events(\Delta)$ is a configuration of β, such that $Mark(C)$ is a reachable marking of Σ which agrees with ϕ.*

To be more exact, the correspondence is a little stronger than what is stated above, the stable models of the program $P_R(\beta, \phi)$ and the configurations which agree with ϕ have a one-to-one correspondence with each other [8]. It is easy to see that the sizes of all the translations are linear in the size of the prefix β, i.e. $\mathcal{O}(|B| + |E| + |F|)$.

6 Implementation

In this work we have implemented the prefix minimization as an off-line tool to be run after prefix generation. However, the approach can also be used to minimize the prefix during prefix generation.

The prefix minimizer reads a binary file containing the description of a finite complete prefix generated by the ERVunfold algorithm [4]. We have to use the same adequate order the prefix generation algorithm for this approach to work. We denote by $<_{PEP}$ the adequate order used by the ERVunfold algorithm. We experimentally found out (and later got a confirmation from the ERVunfold author) that the only difference from the adequate order presented in [5] (denoted $<_{ERV}$) is in the way Foata normal form levels are compared. The difference is that the $<_{ERV}$ order uses a lexicographical order, while $<_{PEP}$ uses a size-lexicographical order (a shorter string is always smaller than a longer one, only when sizes match are strings compared with lexicographical order), for details see [5].

The minimized prefix is initialized to contain all the minimal conditions of the original prefix. Then minimizer main routine loops over events $e \in E$ of the original prefix, and does the following:

1. If the event is a cut-off event of the original prefix (and has not been removed by the steps below), then e is added as a cut-off event into the minimized prefix together with its postset conditions, and we continue with the next event from step 1.

2. Otherwise, we generate a reachability program from the part of the minimized prefix which includes all events e_i, such that $|[e_i]| < |[e]|$. The reachability program is given the assertion $\phi = \langle \{s \in \mathit{Mark}([e])\}, \{s \in S \setminus \mathit{Mark}([e])\} \rangle$.

3. We add an integrity rule into the program, which disallows all solutions which have more than $|[e]|$ events in the configuration.

4. We find the next stable model Δ of the program with `smodels`, if no solution is found, we add e and its postset conditions into the minimized prefix, and continue with the next event from step 1.

5. If a solution is found, it is compared using a subroutine whether $C = \mathit{Events}(\Delta) <_{PEP} [e]$. If this is true, then e is added to the set of cut-off events of the minimized prefix together with its postset conditions. Then all events e_i of the original prefix, such that $e <_F e_i$, are removed from the original prefix together with their postset conditions. If the comparison was false, we compute the next solution in step 4.

The reachability program generation ensures that the marking $\mathit{Mark}(C) = \mathit{Mark}([e])$ for all stable models found in step 4. The step 3 guarantees that all solutions have at most $|[e]|$ events in them, and rules out most of the solutions which would be larger in the adequate order, and thus rejected by step 5.

Note that our approach can also be adjusted to other adequate orders than the one used in this work. However, the fact that larger configurations are always larger in the adequate order was very useful for implementation. In fact, the implementation only generates one base program for all events whose local configuration size is the same, and only changes the reachability assertion for each event. We can do this, because the prefix generator has already marked as cut-offs all events, whose corresponding configuration is a (smaller in the adequate order) local configuration of the same size. To use this optimization of the algorithm during the prefix generation, we need to compare the local configuration of an event against all other event whose local configuration is of the same size. We do not believe this to be a big problem for implementing minimization in a prefix generator, as this is something which is already done by the current prefix generators.

6.1 Experimental Results

We have made experiments with our approach using examples by Corbett [2], McMillan [11, 12], and Melzer and Römer [13]. They were previously used by Melzer and Römer in [13] and by Best and Römer in [1], where additional information can be found.

The Figures 1-3 present the running times in seconds for the various algorithms used in this work. The figures might be described as the good news, the bad news, and the indifferent news, respectively. The running times have been measured using a Pentium 266MHz, 512MB RAM, Linux 2.2.3, egcs 2.91.60 C++ compiler, `smodels` 2.22, and ERVunfold 4.5.1 by Stefan Römer [4].

