Action Planning for Directed Model Checking of Petri Nets

Stefan Edelkamp [1]

Computer Science Department
University of Dortmund
Dortmund, Germany

Shahid Jabbar [2]

Computer Science Department
University of Dortmund
Dortmund, Germany

Abstract

Petri nets are fundamental to the analysis of distributed systems especially infinite-state systems. Finding a particular marking corresponding to a property violation in Petri nets can be reduced to exploring a state space induced by the set of reachable markings. Typical exploration(reachability analysis) approaches are undirected and do not take into account any knowledge about the structure of the Petri net. This paper proposes heuristic search for enhanced exploration to accelerate the search. For different needs in the system development process, we distinguish between different sorts of estimates.

Treating the firing of a transition as an *action* applied to a set of *predicates* induced by the Petri net structure and markings, the reachability analysis can be reduced to finding a plan to an AI planning problem. Having such a reduction broadens the horizons for the application of AI heuristic search planning technology. In this paper we discuss the transformations schemes to encode Petri nets into PDDL. We show a concise encoding of general place-transition nets in Level 2 PDDL2.2, and a specification for bounded place-transition nets in ADL/STRIPS. Initial experiments with an existing planner are presented.

Key words: Petri nets, PDDL, Action Planning, Directed Model Checking

[1] Email: stefan.edelkamp@cs.uni-dortmund.de
[2] Email: shahid.jabbar@cs.uni-dortmund.de

1 Introduction

Several problems in computer science can be reduced to state exploration problem: given a state space and a pair of start s and goal state t, search for a path that starts at s and ends at t. Sometimes, the state space is provided before-hand, such as in routing systems - we call such state spaces as *explicit* state spaces. On the other hand, a state space can also be provided by an initial state complemented with a set of rules to generate the rest of the states. Such state spaces are known as *implicit* state spaces.

There are several methods to represent the set of rules (a.k.a model) that help us to generate a state space. Some widely used ones are: Büchi automata, Kripke structures, Petri nets, etc. [9]. Petri nets are special in the sense that they can easily be used to represent an infinite state space. *Model Checking* [9] and *Artificial Intelligence Search* [36] are two such disciplines where one frequently encounters implicit state spaces.

Roughly speaking, *Model Checking* is a process to validate the correctness of a property in a given model. The success of model checking is mainly due to detecting subtle bugs in large designs, while delivering counterexamples as witnesses for the errors. In general, correctness properties in concurrent systems are specified in some temporal logic. Frequently, errors to be uncovered are rather simple conditions on individual states such as deadlock appearance or invariant violations.

Artificial Intelligence Search, especially Action Planning aims at finding a sequence of actions leading from an initial situation to a state that satisfies a desired goal condition. Nowadays planning domain description languages such as PDDL2.1 [23] and PDDL2.2 [16] are flexible to concisely describe very different exploration problems, and current AI planning systems are capable of exploring the state spaces in those problem domains. The list of approaches of planning via model checking includes planning with control rules [29,1,30] and symbolic model checking applied to AI planning problems [14,8,7,35,6].

In the last few years, there is a rising trend to unify the exploration approaches that have been developed in Model Checking and Artificial Intelligence Search. One of the most effective approaches, called *Directed Model Checking* [19] explores the state spaces with AI heuristic search algorithms like A* [32]. Basically, the idea is to apply algorithms that exploit the information about the problem being solved in order to guide the exploration process. The benefits are twofold: the search effort is reduced (errors are found faster), and the solution quality is improved (counterexamples are shorter).

Petri nets [34] reflect a simple but effective graphical modeling formalism for a restricted form of model checking, appropriate to test discrete distributed systems for functional correctness. Once modeled as a Petri net, one can analyze the system to find deadlocks, and to validate liveness and boundedness properties.

In this paper we contribute different heuristics for enhanced error detection in Petri nets. We concentrate on finding deadlocks in place-transition nets. Moreover, we provide a suitable encoding in PDDL. This allows to apply directed search directly by the in-built heuristics of current planning systems, bypassing extensive modifications in existing model checkers.

