Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

A Negative Result on Depth-First Net Unfoldings

Javier Esparza[1], Pradeep Kanade[2⋆], and Stefan Schwoon[1]

[1] University of Stuttgart
Institute for Formal Methods in Computer Science
Stuttgart, Germany
e-mail: {esparza,schwoosn}@informatik.uni-stuttgart.de
[2] Department of Computer Science and Engineering, IIT Bombay, India
e-mail: pradeepk@cse.iitb.ac.in

The date of receipt and acceptance will be inserted by the editor

1 Introduction

A successful way of palliating the state-explosion problem when model-checking concurrent systems is to apply partial-order techniques. These techniques allow to verify properties without exhaustively exploring all the reachable states of the system.

Net unfoldings are a partial-order technique based on the theory of true concurrency. Introduced by McMillan in [12,13], it has since then been further analyzed and improved [15,6–8], extended to full LTL model checking [3–5], to symmetrical systems [2] and to nets with read arcs [16], and applied, e.g., to conformance checking [14], analysis of asynchronous circuits [9,10] and graph-grammars [1]. The technique requires the system to be modelled as a collection of communicating automata, as a Petri net, or in any other formalism with a notion of concurrent components and specifying which components participate in each action. In this paper, we use the Petri net formalism.

It is clear that a rooted transition system can be unfolded or unrolled into a (possibly infinite) tree exhibiting the same behaviour. A net unfolding is a true-concurrency equivalent of this notion. Instead of a transition system we now have a Petri net $\mathcal{N}$, and instead of unfolding into a (possibly infinite) tree we unfold $\mathcal{N}$ into a (possibly infinite) acyclic Petri net with tree-like properties. The unfolding can be constructed by a non-deterministic procedure that starts from a net without transitions, whose only reachable marking corresponds to the initial marking of $\mathcal{N}$. The procedure adds new transitions, one at a time. Intutitively, these transitions correspond to particular occurrences of transitions of $\mathcal{N}$.

The unfolding procedure can be extended to decide properties of the net, e.g. whether a given place p of $\mathcal{N}$ can ever become marked. The procedure attaches to each new transition t of the unfolding a reachable marking M_t. When adding t, it checks whether M_t marks p, in which case it signals success and terminates. Otherwise, the algorithm checks whether some transition t' added earlier satisfies $M_{t'} = M_t$, in which case it marks t as a *cut-off*, meaning that this branch of the search will no longer be explored. The search terminates without success when no new transition can be added. The search can be conducted using different strategies, most prominently breadth-first and depth-first. In the best case, it can be exponentially faster than search algorithms acting on the Petri net's reachability graph, and experiments have shown it to be also more efficient in many systems relevant in practice (see e.g. [15,6,9]).

It is well-known that, unlike in the case of transition systems, the correctness of the search, i.e., the property that the search always terminates with the correct answer, *depends on the strategy* [6]. Several papers [12,7,6] have given correct breadth-first strategies. The question whether correct depth-first strategies exist has been open for several years (in fact, this question was already considered by McMillan in his original work [11]). It is an important question, not only because of the fact that depth-first search tends to be more efficient than breadth-first for positive instances (i.e., when the search terminates successfully), but also because because many efficient algorithms for transition systems, like the algorithms for checking Büchi emptiness, are based on depth-first search, and so their generalization from the transition system to the Petri net case relies on the correctness of depth-first search.

In this note we present a counterexample showing that depth-first search in net unfoldings is not correct. The note is organized as follows. In Section 2 we define unfoldings of transition systems. In Section 3 we intro-

⋆ Work done while this author was visiting the University of Stuttgart, partially supported by the project "Algorithms for Software Model Checking" funded by the Deutsche Forschungsgemeinschaft.

duce net unfoldings as a generalization of the transition-system case. Section 4 introduces the notion of correct search algorithms. Section 5 presents the counterexample showing that depth-first-search algorithms are not correct. Finally, in Section 6 we discuss the result.

2 Unfoldings of transition systems

A *transition system* is a pair $\mathcal{A} = (S, T, s_I)$, where S is a set of *states*, $T \subseteq S \times S$ is a set of *transitions*, and $s_I \in S$ is the *initial state*.

