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
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 \textit{preset} of a node \( x \), denoted by \( \ast x \), is the set \( \{ y \in S \cup T \mid F(y, x) = 1 \} \). The \textit{postset} of a node \( x \), denoted by \( x^* \), 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 \( \ast X = \bigcup_{x \in X} \ast x \), and \( X^* = \bigcup_{x \in X} x^* \) respectively.

A \textit{marking} of a net \( \langle S, T, F \rangle \) is a mapping \( S \rightarrow \mathbb{N} \). 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 \textit{reachable} in \( \Sigma \) 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: \( M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_{n-1}} M_{n-1} \xrightarrow{t_n} M_n \). A reachable marking is \textit{1-safe} if \( \forall s \in S : M(s) \leq 1 \). A net system \( \Sigma \) is \textit{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 \textit{conflict}, denoted by \( x_1 \sim x_2 \), if there exist \( t_1, t_2 \in T \) such that \( t_1 \neq t_2, t_1 \cap ^* t_2 \neq \emptyset, t_1 \leq_F x_1 \), and \( t_2 \leq_F x_2 \). An occurrence net is a net \( N = \langle B, E, F \rangle \) such that:

- \( \forall b \in B : |^* b| \leq 1 \),
- $F$ 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: $h(S_1) \subseteq S_2 \land h(T_1) \subseteq T_2$, and for all $t \in T_1$, the restriction of $h$ to $t^*$ is a bijection between $t$ and $h(t)$, and similarly for $t^*$ and $h(t)^*$. A branching process of a net system $\Sigma$ is a tuple $\beta = \langle N', h \rangle$, where $N'$ is an 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 $e_1 = e_2 \land h(e_1) = h(e_2)$ then $e_1 = e_2$. The set of places associated with a configuration $C$ of $\beta$ is denoted by $Mark(C) = h((Min(N) \cup C^\star) \setminus C)$.

It is shown in [3] that a net has a maximal branching process up to isomorphism, 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 branching processes to denote the extension of a branching process $\beta$ by the set of events $E$ and their postset conditions.

2.4 Finite Complete Prefixes

A finite branching process $\beta$ is a finite complete prefix of a net system $\Sigma$ iff for each reachable marking $M$ of $\Sigma$ there exists a configuration $C$ of $\beta$ such that:

- $Mark(C) = M$, and
- for every transition $t$ enabled in $M$ there exists a configuration $C \cup \{e\}$ such that $e \not\in C$ and $h(e) = t$.

Algorithms to obtain a finite complete prefix $\beta$ given a 1-safe net system $\Sigma$ are presented in e.g. [5, 11, 12]. The algorithms will mark some events of the prefix $\beta$ 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 \( \beta \) be prefix of the unfolding containing an event \( e \). The event \( e \) is a cut-off event of \( \beta \) iff \( \beta \) contains a local configuration \([e']\) such that:

(a) \( \text{Mark}(e) = \text{Mark}(e') \), 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 \( \beta \) 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 \( \beta \), sorted in a non-decreasing \( \prec \) order of their local configurations.

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

(a) \( \text{Mark}(e_i) = \text{Mark}(C_i) \),

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

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

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

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

$$h \leftarrow a_1, \ldots, a_n, \text{not}(b_1), \ldots, \text{not}(b_m)$$

where $a_1, \ldots, a_n, b_1, \ldots, b_m$ and $h$ are propositional atoms. Such a rule can be seen as a constraint saying that if atoms $a_1, \ldots, a_n$ are in a model and atoms $b_1, \ldots, b_m$ are not in a model, then the atom $h$ 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 $\text{not}(x)$ in its body such that $x \in A$ 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 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: $h \leftarrow n\{a_1, \ldots, a_n\}$. The semantics of this rule is that if $n$ or more atoms from the set $a_1, \ldots, a_n$ belong to the model, then also the atom $h$ 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: \(\leftarrow a_1, \ldots, a_n, \text{not} (b_1), \ldots, \text{not} (b_m)\). The semantics are defined in such a way that if atoms \(a_1, \ldots, a_n\) are in a model and atoms \(b_1, \ldots, b_m\) 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 \(e_i\) (or \(b_i\)) to refer to the event (condition) number \(i\). In the logic programs \(e_i, (b_i)\) 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 \(\Delta\) of a logic program \(P\) is\n
\[\text{Events}(\Delta) = \{e_i \in E | e_i \in \Delta\}.\]

With assertions we can easily formulate the reachability problem.

**Definition 5.2** An assertion on a marking of a l-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 \(\Sigma\) iff:

\[S^+ \subseteq \{s \in S | M(s) = 1\} \land S^- \subseteq \{s \in S | M(s) = 0\} \].

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 l-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 \(\Sigma\). Let \(P_B(\beta, \phi)\) be a logic program containing the following rules:

1. For all \(e_i \in E \setminus \text{CutOffs}(\beta)\) a rule:

\[e_i \leftarrow e_{p_1}, \ldots, e_{p_s}, \text{not} (b_{e_i}), \]

such that \(\{e_{p_1}, \ldots, e_{p_s}\} = *e_i\).