The paper is structured as follows. First we introduce Petri nets and their analysis. Next we address guiding the analysis through the definition of different heuristic estimates. We distinguish between general estimates that can be used for fault-finding and state-to-state estimates that are used for reduction of firing sequences. Afterwards, we turn to a possible PDDL encoding of the domain. We separate the numerical domain encoding from the propositional one as the latter is limited to encode bounded Petri nets. Based on such planning domain models we present experiments and results that we have obtained with a heuristic search planner on a benchmark set. Last, we discuss related work and draw conclusions.

2 Petri Nets and their Analysis

Petri nets were invented by [34] as a means of describing concurrency and synchronization in distributed systems. A Petri net is a 4-tuple (P, T, I^-, I^+), where $P = \{p_1, \ldots, p_n\}$ is the set of *places*, $T = \{t_1, \ldots, t_m\}$ is the set of *transitions* with $1 \leq n, m < \infty$ and $P \cap T = \emptyset$. The backward and forward incidence mappings I^- and I^+ respectively map elements of $P \times T$ and $T \times P$ to the set of natural numbers and fix the Petri net link structure and the transition labels.[3] A Petri net is an *ordinary place-transition net* if the label set is restricted to only 0 (arc omitted) and 1 (arc present). To ease the exposition, in this paper we assume all place-transition nets to be ordinary.

A *marking* in a place-transition net maps elements of P to a natural number. With $M(p)$ we denote the number of *tokens* at place p. It is natural to assume that M is provided in vector representation. Markings correspond to states in a state space. Petri nets are often supplied with an initial marking M_0, the initial state. A transition t is *enabled*, if all its input places contain at least one token, i.e., $M(p) \geq I^-(p,t)$ for all $p \in P$. If a transition is fired, it deletes one token from each of its input places and generates one on each of its outputs places. A transition t enabled at marking m may *fire* and generate a new marking $M'(p) = M(p) - I^-(p,t) + I^+(p,t)$ for all $p \in P$, written as $M \rightarrow M'$. A marking M' is *reachable* from M, if $M \xrightarrow{*} M'$, where $\xrightarrow{*}$ is the reflexive and transitive closure of $\rightarrow$. The *reachability set* $R(N)$ of a place transition net N is the set of all markings M reachable from M_0. A place-transition net N is *bounded*, if for all places p there exists a natural number k, such that for all M in $R(N)$ we have $M(p) \leq k$. A transition t is *live*, if for all M in $R(N)$ there is a M' in $R(N)$ with $M \xrightarrow{*} M'$ and t is enabled in M'. A place-transition net N is *live*, if all transitions t are *live*. A *firing sequence* $\sigma = t_1, \ldots, t_n$ starting at M_0 is a finite sequence of transitions such that t_i is enabled in M_{i-1} and M_i is the result of firing t_i in M_{i-1}.

In the analysis of complex systems, *places* model conditions or objects such as program variables, *transition* model activities that change the values of conditions

[3] The terms *Petri nets* and *place-transition nets* are used synonymously in this paper.

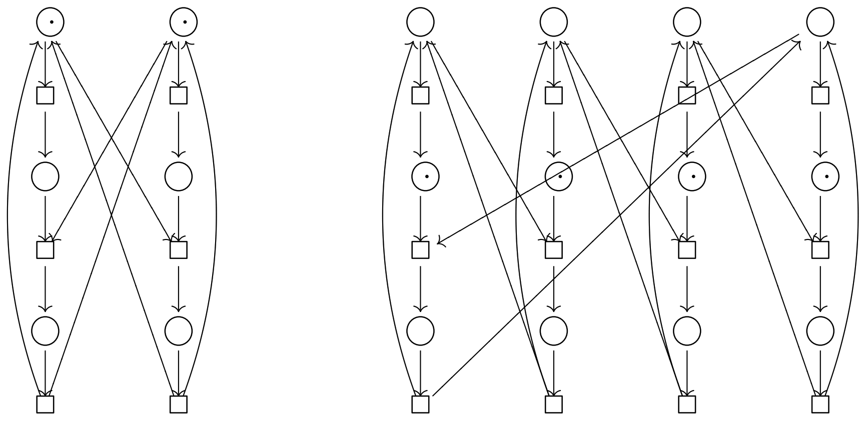
Fig. 1. Place-Transition Petri Nets for 2 and 4 Dining Philosophers.

and objects, and *markings* represent the specific values of the condition or object, such as the value of a program variable.