A transition system $\mathcal{A}$ can be 'unfolded' into a tree. The unfolding can be seen as the 'limit' of the construction that starts with the tree having one single node labeled by s_I, and iteratively extends the current tree as follows: if $\mathcal{A}$ has a transition $t = (s, s')$, and the current tree has a node labeled by s, then we add a new edge to the tree, labeled by t, leading to a new node labeled by s' (to be precise, we only add the edge if it hasn't been added before). We call this edge a *possible extension* of the current tree. If the transition system has a cycle, then its unfolding is an infinite tree. Notice that we can also look at the unfolding as a *labeled transition system*, i.e., as a transition system whose states and transitions carry labels. The states of the unfolding are the nodes of the tree (labeled with states of $\mathcal{A}$), and its transitions are the edges of the tree (labeled with transitions of $\mathcal{A}$). In this note we call the transitions of the unfolding *events*. Many states (possibly infinitely many) of the unfolding may be labeled by the same state of $\mathcal{A}$: they correspond to different visits to the state. Similarly, an event symbolises a particular *occurrence* of a transition.

The reachability problem for a transition system $\mathcal{A}$ is as follows: given a target state s_T, decide if there is a path from s_I to s_T. Clearly, this is the case if and only if the unfolding of $\mathcal{A}$ contains a state labelled by s_T.

A *search algorithm* for the reachability problem, like breadth-first or depth-first search, explores a *prefix* of the unfolding of $\mathcal{A}$. Such an algorithm explores the unfolding, one event at a time. After exploring a new event, the algorithm sometimes marks it as a "cut-off", whose successors are not explored. An event is marked as a cut-off if the label of the state it leads to, say s, satisfies one of the following conditions:

(1) $s = s_T$; in this case the algorithm stops with the result 'reachable'.
(2) 's is already known to be reachable', i.e., some other state of the current tree is also labeled by s.

If the algorithm reaches a point at which no new event can be added and no state of the current tree is labeled by s_T, then it terminates with 'unreachable'.

At this level of abstraction, search algorithms only differ in the way they select the next event to be explored among the possible extensions of the current tree. For instance, breadth-first search algorithms select a possible extension having smallest distance to the root, while chaotic search algorithms choose the new event randomly.

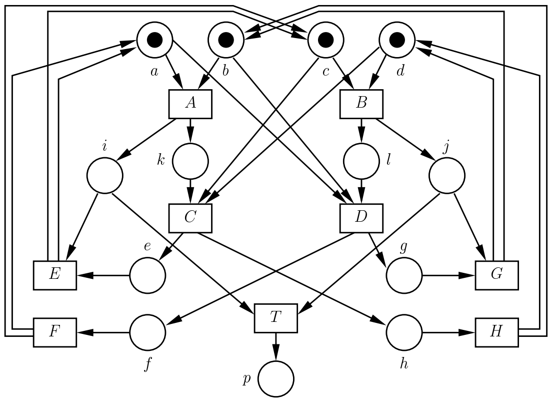

Fig. 1. The safe Petri net $\mathcal{N}_h$.

It is not difficult to prove that, if $\mathcal{A}$ is finite, then *all search algorithms* terminate with the correct answer ('reachable' or 'unreachable'), i.e., the order in which events are explored does not affect correctness.

3 Net unfoldings

In this section we define the unfolding of a Petri net as a generalization of the unfolding of a transition system. The definition will be informal, but hopefully precise. For formal definitions the reader is referred to [7].

3.1 Preliminaries.

A *Petri net* is a pair (P, T, M_I), where P is a set of *places* and $T \subseteq 2^P \times 2^P$ is a set of *transitions*. Given a transition $t = (P_1, P_2)$, we call P_1 and P_2 the sets of *input* and *output places* of t, respectively. A *marking* M is a mapping assigning to each place a number of *tokens*, and M_I is an *initial marking*. If, at a given marking, all the input places of a transition hold at least one token, then the transition can *occur*, which leads to a new marking obtained by removing one token from each input place and adding one token to each output place.