2. For all \(e_i \in E \setminus \text{CutOffs}(\beta)\) a rule:

\[b_{e_i} \leftarrow \text{not} (e_i).\]
3. For all \(b_i \in B\) such that \(|b_i^* \setminus \text{CutOffs} (\beta)| \geq 2\) a rule:
\[b_i \leftarrow \text{not} (e_{p_1}, \ldots, e_{p_n}),\]
such that \(\{e_{p_1}, \ldots, e_{p_n}\} = b_i^* \setminus \text{CutOffs} (\beta)\).

4. For all \(b_i \in \{b_j \in B \mid h(b_j) \in S^+ \cup S^- \wedge b_j \in E \setminus \text{CutOffs} (\beta)\}\) a rule:
\[b_i \leftarrow e_1, \text{not} (e_{p_1}, \ldots, \text{not} (e_{p_n})),\]
such that \(\{e_1\} = b_i^*\) and \(\{e_{p_1}, \ldots, e_{p_n}\} = b_i^* \setminus \text{CutOffs} (\beta)\).

5. For all \(s_i \in \{b_j \in B \mid h(b_j) \in S^+ \cup S^- \wedge \exists b_j \in E \setminus \text{CutOffs} (\beta)\}\) a rule:
\[s_i \leftarrow b_i,\]
such that \(s_i = h(b_i)\).

6. For all \(s_i \in S^+\) a rule:
\[\leftarrow \text{not} (s_i).\]

7. For all \(s_i \in S^-\) a rule:
\[\leftarrow s_i.\]

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 \(\Sigma\) which agrees with \(\phi\). Additionally, for any stable model \(\Delta\) of \(P_R(\beta, \phi)\), the configuration \(C = \text{Events} (\Delta)\) is a configuration of \(\beta\), such that \(\text{Mark} (C)\) is a reachable marking of \(\Sigma\) which agrees with \(\phi\).

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 \(\phi\) 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 \(\beta\), i.e., \(O(|B| + |E| + |F|)\).

## 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 \(<_{PEF}\) 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 $\phi = \{s \in Mark([e])\}$, $\{s \in S \setminus Mark([e])\}$.

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 $\Delta$ 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 $C = Events(\Delta) <_{PEP} [e]$. 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 $e <_{P} e_i$, 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 $Mark(C) = Mark([e])$ 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 2.9.60 C++ compiler, smodels 2.22, and ERVunfold 4.5.1 by Stefan Römer [4].

<table>
<thead>
<tr>
<th>Problem (size)</th>
<th>Original Prefix</th>
<th>Minimized Prefix</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Table showing the results of the minimization process](image)

Figure 1: Minimization results
<table>
<thead>
<tr>
<th>Problem (size)</th>
<th>Original Prefix</th>
<th>Minimized Prefix</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(</td>
<td>B</td>
<td>)</td>
</tr>
<tr>
<td>CYCL(3)</td>
<td>52</td>
<td>23</td>
<td>4</td>
</tr>
<tr>
<td>CYCL(6)</td>
<td>112</td>
<td>50</td>
<td>7</td>
</tr>
<tr>
<td>CYCL(9)</td>
<td>172</td>
<td>77</td>
<td>10</td>
</tr>
<tr>
<td>CYCL(12)</td>
<td>232</td>
<td>104</td>
<td>13</td>
</tr>
<tr>
<td>DME(2)</td>
<td>481</td>
<td>122</td>
<td>4</td>
</tr>
<tr>
<td>DME(3)</td>
<td>1210</td>
<td>321</td>
<td>9</td>
</tr>
<tr>
<td>DME(4)</td>
<td>2381</td>
<td>652</td>
<td>16</td>
</tr>
<tr>
<td>DME(5)</td>
<td>4096</td>
<td>1145</td>
<td>25</td>
</tr>
<tr>
<td>DME(6)</td>
<td>6451</td>
<td>1830</td>
<td>36</td>
</tr>
<tr>
<td>DME(7)</td>
<td>9542</td>
<td>2737</td>
<td>49</td>
</tr>
<tr>
<td>DME(8)</td>
<td>13465</td>
<td>3896</td>
<td>64</td>
</tr>
<tr>
<td>DME(9)</td>
<td>18316</td>
<td>5337</td>
<td>81</td>
</tr>
<tr>
<td>DME(10)</td>
<td>24191</td>
<td>7060</td>
<td>100</td>
</tr>
<tr>
<td>DME(11)</td>
<td>31186</td>
<td>9185</td>
<td>121</td>
</tr>
<tr>
<td>DP(6)</td>
<td>204</td>
<td>96</td>
<td>30</td>
</tr>
<tr>
<td>DP(8)</td>
<td>368</td>
<td>176</td>
<td>56</td>
</tr>
<tr>
<td>DP(10)</td>
<td>580</td>
<td>280</td>
<td>90</td>
</tr>
<tr>
<td>DP(12)</td>
<td>840</td>
<td>408</td>
<td>132</td>
</tr>
<tr>
<td>DPFM(2)</td>
<td>12</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>DPFM(5)</td>
<td>67</td>
<td>31</td>
<td>20</td>
</tr>
<tr>
<td>DPFM(8)</td>
<td>426</td>
<td>209</td>
<td>162</td>
</tr>
<tr>
<td>DPFM(11)</td>
<td>2433</td>
<td>1211</td>
<td>1012</td>
</tr>
<tr>
<td>HART(25)</td>
<td>179</td>
<td>102</td>
<td>1</td>
</tr>
<tr>
<td>HART(50)</td>
<td>354</td>
<td>202</td>
<td>1</td>
</tr>
<tr>
<td>HART(75)</td>
<td>529</td>
<td>302</td>
<td>1</td>
</tr>
<tr>
<td>HART(100)</td>
<td>704</td>
<td>402</td>
<td>1</td>
</tr>
<tr>
<td>RV(6)</td>
<td>806</td>
<td>397</td>
<td>327</td>
</tr>
<tr>
<td>RV(9)</td>
<td>9272</td>
<td>4627</td>
<td>4106</td>
</tr>
<tr>
<td>RV(12)</td>
<td>98378</td>
<td>49177</td>
<td>45069</td>
</tr>
</tbody>
</table>