	Original Prefix			Minimized Prefix			Time (s)	
Problem(size)	$\|B\|$	$\|E\|$	#c	$\|B\|$	$\|E\|$	#c	Unf	$\mathrm{Min}_{\mathrm{smo}}$
BDS(1)	12310	6330	3701	3167	1660	832	2.5	11.6
DPD(4)	594	296	81	318	158	62	0.0	0.3
DPD(5)	1582	790	211	652	325	130	0.1	1.0
DPD(6)	3786	1892	499	1282	640	258	0.5	3.6
DPD(7)	8630	4314	1129	2488	1243	502	2.2	14.6
DPH(4)	680	336	117	472	232	75	0.0	0.4
DPH(5)	2712	1351	547	1326	658	235	0.2	2.5
DPH(6)	14590	7289	3407	3338	1663	636	4.1	17.0
DPH(7)	74558	37272	19207	7840	3913	1580	101.4	117.9
FURNACE(1)	535	326	189	305	183	99	0.0	0.2
FURNACE(2)	4573	2767	1750	1966	1168	688	0.4	4.6
FURNACE(3)	30820	18563	12207	10177	5995	3710	14.3	162.3
GASN(2)	338	169	46	194	97	22	0.0	0.1
GASN(3)	2409	1205	401	837	419	109	0.2	1.6
GASN(4)	15928	7965	2876	3360	1681	460	8.1	28.3
GASN(5)	100527	50265	18751	12867	6435	1783	399.4	521.9
GASQ(1)	43	21	4	39	19	4	0.0	0.0
GASQ(2)	346	173	54	186	93	26	0.0	0.1
GASQ(3)	2593	1297	490	941	471	150	0.3	1.7
GASQ(4)	19864	9933	4060	5488	2745	916	14.1	65.7
MMGT(1)	118	58	20	118	58	20	0.0	0.0
MMGT(2)	1280	645	260	834	420	170	0.1	0.8
MMGT(3)	11575	5841	2529	4893	2470	1046	3.4	31.1
MMGT(4)	92940	46902	20957	25944	13100	5650	307.8	1104.2
OVER(2)	83	41	10	59	29	7	0.0	0.0
OVER(3)	369	187	53	166	84	24	0.0	0.1
OVER(4)	1536	783	237	386	198	62	0.1	0.4
OVER(5)	7266	3697	1232	951	491	165	1.7	2.2
SYNC(2)	3884	2091	474	1276	700	137	0.6	5.2
SYNC(3)	28138	15401	5210	9431	5233	1591	22.9	270.1

Figure 1: Minimization results

	Original Prefix			Minimized Prefix			Time (s)	
Problem(size)	$\|B\|$	$\|E\|$	#c	$\|B\|$	$\|E\|$	#c	Unf	Min_{smo}
CYCL(3)	52	23	4	52	23	4	0.0	0.0
CYCL(6)	112	50	7	112	50	7	0.0	0.1
CYCL(9)	172	77	10	172	77	10	0.0	0.2
CYCL(12)	232	104	13	232	104	13	0.0	0.3
DME(2)	487	122	4	487	122	4	0.1	0.6
DME(3)	1210	321	9	1210	321	9	0.2	2.9
DME(4)	2381	652	16	2381	652	16	0.5	11.4
DME(5)	4096	1145	25	4096	1145	25	1.5	37.4
DME(6)	6451	1830	36	6451	1830	36	4.2	100.9
DME(7)	9542	2737	49	9542	2737	49	10.4	231.2
DME(8)	13465	3896	64	13465	3896	64	23.0	470.5
DME(9)	18316	5337	81	18316	5337	81	44.8	881.2
DME(10)	24191	7090	100	24191	7090	100	82.5	1556.0
DME(11)	31186	9185	121	31186	9185	121	142.0	2607.7
DP(6)	204	96	30	204	96	30	0.0	0.1
DP(8)	368	176	56	368	176	56	0.0	0.2
DP(10)	580	280	90	580	280	90	0.0	0.6
DP(12)	840	408	132	840	408	132	0.1	1.1
DPFM(2)	12	5	2	12	5	2	0.0	0.0
DPFM(5)	67	31	20	67	31	20	0.0	0.0
DPFM(8)	426	209	162	426	209	162	0.1	0.0
DPFM(11)	2433	1211	1012	2433	1211	1012	1.9	0.4
HART(25)	179	102	1	179	102	1	0.0	0.2
HART(50)	354	202	1	354	202	1	0.1	1.0
HART(75)	529	302	1	529	302	1	0.1	2.3
HART(100)	704	402	1	704	402	1	0.2	4.0
RW(6)	806	397	327	806	397	327	0.1	0.1
RW(9)	9272	4627	4106	9272	4627	4106	0.5	2.8
RW(12)	98378	49177	45069	98378	49177	45069	25.3	265.7

Figure 2: Minimization results

The rows of the tables correspond to different problems. The columns represent: sum of user and system times measured by `/usr/bin/time` command:

- Unf = time for unfolding (creation of the finite complete prefix) (ERVunfold).
- Min_{smo} = time for the prefix minimization algorithm (currently without prefix output).