The graphical representation of Petri nets consists of *circles* for places, *dots* for tokens, *rectangles* for transitions, and *arrows* for arcs between places and transitions. An example of an ordinary place-transition petri net for the Dining Philosophers example with 2 and 4 philosophers is provided in Figure 1. Different philosophers correspond to different columns, while the places in the rows denote their states: *thinking*, *waiting*, and *eating*. The markings for the 2-philosophers case correspond to the initial state of the system, while for the 4-philosophers case, we show the markings that resulted in a deadlock.

There are two different analysis techniques for Petri nets, the *analysis of the reachability set* and the *invariant analysis*. The latter approach concentrates more on the Petri net structure itself. Unfortunately, invariant analysis is applicable only if studying $|P| \times |T|$ is tractable.

In this paper, we concentrate on the analysis of the reachability set. Recall that the number of tokens for a node in a place transition net is not bounded a priori, so that the number of possible states is infinite. Nonetheless, there is a state space exploration algorithm that terminates even in case of an unbounded place-transition net. The main idea is to include partial markings, that introduce don't care symbol ω into the state vector, denoting a marking of an unbounded place larger than every natural number. The algorithm starts with a reachability set R consisting of marking M_0 and generates a *coverability tree*. Nondeterministically, a partial marking M in R at a leaf is chosen and for all enabled transitions t a new partial marking M' is generated by firing t and included into the tree. If there is a marking M'' on the path from M_0 to M' with $M'' \leq M'$ and $M(p) < M'(p)$, we set $M'(p)$ to ω. Obviously, the place-transition net is bounded if no node contains ω.

Before we proceed to discussing the use of heuristics in Petri nets, we need to clarify what one understands by a *goal condition* in the context of Petri nets. Goal conditions are basically markings that signify some property of the system that a Petri net models. We distinguish between two kinds of goal conditions: *specific* -

that denote an explicit marking within the network, or *general* - that denote a set of different markings e.g., a deadlock in the system.

3 Distance Heuristics for Petri Nets

Heuristics are evaluation functions that estimate the number of transitions necessary to achieve a goal condition. Evaluation functions in the context of Petri nets associate a numerical value to each marking in order to prioritize the exploration of some successors with respect to some others.

The *shortest path distance* $\delta_N(M, M')$ in a net N is defined as the length of the shortest firing sequence between M and M'. The distance is infinite if there exists no firing sequence between M and M'. Moreover, $\delta_N(M, \psi)$ is the shortest path to a marking that satisfies condition ψ starting at M, i.e., $\delta_N(M, \psi) = \min\{\delta_N(M, M') \mid M' \models \psi\}$. Subsequently, heuristic $h(M)$ estimates $\delta_N(M, \psi)$. It is *admissible*, if $h(M) \leq \delta_N(M, \psi)$ and *monotone* if $h(M) - h(M') \leq 1$ for a successor marking M' of M. Monotone heuristics with $h(M') = 0$ for all $M' \models \psi$ are admissible.

We distinguish two search scenarios. In the explanatory mode, we explore the set of reachable markings having just the knowledge on what kind of error ϕ we aim at. In this phase we are just interested in finding such error fast, without aiming at concise counterexample firing sequences. For the fault-finding mode we assume that we know the marking, where the error occurs. This knowledge is to be inferred by simulation, test or a previous run in the explanatory mode. To reduce the firing sequence a heuristic estimate between two markings is needed. The presentation of the heuristics is kept short, as the heuristics and their properties share similarities with the ones that have been designed for communication protocols in Promela [19,20].

3.1 Hamming Distance

A very simple heuristic estimates is the *Hamming distance heuristic*

$$h_H(M, M') = \sum_{p \in P} [M(p) \neq M'(p)].$$

Here, the truth of $[M(p) \neq M'(p)]$ is interpreted as an integer in $\{0, 1\}$. As a transition may add/delete more than one token at a time the heuristic is neither admissible nor consistent. However if we divide $h_H(M, M')$ by the maximum number of infected places of a transition, we arrive at an admissible value. In the example of 4 Dining Philosophers we obtain an initial estimate of 4 that matches the shortest firing distance to a deadlock.