Figure 2: Minimization results

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

- \textit{Unf} = time for unfolding (creation of the finite complete prefix) (ERVunfold).
- \textit{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
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.

<table>
<thead>
<tr>
<th>Problem[size]</th>
<th>Original Prefix</th>
<th>Minimized Prefix</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABP[1]</td>
<td>320 167 56</td>
<td>320 163 54</td>
<td>0.0 0.2</td>
</tr>
<tr>
<td>DAC[6]</td>
<td>92 53 0</td>
<td>62 38 5</td>
<td>0.0 0.0</td>
</tr>
<tr>
<td>DAC[9]</td>
<td>167 95 0</td>
<td>95 9 8</td>
<td>0.0 0.1</td>
</tr>
<tr>
<td>DAC[12]</td>
<td>200 146 0</td>
<td>128 80 11</td>
<td>0.0 0.1</td>
</tr>
<tr>
<td>DAC[15]</td>
<td>371 206 0</td>
<td>161 101 14</td>
<td>0.0 0.1</td>
</tr>
<tr>
<td>ELEVATOR[1]</td>
<td>256 157 50</td>
<td>246 132 41</td>
<td>0.0 0.1</td>
</tr>
<tr>
<td>ELEVATOR[2]</td>
<td>1562 827 331</td>
<td>1480 786 309</td>
<td>0.1 2.5</td>
</tr>
<tr>
<td>ELEVATOR[3]</td>
<td>7398 389 1639</td>
<td>7804 3838 1597</td>
<td>1.4 7.6</td>
</tr>
<tr>
<td>ELEVATOR[4]</td>
<td>32354 16935 7337</td>
<td>32208 16862 7295</td>
<td>27.4 172.9</td>
</tr>
<tr>
<td>FTP[1]</td>
<td>178085 89046 35197</td>
<td>130156 65080 25602</td>
<td>953.4 34135.3</td>
</tr>
<tr>
<td>KEY[2]</td>
<td>1310 653 199</td>
<td>1194 595 175</td>
<td>0.2 2.7</td>
</tr>
<tr>
<td>KEY[3]</td>
<td>139141 69668 2911</td>
<td>12573 6284 2534</td>
<td>4.8 290.7</td>
</tr>
<tr>
<td>KEY[4]</td>
<td>135914 67954 32049</td>
<td>121930 60062 27803</td>
<td>307.6 20092.0</td>
</tr>
<tr>
<td>Q[1]</td>
<td>16123 8417 1188</td>
<td>15770 8240 1090</td>
<td>14.8 1039.2</td>
</tr>
<tr>
<td>RING(3)</td>
<td>97 47 11</td>
<td>67 32 7</td>
<td>0.0 0.0</td>
</tr>
<tr>
<td>RING(5)</td>
<td>339 167 37</td>
<td>223 109 21</td>
<td>0.0 0.2</td>
</tr>
<tr>
<td>RING(7)</td>
<td>813 403 70</td>
<td>523 258 43</td>
<td>0.1 0.7</td>
</tr>
<tr>
<td>RING(9)</td>
<td>1599 795 137</td>
<td>1015 503 73</td>
<td>0.2 2.7</td>
</tr>
<tr>
<td>SENT[25]</td>
<td>383 216 40</td>
<td>290 157 23</td>
<td>0.0 0.3</td>
</tr>
<tr>
<td>SENT[50]</td>
<td>458 241 40</td>
<td>365 182 23</td>
<td>0.1 0.5</td>
</tr>
<tr>
<td>SENT[75]</td>
<td>333 266 40</td>
<td>440 207 23</td>
<td>0.1 0.8</td>
</tr>
<tr>
<td>SENT[100]</td>
<td>608 291 40</td>
<td>515 232 23</td>
<td>0.1 1.1</td>
</tr>
<tr>
<td>SPD[1]</td>
<td>4929 2882 1210</td>
<td>3155 1824 787</td>
<td>0.1 14.8</td>
</tr>
</tbody>
</table>

Figure 3: Minimization results

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 Emil Aaltonen Foundation are gratefully acknowledged.

References


