Optimality Properties of Planning Via Petri Net Unfolding: A Formal Analysis

Sarah Hickmott
School of CS and IT
RMIT University
Melbourne, Australia
sarah.hickmott@rmit.edu.au

Sebastian Sardina
School of CS and IT
RMIT University
Melbourne, Australia
sebastian.sardina@rmit.edu.au

Abstract

We provide a theoretical analysis of planning via Petri net unfolding, a novel technique for synthesising *parallel* plans. Parallel plans are generally valued for their execution *flexibility*, which manifests as alternative choices for the ordering of operators and potentially faster plan executions. Being a relatively new approach, the flexibility properties of plans synthesised via unfolding, and even the concurrency semantics supported by this technique, are particularly unclear and only understood at an informal level. In this paper, we first formally characterise the concurrency semantics of planning via unfolding as a further restriction on the standard notion of independence. More importantly, we then prove that plans obtained using this approach are *optimal deorderings* and *optimal reorderings* in terms of the number of ordering constraints on operators and plan execution time, respectively. These results provide objective guarantees on the quality of plans obtained by the unfolding technique.

Introduction

In this paper, we formally characterise the concurrency semantics supported by the Petri net unfolding approach to automated planning and prove optimality guarantees on the "flexibility" of plans constructed by the technique. Planning via Petri net unfolding (Hickmott et al. 2007; Bonet et al. 2008) is a novel approach for synthesising *parallel plans*, that is, partial-order plans with a *true* notion of concurrency (i.e., actions can temporally overlap). Parallel plans are highly desirable for many practical applications where greater flexibility is required at execution time, see e.g., (Nguyen and Kambhampati 2001; Smith, Frank, and Jónsson 2000). Such plans are, in principle, *flexible* in that they may avoid over-committing to action orderings. Making a plan least constrained is useful in that the scheduler may have several alternative execution realizations—sequences in the case of interleaved concurrency—to choose from. This is clearly desirable, for instance, in contexts where actions have deadlines and earliest release times, and a scheduler is used to post-process or adapt the plan to find a feasible schedule. Most importantly, making a plan least constrained is appealing if certain actions can be executed in parallel in order to reduce the execution time of the plan.

Copyright © 2009, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Planning via unfolding, a verification technique for asynchronous systems (Esparza, Römer, and Vogler 1996; McMillan 1992), was developed as a middle point – on the axis of commitment – between traditional state-based and plan-based approaches to automated planning. It offers the ability to reason about partially ordered actions whilst maintaining access to a fully-specified state. The later facilitates the use of state-based heuristics for guidance and pruning. The tactic of guiding the unfolding in the manner of heuristic search emerged from the planning community, and is referred to as *directed* unfolding.

Nonetheless, precisely what the *concurrency semantics* captured by the technique is and to *what extent* plans obtained via unfolding are flexible are still important open questions. Here, we provide a theoretical analysis to answer both questions. To that end, we rely on Bäckström (1998)'s principles for flexibility, which are based on two operations on plans aimed to reduce plan execution time and degree of commitment, namely *deordering* and *reordering*. Deordering a plan involves lifting existing action ordering constraints, while reordering allows for arbitrary modifications to the orderings.

The contributions of this paper are threefold. First, a characterisation of the the concurrency semantics supported by the unfolding technique is given, as that of *strong independence*, a further restriction on the standard notion of independence that happens to be analogous to the kind of concurrency captured by monitors for thread synchronization (Hoare 1974). To achieve this kind of concurrency, a suitable transformation of operators with persistent preconditions is employed. Second, the space of solutions explored is identified as exactly the set of parallel plans that are optimal with respect to plan deordering—no plan includes unnecessary constraints. This result provides new insight into the form (and size) of the search space of this planning technique. Finally, it is shown that when the search process is appropriately guided to prefer faster plans, planning via directed unfolding is guaranteed to construct a plan that is an optimal parallel reordering for execution time—changing its operator ordering constraints arbitrarily will not make it faster. These results are significant since *optimal* deorderings and reorderings of plans are not tractable operations (Bäckström 1998).

We note that we shall not be concerned here with an empirical evaluation of the planning via unfolding technique. Indeed, such kind of analysis is important and necessary, but it would be orthogonal to the theoretical evaluation carried out in this paper, and has been done—to some extent—elsewhere, e.g., (Hickmott 2008). The results presented here give objective guarantees on how flexible plans obtained via unfolding are, regardless of the domain or specific implementation. We believe, in fact, that a formal evaluation of the kind carried out here is necessary for a real understanding of this partial-order technique.

The rest of the paper is organized as follows. In the next section, we provide an overview of the planning via Petri net unfolding technique for synthesising parallel plans. We then characterise, formally, the semantics of concurrency captured by the planning technique. After that, we discuss—based on (Bäckström 1998)—the formal framework under which we shall perform our evaluation, namely, optimal limits of plan deordering and reordering. Finally, we prove optimality guarantees on the quality of plans constructed by the unfolding approach, w.r.t. to degree of commitment and plan execution time.

Planning via Petri net Unfolding

Here, we quickly review the planning technique that is the focus of our study, namely, planning via Petri net unfolding.

Let V be a set of propositions and $L_V = V \cup \{\neg v \mid v \in V\}$ the set of *literals* over V. The *complement* of a literal v is denoted by $\overline{v}$; this notation trivially extends to sets of literals. A *state* S is a consistent subset of L such that $|S| = |V|$. An *operator* $o = \langle \mathit{Pre}, \mathit{Eff} \rangle$ is characterised by its preconditions Pre and effects Eff, where Pre and Eff are consistent subsets of L_V. Operator o is *executable* in a state S iff $\mathit{Pre} \subseteq S$ (i.e., its preconditions hold); its execution will evolve the state to the new state $\mathit{Eff} \cup (S \setminus \overline{\mathit{Eff}})$.

