OBDD-based Universal Planning: Specifying and Solving Planning Problems for Synchronized Agents in Non-Deterministic Domains

Rune M. Jensen and Manuela M. Veloso

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
{runej,mmv}@cs.cmu.edu

Abstract. Recently model checking representation and search techniques were shown to be efficiently applicable to planning, in particular to non-deterministic planning. Such planning approaches use Ordered Binary Decision Diagrams (OBDDs) to encode a planning domain as a non-deterministic finite automaton (NFA) and then apply fast algorithms from model checking to search for a solution. OBDDs can effectively scale and can provide universal plans for complex planning domains. We are particularly interested in addressing the complexities arising in non-deterministic, multi-agent domains. In this article, we present UMOP,[1] a new universal OBDD-based planning framework for non-deterministic, multi-agent domains, which is also applicable to deterministic single-agent domains as a special case. We introduce a new planning domain description language, *NADL*,[2] to specify non-deterministic multi-agent domains. The language contributes the explicit definition of controllable agents and uncontrollable environment agents. We describe the syntax and semantics of *NADL* and show how to build an efficient OBDD-based representation of an *NADL* description. The UMOP planning system uses *NADL* and different OBDD-based universal planning algorithms. It includes the previously developed strong and strong cyclic planning algorithms [9, 10]. In addition, we introduce our new optimistic planning algorithm, which relaxes optimality guarantees and generates plausible universal plans in some domains where no strong or strong cyclic solution exist. We present empirical results from domains ranging from deterministic and single-agent with no environment actions to non-deterministic and multi-agent with complex environment actions. UMOP is shown to be a rich and efficient planning system.

1 Introduction

Classical planning is a broad area of research which involves the automatic generation of the appropriate choices of actions to traverse a state space to achieve

[1] UMOP stands for Universal Multi-agent OBDD-based Planner.
[2] *NADL* stands for Non-deterministic Agent Domain Language.

specific goal states. A variety of different algorithms have been developed to address the state-action representation and the search for action selection.

Traditionally these algorithms have been classified according to their search space representation as either state-space planners (e.g., PRODIGY, [41]) or plan-space planners (e.g., UCPOP, [35]).

A new research trend has been to develop new encodings of planning problems in order to adopt efficient algorithms from other research areas, leading to significant developments in planning algorithms, as surveyed in [43]. This class of planning algorithms includes GRAPHPLAN [3], which uses a flow-graph encoding to constrain the search and SATPLAN [29], which encodes the planning problem as a satisfiability problem and uses fast model satisfaction algorithms to find a solution.

Recently, another new planner MBP [8] was introduced that encodes a planning domain as a non-deterministic finite automaton (NFA) represented by an Ordered Binary Decision Diagram (OBDD) [5]. In contrast to the previous algorithms, MBP effectively extends to non-deterministic domains producing universal plans as robust solutions. Due to the scalability of the underlying model checking representation and search techniques, it can be shown to be a very efficient non-deterministic planner [9, 10].

One of our main research objectives is to develop planning systems suitable for planning in uncertain, single, or multi-agent environments [25, 42, 39]. The universal planning approach, as originally developed [38], is appealing for this type of environments. A universal plan is a set of state-action rules that aim at covering the possible multiple situations in the non-deterministic environment. A universal plan is executed by interleaving the selection of an action in the plan and observing the resulting effects in the world. Universal planning resembles the outcome of reinforcement learning [40], in that the state-action model captures the uncertainty of the world. Universal planning is a precursor approach,[3] where all planning is done prior to execution, building upon the assumption that a non-deterministic model can be acquired, and leading therefore to a sound and complete planning approach.

However, universal planning has been criticized (e.g., [22]), due to a potential exponential growth of the universal plan size with the number of propositions defining a domain state. An important contribution of MBP is thus the use of OBDDs to represent universal plans. In the worst case, this representation may also grow exponential with the number of domain propositions, but because OBDDs are very compact representations of boolean functions, this is often not the case for domains with a regular structure [9]. Therefore, OBDD-based planning seems to be a promising approach to universal planning.

MBP specifies a planning domain in the action description language $\mathcal{AR}$ [23] and translates it to a corresponding NFA, hence limited to planning problems with finite state spaces. The transition relation of the automaton is encoded as an OBDD, which allows for the use of model checking parallel breadth-first

[3] The term *precursor* originates from [13] in contrast to *recurrent* approaches which replan to recover from execution failures.

search. MBP includes two algorithms for universal planning. The *strong planning* algorithm tries to generate a plan that is guaranteed to achieve the goal for all of the possible outcomes of the non-deterministic actions. If no such strong solution exists, the algorithm fails. The *strong cyclic planning* algorithm returns a strong solution, if one exists, or otherwise tries to generate a plan that may contain loops but is guaranteed to achieve the goal, given that all cyclic executions eventually terminate. If no such strong cyclic solution exists, the strong cyclic planning algorithm fails.

In this article we present our OBDD-based planning system, UMOP, which uses a new OBDD-based encoding, generates universal plans in multi-agent non-deterministic domains, and includes a new "optimistic" planning algorithm (UMOP stands for Universal Multi-agent OBDD-based Planner).

Our overall approach for designing an OBDD-based planner is similar to the approach developed in [9, 10]. Our main contribution is an efficient encoding of a new front end domain description language, *NADL* (*NADL* stands for Non-deterministic Agent Domain Language.). *NADL* has more resemblance with previous planning languages than the action description language $\mathcal{AR}$ currently used by MBP. It has powerful action descriptions that can perform arithmetic operations on numerical domain variables. Domains comprised of synchronized agents can be modelled by introducing concurrent actions based on a multi-agent decomposition of the domain.

In addition, *NADL* introduces a separate and explicit environment model defined as a set of *uncontrollable* agents, i.e., agents whose actions cannot be a part of the generated plan. *NADL* has been carefully designed to allow for efficient OBDD-encoding. Thus, in contrast to MBP, UMOP can generate a partitioned transition relation representation of the NFA, which is known from model checking to scale up well [6, 37]. Our empirical experiments suggest that this is also the case for UMOP.

UMOP includes the previously developed algorithms for OBDD-based universal planning. In addition, we introduce a new "optimistic" planning algorithm, which relaxes optimality guarantees and generates plausible universal plans in some domains where no strong or strong cyclic solution exists.

The article is organized as follows. Section 2 gives a brief overview of OBDDs and may be skipped by readers already familiar with the subject. Section 3 introduces *NADL*, shows how to encode a planning problem, and formally describes the syntax and semantics of this description language in terms of an NFA. We also discuss the properties of the language based on an example and argue for our design choices. Section 4 presents the OBDD representation of *NADL* domain descriptions. Section 5 describes the different algorithms that have been used for OBDD-based planning and introduces our optimistic planning algorithm. Section 6 presents empirical results in several planning domains, ranging from single-agent and deterministic ones to multi-agent and non-deterministic ones. We experiment with previously used domains and introduce two new ones, namely a power plant and a soccer domain, as non-deterministic multi-agent planning problems. Section 7 discusses previous approaches to planning in non-

deterministic domains. Finally, Section 8 draws conclusions and discusses directions for future work.

2 Introduction to OBDDs

An Ordered Binary Decision Diagram [5] is a canonical representation of a boolean function with n linear ordered arguments $x_1, x_2, ..., x_n$.

An OBDD is a rooted, directed acyclic graph with one or two terminal nodes of out-degree zero labeled 1 or 0, and a set of variable nodes u of out-degree two. The two outgoing edges are given by the functions $high(u)$ and $low(u)$ (drawn as solid and dotted arrows). Each variable node is associated with a propositional variable in the boolean function the OBDD represents. The graph is ordered in the sense that all paths in the graph respect the ordering of the variables.

An OBDD representing the function $f(x_1, x_2) = x_1 \wedge x_2$ is shown in Figure 1. Given an assignment of the arguments x_1 and x_2, the value of f is determined by a path starting at the root node and iteratively following the high edge, if the associated variable is true, and the low edge, if the associated variable is false. The value of f is *True* if the label of the reached terminal node is 1; otherwise it is *False*.

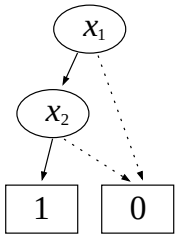

Fig. 1. An OBDD representing the function $f(x_1, x_2) = x_1 \wedge x_2$. High (true) and low (false) edges are drawn solid and dotted, respectively.

An OBBD graph is reduced so that no two distinct nodes u and v have the same variable name and low and high successors (Figure 2(a)), and no variable node u has identical low and high successors (Figure 2(b)).

The OBDD representation has two major advantages: First, it is an efficient representation of boolean functions because the number of nodes often is much smaller than the number of truth assignments of the variables. The number of nodes can grow exponential with the number of variables, but most commonly encountered functions have a reasonable representation [5]. Second, any operation on two OBDDs, corresponding to a boolean operation on the functions they represent, has a low complexity bounded by the product of their node counts. A disadvantage of OBDDs is that the size of an OBDD representing some function is very dependent on the ordering of the variables. To find an optimal variable

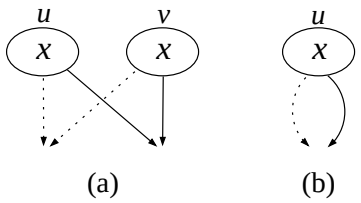

Fig. 2. Reductions of OBDDs. (a): nodes associated to the same variable with equal low and high successors will be converted to a single node. (b): nodes causing redundant tests on a variable, are eliminated.

ordering is a co-NP-complete problem in itself, but fortunately a good heuristic is to locate dependent variables near each other in the ordering [7].

OBDDs have been successfully applied to model checking. In model checking the behavior of a system is modelled by a finite state automaton with a transition relation represented as an OBDD. Desirable properties of the system is checked by analyzing the state space of the system by means of OBDD manipulations.

As introduced in [9, 10], a similar approach can be used for a non-deterministic planning problem. Given an NFA representation of the planning domain with the transition relation represented as an OBDD, the algorithms used to verify CTL properties in model checking [11, 34] can be used to find a universal plan solving the planning problem.

3 *NADL*

In this section, we first discuss the properties of *NADL* based on an informal definition of the language and a domain encoding example. We then describe the formal syntax and semantics of *NADL*.

An *NADL* domain description consists of: a definition of *state variables*, a description of *system* and *environment agents*, and a specification of an *initial* and *goal conditions*.

The set of state variable assignments defines the state space of the domain. An agent's description is a set of *actions*. The agents change the state of the world by performing actions, which are assumed to be executed synchronously and to have a fixed and equal duration. At each step, all of the agents perform exactly one action, and the resulting action tuple is a *joint action*. The system agents model the behavior of the agents controllable by the planner, while the environment agents model the uncontrollable world. A valid domain description requires that the system and environment agents constrain a disjoint set of variables.

