Planning Via Petri Net Unfolding

Sarah Hickmott[1], Jussi Rintanen[2], Sylvie Thiébaux[2], Lang White[1]
National ICT Australia
[1]School of Electrical & Electronic Engineering, The University of Adelaide, &
[2]Computer Sciences Laboratory, The Australian National University

Abstract

The factored state representation and concurrency semantics of Petri nets are closely related to those of concurrent planning domains, yet planning and Petri net analysis have developed independently, with minimal and usually unconvincing attempts at cross-fertilisation. In this paper, we investigate and exploit the relationship between the two areas, focusing on Petri net unfolding, which is an attractive reachability analysis method as it naturally enables the recognition and separate resolution of independent subproblems. On the one hand, based on unfolding, we develop a new forward search method for cost-optimal partial-order planning which can be exponentially more efficient than state space search. On the other hand, inspired by well-known planning heuristics, we investigate the automatic generation of heuristics to guide unfolding, resulting in a more efficient, directed reachability analysis tool for Petri nets.

1 Introduction

Petri nets are traditionally used for modelling and analysing distributed systems [Murata, 1989]. They provide a compact description of the state space in much the same way as planning operators do, but additionally represent independence (concurrency) and causal relations between actions in a way that enables the recognition and separate resolution of independent subproblems. This confers, e.g., the ability to reason about partially ordered sets of actions without having to consider their interleavings. This was exploited early on by Godefroid and Kabanza [1991] to synthesize reactive plans.

Nevertheless, it is fair to say that work since then has failed to sufficiently develop and utilize the connections between the two areas. A recent exception is Edelkamp and Jabbar's [2006] work on applying planning via heuristic search to detecting deadlocks in Petri nets. The primary goal of our work, by contrast, is to determine whether techniques developed for Petri net analysis could be successfully applied to planning.

We focus on Petri net unfolding [McMillan, 1992; Esparza *et al.*, 2002], an *exact* reachability analysis technique which is particularly attractive in that it preserves and exploits much of the structure inherent in the Petri net. The unfolding process generates, forward, a simpler type of net called an occurrence net, which is acyclic and avoids certain conflicts. In Petri net analysis, where the problem is often to prove the absence of deadlocks, unfolding amounts to a breadth-first search which stops when the generated occurrence net represents all markings reachable in the original net. The size of the unfolded net is bounded below by and is typically exponentially larger than the size of the original Petri net, but is bounded above by and is typically exponentially smaller than the size of the state space it represents. Hence searching in unfolding space offers potential gains over state space search.

In Section 2, we provide the necessary background on Petri nets and unfolding, pointing out the differences with the *approximate* reachability analysis performed by the planning graph [Blum and Furst, 1997]. In Section 3, we give a translation from planning problems to 1-safe place transition nets, low level nets to which off-the-shelf unfolding tools apply.

In Section 4, we describe our new planning method. A rather costly option would be to first build the complete unfolded net, and then extract from it the partially ordered plans of interest in time linear in their size. To capitalise on the approach for planning, we instead embed heuristic search into the unfolding, resulting in a directed model-checker [Edelkamp *et al.*, 2001] for Petri nets. We show that monotonic planning heuristics such as h^m [Haslum and Geffner, 2000] can be directly computed from the original Petri net, and guide the unfolding towards minimal cost plans without loss of completeness.

Finally, in Section 5, we present and analyse experimental results obtained with benchmark problems from the International Planning Competition and with a standard Petri net benchmark. Proofs are omitted on grounds of space and are available in the technical report [Hickmott *et al.*, 2006].

2 Petri Nets and Unfolding

2.1 Place-Transition Net

We consider low level Petri nets called *place-transition* (PT) nets. A PT-net (see left-hand side of Figure 1) consists of a net N and its initial marking M_0. The net is a directed, bipartite graph. The two types of nodes are places and transitions, which represent the state variables and the events of the underlying system. Arcs, which capture the dynamics of the system, are directed from places to transitions and vice versa. The marking M of a PT-net represents the state of the system it models. It assigns to each place 0 or more tokens.

Definition 1 *A PT-net is a 4-tuple* (P, T, F, M_0) *where* P *and* T *are disjoint finite sets of places and transitions, respectively,* $F : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ *is a flow relation indicating the presence (1) or absence (0) of arcs, and* $M_0 : P \rightarrow \mathbb{N}$ *is the initial marking.*

The *preset* ${}^\bullet x$ of a node x in the net is the set $\{y \in P \cup T | F(y, x) = 1\}$. The *postset* $x^\bullet$ of a node is the set $\{y \in P \cup T | F(x, y) = 1\}$. For simplicity, we assume that every transition has non-empty preset and postset. A particular marking M enables a transition t if $\forall p \in P\ F(p, t) \leq M(p)$. The occurrence, or *firing*, of transition t absorbs a token from each of its preset places and produces a token in each of its postset places, thus moving the net from M to the new marking $M'(p) = M(p) - F(p, t) + F(t, p)\ \forall p \in P$. This corresponds to a state transition of the modelled system. A set of transitions T' is concurrently enabled at the marking M if it is possible for all $t \in T'$ to occur simultaneously, viz. $\forall p \in P\ \sum_{t \in T'} F(p, t) \leq M(p)$. For instance, in the net of Figure 1, transitions 1 and 3 are concurrently enabled for the given marking, as are transitions 2 and 3. Conversely, transitions 1 and 2 are in *forward conflict*, which means that, whilst each is individually enabled, only one of them can fire. Firing transitions 2 and 3 (in any order or concurrently) followed by transition 5 results in one token each in places f and g. We say that a PT-net is n-safe if the number of tokens in each place can never exceed n. In this paper, we consider 1-safe nets.