A *planning problem* $\mathcal{P}$ is a tuple $\langle V, S_0, O, G \rangle$, where V is a set of propositions, S_0 is the initial state, O is a set of planning operators, and $G \subseteq L_V$ is a a set of literals representing the goal. A *parallel plan* is a tuple $\pi = \langle A, \prec \rangle$, where A is a multi-set of planning operators in O and $\prec$ is a strict partial order relation over A. We assume such plans have *true concurrency semantics*, meaning that un-ordered operators may temporally overlap their executions.[1] A *linearisation* of plan π is any sequential ordering of A which respects $\prec$. Plan π is a *solution plan* for planning problem $\mathcal{P}$ if any linearization of π will transition the system from S_0 to a state S_g where all goal propositions in G hold (i.e., $G \subseteq S_g$). Finally, a plan can have a *cost* associated, by suitably aggregating the cost of operators. In particular, when the cost of an operator is taken to be its execution duration (i.e., durative actions), then the parallel cost of a plan amounts to its (minimum) execution time.

Planning via Petri net unfolding is a novel method for synthesising parallel solution plans, which utitlises Petri net techniques for reasoning about concurrent systems. Roughly speaking, a planning domain is represented as a Petri net, and reachability analysis (via unfolding) determines if and how the net can be transitioned from the state representing S_0 to one in which G holds. Below, we briefly go over these notions by following (Hickmott et al. 2007).

Place transition net

A *Petri net* or *place transition net* (PT-net) is a tuple $\Sigma = \langle N, M_0 \rangle$, where N is a directed bipartite graph with place nodes P and transition nodes T, and M_0 is the *initial marking* of N. Dynamic behavior is captured by a flow relation, $F \subseteq (P \times T) \cup (T \times P)$, indicating the presence (or absence) of arcs between places and transitions, and vice-versa. A *marking* $M : P \mapsto \mathbb{N}$ of a PT-net captures a state of the modeled system by assigning zero, one, or more tokens to each place. In this paper, we only consider 1-safe nets, meaning it is never possible for more than one token to exist in a place, and for easier reading we will often depict a marking by the set of places containing a token.

The preset ${}^{\bullet}x$ of node x is the set $\{y \mid y \in P \cup T, F(y,x) = 1\}$, and its postset $x^{\bullet}$ is the set $\{y \mid y \in P \cup T, F(x,y) = 1\}$. Marking M *enables* transition t if $M(p) > 0$ for all $p \in {}^{\bullet}t$. The *occurrence* of an enabled transition t absorbs a token from each of its preset places and puts one token in each postset place. This corresponds to a state transition $M \xrightarrow{t} M'$ in the system, moving the net from M to the new marking $M' = (M \setminus {}^{\bullet}t) \cup t^{\bullet}$. An *occurrence sequence* $\sigma = t_1, \ldots, t_n$ is a sequence of state transitions, i.e., there exist markings $M_1, \ldots, M_n$ such that $M_0 \xrightarrow{t_1} \cdots \xrightarrow{t_n} M_n$; denoted as $M_0 \xrightarrow{\sigma} M_n$. A marking M is *reachable* if there is a sequence σ such that $M_0 \xrightarrow{\sigma} M$. The *reachability problem* seeks to find σ for a given PT-net and a (partially specified) marking M. Figure 1(b) depicts a PT-net with place nodes $P = \{a_1, a_2, b, c, \widehat{a}, \widehat{b}, \widehat{c}\}$, transition nodes $T = \{o_1, o_2, o_3, o_4, o_5\}$, and initial marking $M_0 = \{a_1, a_2, b, c\}$. In this PT-net, transitions o_1 and o_2 are concurrently enabled by M_0; but transitions o_1 and o_3 are not since they conflict on place a_1. Also, marking $M = \{\widehat{a}, \widehat{b}, \widehat{c}\}$ is reachable since $\sigma = o_1, o_2, o_3$ is an occurrence sequence such that $M_0 \xrightarrow{\sigma} M$ applies.

Petri net representation of planning problem

Hickmott et al. (2007) propose a translation from a planning problem $\mathcal{P} = \langle V, S_0, O, G \rangle$ to PT-net $\mathsf{pnet}(\mathcal{P})$, such that the problem of finding a solution plan for $\mathcal{P}$ is cast as a reachability problem for $\mathsf{pnet}(\mathcal{P})$. The translation proceeds in three steps: *(i)* make the operators toggling; *(ii)* eliminate negative preconditions; and *(iii)* map the planning problem to a PT-net. The purpose of the first two steps is to make operators compatible with the syntax and semantics of a PT-net.

Informally, an operator is "toggling" if all its effects imply actual changes (in the truth of propositions).[2]

Definition 1. An operator $o = \langle \mathit{Pre}, \mathit{Eff} \rangle$ is *toggling* iff $\overline{\mathit{Eff}} \subseteq \mathit{Pre}$. ■

Translating the original operators into toggling ones helps to maintain logical consistency with respect to the state

[1] Note the difference with *partial-order* plans where an *interleaved* notion of concurrency is assumed.

[2] Toggling operators were referred to as "1-safe" in (Hickmott et al. 2007).

propositions and model negative effects. A non-toggling operator can be converted into a collection of toggling ones, by including in the preconditions the complement of every literal appearing in the effects, one per combination of literals missing from the preconditions. For example, the operator $o = \langle\{a, \neg b, d\}, \{c, \neg d\}\rangle$ is accounted for by the two toggling operators $o_1 = \langle\{a, \neg b, c, d\}, \{\neg d\}\rangle$ and $o_2 = \langle\{a, \neg b, \neg c, d\}, \{c, \neg d\}\rangle$.

Compiling away negative preconditions is necessary because a transition in a Petri net can only be conditioned on the presence of tokens in places, not their absence. This is achieved by introducing the set of propositions $\widehat{A} = \{\widehat{a} \mid a \in A\}$; the idea is that $\widehat{a}$ is true exactly when a is false. Thus, the operator o_1 above becomes $\langle\{a, \widehat{b}, c, d\}, \{\widehat{d}\}\rangle$. Let μ be the function which takes a set of operators and produces a set of toggling operators with no negative preconditions.

Finally, in the third step, the planning problem is mapped to a PT-net $\mathsf{pnet}(\mathcal{P}) = \langle N, M_0\rangle$ as follows. Place nodes are the state propositions $A \cup \widehat{A}$ and transition nodes are the operators $\mu(O)$. For each transition $o = \langle \mathit{Pre}, \mathit{Eff}\rangle \in \mu(O)$, ${}^\bullet o = \mathit{Pre}$ and $o^\bullet = \mathit{Eff} \cup \{p \mid p \in \mathit{Pre}, \neg p \notin \mathit{Eff}\}$. Note that the postset of a transition explicitly includes the persistent (non-deleted) preconditions. The initial marking is specified as $M_0(p) = 1$ iff $p \in S_0$.