An action has three parts: a set of *state variables*, a *precondition* formula, and an *effect* formula. Intuitively the action takes responsibility of constraining the values of the set of state variables in the next state. It further has exclusive access to these variables during execution. In order for the action to be applicable, the precondition formula must be satisfied in the current state. The effect of

the action is defined by the effect formula which must be satisfied in the next state. To allow conditional effects, the effect expression can refer to both current and next state variables, which need to be a part of the set of variables of the action. All next state variables not constrained by any action in a joint action maintain their value. Furthermore only joint actions containing a set of actions with consistent effects and a disjoint set of state variable sets are allowed. System and environment agents must be independent in the sense that the two sets of variables, their actions constrain, are disjoint.

The initial and goal conditions are formulas that must be satisfied in the initial state and the final state, respectively.

There are two causes for non-determinism in *NADL* domains: (1) actions not restricting all their constrained variables to a specific value in the next state, and (2) the non-deterministic selection of environment actions.

A simple example of an *NADL* domain description is shown in Figure 3.[4] The domain describes a planning problem for Schoppers' (1987) robot-baby domain. The domain has two state variables: a numerical one, *pos*, with range $\{0, 1, 2, 3\}$ and a propositional one, *robot_works*. The robot is the only system agent and it has two actions *Lift-Block* and *Lower-Block*. The baby is the only environment agent and it has one action *Hit-Robot*. Because each agent must perform exactly one action at each step, there are two joint actions (*Lift-Block*,*Hit-Robot*) and (*Lower-Block*,*Hit-Robot*).

Initially the robot is assumed to hold a block at position 0, and its task is to lift it up to position 3. The *Lift-Block* (and *Lower-Block*) action has a conditional effect described by an if-then-else operator: if *robot_works* is true, *Lift-Block* increases the block position with one, otherwise the block position is unchanged.

Initially *robot_works* is assumed to be true, but it can be made false by the baby. The baby's action *Hit-Robot* is non-deterministic, as it only constrains *robot_works* by the effect expression $\neg robot\_works \Rightarrow \neg robot\_works'$. Thus, when *robot_works* is true in the current state, the effect expression of *Hit-Robot* does not apply, and *robot_works* can either be true or false in the next state. On the other hand, if *robot_works* is false in the current state, *Hit-Robot* keeps it false in the next state. The *Hit-Robot* models an uncontrollable environment, in this case a baby, by its effects on *robot_works*. In the example above, *robot_works* stays false when it, at some point, has become false, reflecting that the robot cannot spontaneously be fixed by a hit of the baby.

An NFA representing the domain is shown in Figure 4. The calculation of the next state value of *pos* in the *Lift-Block* action shows that numerical variables can be updated by an arithmetic expression on the current state variables. The update expression of *pos* and the use of the if-then-else operator further demonstrate the advantage of using explicit references to current state and next state variables in effect expressions. *NADL* does not restrict the representation

[4] Unquoted and quoted variables refer to the current and next state, respectively. Another notation like v_t and v_{t+1} could have been used. We have chosen the quote notation because it is the common notation in model checking.

variables
  nat(4) *pos*
  bool *robot_works*
system
  agt: Robot
    Lift-Block
      con: *pos*
      pre: $pos < 3$
      eff: $robot\_works \rightarrow pos' = pos + 1, pos' = pos$
    Lower-Block
      con: *pos*
      pre: $pos > 0$
      eff: $robot\_works \rightarrow pos' = pos - 1, pos' = pos$
environment
  agt: Baby
    Hit-Robot
      con: *robot_works*
      pre: *true*
      eff: $\neg robot\_works \Rightarrow \neg robot\_works'$
initially
  $pos = 0 \wedge robot\_works$
goal
  $pos = 3$

Fig. 3. An *NADL* domain description.

by enforcing a structure separating current state and next state expressions. The if-then-else operator has been added to support complex, conditional effects that often are efficiently and naturally represented as a set of nested if-then-else operators.

The explicit representation of constrained state variables enables any non-deterministic or deterministic effect of an action to be represented, as the constrained variables can be assigned to any value in the next state that satisfies the effect formula. It further turns out to have a clear intuitive meaning as the action takes the "responsibility" of specifying the values of the constrained variables in the next state.

Compared to the action description language $\mathcal{A}$ and $\mathcal{AR}$ [20, 23] that are the only prior languages used for OBDD-based planning [14, 9, 10, 8], *NADL* introduces an explicit environment model, a multi-agent decomposition and numerical state variables. It can further be shown that *NADL* can be used to model any domain that can be modelled with $\mathcal{AR}$ (see Appendix A).

The concurrent actions in *NADL* are assumed to be synchronously executed and to have fixed and equal duration. A general representation allowing partially overlapping actions and actions with different durations has been avoided, as it requires more complex temporal planning (see e.g., O-PLAN or PARCPLAN,

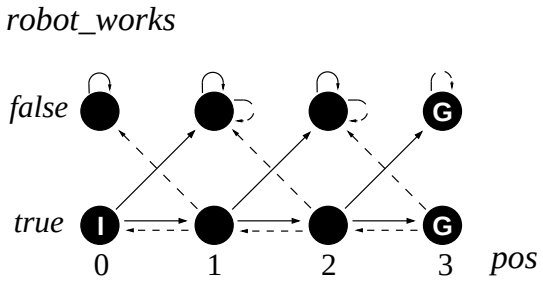

Fig. 4. The NFA of the robot-baby domain (see Figure 3). There are two state variables: a propositional state variable *robot_works* and a numerical state variable *pos* with range $\{0, 1, 2, 3\}$. The (*Lift-Block,Hit-Robot*) and (*Lower-Block,Hit-Robot*) joint actions are drawn with solid and dashed arrows respectively. States marked with "I" and "G" are initial and goal states.

[12, 31]). Our joint action representation has more resemblance with $\mathcal{A}_\mathcal{C}$ and $\mathcal{C}$ [2, 24], where sets of actions are performed at each time step. In contrast to these approaches, though, we model multi-agent domains.

An important issue to address when introducing concurrent actions is synergetic effects between simultaneously executing actions [33]. A common example of destructive synergetic effects is when two or more actions require exclusive use of a single resource or when two actions have inconsistent effects like $pos' = 3$ and $pos' = 2$. In *NADL* actions cannot be performed concurrently if: 1) they have inconsistent effects, or 2) they constrain an overlapping set of state variables. The first condition is due to the fact that state knowledge is expressed in a monotonic logic which cannot represent inconsistent knowledge. The second rule addresses the problem of sharing resources. Consider for example two agents trying to drink the same glass of water. If only the first rule defined interfering actions both agents, could simultaneously empty the glass, as the effect *glass_empty* of the two actions would be consistent. With the second rule added, these actions are interfering and cannot be performed concurrently.

The current version of *NADL* only avoids destructive synergetic effects. It does not include ways of representing constructive synergetic effects between simultaneous acting agents [33]. A constructive synergetic effect is illustrated in [2], where an agent spills soup from a bowl when trying to lift it up with one hand, but not when lifting it up with both hands. In $\mathcal{C}$ and $\mathcal{A}_\mathcal{C}$ this kind of synergetic effects can be represented by explicitly stating the effect of a compound action. A similar approach could be used in *NADL*, but is currently not supported.

3.1 Syntax

Formally, an *NADL* description is a 7-tuple $D = (SV, S, E, Act, d, I, G)$, where:

- $SV = PVar \cup NVar$ is a finite set of state variables comprised of a finite set of propositional variables, $PVar$, and a finite set of numerical variables,

$NVar$.

- S is a finite, nonempty set of system agents.
- E is a finite set of environment agents.
- Act is a set of action descriptions (c, p, e) where c is the state variables constrained by the action, p is a precondition state formula in the set $SForm$ and e is an effect formula in the set $Form$. Thus $(c, p, e) \in Act \subset 2^{SV} \times SForm \times Form$. The sets $SForm$ and $Form$ are defined below.
- $d : Agt \to 2^{Act}$ is a function mapping agents ($Agt = S \cup E$) to their actions. Because an action is associated to one agent, d must satisfy the following conditions:

$$\bigcup_{\alpha \in Agt} d(\alpha) = Act$$

$$\forall \alpha_1, \alpha_2 \in Agt \,.\, d(\alpha_1) \cap d(\alpha_2) = \emptyset$$

- $I \in SForm$ is the initial condition.
- $G \in SForm$ is the goal condition.

For a valid domain description, we require that actions of system agents are independent of actions of environment agents:

$$\bigcup_{\substack{e \in E \\ a \in d(e)}} c(a) \quad \cap \bigcup_{\substack{s \in S \\ a \in d(s)}} c(a) = \emptyset,$$

where $c(a)$ is the set of constrained variables of action a. The set of formulas $Form$ are constructed from the following alphabet of symbols:

- A finite set of current state v and next state v' variables, where $v \in SV$.
- The natural numbers $\mathbf{N}$.
- The arithmetic operators $+$, $-$, $/$, $*$ and mod.
- The relation operators $>$, $<$, $\leq$, $\geq$, $=$ and $\neq$.
- The boolean operators $\neg$,$\vee$,$\wedge$,$\Rightarrow$,$\Leftrightarrow$ and $\rightarrow$.
- The special symbols *true*, *false*, parenthesis and comma.

The set of arithmetic expressions is constructed from the following rules:

1. Every numerical state variable $v \in NVar$ is an arithmetic expression.
2. A natural number is an arithmetic expression.
3. If e_1 and e_2 are arithmetic expressions and $\oplus$ is an arithmetic operator, then $e_1 \oplus e_2$ is an arithmetic expression.

Finally, the set of formulas $Form$ is generated by the rules:

1. *true* and *false* are formulas.
2. Propositional state variables $v \in PVar$ are formulas.
3. If e_1 and e_2 are arithmetic expressions and $\mathcal{R}$ is a relation operator then $e_1 \; \mathcal{R} \; e_2$ is a formula.
4. If f_1, f_2 and f_3 are formulas, so are $(\neg f_1)$, $(f_1 \vee f_2)$, $(f_1 \wedge f_2)$, $(f_1 \Rightarrow f_2)$, $(f_1 \Leftrightarrow f_2)$ and $(f_1 \rightarrow f_2, f_3)$.

Parenthesis have their usual meaning and operators have their usual priority and associativity with the if-then-else operator "$\rightarrow$" given lowest priority. $SForm \subset Form$ is a subset of the formulas only referring to current state variables. These formulas are called *state formulas*.