2.2 Unfolding the Place-Transition Net

Unfolding is a method for reachability analysis which exploits and preserves concurrency information. In planning terms, the unfolding approach allows searching for partially ordered plans without considering unnecessary interactions between actions. The unfolding of a PT-net produces an *occurrence* net whose nodes are called conditions and events. These represent particular occurrences of the places and transitions, respectively, in possible runs of the original net from the initial marking. The unfolding achieves this by eliminating cycles and *backward conflicts*. Two transitions that output to the same place are in backward conflict; by eliminating this we know exactly which transitions were fired to obtain a particular marking. In planning terms, the elimination of backward conflicts achieves the property of *post-uniqueness* of the action set [Backstrom and Nebel, 1995], which implies that we know the exact set of actions that causes a state variable to have a certain value at some point in the plan.

The unfolding of a PT-net $N = (P, T, F, M_0)$ is $\beta = (ON, \varphi)$, where $ON = (B, E, F')$ is an occurrence net and φ is a homomorphism from ON to N, a mapping from conditions B and events E to places P and transitions T respectively. The occurrence net starts with conditions representing the places initially marked in the PT-net, that is, φ maps the set B_0 of conditions which have an empty preset one-one onto the set of places p such that $M_0(p) \geq 1$.

The right-hand side of Figure 1 shows a prefix of the unfolding of the PT-net example in the left-hand side. Notice the multiple instances of place g for example, due to the different paths through which it can be reached. Note also that transition 0 does not appear in the unfolding, as there exists no path through the net in which the events in its causal history are not in conflict.

2.3 Configurations

To understand how the unfolding is built, the most important notions are that of a configuration and the local configuration of an event. A *configuration* represents a possible partial run of the net. It is any set of events C such that:

1. C is causally closed, that is if any event is in the configuration, then so are all its ancestors in the occurrence net: $\forall e' \leq e,\ e \in C \Rightarrow e' \in C$.
2. C contains no forward conflict — this is motivated by the fact that two events in forward conflict cannot both occur (in any order or simultaneously) in the same run of the net: $\forall\, e_1, e_2 \in C,\ e_1 \neq e_2 \Rightarrow {}^\bullet e_1 \cap {}^\bullet e_2 = \emptyset$.

For instance, in the finite prefix in Figure 1, $\{\mathsf{e1}, \mathsf{e3}, \mathsf{e4}, \mathsf{e5}\}$ is a configuration. A configuration C can be associated with a marking Mark(C) of the original net by identifying which conditions will contain a token after the events in C are fired from the initial marking: Mark$(C) = \varphi((B_0 \cup C^\bullet) \backslash {}^\bullet C)$, where $C^\bullet = \{e^\bullet | e \in C\}$ and ${}^\bullet C = \{{}^\bullet e | e \in C\}$. That is, the marking of configuration C identifies the resultant state of the original Petri net when (only) the events in C occur. For instance, in Figure 1, the marking of configuration $\{\mathsf{e1}, \mathsf{e3}, \mathsf{e4}, \mathsf{e5}\}$ is $\varphi(\{\mathsf{c6}, \mathsf{c8}\}) = \{\mathsf{g}, \mathsf{b}\}$.

The *local configuration* of an event e, denoted $[e]$, consists of that event and all of its ancestors. It is the minimal configuration containing e. For example, $[\mathsf{e5}] = \{\mathsf{e1}, \mathsf{e3}, \mathsf{e4}, \mathsf{e5}\}$. A set of conditions can be simultaneously marked if the union of the local configurations of their presets forms a configuration. The unfolding process involves identifying which transitions are enabled by those conditions, currently in the occurrence net, that can be simultaneously marked. The identified transitions are referred to as the possible events. A new instance of each is added to the occurrence net, as are instances of the places in each of their postsets.

2.4 Finite Complete Prefix of Unfolded net

In most cases, the unfolding β of a Petri-net is infinite. For this reason, we seek a complete finite prefix β' of β, one which contains as much information as β. Formally, the prefix β' of β is *complete* if for every reachable marking M, there exists a configuration $C \in \beta'$ such that

1. Mark$(C) = M$, and
2. for every transition t enabled by M there exists a configuration $C \cup \{e\}$ such that $e \notin C$ and $\varphi(e) = t$.

The key to obtaining a complete finite prefix is to identify those events at which we can cease unfolding without loss of information. Such events are referred to as *cut-off events* and are defined in terms of an *adequate order* on configurations [McMillan, 1992; Esparza *et al.*, 2002]. In the following, $C \oplus \mathcal{E}$ denotes a configuration that extends C with the finite set of events $\mathcal{E}$ disjoint from C.

Definition 2 *A partial order* $\prec$ *on finite configurations is adequate if*

1. $\prec$ *is well founded,*
2. $C_1 \subset C_2 \Rightarrow C_1 \prec C_2$*, and*

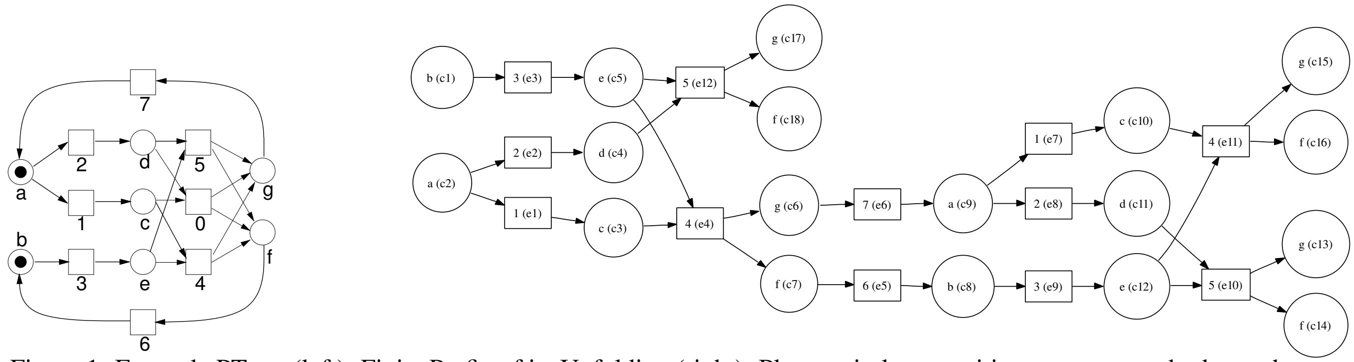

Figure 1: Example PT-net (left). Finite Prefix of its Unfolding (right). Places=circles, transitions=squares and tokens=dots.