An *occurrence sequence* is a sequence of transitions that can occur from M_I in the order specified by the sequence. A marking M of $\mathcal{N}$ is *reachable* if some occurrence sequence leads from M_I to M. In this note we consider *safe* Petri nets, which are those in which $M(p) \leq 1$ for every reachable marking M and every place p. Figure 1 shows an example of a safe Petri net. (This net will later serve as our counterexample.) We often identify a marking M with the set that contains all places p for which $M(p) = 1$.

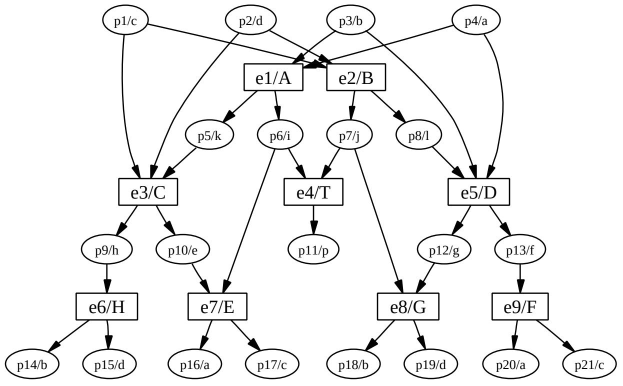

Fig. 2. A prefix of the unfolding of the net of Figure 1

3.2 Unfoldings

We have defined the unfolding of a transition system as a labeled transition system; analogously, the unfolding of a Petri net $\mathcal{N}$ is a *labeled Petri net* whose places and transitions are labeled with the names of places and transitions of $\mathcal{N}$. We also speak of the *label of a marking* m of the unfolding, meaning the set of all the labels of the elements of m. As in the case of transition systems, we call the transitions of the unfolding *events*.

When unfolding $\mathcal{A}$, we start with one state, labeled with the initial state of $\mathcal{A}$. In the same way, when unfolding $\mathcal{N}$, we start with one *place* for each element of the initial marking M_I. For the Petri net $\mathcal{N}_h$ of Figure 1, we have $M_I = \{a, b, c, d\}$, so we start with the four places at the top of Figure 2.

We now generalize the notion of possible extension: if in the current labeled net we can reach a marking m labeled by a marking M of the original net, and M enables a transition t leading to a marking M', then we add to the unfolding a new event labeled by t, and for each output place p of t in $\mathcal{N}$ a new place labeled by p.

Figure 2 shows a snapshot of the unfolding process of $\mathcal{N}_h$ after the addition of nine events. We call these snapshots *prefixes* of the unfolding. The initial marking $\{p_1, p_2, p_3, p_4\}$ with label $\{a, b, c, d\}$ enables the transitions A and B, so we have two possible extensions. We choose to add event e_1 first, labeled by A. Now the prefix has a new reachable marking, corresponding to marking $\{c, d, i, k\}$ in $\mathcal{N}_h$, which leads to a new possible extension with label C. Suppose that we add event e_2 with label B next. Now the prefix has two additional reachable markings, $\{p_3, p_4, p_7, p_8\}$ (labelled by $\{a, b, j, l\}$) and $\{p_5, p_6, p_7, p_8\}$ (labelled by $\{i, j, k, l\}$). These two markings enable extensions labelled with D and T, respectively. After event e_3 with label C we have a new marking labelled by $\{e, h, i\}$, enabling possible extensions with labels E and H, etc.

As in the transition-system case, the unfolding of $\mathcal{N}$ is defined as the 'limit' of the unfolding procedure. It is easy to show that the 'limit' is unique up to isomorphism, and that it is a safe labeled Petri net. Moreover, the occurrence sequences of $\mathcal{N}$ and its unfolding coincide, i.e. $t_1 \dots t_n$ is an occurrence sequence of $\mathcal{N}$ if and only if its unfolding contains an occurrence sequence $e_1 \dots e_n$ labeled by $t_1 \dots t_n$.

3.3 The place-reachability problem

For convenience, we consider a special version of the reachability problem that we call the *place-reachability* problem: Given a safe Petri net $\mathcal{N}$ and a target place p_T of N, does some reachable marking M put a token on p_T, i.e., does $M(p_T) = 1$ hold for some reachable marking M? It is well-known that for safe Petri nets, the usual reachability problem, where we ask for the reachability of a marking M_T, can be reduced in linear time to the place-reachability problem, and vice versa.