3.2 Semantics

All of the symbols in the alphabet of formulas have their usual meaning with the if-then-else operator $f_1 \rightarrow f_2, f_3$ being an abbreviation for $(f_1 \wedge f_2) \vee (\neg f_1 \wedge f_3)$. Each numerical state variable $v \in NVar$ has a finite range $rng(v) = \{0, 1, \cdots, t_v\}$, where $t_v > 0$.

The formal semantics of a domain description $D = (SV, S, E, Act, d, I, G)$ is given in terms of an NFA M:

Definition1 NFA. A Non-deterministic Finite Automaton is a 3-tuple, $M = (Q, \Sigma, \delta)$, where Q is the set of states, Σ is a set of input values and $\delta : Q \times \Sigma \rightarrow 2^Q$ is a next state function.

In the following construction of M we express the next state function as a transition relation. Let $\mathcal{B}$ denote the set of boolean values $\{True, False\}$. Further, let the *characteristic function* A: $B \rightarrow \mathcal{B}$ associated to a set $A \subseteq B$ be defined by: $A(x) = (x \in A)$.[5] Given an NFA M we define its *transition relation* $T \subseteq Q \times \Sigma \times Q$ as a set of triples with characteristic function $T(s, i, s') = (s' \in \delta(s, i))$.

The states Q of M equals the set of all possible variable assignments $Q = (PVar \rightarrow \mathcal{B}) \times (Nvar \rightarrow \mathbf{N})$. Σ of M is the set of joint actions of system agents represented as sets. That is, $\{a_1, a_2, \cdots, a_{|S|}\} \in \Sigma$ if and only if $(a_1, a_2, \cdots, a_{|S|}) \in \prod_{\alpha \in S} d(\alpha)$, where $|S|$ denotes the number of elements in S.

To define the transition relation $T : Q \times \Sigma \times Q \rightarrow \mathcal{B}$ of M we constrain a transition relation $t : Q \times J \times Q \rightarrow \mathcal{B}$ with the joint actions J of all agents as input by existential quantification to the input Σ.

$$T(s, i, s') = \exists j \in J \,.\, i \subset j \wedge t(s, j, s')$$

The transition relation t is a conjunction of three relations A, F and I. Given an action $a = (c, p, e)$ and a current state s, let $P_a(s)$ denote the value of the precondition formula p of a. Similarly, given an action $a = (c, p, e)$ and a current and next state s and s', let $E_a(s, s')$ denote the value of the effect formula e of a. $A : Q \times J \times Q \rightarrow \mathcal{B}$ is then defined by:

$$A(s, j, s') = \bigwedge_{a \in j} \Big(P_a(s) \wedge E_a(s, s') \Big)$$

A defines the constraints on the current state and next state of joint actions. A further ensures that actions with inconsistent effects cannot be performed concurrently as A reduces to false if any pair of actions in a joint action have

[5] Note: the characteristic function has the same name as the set.

inconsistent effects. Thus, A also states the first rule for avoiding interference between concurrent actions.

$F : Q \times J \times Q \to \mathcal{B}$ is a frame relation ensuring that unconstrained variables maintain their value. Let $c(a)$ denote the set of constrained variables of action a. We then have:

$$F(s, j, s') = \bigwedge_{v \notin C} (v = v'),$$

where $C = \bigcup_{a \in j} c(a)$.

$I : J \to \mathcal{B}$ ensures that concurrent actions constrain a non overlapping set of variables and thus states the second rule for avoiding interference between concurrent actions:

$$I(j) = \bigwedge_{(a_1, a_2) \in j^2} \Big(c(a_1) \cap c(a_2) = \emptyset \Big),$$

where j^2 denotes the set $\{(a_1, a_2) \,|\, (a_1, a_2) \in j \times j \wedge a_1 \neq a_2\}$. The transition relation t is thus given by:

$$t(s, j, s') = A(s, j, s') \wedge F(s, j, s') \wedge I(j)$$

4 OBDD Representation of *NADL* Descriptions

To build an OBDD $\tilde{T}$ representing the transition relation $T(s, i, s')$ of the NFA of a domain description $D = (SV, S, E, Act, d, I, G)$, we must define a set of boolean variables to represent the current state s, the joint action input i and the next state s'. As in Section 3.2 we first build a transition relation with the joint actions of both system and environment agents as input and then reduces this to a transition relation with only joint actions of system agents as input.

Joint action inputs are represented in the following way: assume action a is identified by a number p and can be performed by agent α. a is then defined to be the action of agent α, if the number expressed binary by a set of boolean variables A_α, used to represent the actions of α, is equal to p. Propositional state variables are represented by a single boolean variable, while numerical state variables are represented binary by a set of boolean variables.

Let A_{e_1} to $A_{e_{|E|}}$ and A_{s_1} to $A_{s_{|S|}}$ denote sets of boolean variables used to represent the joint action of system and environment agents. Further, let $x^k_{v_j}$ and $x'^k_{v_j}$ denote the *k'th* boolean variable used to represent state variable $v_j \in SV$ in the current and next state. An ordering of the boolean variables, known to be efficient from model checking, puts the input variables first followed by an interleaving of the boolean variables of current state and next state variables:

$$\begin{aligned} A_{e_1} \prec \cdots \prec A_{e_{|E|}} \prec A_{s_1} \prec \cdots \prec A_{s_{|S|}} \\ \prec x^1_{v_1} \prec x'^1_{v_1} \prec \cdots \prec x^{m_1}_{v_1} \prec x'^{m_1}_{v_1} \\ \cdots \\ \prec x^1_{v_n} \prec x'^1_{v_n} \prec \cdots \prec x^{m_n}_{v_n} \prec x'^{m_n}_{v_n} \end{aligned}$$

where m_i is the number of boolean variables used to represent state variable v_i and n equals $|SV|$. The construction of an OBDD representation $\tilde{T}$ is quite similar to the construction of T in Section 3.2. An OBDD representing a logical expression is built in the standard way. Arithmetic expressions are represented as lists of OBDDs defining the corresponding binary number. They collapse to single OBDDs when related by arithmetic relations.

To build an OBDD $\tilde{A}$ defining the constraints of the joint actions we need to refer to the values of the boolean variables representing the actions. Let $i(\alpha)$ be the function that maps an agent α to the value of the boolean variables representing its action and let $b(a)$ be the identifier value of action a. Further let $\tilde{P}(a)$ and $\tilde{E}(a)$ denote OBDD representations of the precondition and effect formula of an action a. $\tilde{A}$ is then given by:

$$\tilde{A} = \bigwedge_{\substack{\alpha \in Agt \\ a \in d(\alpha)}} \Big(i(\alpha) = b(a) \Rightarrow \tilde{P}(a) \wedge \tilde{E}(a) \Big)$$

Note that logical operators now denote the corresponding OBDD operators. An OBDD representing the frame relation $\tilde{F}$ changes in a similar way:

$$\tilde{F} = \bigwedge_{v \in SV} \Big(\big(\bigwedge_{\substack{\alpha \in Agt \\ a \in d(\alpha)}} (i(\alpha) = b(a) \Rightarrow v \notin c(a)) \big) \Rightarrow s'_v = s_v \Big),$$

where $c(a)$ is the set of constrained variables of action a and $s_v = s'_v$ expresses that all current and next state boolean variables representing v are pairwise equal. The expression $v \notin c(a)$ evaluates to *True* or *False* and is represented by the OBDD for *True* or *False*.

The action interference constraint $\tilde{I}$ is given by:

$$\begin{aligned} \tilde{I} = & \bigwedge_{\substack{(\alpha_1, \alpha_2) \in S^2 \\ (a_1, a_2) \in c(\alpha_1, \alpha_2)}} \Big(i(\alpha_1) = b(a_1) \Rightarrow i(\alpha_2) \neq b(a_2) \Big) \wedge \\ & \bigwedge_{\substack{(\alpha_1, \alpha_2) \in E^2 \\ (a_1, a_2) \in c(\alpha_1, \alpha_2)}} \Big(i(\alpha_1) = b(a_1) \Rightarrow i(\alpha_2) \neq b(a_2) \Big), \end{aligned}$$

where $c(\alpha_1, \alpha_2) = \{(a_1, a_2) \mid (a_1, a_2) \in d(\alpha_1) \times d(\alpha_2) \wedge c(a_1) \cap c(a_2) \neq \emptyset\}$.

Finally the OBDD representing the transition relation $\tilde{T}$ is the conjunction of $\tilde{A}$, $\tilde{F}$ and $\tilde{I}$ with action variables of the environment agents existentially quantified:

$$\tilde{T} = \exists A_{e_1}, \cdots, A_{e_{|E|}} . \, \tilde{A} \wedge \tilde{F} \wedge \tilde{I}$$

Partitioning the transition relation

The algorithms we use for generating universal plans all consist of some sort of backward search from the states satisfying the goal condition to the states satisfying the initial condition (see Section 5). Empirical studies in model checking have shown that the most complex operation for this kind of algorithms normally is to find the preimage of a set of visited states V.

Definition 2 Preimage. Given an NFA $M = (Q, \Sigma, \delta)$ and a set of states $V \subseteq Q$, the *preimage* of V is the set of states $\{s \mid s \in Q \wedge \exists i \in \Sigma, s' \in \delta(s, i) \,.\, s' \in V\}$.

Note that states already belonging to V can also be a part of the preimage of V. Assume that the set of visited states are represented by an OBDD expression $\tilde{V}$ on next state variables and that we for iteration purposes, want to generate the preimage $\tilde{P}$ also expressed in next state variables. For a monolithic transition relation $\tilde{T}$ we then calculate:

$$\tilde{U} = (\exists \mathbf{x}' \,.\, \tilde{T} \wedge \tilde{V})[\mathbf{x}/\mathbf{x}']$$
$$\tilde{P} = \exists \mathbf{i}' \,.\, \tilde{U}$$

where $\mathbf{i}$, $\mathbf{x}$ and $\mathbf{x}'$ denote input, current state and next state variables, and $[\mathbf{x}'/\mathbf{x}]$ denotes the substitution of current state variables with next state variables. The set expressed by $\tilde{U}$ consists of state input pairs (s, i), for which the state s belongs to the preimage of V and the input i may cause a transition from s to a state in V. In the universal planning algorithms presented in the next section, the universal plans are constructed from elements in $\tilde{U}$.