Planning via unfolding

Having cast a planning problem as a PT-net reachability problem, one can synthesise a solution plan via *unfolding*. Unfolding $\mathsf{pnet}(\mathcal{P})$ essentially enumerates, in a forward manner, the space of parallel plans for planning problem $\mathcal{P}$.

Unfolding a PT-net $\Sigma = \langle N, M_0\rangle$ produces a pair $\mathit{Unf}(\Sigma) = \langle ON, \varphi\rangle$, where $ON = \langle B, E, F'\rangle$ is an occurrence net and φ is a homomorphism that maps the nodes in ON to nodes in N. An *occurrence net* is a PT-net without cycles or backward conflicts. Multiple transitions are in *backward conflict* if they all feed into some (same) place p; to eliminate backward conflicts, the unfolding process replicates p so that each instance of p has a clearly identifiable history. Note that net ON may still contain *forward conflicts*, that is, cases where a single place feeds into multiple transitions. In a planning context, forward conflicts may represent a choice between non-independent operators; $\mathit{Unf}(\mathsf{pnet}(\mathcal{P}))$ captures every possible resolution of conflict between operators in $\mathcal{P}$.

In the occurrence net ON, places and transitions are called *conditions* B and *events* E, respectively; each event (condition) in ON is a particular instance of a transition (place) in N, uniquely defined by the system transformations which led to it being executed (containing a token). For example, Figure 1(c) depicts part of the unfolding of the PT-net in Figure 1(b); observe that there are multiple instances of place a_2, as there are multiple ways to make this proposition true.

To understand the unfolding process, the most important notions are that of a configuration and its marking. A configuration represents a possible partial run of the original PT-net, beginning at M_0. Formally, a *configuration* C is a set of events in ON such that C is causally closed and contains no forward conflict. The *local configuration* of an event e, denoted $[e]$, is the minimal configuration containing event e. Intuitively, $[e]$ stands for one possible (parallel) history of events leading to the occurrence of transition $\varphi(e)$. In our example, $[e_5] = \{e_1, e_2, e_5\}$ is a (local) configuration, but $\{e_1, e_3\}$ is *not* a configuration because e_1 and e_3 are in forward conflict.

From the planning perspective, a configuration C *induces* a unique parallel plan $\pi_C = \langle A, \prec\rangle$, where A is the set of operator instances represented by C and $\prec$ is the partial-order relation induced by the relation F' on the set C, namely, $a \prec b$ iff there exists condition x such that $aF'x$ and xFb. For instance, $\pi_{[e_5]} = \langle\{o_1, o_2, o_3\}, \{o_1 \prec o_3, o_2 \prec o_3\}\rangle$.

In addition, a configuration C can be associated with a marking $\mathsf{Mark}(C)$ of the original PT-net by identifying which places will contain a token after the occurrence of all transitions represented by the events in C. Formally, $\mathsf{Mark}(C) = \varphi((B_0 \cup C^\bullet) \setminus {}^\bullet C)$, where $C^\bullet$ (resp. ${}^\bullet C$) is the union of postsets (resp. presets) of all events in C. For instance, the marking for event e_5's local configuration is $\mathsf{Mark}([e_5]) = \{\widehat{a}, \widehat{b}, \widehat{c}\}$.

The places initially marked in N have a 1-1 mapping with a set of initial conditions B_0 in ON. Beginning with $ON = \langle B_0, \emptyset, \emptyset\rangle$, the unfolding process builds the occurrence net by repeatedly identifying any transition $o = \langle \mathit{Pre}, \mathit{Eff}\rangle$ such that ON contains conditions representing *Pre*, and the causal closure of these conditions has no forward conflict. If satisfied, then instances of o and *Eff* are added to ON accordingly. If, at some point, there is a configuration C such that $\mathsf{Mark}(C) = M$, then C captures a solution to the reachability problem defined by marking M. It easy to see that reachable markings correspond one-to-one to reachable states in the planning problem, and hence, we should sometime blur their distinction. It follows then that if C is a configuration such that $G \subseteq \mathsf{Mark}(C)$, then its induced plan $\pi_C = \langle A, \prec\rangle$ solves the planning problem of interest, where A is the set of operator instances in C and $\prec$ is the partial order relation captured by F' on the set C, i.e. $a \prec b$ iff $\exists x$ such that $aF'x$ and xFb.

Concurrency Semantics of PN Unfolding

In the previous section, we defined the notion of a solution plan based on the set of all possible linearisations of a parallel plan. However, when considering plans with true concurrency, this is not enough: one must also take into account which operators can reasonably be allowed to temporally overlap.[3] In fact, different concurrency semantics may be used to specify under which conditions two or more operators can execute *at the same time*.

Currently, the notion of concurrency supported by the PT-net unfolding approach to planning, that is, the one resulting from the planning problem to PT-net translation and the PT-net dynamics, is not well understood. In this section, we address this issue by formally characterising the tech-

[3]What is "reasonable" could depend on various factors, ranging from the semantic assumptions which were made during formulation of a planning problem to the physical limitations of the particular artifacts which are to execute the plan.

nique's concurrency semantics and, subsequently, identifying the search space of plans explored.

Comparison with Planning Independence

One of the standard notions of concurrency used by parallel planners is that of independence—two operators can only execute at the same time if they are independent.

Definition 2. Two operators $o_1 = \langle Pre_1, Eff_1 \rangle$ and $o_2 = \langle Pre_2, Eff_2 \rangle$ are *independent*, denoted by $o_1 \sim_I o_2$, iff for all $i, j \in \{1, 2\}$ and $i \neq j$, it is the case that *(i)* $Pre_i \cap \overline{Eff_j} = \emptyset$; *(ii)* $Eff_i \cap \overline{Eff_j} = \emptyset$; and *(iii)* $Pre_i \cap \overline{Pre_j} = \emptyset$. ■

The first condition guarantees that the precondition of operator o_i will not be threatened by the effects of operator o_j; the second condition is the usual post-exclusion principle (Bäckström 1998, Definition 5.2) stating that both operators do not contradict themselves in their effects; and the last condition rules out situations requiring inconsistent preconditions. Pairs of operators which do not satisfy the conditions for independence have been described as *eternally mutually exclusive* in the context of Graphplan techniques (Smith and Weld 1999), as the mutex relationship persists regardless of the state of execution. Hence, we say that a plan $\pi = \langle A, \prec \rangle$ is *independent* if for any two different operator (instances) $o_1, o_2 \in A$ such that $o_1 \nsim_I o_2$, it is the case that either $o_1 \prec o_2$ or $o_2 \prec o_1$.