Since the occurrence sequences of a net $\mathcal{N}$ and its unfolding coincide, some reachable marking of $\mathcal{N}$ puts a token on p_T if and only if some place of the unfolding of $\mathcal{N}$ is labeled by p_T. In the net $\mathcal{N}_h$ of Figure 1 the place p is reachable.

3.4 Search algorithms for place-reachability

We study search algorithms for the place-reachability problem. These algorithms explore (or construct) increasingly large prefixes of the unfolding of N. As in the case of transition systems, they explore one event at a time, and some events are marked as cut-offs, whose successors are not explored anymore. The problem is how to generalize the definition of cut-off events to Petri nets. A solution was presented by McMillan in [12,13]. The key is to attach to each event e a suitable reachable marking M_e of the original Petri net $\mathcal{N}$. This is done in three steps, which we illustrate on event e_3 of Figure 2:

- Compute the set $[e]$ of all predecessors of e, i.e, the set of all events e' such that the unfolding contains a path from e' to e. We have $[e_3] = \{e_1, e_3\}$.
- Choose any occurrence sequence σ containing each element of $[e]$ exactly once (which is guaranteed to exist), and let it occur. In our case, $\sigma = e_1 e_3$.
- Let m be the marking of the unfolding reached by firing σ (which can be shown to be independent of the choice of σ), and define M_e as the label of m. In our case, $m = \{p_6, p_9, p_{10}\}$ and $M_{e_3} = \{e, h, i\}$.

For each new event e the algorithms compute and store the marking M_e. Notice that these are the only markings of $\mathcal{N}$ which the algorithms know to be reachable.

Now we can easily generalize conditions (1) and (2). An event e is marked as a cut-off if the marking M_e satisfies one of the following conditions:

(1') $M_e(p_T) = 1$; in this case the algorithm stops with the result 'reachable'.

(2') 'M_e is already known to be reachable', i.e. either $M_e = M_I$, or $M_{e'} = M_e$ for some other event e'. In that case, e' is called the *corresponding event* of e.

The unfolding prefix in Figure 2 has a reachable marking $\{p_{14}, p_{15}, p_{16}, p_{17}\}$, corresponding to $\{a, b, c, d\}$. This enables a possible extension labelled by A, which would be associated with the marking $\{c, d, i, k\}$. Since that is equal to M_{e_1}, such an extension would be a cut-off.

4 Search algorithms for place-reachability

We have seen that chaotic search is a correct algorithm for the reachability problem in transition systems. Is this still the case for place reachability in Petri nets?

It is easy to see that chaotic search in the unfolding of a safe Petri net always terminates: Since an event e can only be added if its marking M_e was not known before, the algorithm can only explore as many events as the (finite) number of reachable markings of $\mathcal{N}$. Also, if the algorithm answers 'reachable', then the place p_T is indeed reachable. Unfortunately, the algorithm may produce false negatives. This fact was already implicit in [12], and an explicit counterexample is shown in [7].

In [12], McMillan gave a correct search algorithm which generalized breadth-first search.

Definition 1. The *size* of an event e is defined as the number of elements of $[e]$. A search algorithm for place-reachability works *small-first* if it explores events in order of increasing size. I.e., if the algorithm adds events $e_1, \ldots, e_n$, in this order, then the size of e_i is smaller than or equal to the size of e_j for every $1 \leq i < j \leq n$.

Unfortunately, McMillan's algorithm used a weaker notion of cut-off than Definition 1. With that notion, the algorithm may explore up to 2^n events for a net with n reachable markings. Correct small-first algorithms with the above definition of cut-off event, and therefore exploring at most n events, were presented in [7] and [6].

The question whether there exist correct search algorithms that generalize depth-first search has been open for several years. Already McMillan considered it [11]. The question is important from an algorithmic point of view. In [4] a model-checking algorithm for LTL based on net unfoldings was described. The algorithm checks emptiness of a Büchi net (a Petri net with accepting places and a Büchi acceptance condition). In the worst case, the algorithm has quadratic complexity in the number of reachable markings of the Petri net. This is due to the fact that the emptiness check has to be carried out using the small-first search of [6,5], and breadth-first-like algorithms for Büchi emptiness have quadratic complexity. A correct depth-first-search algorithm could lead to a linear algorithm, as in the transition system case.