The OBDD representing the transition relation $\tilde{T}$ and the set of visited states $\tilde{V}$ tend to be large, and a more efficient computation can be obtained by performing the existential quantification of next state variables early in the calculation [6, 37]. To do this the transition relation has to be split into a conjunction of partitions $T_1, T_2, ..., T_n$ allowing the modified calculation:

$$\tilde{U} = (\exists \mathbf{x}'_n \,.\, \tilde{T_n} \wedge \cdots (\exists \mathbf{x}'_2 \,.\, \tilde{T_2} \wedge (\exists \mathbf{x}'_1 \,.\, \tilde{T_1} \wedge \tilde{V})) \cdots)[\mathbf{x}/\mathbf{x}']$$
$$\tilde{P} = \exists \mathbf{i}' \,.\, \tilde{U}$$

That is, $\tilde{T_1}$ can refer to all variables, $\tilde{T_2}$ can refer to all variables except $\mathbf{x}'_1$, $\tilde{T_3}$ can refer to all variables except $\mathbf{x}'_1$ and $\mathbf{x}'_2$ and so on.

As shown in [37] the computation time used to calculate the preimage is a convex function of the number of partitions. The reason for this is that, for some number of partitions, a further subdivision of the partitions will not reduce the total complexity, because the complexity introduced by the larger number of OBDD operations is higher than the reduction of the complexity of each OBDD operation.

NADL has been carefully designed to allow a partitioned transition relation representation. The relations A, F and I all consist of a conjunction of subexpressions that normally only refer to a subset of next state variables. A partitioned transition relation that enables early variable quantification can be constructed by sorting the subexpressions according to which next state variables they refer to and combining them in partitions with near optimal sizes that satisfy the above requirements.

5 OBDD-based Universal Planning Algorithms

In this section we will describe two prior algorithms for OBDD-based universal planning and discuss which kind of domains they are suitable for. Based on this discussion we present a new algorithm called *optimistic planning* that seems to be suitable for some domains not covered by the prior algorithms.

The three universal planning algorithms discussed are all based on an iteration of preimage calculations. The iteration corresponds to a parallel backward breadth first search starting at the goal states and ending when all initial states are included in the set of visited states (see Figure 5). The main difference between the algorithms is the way the preimage is defined.

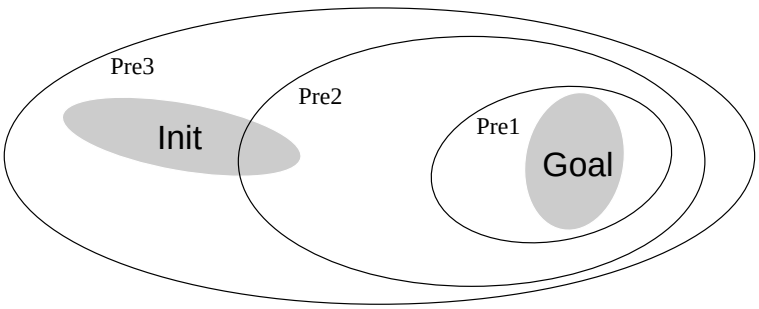

Fig. 5. The parallel backward breadth first search used by universal planning algorithms studied in this article.

5.1 Strong Planning

Strong Planning [10] uses a different preimage definition called strong preimage. For a state s belonging to the strong preimage of a set of states V, there exists at least one input i where all the transitions from s associated to i leads into V. When calculating the strong preimage of a set of visited states V, the set of state input pairs U represents the set of actions for each state in the preimage that, for any non-deterministic effect of the action, causes a transition into V. The universal plan returned by strong planning is the union of all these state-action rules. Strong planning is complete. If a strong plan exists for some planning problem the strong planning algorithm will return it, otherwise, it returns that no solution exists. Strong planning is also optimal due to the breadth first search. Thus, a strong plan with the fewest number of steps in the worst case is returned.

5.2 Strong Cyclic Planning

Strong cyclic planning [9] is a relaxed version of strong planning, as it also considers plans with infinite length. Strong cyclic planning finds a strong plan

if it exists. Otherwise, if the algorithm at some point in the iteration is unable to find a strong preimage it adds an ordinary preimage (referred to as a weak preimage). It then tries to prune this preimage by removing all states that have transitions leading out of the preimage and the set of visited states V. If it succeeds, the remaining states in the preimage are added to V and it again tries to add strong preimages. If it fails, it adds a new, weak preimage and repeats the pruning process. A partial search of strong cyclic planning is shown in Figure 6. A strong cyclic plan only guarantees progress towards the goal in the strong

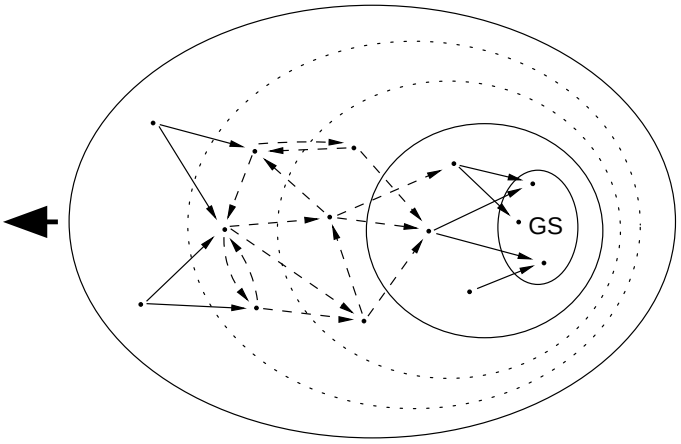

Fig. 6. Preimage calculations in strong cyclic planning. Dashed ellipses denote weak preimages while solid ellipses denote strong preimages. Only one action is assumed to exist in the domain. All the shown transitions are included in the universal plan. Dashed transitions are from "weak" parts of the plan while solid transitions are from "strong" parts of the plan.

parts. In the weak parts, cycles can occur. To keep the plan length finite, it must be assumed that a transition leading out of the weak parts eventually will be taken. The algorithm is complete as a strong solution will be returned if it exists. If no strong or strong cyclic solution exist the algorithm returns that no solution exists.

5.3 Strengths and Limitations

An important reason for studying universal planning is that universal planning algorithms can be made generally complete. Thus, if a plan exists for painting the floor, an agent executing a universal plan will always avoid to paint itself into the corner or reach any other unrecoverable dead-end. Strong planning and strong cyclic planning algorithms contribute by providing complete OBDD based algorithms for universal planning.

A limitation of strong and strong cyclic planning is their criteria for plan existence. If no strong or strong cyclic plan exist, these algorithms fail. The domains that strong and strong cyclic planning fail in are characterized by having unrecoverable dead-ends that cannot be guaranteed to be avoided.

Unfortunately, real world domains often have these kinds of dead-ends. Consider, for example, Schoppers' robot-baby domain described in Section 3. As depicted in Figure 4 no universal plan represented by a state-action set can guarantee the goal to be reached in a finite or infinite number of steps, as all relevant actions may lead to an unrecoverable dead-end.

A more interesting example is how to generate a universal plan for controlling, e.g., a power plant. Assume that actions can be executed that can bring the plant from any bad state to a good state. Unfortunately the environment can simultaneously fail subsystems of the plant which makes the resulting joint action non-deterministic, such that the plant may stay in a bad state or even change to an unrecoverable failed state (see Figure 7). No strong or strong cyclic solution can be found because an unrecoverable state can be reached from any initial state. An *NADL* description of a power plant domain is studied in Section 6.2.

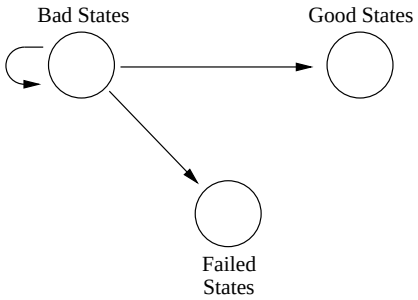

Fig. 7. Abstract description of the NFA of a power plant domain.

Another limitation of strong and strong cyclic planning is the inherent pessimism of these algorithms. Strong cyclic planning will always prefer to return a strong plan if it exists, even though a strong cyclic plan may exist with a shorter, best case plan length. Consider for example the domain described in Figure 8. The strong cyclic algorithm would return a strong plan only considering solid

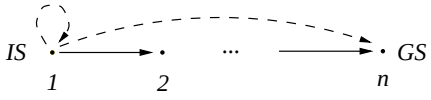

Fig. 8. The NFA of a domain with two actions (drawn as solid and dashed arrows) showing the price in best case plan length when preferring strong solutions. IS is the initial state while GS is the goal state.

actions. This plan would have a best and worst case length of n. But a strong cyclic plan considering both solid and dashed actions also exists and could be preferable because the best case length of 1 of the cyclic solution may have a much higher probability than the infinite worst case length.

By adding a unrecoverable dead-end for the dashed action and making solid actions non-deterministic (see Figure 9) strong cyclic planning now returns a strong cyclic plan considering only solid actions. But we might still be interested in a plan with best case performance even though the goal is not guaranteed to be achieved.

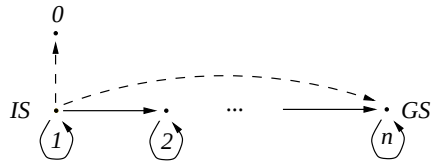

Fig. 9. The NFA of a domain with two actions (drawn as solid and dashed arrows) showing the price in best case plan length when preferring strong cyclic solutions. IS is the initial state while GS is the goal state.

5.4 Optimistic Planning

The analysis in the previous section shows that there exist domains and planning problems for which we may want to use a fully relaxed algorithm, that always includes the best case plan and returns a solution even if it includes dead-ends which cannot be guaranteed to be avoided. An algorithm similar to the strong planning algorithm, that adds an ordinary preimage in each iteration has these properties. Because state-action pairs that can have transitions to unrecoverable dead-ends are added to the universal plan, we call this algorithm *optimistic planning*. The algorithm is shown in Figure 10.

The optimistic planning algorithm is incomplete because it does not necessarily return a strong solution if it exists. Intuitively, optimistic planning only guarantees that there exists some effect of a plan action leading to the goal, where strong planning guarantees that all effects of plan actions lead to the goal.

The purpose of optimistic planning is not to substitute strong or strong cyclic planning. In domains where strong or strong cyclic plans can be found and goal achievement has the highest priority these algorithms should be used. On the other hand, in domains where goal achievement cannot be guaranteed or the shortest plan should be included in the universal plan, optimistic planning might be the better choice.