3. *$\prec$ is preserved by finite extensions: if $C_1 \prec C_2$ and Mark(C_1) = Mark(C_2), then for all finite extensions $C_1 \oplus \mathcal{E}_1$ and $C_2 \oplus \mathcal{E}_2$ such that $\mathcal{E}_1$ and $\mathcal{E}_2$ are isomorphic, we have $C_1 \oplus \mathcal{E}_1 \prec C_2 \oplus \mathcal{E}_2$*

Without loss of information, or in other terms, without threat to completeness, we can cease unfolding from an event e, if e takes the net to a marking which can be caused by some other event e' such that $[e'] \prec [e]$. This is because the events (and thus markings) which proceed from e will also proceed from e'. Relevant proofs can be found in [Esparza *et al.*, 2002]:

Definition 3 *Let $\prec$ be an adequate partial order. An event e is a cut-off event with respect to $\prec$ if the prefix contains some event e' such that Mark$([e])$ = Mark$([e'])$ and $[e'] \prec [e]$.*

MOLE[1] is a freeware program which unfolds 1-safe PT-nets. It uses an adequate order $\prec$ on configurations which is based on comparing their cardinality. This is refined by comparisons based on Parikh-vectors and the Foata normal form to make the order strict and thus minimise the size of the generated prefix [Esparza *et al.*, 2002]. The prefix on the right-hand side of Figure 1 is the complete finite prefix that MOLE generates for our example. The events e10, e11, and e12 are all cut-off events. This is because each of their local configurations, firstly, has the same marking as the local configuration of event e4, ie. {f, g}, and, secondly, is greater than the local configuration of event e4 with respect to the adequate partial order implemented by MOLE. Notice that the finite prefix of the unfolding ceases at cut-off events, even though resulting conditions could indeed enable other actions.

2.5 Unfolding Algorithm

MOLE builds the complete finite prefix following Algorithm 1. The algorithm maintains a priority queue of possible events in increasing order of $\prec$ wrt. their local configuration. The expensive part of the algorithm is the computation of the possible events which is exponential in the maximal size of the presets of the transitions; see [Esparza *et al.*, 2002] for details. The size of the prefix obtained decreases with the strength of the ordering and with the amount of concurrency in the original net. When the ordering is strict, the size of the unfolding is bounded above by that of the reachable state space of the net (up to a small factor) and only equals that bound if there is no concurrency at all [Esparza *et al.*, 2002]. The presence of concurrency typically leads to prefixes exponentially smaller. This is because the unfolding builds a space of partially ordered sets of events and avoids the combinatorial interleavings of events that can be handled concurrently.

[1] http://www.fmi.uni-stuttgart.de/szs/tools/mole/

Algorithm 1 The MOLE Unfolding Algorithm

Add the conditions in B_0 to the prefix
Initialise the priority queue with the events possible in B_0
while the queue is not empty:
    remove the first event in the queue
    if it is not a cut-off
        Add the event and its postset to the prefix
        Identify the new possible events and
        insert them in the queue
    endif
endwhile
Add the postsets of all cut-off events to the prefix

2.6 Unfolding vs Planning Graph

The reader might find it useful to view the unfolding as a powerful planning graph [Blum and Furst, 1997], where conditions and events play the role of the graph's proposition and action nodes, respectively. There are a number of important differences, however. Firstly, whilst the planning graph performs an approximate reachability analysis, the unfolding computes reachability exactly: a by-product of the Petri net semantics is that all mutexes (not just binary ones) are propagated and accounted for when determining sets of possible events. Secondly, the unfolding duplicates nodes as needed to guarantee post-uniqueness, i.e., that conditions (proposition nodes) have a unique event (action node) as predecessor. A consequence of these differences is that plans can be extracted from the unfolding in time linear in their length, while plan extraction from the planning graph requires search. Finally, there is no global notion of level in the unfolding. Instead, there is an asynchronous vision of time which confers on independent subproblems their own local levels. Consequently, the unfolding lends itself more easily to the generation of partially-ordered plans with optimal cost, while the graph is better suited to producing step-optimal parallel plans.

3 Translating Planning Problems into PT-Nets

To use an unfolding tool such as MOLE for planning, we need to turn planning problems into 1-safe place-transition nets, which these tools accept as an input. In fact, 1-safety rather helps in representing propositional planning operators. When reading the truth value of a boolean variable as the presence or absence of a token, allowing multiple tokens in a place would be meaningless. At best, it would require non-trivial book-keeping, since multiple tokens in a place resulting from repeatedly making a variable true would all need to be removed to make this variable false.

Our translation operates in three steps. In the first step 1-safety is established by replacing every planning operator by several 1-safe ones (the concept of 1-safe operator is defined below). In the second step, we eliminate negative preconditions which are lacking in PT-nets. In the third step, the resulting problem is finally mapped onto a PT-net. We prove that our translation is correct. We also characterise the extent to which the notion of concurrency in the PT-net we obtain matches the independence-based notion of concurrency commonly used in planning.

3.1 Establishing 1-safety

Let A be a set of state variables. The set of *literals* over A is $L = A \cup \{\neg a | a \in A\}$. The *complement* $\bar{l}$ of a literal $l \in L$ is defined by $\overline{a} = \neg a$ and $\overline{\neg a} = a$ for $a \in A$. For sets e of literals, we define $\overline{e} = \{\bar{l} | l \in e\}$. A *state* $s : A \rightarrow \{0, 1\}$ assigns values 0 or 1 to the state variables. A planning *operator* over A is a pair $\langle p, e\rangle$ such that $p \cup e \subseteq L$. A planning operator $\langle p, e\rangle$ has *positive preconditions* if $p \subseteq A$. It is *1-safe* if $\overline{e} \subseteq p$, that is, if all effect literals appear (negatively) in the preconditions. A *planning problem* is a quadruple $\langle A, I, O, G\rangle$ where A is a set of state variables, $I : A \rightarrow \{0, 1\}$ is an initial state, O is a set of planning operators, and G is a set of goal literals.