3.2 Subnet Distance

A more elaborate heuristic that approximates the distance between M and M' works as follows. Via abstraction function ϕ it projects the place transition network N to $\phi(N)$ by omitting some places, transitions and corresponding arcs. In addition, the initial set of marking M and M' is reduced to $\phi(M)$ and $\phi(M')$. As an example the 2-Dining Philosopher place-transition net in Figure 1 is in fact an abstraction of the 4-Dining Philosophers place transition net to its right.

The *subnet distance heuristic* can now be defined as the shortest path distance required to reach $\phi(M')$ from $\phi(M)$, formally

$$h_\phi(M, M') = \delta_{\phi(N)}(\phi(M), \phi(M')).$$

In the example of 4 dining philosophers we obtain an initial estimate of 2.

It is not difficult to see that the heuristic estimate thus obtained is admissible, i.e., $\delta_N(M, M') \geq \delta_{\phi(N)}(\phi(M), \phi(M'))$. Let M be the current marking and M'' be its immediate successor. In order to prove that the heuristic h_ϕ is consistent we need to show that $h_\phi(M) - h_\phi(M'') \leq 1$. Using the def. of h_ϕ, we can say that

$$\delta_{\phi(N)}(\phi(M), \phi(M')) \leq 1 + \delta_{\phi(N)}(\phi(M''), \phi(M'))$$

The above inequality is always true since the shortest path cost from $\phi(M)$ to $\phi(M')$ cannot be greater than the shortest path cost that traverses $\phi(M'')$ (Triangular property).

To avoid recomputations, it is appropriate to pre-compute the distance prior to the search and to use table lookups to guide the exploration. The subnet distance heuristic completely explore the coverability graph of $\phi(N)$ and run a shortest-path algorithm on top of it. This idea is referred to as *pattern database* construction and goes back to [11].

If we apply two different abstractions ϕ_1 and ϕ_2, to preserve admissibility, we can only take their maximum, i.e.,

$$h^{\max}_{\phi_1,\phi_2}(M, M') = \max\{h_{\phi_1}(M, M'), h_{\phi_2}(M, M')\}.$$

However, if the support of ϕ_1 and ϕ_2 are disjoint, i.e., the corresponding set of places and the set of transitions are disjoint $\phi_1(P) \cap \phi_2(P) = \emptyset$ and $\phi_1(T) \cap \phi_2(T) = \emptyset$, the sum of the two individual heuristics

$$h^{\text{add}}_{\phi_1,\phi_2}(M, M') = h_{\phi_1}(M, M') + h_{\phi_2}(M, M')$$

is still admissible. If we use an abstraction for the first two and the second two philosophers we obtain the perfect estimate of 4 firing transitions.

3.3 Activeness Heuristic

While the above two heuristics measure the distance from one marking to another, it is not difficult to extend them for a goal by taking the minimum of the distance of the current state to all possible markings that satisfy the desired goal. However, as we concentrate on deadlocks, specialized heuristics can be established that bypass the enumeration of the goal set.

A deadlock in a Petri net occurs if no transition can fire. Therefore, a simple distance estimate to the deadlock is simply to count the number of active transitions.

```
(:action fire-transition
 :parameters (?t - transition)
 :preconditions
   (forall (?p - place)
     (or (not (incoming ?p ?t))
         (> (number-of-tokens ?p) 0)))
 :effects
   (forall (?p - place)
     (when (incoming ?p ?t)
           (decrease (number-of-tokens ?p))))
   (forall (?p - place)
     (when (outgoing ?t ?p)
           (increase (number-of-tokens ?p)))))
```

Fig. 2. Numerical planning operator of a Petri net transition.

In other words, we have

$$h_A(M) = \sum_{t \in T} \textit{enabled}(t).$$

As with the Hamming distance the heuristic is not consistent nor admissible, since one firing transition can change the enableness of more than one transition. For our running example we find 4 active transitions in the initial states.

4 A PDDL Model for Petri Nets

PDDL provides a modeling formalism for planning domains and their corresponding problems. A PDDL-based planner takes a domain file and a problem file written in PDDL to compute a plan. In the domain file, object types, predicates and actions are provided, while the problem file contains objects themselves the initial state and the goal specification.

In the following we derive a Level 2, PDDL2.2 model [16] for Petri nets, which is then simplified to be compatible to a propositional planning formalism [22].