Consider again, as an example, the robot-baby domain described in Section 3. For this problem an optimistic solution makes the robot try to lift the block

```
procedure OptimisticPlanning(Init, Goal)
        VisitedStates := Goal
        UniversalPlan := ∅
        while (Init ⊄ VisitedStates)
                StateActions := Preimage(VisitedStates)
                PrunedStateActions := Prune(StateActions, VisitedStates)
                if StateActions ≠ ∅ then
                        UniversalPlan := UniversalPlan ∪ PrunedStateActions
                        VisitedStates := VisitedStates ∪ StatesOf(PrunedStateActions)
                else
                        return "No optimistic plan exists"
return UniversalPlan
```

Fig. 10. The optimistic planning algorithm. All sets in this algorithm are represented by their characteristic function which is implemented as an OBDD. Preimage(*VisitedStates*) returns the set of state-action pairs U associated with the preimage of the visited states. Prune(*StateActions*, *VisitedStates*) removes the state-action pairs, where the state already is included in the set of visited states. StatesOf(*PrunedStateActions*) returns the set of states of the pruned state-action pairs.

as long as it is working. A similar optimistic plan is generated in the power plant domain. For all bad states the optimistic plan recommend an action that brings the plant to a good state in one step. This continues as long as the environment keeps the plant in a bad state. Because no strategy can be used to avoid the environment from bringing the block lifting robot and power plant to an unrecoverable dead-end, the optimistic solution is quite sensible.

For the domains shown in Figure 8 and 9 optimistic planning would return a universal plan with two state-action pairs: $(1, \textit{dotted})$ and $(n-1, \textit{solid})$. For both domains this is a universal plan with the shortest best case length. Compared to the strong cyclic solution the price in the first domain is that the plan may have an infinite length, while the price in the second domain is that a dead-end may be reached.

6 Results

The UMOP planning system is implemented in C/C++ and uses the BUDDY package [32] for OBDD manipulations. The input to UMOP is an *NADL* description and a specification of which planning algorithm to use. The output is a universal plan or sequential plan depending on the planning algorithm. The current implementation of *NADL* only includes the arithmetic operators + and −, but an implementation of the remaining operators is straight forward and has only been omitted due to time limitations. UMOP generates an OBDD representation

of the partitioned transition relation as described in Section 4, which is used to generate the universal plan. During planning the dynamic variable reordering facility of the BUDDY package can be used to speed up the OBDD operations. A universal plan is represented by an OBDD and defines for each domain state a set of joint actions that the system agents must execute synchronously in order to achieve the goal. The implemented planning algorithms are:

1. Classical deterministic planning (see description below).
2. Strong planning.
3. Strong cyclic planning.
4. Optimistic planning.

The backward search of the deterministic planning algorithm is similar to the optimistic planning algorithm. A sequential plan is generated from the universal plan by choosing an initial state and iteratively adding an action from the universal plan until a goal state is reached. The deterministic planning algorithm has been implemented to verify the performance of UMOP compared to other classical planners. It has not been our intention though, to develop a fast OBDD-based classical planning algorithm like [14] as our main interest is non-deterministic universal planning.

In the following four subsections we present results obtained with the UMOP planning system in nine different domains ranging from deterministic and single-agent with no environment actions to non-deterministic and multi-agent with complex environment actions.[6] A more detailed description of the experiments can be found in [26].

6.1 Deterministic Domains

A number of experiments have been carried out in deterministic domains in order to verify UMOP's performance and illustrate the generality of universal plans versus classical, sequential plans. In the next section, we compare run time results obtained with UMOP in some of the AIPS'98 competition domains to the results of the competition planners. We then generate a universal plan in a deterministic obstacle domain to show that a large number of classical sequential plans are contained in the universal plan.

AIPS'98 Competition Domains Five planners BLACKBOX, IPP, STAN, HSP and SGP participated in the competition. Only the first four of these planners competed in the three domains, we have studied. BLACKBOX is based on SATPLAN [29], while IPP and STAN are graphplan-based planners [3]. HSP uses a heuristic search approach based on a preprocessing of the domain. The AIPS'98

[6] All experiments were carried out on a 350 MHz Pentium PC with 1 GB RAM running Red Hat Linux 4.2.

planners were run on 233/400 MHz[7] Pentium PCs with 128 MB RAM equipped with Linux.

The Gripper Domain. The gripper domain consists of two rooms A and B, a robot with a left and right gripper and a number of balls that can be moved by the robot. The task is to move all the balls from room A to room B, with the robot initially in room A. The state variables of the *NADL* encoding of the domain are the position of the robot and the position of the balls. The position of the robot is either 0 (room A) or 1 (room B), while the position of a ball can be 0 (room A), 1 (room B), 2 (in left gripper) or 3 (in right gripper). For the AIPS'98 gripper problems the number of plan steps in an optimal plan grows linear with the problem number. Problem 1 contains 4 balls, and the number of balls grow with two for each problem. The result of the experiment is shown in Table 1 together with the results of the planners in the AIPS'98 competition. A graphical representation of the execution time in the table is shown in Figure 11. UMOP generates shortest plans due to its parallel breadth first search algorithm. As depicted in Figure 11, it avoids the exponential growth of the execution time that characterizes all of the competition planners except HSP. When using a partitioned transition relation UMOP is the only planner capable of generating optimal plans for all the problems. For this domain the transition relation of an *NADL* description can be divided into $n + 1$ basic partitions, where n is the number of balls. As discussed in Section 4, the optimal number of partitions is not necessarily the largest number of partitions. For the results in Table 1 each partition equaled a conjunction of 10 basic partitions. Compared to the monolithic transition relation representation the results obtained with the partitioned transition relation was significantly better on the larger problems. The memory usage for problem 20 with a partitioned transition relation was 87 MB, while it, for the monolithic transition relation, exceeded the limit of 128 MB at problem 17.

The Movie Domain. In the movie domain the task is to get chips, dip, pop, cheese and crackers, rewind a movie and set the counter to zero. The only interference between the subgoals is that the movie must be rewound, before the counter can be set to zero. The problems in the movie domain only differs by the number of objects of each type of food. The number of objects increases linear from 5 for problem 1 to 34 for problem 30.

Our *NADL* description of the movie domain represents each type of food as a numerical state variable with a range equal to the number of objects of that type of food. Table 2 shows the execution time for UMOP and the competition planners for the movie domain problems. In this experiment and the remaining

[7] Unfortunately no exact record has been kept on the machines and there is some disagreement about their clock frequency. According to Drew McDermott, who chaired the competition, they were 233 MHz Pentiums, but Derek Long (STAN) believes, they were at least 400 MHz Pentiums, as STAN performed worse on a 300 MHz Pentium than in the competition.

Problem	UMOP Part.			UMOP Mono.		STAN		HSP		IPP		BLACKBOX	
1	20	11	1	20	11	46	11	2007	13	50	15	113	11
2	150	17	1	130	17	1075	17	2150	21	380	23	7820	17
3	710	23	1	740	23	54693	23	2485	31	3270	31	-	-
4	1490	29	2	2230	29	3038381	29	3060	37	26680	39	-	-
5	3600	35	2	6040	35	-	-	3320	47	226460	47	-	-
6	7260	41	2	11840	41	-	-	3779	53	-	-	-	-
7	13750	47	2	24380	47	-	-	4797	63	-	-	-	-
8	23840	53	2	38400	53	-	-	5565	71	-	-	-	-
9	36220	59	3	68750	59	-	-	6675	79	-	-	-	-
10	56200	65	3	95140	65	-	-	7583	85	-	-	-	-
11	84930	71	3	145770	71	-	-	9060	93	-	-	-	-
12	127870	77	3	216110	77	-	-	10617	101	-	-	-	-
13	197170	83	3	315150	83	-	-	12499	109	-	-	-	-
14	290620	89	4	474560	89	-	-	15050	119	-	-	-	-
15	411720	95	4	668920	95	-	-	16886	125	-	-	-	-
16	549610	101	4	976690	101	-	-	20084	135	-	-	-	-
17	746920	107	4	-	-	-	-	23613	143	-	-	-	-
18	971420	113	4	-	-	-	-	26973	151	-	-	-	-
19	1361580	119	5	-	-	-	-	29851	157	-	-	-	-
20	1838110	125	5	-	-	-	-	33210	165	-	-	-	-

Table 1. Gripper domain results. Column one and two show the execution time in milliseconds and the plan length. UMOP Part. and UMOP Mono. show the execution time for UMOP using a partitioned and a monolithic transition relation respectively. For UMOP with partitioned transition relation the third column shows the number of partitions. (- -) means the planner failed. Only results for executions using less than 128 MB are shown for UMOP.

experiments UMOP used its default partitioning of the transition relation. For every problem all the planners find the optimal plan. Like the competition planners UMOP has a low computation time, but it is the only planner not showing any increase in computation time even though, the size of the state space of its encoding increases from 2^{24} to 2^{39}.

The Logistics Domain. The logistics domain consists of cities, trucks, airplanes and packages. The task is to move packages to specific locations. Problems differ by the number of packages, cities, airplanes and trucks. The logistics domain is hard and only problem 1,2,5,7 and 11 of the 30 problems were solved by any planner in the AIPS'98 competition (see Table 3). The *NADL* description of the logistics domain uses numerical state variables to represent locations of packages, where trucks and airplanes are treated as special locations. Even though, the state space for the small problems is moderate, UMOP fails to solve any of the problems in the domain. It succeeds to generate the transition relation but fails to finish the preimage calculations. The reason for this might be a bad representation or variable ordering. It might also be that no compact OBDD

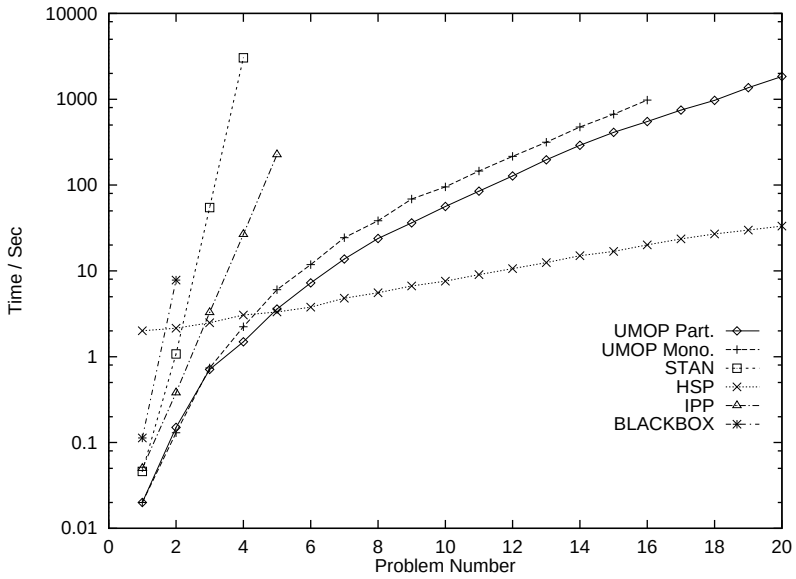

Fig. 11. Execution time for UMOP and the AIPS'98 competition planners for the gripper domain problems. UMOP Part. and UMOP Mono. show the execution time for UMOP using a partitioned and a monolithic transition relation respectively.

representation exists for this domain in the same way, that no compact OBDD representation exists for the integer multiplier [5]. More research is needed to decide this.