We map every planning problem to an equivalent one with the property that every operator has positive preconditions and is 1-safe. We start by establishing 1-safety. An operator $o = \langle p, e\rangle$ is first replaced by $2^{|e\setminus\overline{p}|}$ 1-safe operators as follows. Let $e' \subseteq e \setminus \overline{p}$ be a set of effect literals. We define a new operator that works like o when o changes exactly the literals e' (in addition to those literals in $e \cap \overline{p}$ which o clearly requires to change). A 1-safe operator that changes exactly these literals and retains the values of other effects of o is $\langle p \cup \overline{e'} \cup (e\setminus\overline{p})\setminus e', e' \cup e \cap \overline{p}\rangle$.

Take e.g. $p{=}\{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}\}$ and $e{=}\{\neg\mathsf{a}, \mathsf{b}, \mathsf{d}, \neg\mathsf{e}\}$. The operator $o{=}\langle p, e\rangle$ is replaced with the four 1-safe operators $o_i{=}\langle p_i, e_i\rangle$ given below along with the respective values for e'.

$p = \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}\}$	$e = \{\neg\mathsf{a}, \mathsf{b}, \mathsf{d}, \neg\mathsf{e}\}$	
$p_1 = \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}, \mathsf{d}, \neg\mathsf{e}\}$	$e_1 = \{\neg\mathsf{a}, \mathsf{b}\}$	$e'_1 = \{\}$
$p_2 = \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}, \neg\mathsf{d}, \neg\mathsf{e}\}$	$e_2 = \{\neg\mathsf{a}, \mathsf{b}, \mathsf{d}\}$	$e'_2 = \{\mathsf{d}\}$
$p_3 = \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\}$	$e_3 = \{\neg\mathsf{a}, \mathsf{b}, \neg\mathsf{e}\}$	$e'_3 = \{\neg\mathsf{e}\}$
$p_4 = \{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}, \neg\mathsf{d}, \mathsf{e}\}$	$e_4 = \{\neg\mathsf{a}, \mathsf{b}, \mathsf{d}, \neg\mathsf{e}\}$	$e'_4 = \{\mathsf{d}, \neg\mathsf{e}\}$

3.2 Eliminating Negative Preconditions

For a given set A of state variables, we introduce the set $\widehat{A} = \{\widehat{a} | a \in A\}$ of new state variables. The idea is that $\widehat{a}$ is true exactly when a is false.

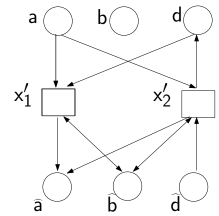

Figure 2: The PT-Net translation of operator $\mathsf{x} = \langle\{\mathsf{a}, \neg\mathsf{b}\}, \{\neg\mathsf{a}, \mathsf{d}\}\rangle$ (after transformation into two 1-safe operators with positive preconditions $\mathsf{x}'_1 = \langle\{\mathsf{a}, \widehat{\mathsf{b}}, \mathsf{d}\}, \{\neg\mathsf{a}, \widehat{\mathsf{a}}\}\rangle$ and $\mathsf{x}'_2 = \langle\{\mathsf{a}, \widehat{\mathsf{b}}, \widehat{\mathsf{d}}\}, \{\neg\mathsf{a}\ \widehat{\mathsf{a}}, \neg\widehat{\mathsf{d}}, \mathsf{d}\}\rangle$).

In the second step of our translation, negative preconditions $\neg a$ are eliminated in the usual way [Gazen and Knoblock, 1997], by replacing them by corresponding positive preconditions $\widehat{a}$ and forcing every state variable $\widehat{a}$ to have the value opposite to the value of a. An operator $\langle p, e\rangle$ over A is replaced by $\langle p', e'\rangle$ over $A \cup \widehat{A}$ where $p' = (p \cap A) \cup \{\widehat{a} | \neg a \in p\}$, and $e' = e \cup \{\neg\widehat{a} | a \in e \cap A\} \cup \{\widehat{a} | \neg a \in e\}$.

For instance, the operator $o_1 = \langle\{\mathsf{a}, \neg\mathsf{b}, \mathsf{c}, \mathsf{d}, \neg\mathsf{e}\}, \{\neg\mathsf{a}, \mathsf{b}\}\rangle$ above is replaced with $o'_1 = \langle\{\mathsf{a}, \widehat{\mathsf{b}}, \mathsf{c}, \mathsf{d}, \widehat{\mathsf{e}}\}, \{\neg\mathsf{a}, \mathsf{b}, \widehat{\mathsf{a}}, \neg\widehat{\mathsf{b}}\}\rangle$.

3.3 Correctness

We define $S(o)$ as the set of operators obtained from o by performing the above two steps. Since a is an effect literal iff $\neg\widehat{a}$ is an effect literal, and $\neg a$ is an effect literal iff $\widehat{a}$ is an effect literal, executing every operator in $S(o)$ preserves the property that for every state s and $a \in A$, $s(a) + s(\widehat{a}) = 1$.

Instead of executing the operator o, we can always execute exactly one of the operators in $S(o)$ with the same effects. This operator depends on the current state and has the property that every state variable mentioned in its effects actually changes when the operator is executed, which is what the definition of 1-safety requires.

The following theorem establishes the correctness of our translation. The proof is based on the fact that in any operator sequence any $o' \in S(o)$ can be replaced by o, and o can be replaced by exactly one operator in $S(o)$.