4.1 Numerical Encoding

In the PDDL2.2 encoding of a place transition net we declare two object types `place` and `transition`. To describe the topology of the net we work with the predicates `(incoming ?s - place ?t - transition)` and `(outgoing ?s - place ?t - transition)`, representing the two sets I^- and I^+. For the sake of simplicity all transitions have weight 1. The only numerical information that is needed is the number of tokens at a place. This marking mapping is realized via the fluent predicate `(number-of-tokens ?p - place)`. The transition firing operator is shown in Figure 2.

The initial state encodes the net topology and the initial markings. It specifies instances to the predicates `incoming` and `outgoing` and a numerical predicate `(number-of-tokens)` to specify M_0.

The condition that a transition is blocked, can be modeled in PDDL2.2 with a derived predicate as follows

```
(:derived block (?t - transition)
   (exists (?p - place)
      (and (incoming ?p ?t) (= (number-of-tokens ?p) 0))))
```

Consequently, a deadlock to be specified as the goal condition is derived as follows

```
(:derived deadlock (forall (?t - transition) (blocked ?t)))
```

It is obvious, that the PDDL encoding inherits a one-to-one correspondence to the original place-transition net.

4.2 Propositional Encoding

ADL [33] descriptions are flexible planning formalisms that allow for more involved propositional operator declarations including negated and disjunctive preconditions, conditional effects, and universal/existential quantification of objects. They are included in Level 1, PDDL2.1/2.2.

To simplify the above encoding to a propositional one, we may use an unary encoding of the tokens at a place, with objects `zero`, `one`, `two`, etc. of type `number`, and predicates `(is-not-zero ?n - number)`, `(inc ?n1 ?n2 - number)`, with obvious semantics. The firing operator is shown in Figure 3.

4.3 Implicit Planning Heuristics

The foremost advantage of modeling Petri nets as a planning problem is to be able to utilize the huge portfolio of heuristics proposed for planning problems.

Most recent forward chaining planners apply variants of the *relaxed planning heuristic* [27]. The *relaxation* a^+ of (STRIPS) action $a = (\textit{pre}(a), \textit{add}(a), \textit{del}(a))$ is defined as $a^+ = (\textit{pre}(a), \textit{add}(a), \emptyset)$. The relaxation of a planning problem is done by substituting all actions by their relaxed counterparts. Any solution that solves the original plan also solves the relaxed one; and all preconditions and goals can be achieved if and only if they can be in the relaxed task. Value h^+ is defined as the length of the shortest plan that solves the relaxed problem. Solving relaxed plans optimally is still computationally hard [5], but the decision problem to determine, if a relaxed planning problem has at least one solution, is computationally tractable. The estimate has been extended to planning problems with numerical state variables [25] and further to non-linear tasks [15].

All the above heuristics are not admissible. Alternative design of a consistent heuristic are the HSP-heuristic [4], and the *planning pattern database heuristic* [12] that introduces *don't cares* to the state vector description.

```
(:action fire-transition
 :parameters (?t - transition)
 :precondition
  (forall (?p - place)
    (or (not (incoming ?p ?t))
        (exists (?n - number)
           (and (number-of-tokens ?p ?n)
                (is-not-zero ?n)))))
 :effect
  (and
    (forall (?p - place ?n1 ?n2 - number)
      (when
        (and (incoming ?p ?t) (inc ?n1 ?n2)
             (number-of-tokens ?p ?n2))
        (and (not (number-of-tokens ?p ?n2))
             (number-of-tokens ?p ?n1))))
    (forall (?p - place ?n1 ?n2 - number)
      (when
        (and (outgoing ?t ?p) (inc ?n1 ?n2)
             (number-of-tokens ?p ?n1))
        (and (not (number-of-tokens ?p ?n1))
             (number-of-tokens ?p ?n2)))))))
```

Fig. 3. Propositional planning operator of a Petri net transition.

5 Experiments

We performed our experiments on a Pentium 4, 3.2 GHz with 2 GB of main memory running Linux operating system. As the planner we use the propositional heuristic search forward planning system FF. It applies the relaxed planning heuristic to guide the search. This system does not support derived predicates, we encoded the blocking and the deadlock condition with ordinary operators having the derived predicate as the only effect.