The concurrency semantics of plans synthesised via PT-net unfolding differs from that of independence in two ways. The first one, already identified in (Hickmott et al. 2007), involves two operators sharing the same *persistent precondition*. For instance, $o_1 = \langle \{a\}, \{b\} \rangle$ and $o_2 = \langle \{a\}, \{d\} \rangle$ are independent but any two events in the unfolding corresponding to such operators may be considered in forward conflict—the corresponding transitions in the Petri net take a token from place a and therefore, due to the 1-safeness property of the constructed net,[4] they can never fire concurrently.

The second source of mismatch between the notion of independence and the concurrency semantics of planning via unfolding involves two operators having *common effects*. Take for instance $o_1 = \langle \{a\}, \{b\} \rangle$ and $o_2 = \langle \{d\}, \{b\} \rangle$. Clearly o_1 and o_2 are independent. However, subject to the PT-net translation the transitions corresponding to $o_1^1 = \langle \{a, \widehat{b}\}, \{b, \neg\widehat{b}\} \rangle \in \mu(o_1)$ and $o_2^1 = \langle \{\widehat{b}, d\}, \{b, \neg\widehat{b}\} \rangle \in \mu(o_2)$ would appear in forward conflict in the unfolding, as they share the same precondition $\widehat{b}$. (Notice in fact that the auxiliary operators o_1^1 and o_2^1 are *not* independent).

While it is not clear how to address the latter source of mismatch, one can resolve the former by extending the PT-net translation to avoid conflict between operators with the same persistent precondition. In (Hickmott et al. 2007), it was observed that the use of *read-arcs* (Christensen and Hansen 1993) in the Petri net would remove the conflict between two operators with a shared persistent precondition. Read-arcs allow reading a token without consuming it, like reading in a database. However, as Hickmott et al. pointed out, directly incorporating read-arcs in the PT-net would complicate the unfolding process considerably (see (Vogler, Semenov, and Yakovlev 1998)). So, we shall instead capture read-arc semantics with regular arcs by applying the place replication technique described in (Vogler, Semenov, and Yakovlev 1998). The idea behind this transformation is that a place p is replicated as many times as there are transitions "reading" it, such that each of these transitions has its own copy of p to "read." For consistency, any transition consuming/writing to p must now consume/write to *all* these replications of p.

Figure 1(a) depicts the PT-net representation of a planning problem. There, transitions o_1, o_2, o_3 each contain place a in their preset; subsequently, they can never execute concurrently. However, o_1 and o_2 do not change the value of a, i.e., a is a persistent precondition for these operators. To support concurrency between o_1 and o_2, place a is replaced with places a_1 and a_2 such that $a_1 \in {}^\bullet o_1 \cap o_1{}^\bullet$ and $a_2 \in {}^\bullet o_2 \cap o_2{}^\bullet$, as shown in Figure 1(b). In addition, transition o_3 now includes both these places in its preset, and transitions o_4 and o_5 include both a_1 and a_2 in their postset, instead of a. See that o_1 and o_2 are now concurrently enabled when places a_1 and a_2 contain a token, though still in forward conflict with o_3, and that a_1 and a_2 will have a token iff $\widehat{a}$ has no token.

From now on, this paper will assume the *extended* PT-net encoding for a given planning problem $\mathcal{P}$, denoted $\mathsf{pnet}^+(\mathcal{P})$, which amounts to applying the aforementioned persistent-precondition transformation to the original encoding $\mathsf{pnet}(\mathcal{P})$., i.e., $\mathsf{pnet}^+(\mathcal{P}) = \mathsf{persprec}(\mathsf{pnet}(\mathcal{P}))$.

Strong Independence

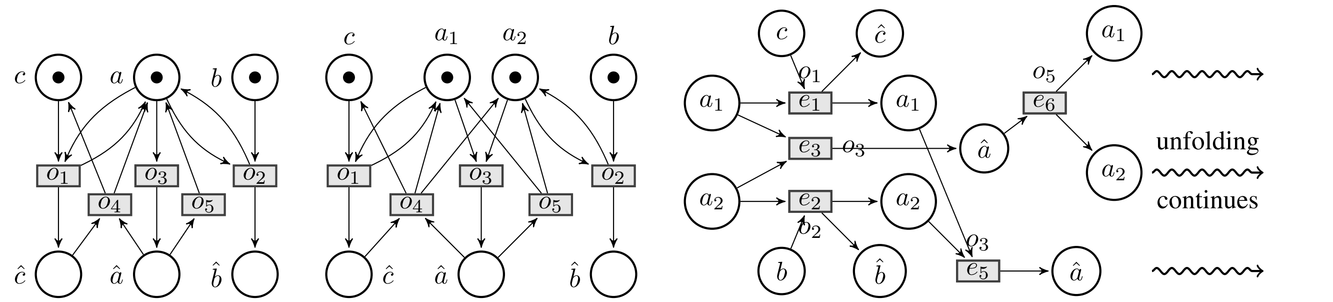
Let us now *characterise* the concurrency semantics supported by the extended PT-net encoding $\mathsf{pnet}^+(\mathcal{P})$. To that end, we first define a stronger version of independence.

Definition 3. Two operators $o_1 = \langle Pre_1, Eff_1 \rangle$ and $o_2 = \langle Pre_2, Eff_2 \rangle$ are *strongly independent in a state* S, denoted by $o_1 \sim_{SI}^S o_2$, iff $o_1 \sim_I o_2$ and $\overline{S} \cap (Eff_1 \cap Eff_2) = \emptyset$. ■

Observe that, unlike the notion of independence, strong independence is relative to a particular state. The extra condition requires that two actions having a shared effect may execute concurrently only if such effect is already true—the actions are *not* actually updating the proposition in question. This condition, together with those from independence, makes strong independence analogous to *monitors* (Hoare 1974) for thread synchronization. Intuitively, actions are assumed to "*lock*" the propositions they refer to, either in the preconditions or effects. A proposition is locked in shared mode if the action does not change its truth value (read only access), whereas the proposition is locked in exclusive mode if its truth value is to be changed by the action (read and write access). This behavior may be natural, for instance, in domains where the actual implementation of an action requires knowing the state of those variables the action depends on.