Theorem 1 *Let $R = \langle A, I, O, G\rangle$ be any planning problem. Let $R' = \langle A \cup \widehat{A}, I, \cup_{o\in O} S(o), G\rangle$. Then for all states $s : A \rightarrow \{0, 1\}$ and $s' : A \cup \widehat{A} \rightarrow \{0, 1\}$ such that $s'(a) + s'(\widehat{a}) = 1$ and $s(a) = s'(a)$ for all $a \in A$, s is a reachable state of R if and only if s' is a reachable state of R'.*

3.4 Mapping to PT-Nets

Finally we map the resulting planning problem to a PT-net as follows. Let $R = \langle A, I, O, G\rangle$ be a planning problem. We define a PT-net $pnet(R) = \langle P, T, F, M_0\rangle$ such that

- the places are $P = A \cup \widehat{A}$,
- the transitions are $T = \cup_{o\in O} S(o)$
- the set F of arcs is obtained from $t = \langle p, e\rangle \in T$ as $\{(a, t) \mid a \in p\} \cup \{(t, a) \mid a \in e \text{ or } a \in p \text{ and } \neg a \notin e\}$

- for all $a \in A$, $M_0(a) = 1$ iff $I(a) = 1$ and $M_0(\widehat{a}) = 1$ iff $I(a) = 0$, and for all $a \in A \cup \widehat{A}$, $M_0(a) = 0$ or $M_0(a) = 1$.

Figure 2 illustrates this mapping for a single operator.

For every reachable marking M and every place $a \in P$ in the resulting PT-net, $M(a) \leq 1$. The proof of the following theorem is by induction on the length of transition sequences leading to M.

Theorem 2 *Let R be a planning problem. Then the PT-net $pnet(R)$ is 1-safe.*

3.5 Concurrency

We are interested in the notion of concurrent or partially-ordered plans which allow the simultaneous execution of several operators. The question arises if the notion of concurrency used in connection with the PT-nets obtained by our translation coincides with the standard notion of concurrency in AI planning. It turns out that this is not the case.

The standard notion of concurrency in planning is *independence*: two operators $\langle p_1, e_1 \rangle$ and $\langle p_2, e_2 \rangle$ are independent iff $p_i \cap \overline{e_j} = \emptyset$ and $e_i \cap \overline{e_j} = \emptyset$ for $i, j \in \{1, 2\}$ and $i \neq j$. This captures the intuition that they can be executed in any order, yielding the same result in both cases.

Independence does not in general imply concurrency in the PT-net sense. For instance, consider the two independent planning operators $\langle \{\mathsf{a}\}, \{\mathsf{b}\} \rangle$ and $\langle \{\mathsf{a}\}, \{\mathsf{c}\} \rangle$. The corresponding Petri net transitions both take a token from a and therefore cannot fire concurrently. This could be remedied by considering Petri nets with read-arcs, but this complicates the unfolding process, and is not supported by MOLE.

For PT-nets in general, the converse implication does not hold either, ie. in some cases, transitions that could not take place simultaneously in the planning context can be simultaneous. For instance, consider two Petri net transitions t and t' such that ${}^\bullet t = \{\mathsf{a}\}$, $t^\bullet = \{\mathsf{b}\}$, ${}^\bullet t' = \{\mathsf{c}\}$, and $t'^\bullet = \{\mathsf{a}\}$. In markings in which places a and c contain a token these two transitions can fire in any order and concurrently. If these transitions are interpreted as planning operators $\langle \{\mathsf{a}\}, \{\neg \mathsf{a}, \mathsf{b}\} \rangle$ and $\langle \{\mathsf{c}\}, \{\neg \mathsf{c}, \mathsf{a}\} \rangle$, no concurrency is possible because the operators are dependent. However, unlike in the general case, the concurrency relation arising out of our translation is strictly stronger than independence:

Theorem 3 *Let $R = \langle A, I, O, G \rangle$ be a planning problem, let $pnet(R) = \langle P, T, F, M_0 \rangle$, and let o_1 and o_2 be operators in O. If there are transitions $t_1, t_2 \in T$ such that $t_1 \in S(o_1)$, $t_2 \in S(o_2)$ and t_1 and t_2 can fire simultaneously, then o_1 and o_2 are independent (and can be executed simultaneously).*

This can be proven contrapositively, assuming that o_1 and o_2 are not independent, and showing that together with 1-safety and the complementary role of places in A and $\widehat{A}$, this implies that t_o and $t_{o'}$ cannot fire simultaneously.

4 Directing Mole for Planning

Once the problem is translated to a PT-net, it is easy to let MOLE produce a partially ordered plan for that problem. Algorithm 1 can be slightly altered to stop whenever the event taken out of the queue is labelled by a designated transition. MOLE actually already supports this option. Therefore, it suffices to augment the planning operator set with a dummy operator whose precondition is the goal, and to require MOLE to stop whenever an event labelled with the corresponding transition is dequeued. The local configuration of this event is a partially ordered plan for the problem. Further, owing to the fact that MOLE's queue orders events by increasing local configuration cardinality, this plan contains the fewest actions.

The cardinality-based ordering relation used by MOLE has a serious drawback for planning however, as it leads MOLE to perform a breadth-first search. A natural idea is to change the ordering to provide better guidance towards the goal, while generalising from the restricted notion of optimality currently in place by considering arbitrary additive action costs.

It turns out that given an arbitrary monotonic heuristic, it is possible to build an adequate order which implements A*, letting the heuristic guide the unfolding towards optimal plans (adequacy ensures that we are retaining completeness of the prefix generated). This rejoins the work on directed model-checking pioneered by Edelkamp et al. [Edelkamp *et al.*, 2001]. A heuristic h estimates the optimal cost of reaching the goal from a given state and is such that $h(s) = 0$ at goal states. Let $\text{cost}(o)$ be the (positive) cost of operator o, and $\text{res}(o, s)$ be the result of applying o in state s; h is *monotonic* iff $h(s) \leq h(\text{res}(o, s)) + \text{cost}(o)$ for all non-goal states s and operators o applicable in s. These definitions easily transfer to the PT-net case, by identifying each operator with the corresponding transition and considering a set of places as the state in which all and only the variables represented by those places are true. Monotonic heuristics which, like h^m [Haslum and Geffner, 2000], can be automatically generated from a planning problem description, are equally easily generated from PT-nets. We then define the following ordering on configurations:

Definition 4 ($\prec_h$) *Let h be a heuristic. For a configuration C, define $g(C) = \sum_{e \in C} cost(\varphi(e))$, and $f(C) = g(c) + h(Mark(C))$. Define $C \prec_h C'$ if and only if $f(C) < f(C')$ or $f(C) = f(C')$ and $|C| < |C'|$.*

Theorem 4 *If h is monotonic, the ordering $\prec_h$ is adequate.*

The proof is a matter of checking the 3 conditions required for adequacy. Only the 2nd condition is non-trivial to prove, and makes use of the monotonicity of the heuristic.