To answer the question we must make precise the meaning of 'generalize depth-first search'. We present a condition which (we think) any depth-first-search algorithm should reasonably satisfy. Let P be an unfolding prefix and E the set of possible extensions of P. Assume that we add to P the event $e \in E$ to yield the prefix P'. Now, let E' be the possible extensions of P'. We call $E' \setminus E$ the possible extensions of P' *enabled by* e.

Definition 2. An algorithm A for place-reachability is *depth-first* if the following holds: Let P be the current prefix, and let e be the last event added by A. The possible extensions of P enabled by e are explored by A before any other possible extension of P.

This condition is a straightforward generalization of the idea that depth-first search tries to extend the current computation before branching to a new one.

5 Depth-first-search algorithms are incorrect

In the rest of this note we demonstrate an instance of the place-reachability problem such that all depth-first search algorithms give the wrong answer to the problem. This shows that none of these algorithms are correct.

Proposition 1. *Every depth-first-search algorithm answers 'unreachable' for the instance $(\mathcal{N}_h, p)$, where $\mathcal{N}_h$ is the net shown in Figure 1.*

Proof. Clearly, p is reachable in $\mathcal{N}_h$, e.g. by firing the sequence A, B, T. However, we show that a particular depth-first exploration answers 'unreachable', and argue that the same holds for any other depth-first traversal. For reference, Figure 3 shows the resulting unfolding.

Initially, there are two possible extensions e_1, e_2, labelled by A and B, respectively. Let us say that the algorithm picks e_1 first (the other case is symmetric). Adding e_1 enables a new event e_3 labelled by C. Since e_3 is the only event enabled by e_1, the algorithm adds it. Now, e_3 enables two further events e_4, e_5 labelled by E and H, respectively. By the depth-first policy, the algorithm adds one of the two, say e_4, and, since e_4 does not enable any new event, it then adds e_5 (adding e_5 first leads to the same result). After e_4 and e_5 have both been added, new events e_6 and e_7 are enabled, labelled by A and B, respectively. Since we have $M_{e_6} = M_{e_1}$, the algorithm can, w.l.o.g., add e_6 first and mark it as a cut-off (this has no consequences, it does not enable any event). The algorithm now adds e_7, which enables event e_8 labelled by D, and e_8 in turn enables e_9 and e_{10}, labelled by F and G, respectively. As in the case of e_4 and e_5, the algorithm adds both e_9 and e_{10}, enabling two new events e_{11}, e_{12} labelled by A and B, respectively. Since $M_{e_{11}} = M_{e_1}$ and $M_{e_{12}} = M_{e_7}$, both become cut-offs. Now the algorithm finally adds e_2, marking it, too, as a cut-off because $M_{e_2} = M_{e_7}$. At this point no more events can be added and the algorithm stops. No event labelled by T has been added.

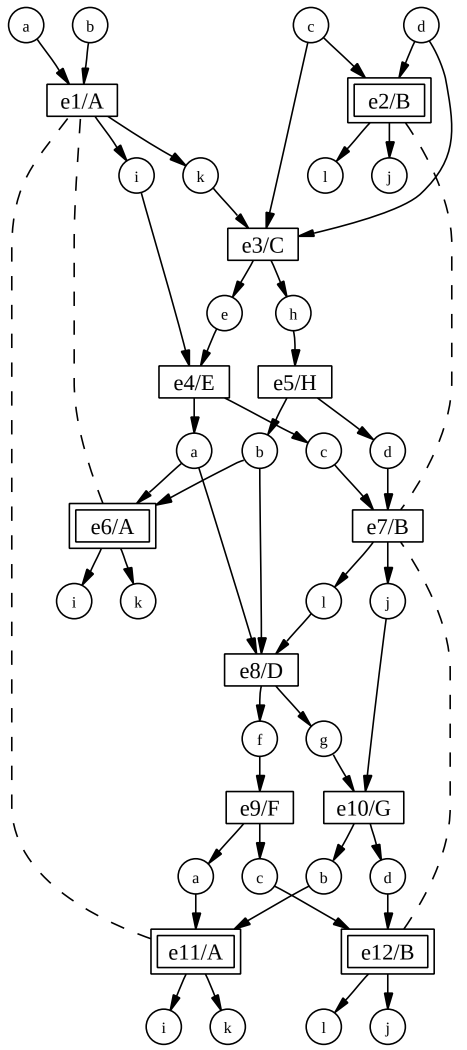

Fig. 3. A depth-first unfolding of $\mathcal{N}_h$. The dashed lines connect cut-offs to their corresponding events.