As done with independence, let us next define the set of plans that are strongly independent. We use *state*(π, S, o) to denote the set of all those states in which operator o in plan

[4] By construction, the PT-net representation of a planning problem is *1-safe* (Hickmott et al. 2007, Theorem 2), which means a place can never contain more than one token.



(a) PT-net encoding $\mathsf{pnet}(\mathcal{P})$ (b) Extended encoding $\mathsf{pnet}^+(\mathcal{P})$ (c) Part of the PT-net unfolding $\mathit{Unf}(\mathsf{pnet}^+(\mathcal{P}))$

Figure 1: (a) The PT-net encoding for a planning problem; (b) the extended encoding with persistent preconditions; and (c) part of its PT-net unfolding. In (c), a condition node x is labeled with $\varphi(x)$; and $\varphi(e)$ is written besides each event node e.

π may potentially be executed when a linearisation of plan π is executed in state S.

Definition 4. Let S be a state and $\pi = \langle A, \prec \rangle$ a parallel plan. Plan π is *strongly independent in* S iff for any two different operator (instances) $o_1, o_2 \in A$ such that $o_1 \nsim_{SI}^{S'} o_2$, for some $S' \in \mathit{state}(\pi, S, o_1)$, either $o_1 \prec o_2$ or $o_2 \prec o_1$. ■

Thus, when considering the strong independence semantics of concurrency, a plan is *valid* for a planning problem when it solves the planning problem and satisfies the concurrency constraints.

Definition 5. Let $\mathcal{P} = \langle V, S_0, O, G \rangle$ be a planning problem. A parallel plan π is $\mathcal{P}$*-valid* iff π is a solution plan for $\mathcal{P}$ and π is strongly independent in S_0. ■

Observe that it could happen that *every linearisation* of a plan may legally execute in the domain (and solve the planning problem), while the plan does *not* meet the concurrency semantics. The fact that a plan π is $\mathcal{P}$-valid means all possible executions of π respect the strong independence notion of concurrency. Notice also that, in solving $\mathcal{P}$, we restrict our attention to plans that are strongly independent in state S_0, the initial state of $\mathcal{P}$.

Let us next prove that the unfolding of $\mathsf{pnet}^+(\mathcal{P})$ actually captures *exactly* the above notion of concurrency, namely, strong independence. We start by relating the notions of strong independence (defined at the planning problem level) and forward conflict (defined for the unfolding).

Lemma 1. *Let e_{o_1} and e_{o_2} be two events in $\mathit{Unf}(\mathsf{pnet}^+(\mathcal{P}))$, representing operators o_1 and o_2 in $\mathcal{P}$, respectively. If $o_1 \sim_{SI}^{S} o_2$, where $S = \mathsf{Mark}([e_{o_1}] \setminus \{e_{o_1}\})$, then e_{o_1} and e_{o_2} are not in forward conflict.*

Proof (Sketch). This is proved by showing that, considering that these events capture particular transformations of the original domain operators o_1 and o_2, if e_{o_1} and e_{o_2} are in forward conflict, then either o_1 and o_2 were not strongly independent or the persistent precondition transformation was not carried out. □

In words, two events in the PT-net unfolding representing operators which are strongly independent in any state they could be executed, will not be in forward conflict with each other. This result is important in that it guarantees that the events corresponding to two strongly independent operators will not be in *direct* conflict in the unfolding, which implies that, in principle, they may be performed concurrently. Of course, they may end up ordered due to additional constraints with other events (i.e., operators).

The following two results characterise the concurrency supported by the planning via unfolding approach, as that of strong independence. Informally, we prove that all solutions constructed by the unfolding technique are in fact strongly independent plans and that every strongly independent plan that solves the problem is accounted for by some plan induced in the unfolding.

Theorem 1. *Let $\mathcal{P} = \langle V, S_0, O, G \rangle$ be a planning problem. If there exists configuration $C \in \mathit{Unf}(\mathsf{pnet}^+(\mathcal{P}))$ such that $G \subseteq \mathsf{Mark}(C)$, then plan π_C is $\mathcal{P}$-valid.*

Proof (Sketch). Plan π_C solves $\mathcal{P}$ due to Theorem 1 in (Hickmott et al. 2007) and the soundness of the persistent-precondition transformation. The second part involves proving that if π_C were not strongly independent in S_0 then either *(i)* there are two events in C that are in forward conflict; or *(ii)* there exists an (inconsistent) reachable marking M and a proposition a such that $M(a) \geq 1$ and $M(\widehat{a}) \geq 1$, implying then that there is a reachable state in which both a and $\neg a$ hold. □

Theorem 2. *Let $\mathcal{P} = \langle V, S_0, O, G \rangle$ be a planning problem. If plan π is $\mathcal{P}$-valid, then there exists a configuration $C \in \mathit{Unf}(\mathsf{pnet}^+(\mathcal{P}))$ such that $G \subseteq \mathsf{Mark}(C)$ and such that all linearisations of π are linearisations of π_C.*

The fact that a solution may not show up itself as a plan in the unfolding, but only implicitly within another plan, is due to some optimality properties of the technique that are the focus of the next section. (Theorem 2 actually follows almost directly from Theorem 5 below.)

We close this section by noting an interesting observation that will come handy later on. Namely, the notions of independence and strong independence coincide when at least one of the operators is *toggling*. Recall that a toggling operator requires all its effects to be false before its execution.