When ordering MOLE's queue with $\prec_h$ for some monotonic heuristic h, we obtain a planner that generates partially ordered plans with the smallest total action cost. In contrast, most state of the art deterministic planners optimise parallel plan length. Moreover, we are not aware of any partial-order planner able to optimise the sum of arbitrary action costs. Finally, our heuristic search in unfolding space substantially differs from existing partial-order planning algorithms.

5 Experimental Results

We implemented Petrify, an extended version of our translation from planning operators to PT-nets. Petrify handles most of the ADL fragment of PDDL. We modified MOLE to implement a variety of search strategies and heuristics defined by their respective ordering relations. All these orderings are complemented with comparisons based on Parikh-vectors and the Foata normal form in case of equality [Esparza *et al.*, 2002], so as to make the order strict. In our experiments below we use the A* strategy, i.e., the $\prec_h$ ordering, with the following heuristics[2] [Bonet and Geffner, 2001; Haslum and Geffner, 2000]:

$\prec_0$: uniform cost, i.e., $\prec_h$ with $h(s) = 0$
$\prec_{h^1}$: the h^1 heuristic, i.e., $\prec_h$ with $h(s) = h^1(s, G)$
$\prec_{h^1_+}$: the h^1_+ heuristic, i.e. $\prec_h$ with $h(s) = h^1_+(s, G)$

Unlike h^1_+, the 0 and h^1 heuristics are monotonic, which guarantees not only optimality but also, by theorem 4, completeness. Nonetheless, $\prec_{h^1_+}$ works well in practice.

We call PUP (Planning via Unfolding of Petri nets), the planner resulting from running our guided version of MOLE on the Petri net encoding produced by Petrify.

All experiments were conducted on a Pentium M 1.7 GHz with 1Gb of memory.

5.1 Artificial Benchmarks

Our first experiment illustrates the claim that planning via unfolding can be exponentially more efficient than planning via state-space search. Consider an artificially constructed problem in which the goal is a conjunction of n subgoals. The ith subgoal is achievable by a sequence A_i of length i. The A_is are disjoint. Each action $a_{i,j}$ in A_i has a unique precondition $e_{i,j-1}$ which it deletes, and one positive effect $e_{i,j}$, which acts as precondition of the next action $a_{i,j+1}$ and so on. Proposition $e_{i,0}$ is true in the initial state, $e_{i,i}$ is the ith subgoal, and the $e_{i,j}$ are all different propositions. The degree c of concurrency in the problem varies from 1 (sequential) to n (fully concurrent) by making the ith subgoal, $i = c \dots n-1$, a precondition to the execution of the $i+1$th sequence (i.e., $e_{i,i}$ is a precondition of $a_{i+1,1}$).

The left-hand graph of Figure 3 shows the number of nodes expanded by forward state space search (sps) and unfolding (unf), each using the 0 and h^1 heuristics, for $n = 3 \dots 10$ and c varying from 1 to n in each case. To ensure fairness, we report the number of nodes expanded to prove optimality (to prove that there is no solution of cost less than the optimal) rather than to find the optimal solution. The figure clearly shows that, as c increases, the performance of state-space search degrades exponentially, while the number of nodes expanded by the unfolding is constant (it equals $n(n+1)/2$, the number of actions in the plan). The h^1 heuristic makes no significant difference except in the purely sequential case where it enables both techniques to prove optimality without search. State space search fails to solve some of the problems as early as $n = 9$, while unfolding solves all problems of size $n = 100$ (not shown in the figure) in a couple of minutes each, producing plans over 5000 actions long.

[2] For a set of literals g and an integer $m \geq 1$, $h^m(s, g) =$

$$\begin{cases} 0 & \text{if } s \text{ satisfies } g \\ \max_{\{g' \subseteq g, |g'| = m\}} h^m(s, g') & \text{if } |g| > m \\ \min_{\{o = \langle p, e \rangle \in O | g \cap e \neq \emptyset\}} cost(o) + h^m(s, p) & \text{if } |g| \leq m \end{cases}$$

h^m_+ is defined as above but using $\sum$ instead of max. This heuristic quickly becomes non-informative for $m > 1$.

5.2 IPC Benchmarks

Next, we look at the gains we typically obtain with more realistic problems taken from the International Planning Competition. We start by comparing the performance of unfolding vs state-space search and by demonstrating the effect of the heuristics h^1 and h^1_+ on the unfolding. In Figure 3, we present results for the first 21 IPC-4 AIRPORT instances, and for OPENSTACKS instances Warwick 91-120 which feature 10 products, 10 orders and an increasing ratio r =3 to 5 of products per order. We use the natural encoding of OPENSTACKS which allows several products to be produced in parallel. In contrast, the IPC-5 "propositional" version disables concurrency.

In AIRPORT, the unfolding expands up to 3 orders of magnitude fewer nodes than state-space search for the hardest instances. h^1 and h^1_+ further reduce this by up to 2 orders of magnitude, except for the easiest problems where h^1_+ underperforms. In OPENSTACKS, the gap between the unfolding and state space search is less spectacular, and decreases with r as the problems get easier. However, the benefits from using h^1_+ are striking: it systematically expands 100 ± 20 nodes across all problems. This shows that our guided unfolding is able to exploit the fact that non-optimal OPENSTACKS is an easy problem, and solve much larger instances than were previously within the reach of the unfolding technique.

We made similar observations in a range of domains that allow some degree of concurrency, from ROVERS to PIPESWORLD. In domains that fully disallow concurrency, such as PSR, the number of nodes expanded by unfolding and state space search is always identical for a given heuristic, and so unfolding gives no advantage.