The Obstacle Domain The obstacle domain has been constructed to demonstrate the generality of universal plans. It consists of a 8×4 grid world, n obstacles and a robot agent. The position of the obstacles are not defined. The goal position of the robot is the upper right corner of the grid, and the task for the robot is to move from any position in the grid, different from the goal position, to the goal position. Because the initial location of obstacles is unknown, the universal plan must take any possible position of obstacles into account, which gives $2^{5(n+1)} - 2^{5n}$ initial states. For a specific initial state a sequential plan can be generated from the universal plan. Thus, $2^{5(n+1)} - 2^{5n}$ sequential plans are comprised in one universal plan. Note that a universal plan with n obstacles includes any universal plan with 1 to n obstacles, as obstacles can be placed at the same location. Note moreover, that the universal plans never covers all initial states, because obstacles can be placed at the goal position, and obstacles can block the way for the agent.

A universal plan for an obstacle domain with 5 obstacles was generated with UMOP in 420 seconds and contained 488296 OBDD nodes (13.3 MB). Sequential plans were extracted from the universal plan for a specific position of the obstacles, for which 16 step plans existed. Figure 12 shows the extraction time

Problem	UMOP		STAN		HSP		IPP		BLACKBOX	
1	14	7	19	7	2121	7	10	7	11	7
2	12	7	18	7	2104	7	10	7	12	7
3	14	7	19	7	2144	7	10	7	14	7
4	4	7	20	7	2188	7	10	7	16	7
5	14	7	21	7	2208	7	10	7	18	7
6	16	7	22	7	2617	7	10	7	20	7
7	14	7	22	7	2316	7	20	7	22	7
8	12	7	23	7	2315	7	20	7	24	7
9	14	7	25	7	2357	7	-	-	26	7
10	14	7	26	7	2511	7	10	7	29	7
11	14	7	27	7	2427	7	30	7	30	7
12	4	7	28	7	2456	7	30	7	32	7
13	16	7	29	7	3070	7	20	7	36	7
14	14	7	31	7	2573	7	30	7	35	7
15	16	7	32	7	2577	7	30	7	38	7
16	14	7	34	7	2699	7	10	7	39	7
17	16	7	35	7	2645	7	30	7	41	7
18	14	7	37	7	2686	7	10	7	43	7
19	16	7	39	7	2727	7	30	7	45	7
20	12	7	40	7	2787	7	20	7	47	7
21	16	7	42	7	2834	7	20	7	49	7
22	14	7	45	7	2834	7	20	7	51	7
23	16	7	48	7	2866	7	20	7	53	7
24	14	7	50	7	3341	7	20	7	55	7
25	16	7	52	7	2997	7	30	7	57	7
26	16	7	54	7	3013	7	40	7	58	7
27	16	7	57	7	3253	7	50	7	60	7
28	4	7	62	7	3049	7	40	7	63	7
29	18	7	64	7	3384	7	50	7	64	7
30	16	7	67	7	3127	7	40	7	66	7

Table 2. Movie domain results. For each planner column one and two show the run time in milliseconds and the plan length. (- -) means the planner failed. UMOP used far less than 128 MB for any problem in this domain.

Problem	STAN		HSP		IPP		BLACKBOX	
1	767	27	79682	43	900	26	2062	27
2	4319	32	97114	44	-	-	6436	32
5	364932	29	144413	26	2400	24	-	-
7	-	-	788914	112	-	-	-	-
11	12806	34	86195	30	6940	33	6544	32

Table 3. Logistics domain results. For each planner column one and two show the run time in milliseconds and the plan length. (- -) means the planner was unable to find a solution.

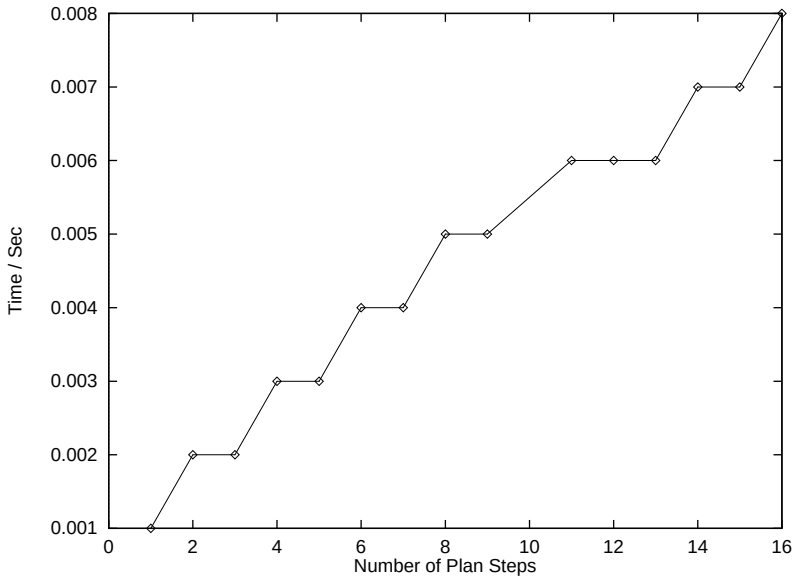

Fig. 12. Time for extracting sequential plans from a universal plan for the obstacle domain with 5 obstacles.

of sequential plans for an increasing number of steps in the plan. Even though the OBDD representing the universal plan is large, the extraction is very fast and only grows linear with the plan length. The set of actions associated with a state s in a universal plan p is extracted by computing the conjunction of the OBDD representation of s and p. As described in Section 2, this operation has an upper bound complexity of $O(|s||p|)$. For the universal plan in the obstacle domain with five obstacles this computation was fast (less than one millisecond) and would allow an executing agent to meet low reaction time constraints, but in the general case, it depends on the structure of the universal plan and might be more time consuming.

6.2 Non-Deterministic Domains

In this section we first test UMOP's performance for some of the non-deterministic domains solved by MBP [9, 10]. Next, we present a version of the power plant domain briefly described in Section 5.2 and finally, we show results from a multi-agent soccer domain.

Domains Tested by MBP One of the domains introduced in [9, 10] is a non-deterministic transportation domain. The domain consists of a set of locations and a set of actions like drive-truck, drive-train and fly to move between the locations. Non-determinism is caused by non-deterministic actions (e.g., a truck

may use the last fuel) and environmental changes (e.g., fog at airports). We defined the two domain examples from [9, 10] for strong and strong cyclic planning in *NADL* and ran UMOP using strong and strong cyclic planning. Both examples were solved in less than 0.05 seconds. Similar results were obtained with MBP. In [9] a general version of the hunter and prey domain [30] and a beam walk domain is also studied. Their generalization of the hunter and prey domain is not described in detail. Thus, we have not been able to make an *NADL* implementation of this domain.

The problem in the beam walk domain is for an agent to walk from one end of a beam to the other without falling down. If the agent falls, it has to walk back to the end of the beam and try again. The finite state machine of the domain is shown in Figure 13. The edges denotes the outcome of a walk action. When the agent is on the beam, the walk action can either move it one step further on the beam or make it fall to a location under the beam. We implemented a generator

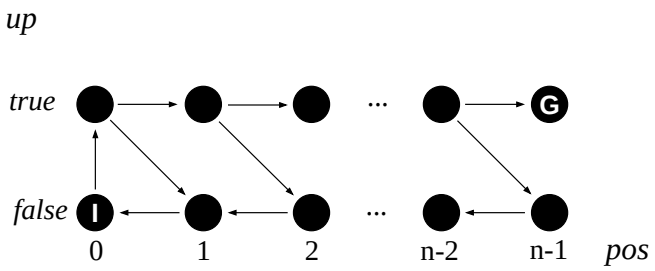

Fig. 13. The beam walk domain. The *NADL* encoding of the beam walk domain has one propositional state variable *up*, which is true if the agent is on the beam and a numerical state variable *pos*, which denotes the position of the agent either on the beam or on the ground. "I" and "G" are the intial state and goal state repspectively.

program for *NADL* descriptions of beam walk domains and produced domains with 4 to 4096 positions. Because the domain only contains two state variables, UMOP cannot exploit a partitioned transition relation for this domain, but have to use a monolithic representation. As shown in Figure 14 the execution time of UMOP was a little smaller than MBP. When discounting that we used a 75% faster machine MBP performs better in this domain. We do not believe this to be caused by an inefficient representation, as UMOP exploits the regularity of the domain in the same way as MBP. A more reasonable explanation is that UMOP uses a less efficient implementation of the strong cyclic planning algorithm.

A detailed comparison of UMOP and MBP is very interesting, as the two systems represent planning problems in a quite different way. Currently MBP is unable to use a partitioned transition relation representation, but it is still an open question if UMOP is able to solve larger problems than MBP due to this feature.

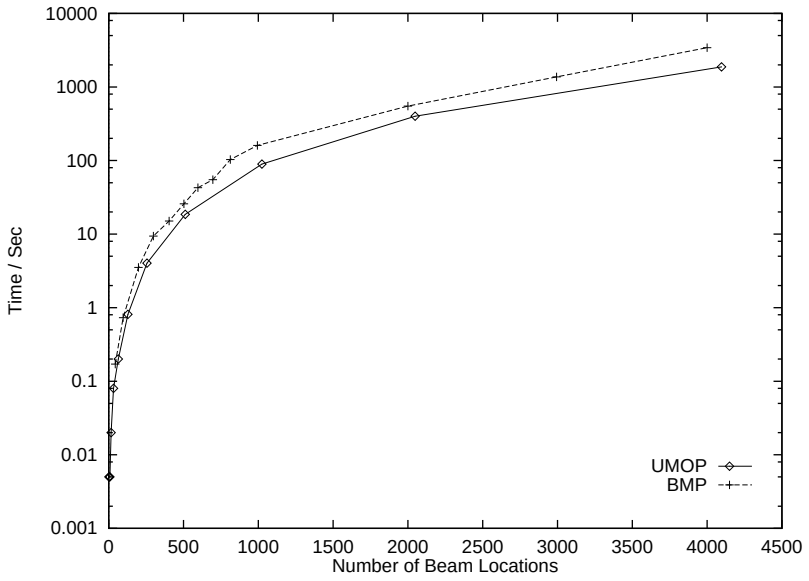

Fig. 14. Execution time of UMOP and MBP in the beam walk domain. The MBP data has been extracted with some loss of accuracy from [9].

The Power Plant Domain Until now we have concentrated on presenting results that show UMOP's performance compared to other planners. The purpose of the remaining experiments is to show universal planning results for domains, where the multi-agent and environment modelling features of *NADL* have been used.