The other fields of the figures are as follows: $|B|$: number of conditions, $|E|$: number of events, #c: number of cut-off events.

Note that the original prefix sizes differ from those presented in previous work [1, 13, 8]. When we implemented the code to compare configurations with respect to the adequate order $<_{PEP}$, we ran into a bug in the prefix generator of the PEP tool, which was promptly fixed by Stefan Römer in the ERVunfold algorithm version 4.5.1.

The Fig. 1 shows quite large savings in the prefix size, with prefix size reductions of up-to almost 90%. Also, the gap between the original prefix and

	Original Prefix			Minimized Prefix			Time (s)	
Problem(size)	$\|B\|$	$\|E\|$	#c	$\|B\|$	$\|E\|$	#c	Unf	Min_{smo}
ABP(1)	337	167	56	329	163	54	0.0	0.2
DAC(6)	92	53	0	62	38	5	0.0	0.0
DAC(9)	167	95	0	95	59	8	0.0	0.1
DAC(12)	260	146	0	128	80	11	0.0	0.1
DAC(15)	371	206	0	161	101	14	0.0	0.1
ELEVATOR(1)	296	157	59	246	132	47	0.0	0.1
ELEVATOR(2)	1562	827	331	1480	786	309	0.1	2.5
ELEVATOR(3)	7398	3895	1629	7284	3838	1597	1.4	78.6
ELEVATOR(4)	32354	16935	7337	32208	16862	7295	27.4	1721.9
FTP(1)	178085	89046	35197	130156	65080	25902	953.4	34135.3
KEY(2)	1310	653	199	1194	595	175	0.2	2.7
KEY(3)	13941	6968	2911	12573	6284	2534	4.8	290.7
KEY(4)	135914	67954	32049	121930	60962	27803	397.6	26992.0
Q(1)	16123	8417	1188	15770	8240	1060	14.8	1059.2
RING(3)	97	47	11	67	32	7	0.0	0.0
RING(5)	339	167	37	223	109	21	0.0	0.2
RING(7)	813	403	79	523	258	43	0.1	0.7
RING(9)	1599	795	137	1015	503	73	0.2	2.7
SENT(25)	383	216	40	290	157	23	0.0	0.3
SENT(50)	458	241	40	365	182	23	0.1	0.5
SENT(75)	533	266	40	440	207	23	0.1	0.8
SENT(100)	608	291	40	515	232	23	0.1	1.1
SPD(1)	4929	2882	1219	3155	1824	787	0.7	14.8

Figure 3: Minimization results

the minimized prefix seems to be growing when moving to larger instances of the problem. The running times are acceptable, with all being under 1 hour.

The Fig. 2 shows another side of the story, the prefixes after minimization are identical to those before it. The running times are quite similar to those of the previous Figure, only the largest DME instances being significantly slower.

The Fig. 3 shows prefixes which only partially benefited from the minimization. However, there are two hard cases for the minimization algorithm: the FTP example and the largest KEY example. Their running times are currently unacceptable. We will experiment with alternative minimization strategies, which might help these cases. Also interesting is the case of DAC problems, the minimization introduced some cut-off events. Thus for example deadlock checking is not anymore trivial using the optimized prefix, while it was trivial in the original one. The complexity results of deadlock and reachability checking for prefixes are however not affected by our minimization, and these problems still remain NP-complete.

7 Conclusions

We have demonstrated that refining the cut-off criterion can sometimes help to create much smaller prefixes. Whether the effort needed to generate these smaller prefixes is justified is still under discussion. If the prefix is used as an input to algorithms which have very large running times, then even the current approach might be viable. The minimized prefixes can be directly used for deadlock and reachability checking. Other verification algorithms might also work, but they need to be modified to use the notion of a corresponding configuration instead of the notion of a corresponding event, and their correctness proofs revisited with this change in mind. The complexity results of different model checking questions for non-minimized vs. minimized prefixes is left as further work.