Next, we turn to run times. Generally, the run times we obtain with the 0 heuristic are comparable, and in a number of cases better, than those obtained by the IPC-4 and IPC-5 optimal planners. A fair comparison is delicate because most of these planners, including the IPC-5 optimal track winner SATPLAN06[3], optimise the number of *parallel* plan steps. Cost-optimal planning is usually considered more challenging. Even in the simple case where actions have unit cost, we might produce plans that contains fewer actions than step-optimal parallel planners. To the exception of the HSP* family of planners[4], we are not aware of any IPC planner currently capable of optimising the sum of arbitrary action costs.

The middle and right-hand graphs in Figure 4 give a feel for the run time of PUP (run with the 0 heuristic), using the same AIRPORT and OPENSTACKS instances as previously. For reference, we also present the run times of the state of the art cost-optimal planner HSP0 (run with the `-seq` and `-bfs` options), and those of the state of the art step-optimal parallel planner SATPLAN06 (run with the default options). Note that SATPLAN06 is not able to solve any of the OPENSTACK instances within our 30mn time limit.

5.3 Petri Net Benchmarks

Our final experiment demonstrates the benefits of guiding the unfolding with planning heuristics, when analysing reacha-

[3] http://www.cs.washington.edu/homes/kautz/satplan/

[4] http://www.ida.liu.se/~pahas/hsps/

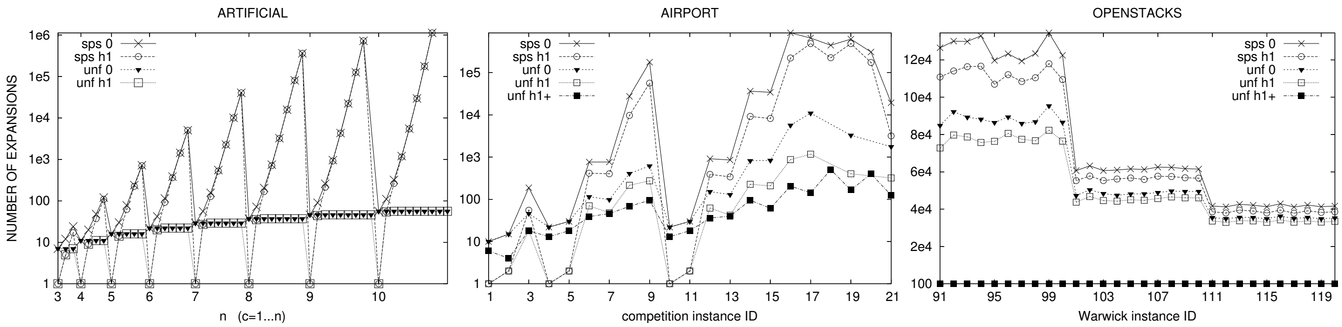

Figure 3: Number of Expansions for ARTIFICIAL (left), AIRPORT (middle), and OPENSTACKS (right).

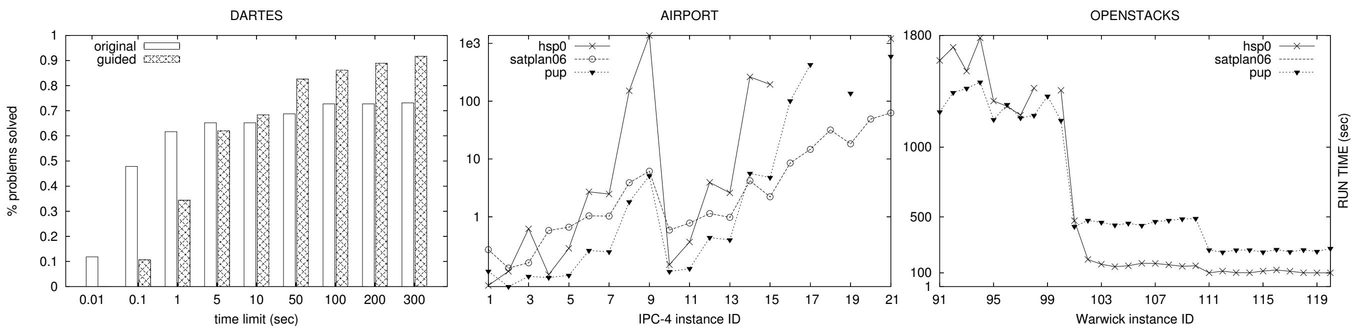

Figure 4: Reachability Coverage for DARTES (left). Run Times for AIRPORT (middle) and OPENSTACKS (right).

bility in Petri nets which have no connection to planning. As before, we are interested in determining whether a given transition of the Petri net is reachable.

We obtained a set of standard Petri net benchmarks from the developers of MOLE. Only one of them, DARTES [Corbett, 1996], which models the communication skeleton of a fairly complex Ada program, turned out to be challenging. MOLE is unable to decide the reachability of certain DARTES transitions in reasonable time, whereas for the other benchmarks in the set, MOLE generates even the complete finite prefix in a matter of seconds.

Figure 4 (left) compares the performance of the original version of MOLE to the version guided by the h^1_+ heuristic. For each of the 253 DARTES transitions, we recorded the time taken by each version to decide reachability; the graph shows the percentage of problems solved within given computation time limits ranging from 0.01 sec to 300 sec. The original breadth-first version of MOLE is quickly able to solve the simplest problems — 50% to 60% of the problems are solved within 0.1 to 1 sec. For those problems, the overhead in computing the heuristic outweighs the benefits. However, if a problem cannot be solved by the original version within 5 secs, the heuristic does help. In total, within our overall 300 sec time limit, the original version solves 185 of the 253 problems (73%), whereas the guided version solves 232 of them (92%). Only 4 of the problems that the original version could solve were unsolved by the guided version. Unsurprisingly, all the solved problems were positive decisions (the transitions were reachable). For sanity, we checked that depth-first search didn't improve on the results obtained with h^1_+. As it turns out, depth-first achieves over 65% coverage extremely quickly (solving the corresponding problems within 0.1 sec), but only reaches 76% coverage overall.