Theorem 3. *Let o_1 and o_2 be two operators such that o_2 is toggling. Then, $o_1 \sim_{SI}^{S} o_2$ for some state S iff $o_1 \sim_I o_2$.*

Proof. ($\Rightarrow$) Holds trivially as strong independence implies independence. ($\Leftarrow$) On the contrary, suppose that $l \in \mathit{Eff}_{o_1} \cap \mathit{Eff}_{o_2}$. Since o_2 is toggling, $\neg l \in \mathit{Pre}_{o_2}$. Therefore, $\neg l \in \overline{\mathit{Eff}_{o_1}} \cap \mathit{Pre}_{o_2}$ and operators o_1 and o_2 are not independent. $\square$

Plan Quality: Deordering and Reordering

In this section, we discuss optimality criteria for evaluating the quality of parallel plans. Consider a parallel plan $\pi = \langle A, \prec \rangle$. A critical question is, are all of the ordering constraints necessary? A less constrained plan may offer more flexibility to the executor. Moreover, if we change the partial order relation $\prec$ to allow different actions to temporally overlap, can the plan then be executed more quickly?

Bäckström (1998) studies the computational aspects of modifying the partial order relation of a plan, via operations referred to as *deordering* and *reordering*, in order to make the plan less constrained or to minimise its execution time. Deordering a plan involves lifting (i.e., deleting) existing ordering constraints from the plan; whereas reordering a plan—a stronger operation—involves the arbitrary modification of the partial order relation. Formally, consider two plans with the same operator set: $\pi' = \langle A, \prec' \rangle$ and $\pi = \langle A, \prec \rangle$. Then, π' is a *parallel reordering* of π (and vice versa). Furthermore, π' is a *parallel deordering* of π iff $\prec' \subseteq \prec$. Consider, for instance, the following three plans that are legally executable in the planning problem underlying the PT-net encoding in Figure 1(b):

$$\pi = \langle \{o_1, o_2, o_3, o_5\}, \{o_1 \prec o_2 \prec o_3 \prec o_5\} \rangle;$$
$$\pi_d = \langle \{o_1, o_2, o_3, o_5\}, \{o_1 \prec o_3, o_2 \prec o_3, o_3 \prec o_5\} \rangle;$$
$$\pi_r = \langle \{o_1, o_2, o_3, o_5\}, \{o_5 \prec o_1, o_5 \prec o_2, o_3 \prec o_5\} \rangle.$$

Here, plan π_d is a parallel deordering of plan π, whereas plan π_r is a parallel reordering of plan π (and also of π_d).

It is not difficult to see that that the above two operations on plans could be used to improve their degree of flexibility as well as to make them faster. It may be possible to deorder or reorder a plan to make it less committed, thus increasing its flexibility and allowing more scheduling and execution options. In the above example, plan π_d is less constrained than π, as the former (but not the latter) allows the concurrent execution of operators o_1 and o_2. In turn, this additional "scheduling" option may yield faster overall plan execution. For example, if operators o_1, o_2, o_3 and o_5 each have a duration of 1 time unit, then the minimum execution time for plan π is 4 units, whereas plan π_d can be executed in only 3 units of time. Similarly, plan π_r has better properties than the original π; it has fewer operator ordering constraints and can be executed in 3 time units. Note that all three plans are strongly independent in the initial state, and all possible linearisations are executable (relative to the initial state implied by Figure 1). Conversely, the deordering $\pi'_d = \langle \{o_1, o_2, o_3, o_5\}, \{o_1 \prec o_3, o_2 \prec o_3\} \rangle$ would allow o_5 to be interleaved at any point in the plan, or executed concurrently with any other operator; however, not all such linearisations are executable and furthermore the plan is not strongly independent in S_0.

So, the idea then is to look for the *optimal limits* of the deordering and reordering operations, in a similar way as done in (Bäckström 1998).[5] Recall from the previous section that a plan is $\mathcal{P}$-valid in our context when it is a solution for $\mathcal{P}$ and respects the strong independence notion of concurrency. Consider then a planning problem $\mathcal{P}$ and a parallel plan $\pi = \langle A, \prec \rangle$ that is $\mathcal{P}$-valid:

- DF Plan π is a *minimal parallel deordering w.r.t. flexibility* iff there exists no parallel deordering $\pi' = \langle A, \prec' \rangle$ of π that is $\mathcal{P}$-valid and such that $|\prec'| < |\prec|$.
- DT Plan π is a *minimal parallel deordering w.r.t. execution time* iff there exists no parallel deordering π' of π that is $\mathcal{P}$-valid and has a shorter execution time than π.
- RF Plan π is a *minimum parallel reordering w.r.t. flexibility* iff there exists no parallel reordering $\pi' = \langle A, \prec' \rangle$ of π' that is $\mathcal{P}$-valid and such that $|\prec'| < |\prec|$.
- RT Plan π is a *minimum parallel reordering w.r.t. execution time* iff there exists no parallel reordering π' of π that is $\mathcal{P}$-valid and has a shorter execution time than π.

We observe that, when considering parallel plans, Bäckström (1998) restricts the attention to execution time only. We, in contrast, consider both execution time and "flexibility" in the sense of the level of operator ordering commitments. The point is that one may not be able to manipulate the ordering constraints to get a faster plan, but it may be possible to get one with less ordering constraints, providing thus greater flexibility to the executor/scheduler. Indeed it is not hard to see that DF implies DT—one cannot make a plan faster if it is not possible to lift any constraints at all. The converse, though, does not apply; one may be able to lift an ordering constraint to make a new plan that is more flexible but has the same execution time. On the the other hand, no relation exists between RF and RT. Finally, as reorderings are also deorderings, it is not difficult to see that RF (RT) trivially implies DF (DT).

Optimality Guarantees

Using the above criteria, we next provide optimality guarantees for the PT-net unfolding approach to planning. Specifically, we show that this technique can synthesise plans which are optimal in terms of DF, DT, and RT, but not RF.

On the Flexibility Properties of Solutions

We first concentrate on characterising the space of plans explored by the unfolding process. The following theorem provides necessary conditions for a solution plan to be represented in the unfolding, namely, it has to amount to a minimal parallel deordering w.r.t. flexibility.

Theorem 4. *Let $\mathcal{P} = \langle V, S_0, O, G \rangle$ be a planning problem. If C is a configuration in $\mathit{Unf}(\mathsf{pnet}^{+}(\mathcal{P}))$ such that $G \subseteq \mathsf{Mark}(C)$, then the plan π_C is a minimal parallel deordering w.r.t. flexibility.*

[5] Bäckström (1998) defines notions of optimality which assess one plan relative to another. Since we are interested here in the assessment of *single* plans, we slightly adapt such notions for our purposes. Nonetheless, it should be clear that our notions are already implicitly accounted in Bäckström's ones.