We first tested the planner on the 2 and 4 philosophers cases. The planner found the deadlock immediately using the minimal number of firing transition.

For an extensive testing, we use a set of deadlock checking benchmarks collected by Corbett [10]. These are *1-safe* place-transition nets converted from communication state machines [31]. Recall that a net is called 1-safe, if $M(p) \leq 1$ for all p. The same benchmark has been addressed with answer set programming based on bounded model checking (BMC) in [24]. The encoding of the firing action for *1-safe* nets is presented in the Appendix.

Table 1 displays the results of our experiments. Besides the size of the network we show the length of the firing sequence to detect the deadlock, the CPU time in seconds and the number of explored states. In the last column we show the run times produced in [24] using the BMC approach. This comparison has to be dealt with care, as the computer they used was a 450 Mhz Pentium III running Linux.

Problem	# Places	# Trans.	Sol. Len.	Time$_{FF}$	# Expl.	Time$_{BMC}$
DARTES(1)	331	257	2	0.28	6	.5
DP(6)	36	24	6	0.08	11	0.1
DP(8)	48	32	8	0.08	15	0.3
DP(10)	60	40	10	0.08	19	3.3
DP(12)	72	48	12	0.08	23	617.4
ELEV(1)	63	99	11	0.03	45	0.4
ELEV(2)	146	299	16	0.2	74	3.9
ELEV(3)	327	783	18	2.08	106	139.0
HART(25)	127	77	26	0.11	27	1.0
HART(50)	252	152	51	0.28	52	5.7
HART(75)	377	227	76	0.71	77	15.5
HART(100)	502	302	101	1.45	102	45.9
MMGT(3)	122	172	10	0.13	15	87.2
MMGT(4)	736	1,939	12	0.2	24	1,874.1
Q(1)	163	194	21	0.25	258	2,733.7
SENT(25)	104	55	3	0.09	5	0.0
SENT(50)	179	80	3	0.1	5	0.0
SENT(75)	254	105	3	0.14	5	0.0
SENT(100)	329	130	3	0.18	5	0.0
SPD(1)	33	39	4	0.08	5	0.0

Table 1
Planner results on Petri net benchmark problems

Compared to [24] our results show a significant gain in the CPU time to establish a trail. We also found that the counterexamples that were produced by the planner are comparable to the ones produced by the model checker.

6 Related Work

To arrive at a smaller net structures, different transformations have been proposed in literature. A transformation is called *consistent*, if it preserves liveness and boundedness. There is no set of rules that is complete in the sense that by applying the rules a finite small set of prototypical place-transitions nets is generated for which liveness and boundedness is known. Reduction rules usually delete nodes in the nets together with all their incoming and outgoing arcs and subsequent isolated nodes. Reduction rules modify the Petri net structure and a marking that is present in the net. For example, simple rules are as follows: delete all transitions without incoming place together with all outgoing places, delete all places without incoming transitions together with all outgoing transitions, and if there are two distinct parallel places (same connection structure) delete the one with more tokens. It is not difficult to see that all these rules are consistent [2]. In Petri net practice, several other local rules have been suggested [3]. In contrast to this work with abstraction heuristics we are concerned on the length of firing sequences rather than preserving liveness and boundedness.

The proposed PDDL modeling approach integrates well with other efforts for converting model checking problem specification into inputs for planners. One of the first approaches is to translate communication protocol specification from Promela, the input language of the model checker SPIN [28], to PDDL [13]. With two scalable protocol designs, namely the deadlock solution to the Dining Philosophers problem and the Optical Telegraph protocol, the domain was entered as a benchmark for the international planning competition IPC-4 [26].

The next approach was to convert probably the most flexible model checking scenario into PDDL: graph transition systems [17]. In graph transition systems (GTS), states are itself graphs and state transitions correspond to (partial) graph morphisms. Goals are either exact matches, subgraphs or graph isomorphisms of subgraphs. Exploiting the parametric description of propositions and actions and the quantification option of current PDDL, the syntax for GTSs and the different goals can be kept in a compact form.The scenario restricts to solve the optimization problems with respect to some cost algebra [18].