6 Conclusion

This paper exploits the relationship between planning and Petri net analysis to the advantage of both fields. On the one hand, we have demonstrated that Petri net unfolding, a form of partial order reduction [Godefroid, 1991], is a promising technique to recognise independent planning subproblems and treat them separately. On the other hand, we have shown that planning heuristics are able to effectively direct unfolding-based reachability analysis. The first product of our work is an original forward heuristic search algorithm for minimal-cost partially-ordered planning. The second is an enhanced reachability analysis tool which might be applicable where existing methods [Esparza and Schröter, 2001] suffer from having to generate a complete finite prefix.

We are not aware of any work that explores the potential of current Petri net analysis techniques for planning, in the depth given here. Meiller and Fabiani [2001] use colored Petri nets to implement a multi-valued version of the planning graph, merely obviating the need to explicitly consider certain types of permanent mutexes. Silva *et al.* [2000] recast plan extraction from the graph as a Petri net submarking reachability problem, yet without demonstrating many benefits.

Our work lays the foundation for planning in unfolding space. There are many possibilities for future work and improvement on our current approach. For instance, the reduction in number of nodes expanded by the h^1 heuristic does not always carry over to runtime, due to the cost of its recomputation at every node – this is a problem inherent to *forward* search, see [Bonet and Geffner, 2001]. Remedying this is critical to improving PUP's run time. Possible ways forward include switching to heuristics which only need to be computed once, such as pattern databases heuristics [Edelkamp, 2002] or h^2 for an inverted dynamics of the domain [Refanidis and Vlahavas, 2001]. Alternatively, we could investigate whether an analogue of regression search would make sense in the unfolding space.

The translation of planning operators into Petri nets is another area where improvements are likely. We first experimented with a translation linear in the number of propositions, but quadratic in the number of actions in the domain as it requires "mutex" places to ensure 1-safety. Unfortunately, we found that in many benchmarks, the number of mutex places greatly dominates the size of the Petri net. This motivated the need for the translation we give in the paper, which is linear when the actions are 1-safe, but is exponential in the number of operators effects in the worst case. Even though Petrify experienced only a few problems with the IPC benchmarks, it would be beneficial to extend it to combine both translations as appropriate. More ambitious developments concern more compact translations into high-level nets making use e.g. of first-order and multi-valued variables.

Finally, we believe that a more exhaustive analysis of the connections between planning (or search) and Petri net unfolding will be fruitful. This includes determining the precise relationship between the size of the unfolding and properties of the causal graph of the planning problem (such as treewidth), and identifying weaker properties of heuristics and orderings that guarantee completeness of the finite prefix.

Acknowledgements

Many thanks to Stefan Schwoon and Patrik Haslum for their help with MOLE and the experiments, respectively. We also thank Jonathan Billington, Blai Bonet, Javier Esparza, Malte Helmert, Rao Kambhampati, Maurice Pagnucco, John Slaney, David Smith, and the anonymous reviewers for interesting discussions and comments. Thanks to National ICT Australia (NICTA) and the Australian Defence Science & Technology Organisation (DSTO) for their support, in particular via the DPOLP (Dynamic Planning, Optimisation & Learning) project. NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian National Research Council.

References

[Backstrom and Nebel, 1995] C. Backstrom and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4), 1995.

[Blum and Furst, 1997] A. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[Bonet and Geffner, 2001] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129:5–33, 2001.

[Corbett, 1996] J. C. Corbett. Evaluating deadlock detection methods for concurrent software. *IEEE Trans. on Software Engineering*, 22(3), 1996.

[Edelkamp and Jabbar, 2006] S. Edelkamp and S. Jabbar. Action planning for directed model checking of Petri nets. *Electr. Notes Theoretical Computer Science*, 149(2), 2006.

[Edelkamp *et al.*, 2001] S. Edelkamp, A. Lluch-Lafuente, and S. Leue. Directed explicit model checking with HSF-SPIN. In *SPIN*, pages 57–79, 2001.

[Edelkamp, 2002] S. Edelkamp. Symbolic pattern databases in heuristic search planning. In *AIPS*, pages 274–283, 2002.

[Esparza and Schröter, 2001] J. Esparza and C. Schröter. Unfolding based algorithms for the reachability problem. *Fundam. Inform.*, 47(3-4), 2001.

[Esparza *et al.*, 2002] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20(3), 2002.

[Gazen and Knoblock, 1997] C. Gazen and C. Knoblock. Combining the expressivity of UCPOP with the efficiency of Graphplan. In *ECP*, 1997.

[Godefroid and Kabanza, 1991] P. Godefroid and F. Kabanza. An efficient reactive planner for synthesizing reactive plans. In *AAAI*, pages 640–645, 1991.

[Godefroid, 1991] P. Godefroid. Using partial orders to improve automatic verification methods. In *CAV*, pages 176–185, 1991.

[Haslum and Geffner, 2000] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *AIPS*, pages 140–149, 2000.

[Hickmott *et al.*, 2006] S. Hickmott, J. Rintanen, S. Thiébaux, and L. White. Planning via Petri net unfolding. Technical report, National ICT Australia, 2006.

[McMillan, 1992] K. L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *CAV*, pages 164–177, 1992.

[Meiller and Fabiani, 2001] Y. Meiller and P. Fabiani. Tokenplan: a planner for both satisfaction and optimization problems. *AI Magazine*, 22(3), 2001.

[Murata, 1989] T. Murata. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, 77(4), 1989.

[Refanidis and Vlahavas, 2001] I. Refanidis and I. Vlahavas. The GRT planning system: Backward heuristic construction in forward state-space planning. *Journal of Artificial Intelligence Resesearch*, 15:115–161, 2001.

[Silva *et al.*, 2000] F. Silva, M. A. Castilho, and L. A. Künzle. Petriplan: A new algorithm for plan generation (preliminary report). In *IBERAMIA-SBIA*, pages 86–95, 2000.