It would be very interesting to know is there a structural property the examples in Fig. 1 have which makes them behave better than the examples in Fig. 2. If this would be possible, then it could be used to choose when to use the refined cut-off criterion with prefixes. One observation is that the problems listed in Fig. 2 have a simpler communication structure (when modelled as state machines) than the problems in Fig. 1, see [2].

One open question is the problem of defining an algorithm which computes the finite complete prefix, which would be minimal in e.g. the number of non-cut-off events. There is an example in an extended version of [5], for which our refined cut-off criterion doesn't create the minimal prefix in this sense. We believe that this work can be used as a starting point for defining such a theoretical algorithm for a minimal (in a stricter sense than [5]) prefix generation algorithm.

Another interesting area for future work is the use of NP-complete problem solvers for prefix generation. They can also be used for other parts of the prefix generation process, with or without using the refined cut-off criterion presented here. This could be of practical interest for the prefix based verification tools.

8 Acknowledgements

The author would like to thank Javier Esparza for the possibility of visiting his research group. The tool `smodels` was created by Patrik Simons, who gave excellent support for it. Stefan Römer provided the example nets, and also Linux binaries for ERVunfold algorithm, which both were invaluable. The financial support of Helsinki Graduate School on Computer Science and Engineering (HeCSE), the Academy of Finland (Project 8309), the Support Foundation of Helsinki University of Technology, and the Eemil Aaltonen Foundation are gratefully acknowledged.

References

[1] E. Best. Partial order verification with PEP. In G. Holzmann, D. Peled, and V. Pratt, editors, *Proceedings of POMIV'96, Workshop on Partial Order Methods in Verification.* American Mathematical Society, July 1996.

[2] J. C. Corbett. Evaluating deadlock detection methods for concurrent software. Technical report, Department of Information and Computer Science, University of Hawaii at Manoa, 1995.

[3] J. Engelfriet. Branching processes of Petri nets. In *Acta Informatica 28*, pages 575–591, 1991.

[4] J. Esparza and S. Römer. An unfolding algorithm for synchronous products of transition systems. In *Proceedings of the 10th International Conference on Concurrency Theory (Concur'99)*. Springer-Verlag, Berlin, 1999. Invited paper, accepted for publication.

[5] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Proceedings of Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 87–106, Passau, Germany, Mar 1996. Springer-Verlag. LNCS 1055.

[6] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, USA, August 1988. The MIT Press.

[7] K. Heljanko. Deadlock checking for complete finite prefixes using logic programs with stable model semantics (extended abstract). In *Proceedings of the Workshop Concurrency, Specification & Programming 1998*, Informatik-Bericht Nr. 110, pages 106–115. Humboldt-University, Berlin, September 1998.

[8] K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. *Fundamenta Informaticae*, 37(3):247–268, 1999.

[9] K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. In *Proceedings of Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, pages 240–254. Springer-Verlag, Berlin, March 1999. LNCS 1579.

[10] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38:588–619, 1991.

[11] K. L. McMillan. Using unfoldings to avoid the state space explosion problem in the verification of asynchronous circuits. In *Proceeding of 4th Workshop on Computer Aided Verification (CAV'92)*, pages 164–174, 1992. LNCS 663.

[12] K. L. McMillan. A technique of a state space search based on unfolding. In *Formal Methods is System Design 6(1)*, pages 45–65, 1995.

[13] S. Melzer and S. Römer. Deadlock checking using net unfoldings. In *Proceeding of 9th International Conference on Computer Aided Verification (CAV'97)*, pages 352–363, Haifa, Israel, Jun 1997. Springer-Verlag. LNCS 1254.

[14] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. In *Proceedings of the Workshop on Computational Aspects of Nonmonotonic Reasoning*, pages 72–79, Trento, Italy, May 1998. Helsinki University of Technology, Digital Systems Laboratory, Research Report A52.

[15] I. Niemelä and P. Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning*, pages 420–429, Dagstuhl, Germany, July 1997. Springer-Verlag.

[16] P. Simons. Extending stable model semantics with more expressive rules. Unpublished manuscript, available on the Internet at http://www.tcs.hut.fi/Publications/papers/simons99.ps.gz.

[17] P. Simons. Towards constraint satisfaction through logic programs and the stable model semantics. Research Report A47, Helsinki University of Technology, Espoo, Finland, August 1997. Licenciate's thesis, Available at http://www.tcs.hut.fi/pub/reports/A47.ps.gz.