The power plant domain demonstrates a multi-agent domain with an environment model and further exemplifies optimistic planning. It consists of reactors, heat exchangers, turbines and valves. A domain example is shown in Figure 15. In the power plant domain each controllable unit is associated with an agent such that all control actions can be executed simultaneously. The environment consists of a single agent that at any time can fail a number of heat exchanges and turbines and ensures that already failed units remain failed. A failed heat exchanger leaks radioactive water from the internal to the external water loop and must be closed by a block action b. For a failed turbine the stop action s must be carried out. The energy production from the reactor can be controlled by p to fit the demand f, but the reactor will always produce two energy units. To transport the energy from the reactor away from the plant at least one heat exchanger and one turbine must be working. Otherwise the plant is in an unrecoverable failed state, where the reactor will overheat.

The state space of the power plant can be divided into three disjoint sets: good, bad and failed states. In the good states the power plant satisfies its safety and activity requirements. In our example the safety requirements ensures that energy can be transported away from the plant and failed units are shut down:

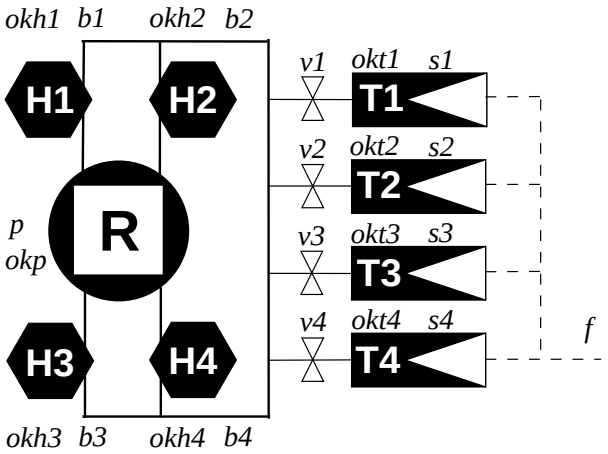

Fig. 15. A power plant domain example. The reactor R is surrounded by the four heat exchangers H1, H2, H3 and H4. The heat exchangers produces high pressure damp to the four electricity generating turbines T1, T2, T3 and T4. A failed heat exchanger must be closed by a block action b. For a failed turbine the stop action s must be carried out. The energy production of the reactor is p and can be controlled to fit the demand f. Each turbine can be closed of by a valve v.

```
% energy can be transported away from the plant
(okh1 \/ okh2 \/ okh3 \/ okh4) /\
(okt1 \/ okt2 \/ okt3 \/ okt4) /\

% heat exchangers blocked if failed
(~okh1 => b1) /\
(~okh2 => b2) /\
(~okh3 => b3) /\
(~okh4 => b4) /\

% turbines stopped if failed
(~okt1 => s1) /\
(~okt2 => s2) /\
(~okt3 => s3) /\
(~okt4 => s4)
```

The activity requirements state that the energy production equals the demand and that all valves to working turbines are open:

```
% power production equals demand
p = f /\

% turbine valve is open if turbine is ok
```

```
(okt1 => v1) /\
(okt2 => v2) /\
(okt3 => v3) /\
(okt4 => v4)
```

In a bad state the plant does not satisfy the safety and activity requirements, but on the other hand is not unrecoverably failed. Finally, in a failed state all heat exchangers or turbines are failed.

The universal planning task is to generate a universal plan to get from any bad state to some good state without ending in a failed state. Assuming that no units fail during execution, it is obvious that only one joint action is needed. Unfortunately, the environment can fail any number of units during execution, thus, as described in Section 5.2, for any bad state the resulting joint action may loop back to a bad state or cause the plant to end in a failed state. (see Figure 7). For this reason no strong or strong cyclic solution exists to the problem.

An optimistic solution simply ignores that joint actions can loop back to a bad state or lead to a failed state and finds a solution to the problem after one preimage calculation. Intuitively, the optimistic plan assumes that no units will fail during execution and always chooses joint actions that lead directly from a bad state to a good state. The optimistic plan is an optimal control strategy, because it always chooses the shortest way to a good state and no other strategy exists that can avoid looping back to a bad state or end in a failed state.

The size of the state space of the above power plant domain is 2^{24}. An optimistic solution was generated by UMOP in 0.92 seconds and contained 37619 OBDD nodes. As an example, a joint action was extracted from the plan for a bad state where H3 and H4 were failed and energy demand f was 2 energy units, while the energy production p was only 1 unit. The extraction time was 0.013 seconds and as expected the set of joint actions included a single joint action changing $b3$ and $b4$ to true and setting p to 2.

The Soccer Domain The purpose of the soccer domain is to demonstrate a multi-agent domain with a more elaborate environment model than the power plant domain. It consists of two teams of players that can move in a grid world and pass a ball to each other. At each time step a player either moves in one of the four major directions or passes the ball to another team player. The task is to generate a universal plan for one of the teams that can be applied, whenever the team possesses the ball in order to score a goal.

A simple *NADL* description of the soccer domain models the team possessing the ball as system agents that can move and pass the ball independent of each other. Thus, a player possessing the ball can always pass to any other team player. The opponent team is modelled as a set of environment agents that can move in the four major directions but have no actions for handling the ball. The goal of the universal plan is to have a player with the ball in front of the opponent goal without having any opponents in the goal area.

Clearly, it is impossible to generate a strong plan that covers all possible initial states. But a strong plan covering as many initial states as possible is useful, because it defines all the "scoring" states of the game and further provides a plan for scoring the goal no matter what actions, the opponent players choose.

We implemented an *NADL* generator for soccer domains with different field sizes and numbers of agents. The Multi-Agent graph in Figure 16 shows UMOP's execution time using the strong planning algorithm in soccer domains with 64 locations and one to six players on each team. The execution time seems to

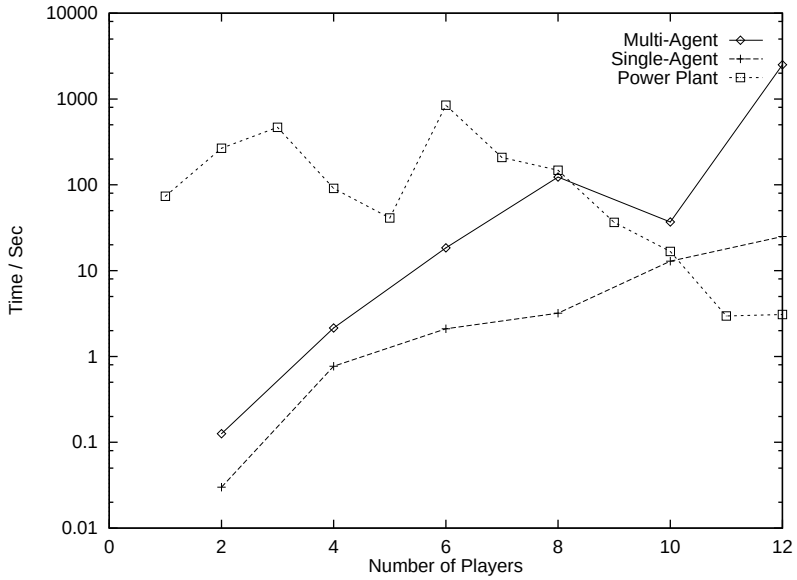

Fig. 16. Execution time of UMOP for generating strong universal plans in soccer domains with one to six players on each team. For the multi-agent experiment each player was associated with an agent, while only a single system and environment agent was used in the single-agent experiment. The power plant graph show execution time for a complex deterministic power plant domain using 1 to 12 system agents.

grow exponential with the number of players. This is not surprising as not only the state space but also the number of joint actions grow exponential with the number of agents. To investigate the complexity introduced by joint actions, we constructed a version of the soccer domain with only a single system and environment agent and ran UMOP again. The Single-Agent graph in Figure 16 shows a dramatic decrease in computation time. Its is not obvious though, that a parallelization of domain actions increases the computational load as this normally also reduces the number of preimage calculations, because a larger number of states is reached in each iteration. Indeed, in a deterministic version of the power plant domain we found the execution time to decrease (see the Power Plant graph in Figure 16), when more agents were added [26]. Again we mea-

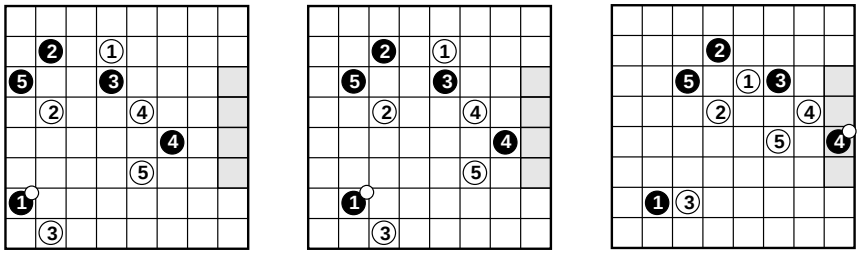

Fig. 17. Plan execution sequence. The three states show a hypothetical attack based on a universal plan. The state (a) is a "scoring" state, because the attackers (black) can extract a nonempty set of joint actions from the universal plan. Choosing some joint actions from the plan the attackers can enter the goal area (shaded) with the ball within two time steps (state (b) and (c)) no matter what actions, the opponent players choose.

sured the time for extracting actions from the generated universal plans. For the multi-agent version of the five player soccer domain the two joint actions achieving the goal shown in Figure 17 were extracted from the universal plan in less than 0.001 seconds.

7 Previous Work

Recurrent approaches performing planning in parallel with execution have been widely used in non-deterministic robotic domains (e.g., [21, 19, 44, 25]). A group of planners suitable for recurrent planning is action selectors based on heuristic search [30, 4]. The min-max LRTA* algorithm [30] can generate suboptimal plans in non-deterministic domains through a search and execution iteration. The search is based on a heuristic goal distance function, which must be provided for a specific problem. The ASP algorithm [4] uses a similar approach and further defines a heuristic function for STRIPS-like [18] action representations. In contrast to min-max LRTA*, ASP does not assume a non-deterministic environment, but is robust to non-determinism caused by action perturbations (i.e., that another action than the planned action is chosen with some probability).

In general recurrent approaches are incomplete, because acting on an incomplete plan can make the goal unachievable. Precursor approaches perform all decision making prior to execution and thus may be able to generate complete plans by taking all possible non-deterministic changes of the environment into account.

The precursor approaches include conditional [17, 36], probabilistic [16, 13] and universal planning [38, 9, 10, 27]. The CNLP [36] partial ordered, conditional planner handles non-determinism by constructing a conditional plan that accounts for each possible situation or contingency that could arise. At execution

time it is determined which part of the plan to execute by performing sensing actions that are included in the plan to test for the appropriate conditions.