In both cases of PDDL encodings, one limit that was encountered was the static state descriptor that is inherent to the current expressiveness of PDDL. Nonetheless, restricting to a finite model is sufficient to many cases that appear in model checking practice.

Petri nets have been used for STRIPS action planning in the planner *TokenPlan* by [21]. It is the inverse approach that applies existing technology for Petri nets to solve propositional planning problems in PDDL. The transcription is automated, each predicate is represented by a place and each action is realized as a transition.

The *add effects* of operators match Petri nets semantics by having a marking set at the corresponding place for the proposition. For the propositions that are deleted no modification is needed, and for the propositions, that are preserved, backward arcs from the transition to the precondition list are inserted. The planning process in *TokenPlan* simulates the working of *Graphplan* and constructs a layered search space with layers for propositions and operators. First empirical results were promising, but in the third international planning competition, the approach was not effective enough to compare positively with state-of-the-art in heuristic search planning.

7 Conclusion

We have shown a flexible approach of analyzing Petri nets with directed search methods and AI planning technology. The heuristics are rather simple, but as shown with other approaches in directed model checking even simple estimates can lead to a drastic reduction in the size of a state space. In contrast to the approaches that incorporate improved planning via model checking we consider model checking via planning. To the authors' knowledge that is the first approach of applying planning technology to the analysis of Petri nets.

In the presentation of the heuristics we have restricted our attention to deadlock properties. However, including assertions on the number of tokens m in a particular place p is not complex. For the numerical encoding we simply add a condition `(<= (number-of-tokens p) m)` to the goal description. In the propositional encoding, we simply substitute this condition with `(number-of-tokens p mark)`, where `mark` is the object of type `number` that represents m.

Although restricted to simple place-transition nets, it is not difficult to extend our setting to colored Petri nets, in which each token at a place has one certain color. Transitions now fire with respect to the color. Every colored Petri net can be uniquely unfolded to a place-transition net, while there are different options to encode a place-transition net with colors.

References

[1] Bacchus, F. and F. Kabanza, *Using temporal logics to express search control knowledge for planning,* Artificial Intelligence **116** (2000), pp. 123–191.

[2] Berthelot, G., *Checking properties of nets using transformations*, in: *Advances in Petri Nets 1985*, LNCS **222** (1985), pp. 19–40.

[3] Berthelot, G., *Transformations and decompositions of nets*, in: *Advances in Petri nets 1986*, LNCS **254** (1987), pp. 359–376.

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

[5] Bylander, T., *The computational complexity of propositional STRIPS planning*, Artificial Intelligence (1994), pp. 165–204.

[6] Cimatti, A., E. Giunchiglia, F. Giunchiglia and P. Traverso, *Planning via model checking: A decision procedure for AR*, in: *European Conference on Planning (ECP)*, 1997, pp. 130–142.

[7] Cimatti, A. and M. Roveri, *Conformant planning via model checking*, in: *European Conference on Planning (ECP),* 1999, pp. 21–33.

[8] Cimatti, A., M. Roveri and P. Traverso, *Automatic OBDD-based generation of universal plans in non-deterministic domains*, in: *National Conference on Artificial Intelligence (AAAI)*, 1998, pp. 875–881.

[9] Clarke, E., O. Grumberg and D. Peled, “Model Checking,” The MIT Press, 1999.

[10] Corbett, J., *Evaluating deadlock detection methods for concurrent software*, IEEE Transactions on Software Engineering **22** (1996), pp. 161–180.

[11] Culberson, J. C. and J. Schaeffer, *Pattern databases*, Computational Intelligence **14** (1998), pp. 318–334.

[12] Edelkamp, S., *Planning with pattern databases*, in: *European Conference on Planning (ECP)*, 2001, pp. 13–24.

[13] Edelkamp, S., *Promela planning*, in: *Workshop on Model Checking Software (SPIN)*, 2003.

[14] Edelkamp, S., *Taming numbers and durations in the model checking integrated planning system*, Journal of Artificial Research (JAIR) **20** (2003), pp. 195–238.

[15] Edelkamp, S., *Generalizing the relaxed planning heuristic to non-linear tasks*, in: *German Conference on Artificial Intelligence (KI)*, 2004, 198–212.

[16] Edelkamp, S. and J. Hoffmann, *The classical part of ipc-4: An Overview*, Journal of Artificial Research (2005), special Track on the 4th International Planning Competition.