Proof (Sketch). We prove that if $\pi_C = \langle A, \prec \rangle$ and $\pi' = \langle A, \prec' \rangle$, such that $\prec' \subset \prec$, then π' is not $\mathcal{P}$-valid (and thus π_C is a minimal parallel deordering w.r.t. flexibility). By Theorem 1, π_C is $\mathcal{P}$-valid. Remove a set of tuples from $\prec$ to obtain plan $\pi' = \langle A, \prec' \rangle$ where $\prec' \subset \prec$. It must be that there exists $e_1, e_2 \in C$, where e_1, e_2 correspond to instances of operators o_1, o_2, such that $e_1^\bullet \cup {}^\bullet e_2 \neq \emptyset$, and $o_1 \prec o_2$ but $o_1 \not\prec' o_2$. It must also be that $X = \varphi({}^\bullet e_1) \in S$ where S is some state the system could be in when the plan preceding these instances of o_1 and o_2 has been executed. So, when the system is in state S, according to π it is possible to execute o_1 and according to π' it is possible to execute either o_1 or o_2 or both concurrently. For π' to be $\mathcal{P}$-valid, it must at least be true that any linearisation of o_1, o_2 is valid in S, and, that $o_1 \sim_{SI}^{S} o_2$ (since, by definition, $S \in \mathit{state}(\pi', S_0, o_2)$). The proof continues by showing that any combination of operators that could cause the construct $e_1^\bullet \cap {}^\bullet e_2 \neq \emptyset$ to appear in C, would break one of the above conditions. □

Note that this result entails that π_C is also a minimal parallel deordering w.r.t. execution time. Furthermore, one can prove that *all* minimal parallel deorderings w.r.t flexibility are indeed represented in the unfolding.

Theorem 5. *Let $\mathcal{P}$ be a planning problem and plan π be $\mathcal{P}$-valid. If π is a minimal parallel deordering w.r.t. flexibility, then π will be represented by a configuration in $\mathit{Unf}(\mathsf{pnet}^+(\mathcal{P}))$.*

Proof (Sketch). This is shown by constructing a legal configuration C in $\mathit{Unf}(\mathsf{pnet}^+(\mathcal{P}))$ by induction on the maximum *rank* of π such that $\pi = \pi_C$. Any plan $\pi = \langle A, \prec \rangle$ induces a *rank* order on A, such that $\mathit{rank}(o) = n$ iff there exist $o_1, \ldots, o_n \in A$ such that $o_1 \prec \ldots \prec o_n \prec o$, i.e., operator o comes after n consecutive operators. The idea is that operators at the same rank level may be executed concurrently, which implies that they are strongly independent. Lemma 1 is then applied to conclude that the corresponding events cannot be in forward conflict. □

Theorems 4 and 5 together characterise the solution space explored by the unfolding technique as the set of all parallel plans that are minimal deorderings w.r.t flexibility under a strong independence notion of concurrency.

We observe that it is straightforward to generalise these results to *all* plans enumerated in the unfolding when only executability is considered (i.e, when the goal is just "`true`"). More concretely, it is possible to prove that no ordering constraint can be lifted from a plan induced by *any* configuration in $\mathit{Unf}(\mathsf{pnet}^+(\mathcal{P}))$ without invalidating it.

On the Execution Time of Solutions

An important feature of the planning via unfolding technique is that it can be *directed* to prefer plans which optimise a specified cost function. Here, we concentrate on analysing the case when this cost function captures execution time. First, though, we need to provide more information on this "directed" technique.

The reader may have observed that the unfolding process described earlier could actually be infinite. However, Esparza, Römer, and Vogler (1996) and McMillan (1992) developed a way to "cut-off" the unfolding at appropriate events, thus allowing us to either construct a solution plan on-the-fly or deem the problem unsolvable. We refer to this as the ERV-Fly algorithm. The algorithm works as follows. When an event is identified for possible addition to the unfolding space, it is added to a queue of "potential extensions," according to some partial order preference $\prec_{[]}$ on local configurations. Then, when event e is actually removed from the front of the queue, if the (current) unfolding space already contains an event e' such that $\mathsf{Mark}([e']) = \mathsf{Mark}([e])$ and $[e'] \prec_{[]} [e]$, then e is *not* added to the unfolding space—event e is deemed a "cut-off" event as everything possible from e will be possible from e'. Hickmott et al. (2007) noted that since preference $\prec_{[]}$ essentially *directs* the unfolding, one could improve efficiency and find minimum-cost solutions by basing $\prec_{[]}$ on a cost function consisting of the *additive* cost of operators contained in the local configuration $[e]$ of an event e, and an admissible estimation of the cost from $\mathsf{Mark}([e])$ to a goal state. In this way, *directed unfolding* operates similarly to heuristic search. Bonet et al. (2008) then showed that $\prec_{[]}$ could alternatively incorporate an inadmissible heuristic function. More recently, Hickmott (2008), proved that the preference orders used to queue events and the determine cut-off events do not need to be the same. The requirements on the queue order were thus weakened, and this opened the door for directing the unfolding to prefer configurations (i.e., plans) with minimal the *parallel* cost (i.e., execution time). Let us refer to the ERV-Fly algorithm, instantiated with appropriate preference relations for directing the unfolding w.r.t. execution time, as $\mathsf{ERV\text{-}Fly}_{min}$. (Detailed descriptions of both ERV-Fly and $\mathsf{ERV\text{-}Fly}_{min}$ algorithms can be found in the aforementioned references.)

The main result here is that by suitably directing the unfolding process, one is guaranteed that the solutions obtained are optimal parallel reorderings w.r.t. execution time.