Notice that the constructed prefix contains three pairs of places labelled by i, j, which could lead to events labelled by T. However, at least one place of each pair is produced by a cut-off, meaning that no pair is explored.

6 Conclusions

We have shown that depth-first-search algorithms for place reachability in Petri nets are not correct or, putting it differently, that depth-first search may fail to construct an unfolding prefix representing all reachable markings. In the terminology of [7], depth-first-search algorithms may not generate a complete prefix. Our result makes the existence of a linear algorithm (in the size of the complete prefix) for model-checking LTL on Petri nets unlikely. Incidentally, there is an interesting analogy with BDD techniques: all known algorithms for checking emptiness of a Büchi automaton encoded as a BDD have quadratic complexity.

Our counterexample indicates that the only hope lies in asking whether Definition 2 really captures the essence of depth-first search in a true concurrent setting. There might be a depth-first-like algorithm which chooses the next event not only among the possible extensions enabled by the event added last, but also among the possible extensions that are concurrent with this event. We think that such algorithms are unlikely to exist due to the non-transitivity of the concurrency relation between events, but we do not have a proof.

References

1. P. Baldan, A. Corradini, and B. König. Verifying finite-state graph grammars: an unfolding-based approach. In *Proc. of CONCUR '04*, LNCS 3170, pages 83–98, 2004.
2. J.-M. Couvreur, S. Grivet, and D. Poitrenaud. Unfolding of products of symmetrical Petri nets. In *Proc. of ICATPN'01*, LNCS 2075. Springer, 2001.
3. J.M. Couvreur, S. Grivet, and D. Poitrenaud. Designing an LTL model-checker based on unfolding graphs. In *Proc. of ICATPN 2000*, LNCS 1825. Springer, 2000.
4. J. Esparza and K. Heljanko. A new unfolding approach to LTL model checking. In *Proc. of ICALP'00*, LNCS 1853, pages 475–486. Springer, July 2000.
5. J. Esparza and K. Heljanko. Implementing LTL model checking with net unfoldings. In *Proc. of SPIN'01*, LNCS 2057, pages 37–56. Springer, May 2001.
6. J. Esparza and S. Römer. An unfolding algorithm for synchronous products of transition systems. In *Proc. of CONCUR'99*, LNCS 1664, pages 2–20. Springer, 1999.
7. J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20:285–310, 2002.
8. V. Khomenko, M. Koutny, and W. Vogler. Canonical prefixes of Petri net unfoldings. *Acta Informatica*, 40(2):95–118, 2003.
9. V. Khomenko, M. Koutny, and A. Yakovlev. Detecting state encoding conflicts in STG unfoldings using SAT. *Fundamenta Informaticae*, 62(2):221–241, 2004.
10. V. Khomenko, M. Koutny, and A. Yakovlev. Logic synthesis for asynchronous circuits based on Petri net unfoldings and incremental SAT. In *Proc. of ACSD'04*. IEEE, 2004.
11. K.L. McMillan. Private communication.
12. K.L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. of CAV'92*, LNCS 663. Springer, 1993.
13. K.L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.
14. K.L. McMillan. Trace theoretic verification of asynchronous circuits using unfoldings. In *Proc. of CAV'95*, LNCS 939. Springer, 1995.
15. S. Melzer and S. Römer. Deadlock checking using net unfoldings. In *Proc. of CAV'97*, LNCS 1254, 1997.
16. W. Vogler, A.L. Semenov, and A. Yakovlev. Unfolding and finite prefix for nets with read arcs. In *Proc. of CONCUR '98*, LNCS 1466. Springer, 1998.