[17] Edelkamp, S., S. Jabbar and A. Lluch-Lafuente, *Action planning for graph transition systems*, in: *International Conference on Automated Planning and Scheduling (ICAPS). Workshop on Verification and Validation of Model-Based Planning and Scheduling Systems*, 2005.

[18] Edelkamp, S., S. Jabbar and A. Lluch-Lafuente, *Cost-algebraic heuristic search*, in: *National Conference on Artificial Intelligence (AAAI)* (2005), pp. 1362–1367.

[19] Edelkamp, S., S. Leue and A. L. Lafuente, *Directed explicit-state model checking in the validation of communication protocols*, International Journal on Software Tools for Technology Transfer (STTT) **5** (2003), pp. 247–267.

[20] Edelkamp, S. and A. Lluch-Lafuente, *Abstraction in directed model checking*, in: *ICAPS-Workshop on Connecting Planning Theory with Practice*, 2004.

[21] Fabiani, P. and Y. Meiller, *Planning with tokens: an approach between satisfaction and optimisation*, in: *European Conference on Artificial Intelligence. Workshop ”New Results in Planning, Scheduling and Design”*, 2000.

[22] Fikes, R. and N. Nilsson, *Strips: A new approach to the application of theorem proving to problem solving*, Artificial Intelligence **2** (1971), pp. 189–208.

[23] Fox, M. and D. Long, *PDDL2.1: An extension to PDDL for expressing temporal planning domains*, Journal of Artificial Research (2003), special issue on the 3rd International Planning Competition.

[24] Heljanko, K. and I. Niemelä, *Answer set programming and bounded model checking*, in: *AAAI Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, 2001, pp. 90–96.

[25] Hoffmann, J., *The Metric FF planning system: Translating "Ignoring the delete list" to numerical state variables*, Journal of Artificial Intelligence Research **20** (2003), pp. 291–341.

[26] Hoffmann, J., R. Englert, F. Liporace, S. Thiebaux and S. Trüg, *Engineering benchmarks for planning: The domains used in the classical part of IPC-4* (2005), draft.

[27] Hoffmann, J. and B. Nebel, *Fast plan generation through heuristic search*, Journal of Artificial Intelligence Research (JAIR) **14** (2001), pp. 253–302.

[28] Holzmann, G., "The Spin Model Checker: Primer and Reference Manual," Addison-Wesley, 2003.

[29] Kabanza, F., M. Barbeau and R. St-Denis, *Planning control rules for reachtive agents*, Artificial Intelligence **95** (1997), pp. 67–113.

[30] Kvarnström, J., P. Doherty and P. Haslum, *Extending TALplanner with concurrency and ressources*, in: *European Conference on Artificial Intelligence (ECAI)*, 2000, pp. 501–505.

[31] Melzer, S. and S. Römer, *Deadlock checking using net unfoldings*, in: *International Conference on Computer Aided Verification (CAV)*, 1997, pp. 172–183.

[32] Pearl, J., "Heuristics," Addison-Wesley, 1985.

[33] Pednault, E. P. D., *ADL: Exploring the middleground between STRIPS and situation calculus*, in: *Knowledge Representation and Reasoning (KR)* (1989), pp. 324–332.

[34] Petri, C. A., "Kommunikation mit Automaten," Ph.D. thesis, Universität Bonn (1962).

[35] Pistore, M. and P. Traverso, *Planning as model checking for extended goals in non-deterministic domains*, in: *International Joint Conference on Artificial Intelligence (IJCAI)*, 2001, pp. 479–486.

[36] Russell, S. and P. Norvig, "Artificial Intelligence: A Modern Approach," Prentice Hall, 2003.

Appendix

```
(:action fire-transition
:parameters (?t - transition)
:precondition
  (forall (?p - place)
    (or (not (incoming ?p ?t))
         (marked ?p)))
:effect
  (and
    (forall (?p - place)
      (when (and (incoming ?p ?t) (marked ?p))
            (not (marked ?p))))

    (forall (?p - place)
      (when (and (outgoing ?t ?p) (not (marked ?p)))
            (marked ?p)))
  )
)
```

Fig. 4. PDDL Encoding of transition operator for *1-safe* Petri nets