Theorem 6. *Let $\mathcal{P}$ be a planning problem and C be the configuration identified by $\mathsf{ERV\text{-}Fly}_{min}(\mathsf{pnet}^+(\mathcal{P}))$. Then, plan π_C is a minimum parallel reordering w.r.t execution time.*

Proof. On the contrary, suppose that $\pi_C = \langle A, \prec \rangle$ can indeed be reordered to a plan $\pi' = \langle A, \prec' \rangle$ that is $\mathcal{P}$-valid and faster. Then, one could always further deorder π' to obtain a plan π'' (possibly π' itself) that is a minimal deordering for $\mathcal{P}$. Clearly, plan π'' would have an execution time smaller or equal to that of π'. In addition, due to Theorem 5, π'' ought to be represented by some configuration in the unfolding of $\mathsf{pnet}^+(\mathcal{P})$. This, however, would contradict Corollary 4.3.5 in (Hickmott 2008), which states that the solution identified by the $\mathsf{ERV\text{-}Fly}_{min}$ algorithm is minimal w.r.t. execution time *over all solutions represented* by configurations in the unfolding of the PT-net translation of $\mathcal{P}$. □

As a matter of fact, one could show a much stronger result: procedure $\mathsf{ERV\text{-}Fly}_{min}(\mathsf{pnet}^+(\mathcal{P}))$ always outputs a parallel plan solution π that is strongly independent in S_0 and such that π has minimum execution time over *all* $\mathcal{P}$-valid plans. In other words, one cannot produce a faster plan even by using a different set of operators.

One may argue that the above result is not significant if plans can be optimally reordered in a tractable way. Bäckström proved that deciding whether a plan is a minimum parallel reordering is, in fact, NP-complete (Bäckström 1998, Theorem 6.7). However, that result was based on an independence notion of concurrency (referred as *simple* concurrency). By appealing to the special case of toggling actions, we show next that the problem does remain hard when *strong* independence is considered.

Theorem 7. *The problem of deciding whether a plan π is a minimum parallel reordering w.r.t. strong independence for a planning problem $\mathcal{P}$ is (still) NP-hard.*

Proof (Sketch). It follows from Theorem 3 above and Theorem 7.10 in (Bäckström 1998) which states that the decision problem remains NP-hard even when restricting to toggling operators under the independence notion of concurrency. □

Conclusion

The technical contributions of this paper are threefold. Firstly, under a suitable transformation of operators with persistent preconditions, we characterised the type of operator concurrency captured by the unfolding technique by defining the so-called notion of *strong independence*. We proved that the plans captured in the unfolding space are *exactly* those that respect strong independence (Theorems 1 and 2). Secondly, we characterised the space of solutions explored by the unfolding technique, as the set of plans that are *optimal deorderings w.r.t flexibility* (Theorems 4 and 5). This result gives insight to the form and size of the search space explored, which is currently understood informally. Finally, and most importantly, we showed that directing the unfolding appropriately guarantees that the solution obtained is an *optimal reordering w.r.t. execution time* (Theorem 6). This result is significant because it can be proved that reordering a plan to achieve time optimality is not tractable (Theorem 7, by means of Theorem 3).

The results of this paper are applicable to temporal planning, if we consider it to be classical planning extended to actions with (arbitrary) durations that may temporarily overlap. Meanwhile, plan flexibility is still relevant to (strict) classical planning in situations where increasing the number of possible linearisations is of value, for example, at scheduling time. It is worth noting also that we have restricted the analysis to what Bäckström (1998) refers to as *definite* parallel plans. In such plans, all un-ordered operators may temporally overlap; there is no notion of unordered operators which can be arbitrarily interleaved, but can not be executed concurrently. Going beyond definite plans is, as far as we know, currently not supported by current PT-net unfolding technique, and would require meta-reasoning about multiple configurations.

Further research is needed to more deeply understand the practical difference between independence and strong independence, as well as the complexity implications of the place replication technique. A starting point for the latter is the work of (Vogler, Semenov, and Yakovlev 1998), which gives some insights on the effect of using transition loops and place replications. In fact, their results (page 3) tell us that, without place replication of a common persistent precondition, there is a combinatorial explosion in the unfolding caused by the interleaving of the "apparently" non-concurrent actions which share this common persistent precondition. So, place replication can result in a substantially smaller unfolding search space. Said so, the initial PT-net encoding would be larger when places are replicated.

Unfortunately, the planning literature reveals few attempts to evaluate and compare the flexibility properties of parallel planning approaches. Two notable exceptions are Nguyen and Kambhampati (2001)'s empirical comparison of the "flexibility" of plans returned by various planners, as measured by the number of constraints between operators, and the solid formal framework developed by Bäckström (1998), on which the work presented here is based.

We believe that a formal understanding of the flexibility properties of partial-order approaches to automated planning is necessary to develop and exploit the full potential of the area, and that this work provides such understanding for one of the latest techniques in the field.

Acknowledgements

We would like to thank Patrik Haslum and Lawrence Cavedon for providing us with detailed feedback that helped improve the paper. We also thank the anonymous reviewers for their good suggestions. The current work has been supported by Agent Oriented Software and the Australian Research Council (under grants LP0882234 & LP0560702), and the National Science and Engineering Research Council of Canada (under a PDF fellowship).

References

Bäckström, C. 1998. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research* 9:99–137.

Bonet, B.; Haslum, P.; Hickmott, S.; and Thiébaux, S. 2008. Directed unfolding of Petri nets. In *Transactions on Petri Nets and Other Models of Concurrency I*, volume 5100 of *LNCS*. 172–198.

Christensen, S., and Hansen, N. D. 1993. Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs. In *Proc. of Petri Nets'93*, 186–205.

Esparza, J.; Römer, S.; and Vogler, W. 1996. An improvement of McMillan's unfolding algorithm. In *Proc. of TACAS'06*, 87–106.

Hickmott, S.; Rintanen, J.; Thiébaux, S.; and White, L. 2007. Planning via Petri net unfolding. In *Proc. of IJCAI*, 1904–1911.

Hickmott, S. 2008. *Directed Unfolding: Reachability Analysis of Concurrent Systems and Applications to Automated Planning*. Ph.D. Dissertation, University of Adelaide, Adelaide, Australia.

Hoare, C. A. 1974. Monitors: an operating system structuring concept. *Communications of the ACM* 17(10):549–557.

McMillan, K. L. 1992. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. of CAV'92*, 164–177.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proc of IJCAI'01*, 459–466.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. of IJCAI'99*, 326–333.

Smith, D. E.; Frank, J.; and Jónsson, A. K. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1).

Vogler, W.; Semenov, A. L.; and Yakovlev, A. 1998. Unfolding and finite prefix for nets with read arcs. In *Proc. of CONCUR'98*, 501–516.