Probabilistic planners try to maximize the probability of goal satisfaction, given conditional actions with probabilistic effects. In [16] plans are represented as a set of Situated Control Rules (SCRs) [15] mapping situations to actions. The planning algorithm begins by adding SCRs corresponding to the most probable execution path that achieves the goal. It then continues adding SCRs for less probable paths, and may end with a complete plan taking all possible paths into account.

Universal plans differs from conditional and probabilistic plans by specifying appropriate actions for every possible state of the domain.[8] Like conditional and probabilistic plans universal plans require the world to be accessible in order to execute the universal plan.

Universal planning was introduced in [38] who used decision trees to represent plans. Recent approaches include [27, 9, 10]. In [27] universal plans are represented as a set of Situated Control Rules [15]. Their algorithm incrementally adds SCRs to a final plan in a way similar to [16]. The goal is a formula in temporal logic that must hold on any valid sequence of actions.

Reinforcement Learning (RL) [28] can also be regarded as a kind of universal planning. In RL the goal is represented by a reward function in a Markov Decision Process (MDP) model of the domain. In the precursor version of RL the MDP is assumed to be known and a control policy maximizing the expected reward is found prior to execution. The policy can either be represented explicitly in a table or implicitly by a function (e.g., a neural network). Because RL is a probabilistic approach, its domain representation is more complex than the domain representation used by a non-deterministic planner. Thus, we may expect non-deterministic planners to be able to handle domains with a larger state space than RL. On the other hand, RL may produce policies with a higher quality, than a universal plan generated by a non-deterministic planner.

All previous approaches to universal planning, except [9, 10], use an explicit representation of the universal plan (e.g., SCRs). Thus, in the general case exponential growth of the plan size with the number of propositions defining a domain state must be expected, as argued in [22].

The compact and implicit representation of universal plans obtained with OBDDs do not necessarily grow exponentially for regular domains as shown in [9]. Further, the OBDD-based representation of the NFA of a non-deterministic domain enables the application of efficient search algorithms from model checking, capable of handling very large state spaces.

[8] The plan solving an *NADL* problem will only be universal if the initial states equals all the states in the domain.

8 Conclusion and Future Work

In this article we have presented a new OBDD-based planning system called UMOP for planning in non-deterministic, multi-agent domains. An expressive domain description language called *NADL* has been developed and an efficient OBDD representation of its NFA semantics has been described. We have analyzed previous planning algorithms for OBDD-based planning and deepened the understanding of when these planning algorithms are appropriate. Finally, we have proposed a planning algorithm called optimistic planning for finding sensible solutions in some domains where no strong or strong cyclic solution exists. The results obtained with UMOP are encouraging, as UMOP has a good performance compared to some of the fastest classical planners known today.

Our research has drawn our attention to a number of open questions that we would like to address in the future. The most interesting of these is how well our encoding of planning problems scales compared to the encoding used by MBP. Currently MBP's encoding does not support a partitioned representation of the transition relation, but the encoding may have other properties that, despite the monolithic representation, makes it a better choice than UMOP's encoding. On the other hand, the two systems may also have an equal performance when both are using a monolithic representation (as in the beam walk example), which should give UMOP an advantage in domains where a partitioning of the transition relation can be defined. We are planning a joint work with Marco Roveri in the Fall of 1999 to address these issues.

Another interesting question is to investigate which kind of planning domains is suitable for OBDD-based planning. It was surprising for us that the logistics domain turned out to be so hard for UMOP. A thorough study of this domain may be the key for defining new approaches and might bring important new knowledge about the strengths and limitations of OBDD-based planning.

The current definition of *NADL* is powerful but should be extended to enable modelling of constructive synergetic effects as described in Section 3. Also, more experiments comparing multi-agent and single-agent domains should be carried out to investigate the complexity of *NADL*'s representation of concurrent actions.

As argued in [1], domain knowledge must be used by a planning system in order to scale up to real world problems. They show how the search tree of a forward chaining planner can be efficiently pruned by stating the goal as formula in temporal logic on the sequence of actions leading to the goal. In this way the goal can include knowledge about the domain (e.g., that towers in the blocks world must be built from bottom to top). A similar approach for reducing the complexity of OBDD-based planning is obvious, especially because techniques for testing temporal formulas already have been developed in model checking.

Other future challenges includes introducing abstraction in OBDD-based planning and defining specialized planning algorithms for multi-agent domains (e.g., algorithms using the least number of agents for solving a problem).

Acknowledgements A special thanks to Paolo Traverso, Marco Roveri and the other members of the IRST group for introducing us to MBP and for many

rewarding discussions on OBDD-based planning and model checking. We also wish to thank Randal E. Bryant, Edmund Clarke, Henrik R. Andersen, Jørn Lind-Nielsen and Lars Birkedal for advice on OBDD issues and formal representation.

This work was carried out while the first author was visiting Carnegie Mellon University from the Technical University of Denmark. The research is sponsored in part by McKinsey & Company, Selmar Tranes Fond, the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under agreement number F30602-97-2-0250. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory (AFRL) or the U.S. Government.

Appendix A. *NADL* Includes the $\mathcal{AR}$ Family

Theorem 3. *If A is a domain description for some $\mathcal{AR}$ language A, then there exists a domain description D in the NADL language with the same semantics as A*

Proof: let $M_a = (Q, \Sigma, \delta)$ denote the NFA (see Definition 1) equal to the semantics of A as defined in [23]. An *NADL* domain description D with semantics equal to M_a can obviously be constructed in the following way: let D be a single-agent domain, where all fluents are encoded as numerical variables and there is an action for each element in the alphabet Σ of M_a. Consider the action a associated to input $i \in \Sigma$. Let the set of constrained state variables of a equal all the state variables in D. The precondition of a is an expression that defines the set of states having an outgoing transition for input i. The effect condition of a is a conjunction of conditional effects $P_s \Rightarrow N_s$. There is one conditional effect for each state that has an outgoing transition for input i. P_s in the conditional effect associated with state s is the characteristic expression for s and N_s is a characteristic expression for the set of next states $\delta(s, i)$. □

References

1. F. Bacchus and F. Kabanza. Using temporal logic to control search in a forward chaining planner. In M. Ghallab and A. Milani, editors, *New directions in AI planning*, pages 141–153. ISO Press, 1996.
2. C. Baral and M. Gelfond. Reasoning about effects of concurrent actions. *The Journal of Logic Programming*, pages 85–117, 1997.
3. A. Blum and M. L. Furst. Fast planning through planning graph analysis. In *Proceedings of the 14'th International Conference on Artificial Intelligence (IJCAI-95)*, pages 1636–1642. Morgan Kaufmann, 1995.
4. B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the 14'th National Conference on Artificial Intelligence (AAAI'97)*, pages 714–719. AAAI Press / The MIT Press, 1997.

5. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 8:677–691, 1986.
6. J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic model checking with partitioned transition relations. In *International Conference on Very Large Scale Integration*, pages 49–58. North-Holland, 1991.
7. E. Carke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999. In Press.
8. A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: A decision procedure for $\mathcal{AR}$. In *Proceedings of the 4'th European Conference on Planning (ECP'97)*, Lecture Notes in Artificial Intelligence, pages 130–142. Springer-Verlag, 1997.
9. A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the 15'th National Conference on Artificial Intelligence (AAAI'98)*, pages 875–881. AAAI Press/The MIT Press, 1998.
10. A. Cimatti, M. Roveri, and P. Traverso. Strong planning in non-deterministic domains via model checking. In *Proceedings of the 4'th International Conference on Artificial Intelligence Planning System (AIPS'98)*, pages 36–43. AAAI Press, 1998.
11. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
12. K. Currie and A. Tate. O-plan: the open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.
13. T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76:35–74, 1995.
14. M. Di Manzo, E. Giunchiglia, and S. Ruffino. Planning via model checking in deterministic domains: Preliminary report. In *Proceedings of the 8'th International Conference on Artificial Intelligence: Methodology, Systems and Applications (AIMSA'98)*, pages 221–229. Springer-Verlag, 1998.
15. M. Drummond. Situated control rules. In *Proceedings of the 1'st International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 103–113. Morgan Kaufmann, 1989.
16. M. Drummond and J. Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the 8'th Conference on Artificial Intelligence*, pages 138–144. AAAI Press / The MIT Press, 1990.
17. O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An approach for planning with incomplete information. In *Proceedings of the 3'rd International Conference on Principles of Knowledge Representation and Reasoning*, 1992.
18. R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
19. E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the 10'th National Conference on Artificial Intelligence (AAAI'92)*, pages 809–815. MIT Press, 1992.
20. M. Gelfond and V. Liftschitz. Representing action and change by logic programs. *The Journal of Logic Programming*, 17:301–322, 1993.
21. M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the 6'th National Conference on Artificial Intelligence (AAAI'87)*, pages 677–682, 1987.

22. M. L. Ginsberg. Universal planning: An (almost) universal bad idea. *AI Magazine*, 10(4):40–44, 1989.
23. E. Giunchiglia, G. N. Kartha, and Y. Lifschitz. Representing action: Indeterminacy and ramifications. *Aritificial Intelligence*, 95:409–438, 1997.
24. E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: Preliminary report. In *Proceedings of the 15'th National Conference on Artificial Intelligence (AAAI'98)*, pages 623–630. AAAI Press/The MIT Press, 1998.
25. K. Z. Haigh and M. M. Veloso. Planning, execution and learning in a robotic agent. In *Proceedings of the 4'th International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, pages 120–127. AAAI Press, 1998.
26. R. M. Jensen. OBDD-based universal planning in multi-agent, non-deterministic domains. Master's thesis, Technical University of Denmark, Department of Automation, 1999. IAU99F02.
27. F. Kabanza, M. Barbeau, and R. St-Denis. Planning control rules for reactive agents. *Artificial Intelligence*, 95:67–113, 1997.
28. L. P. Kaebling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
29. H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the 13'th National Conference on Artificial Intelligence (AAAI'96)*, volume 2, pages 1194–1201. AAAI Press/MIT Press, 1996.
30. S. Koenig and R. G. Simmons. Real-time search in non-deterministic domains. In *Proceedings of the 14'th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1660–1667. Morgan Kaufmann, 1995.
31. J. Lever and B. Richards. *Parc*plan: a planning architecture with parallel actions and constraints. In *Lecture Notes in Artificial Intelligence*, pages 213–222. ISMIS'94, Springer-Verlag, 1994.
32. J. Lind-Nielsen. BuDDy - A Binary Decision Diagram Package. Technical Report IT-TR: 1999-028, Institute of Information Technology, Technical University of Denmark, 1999. `http://cs.it.dtu.dk/buddy`.
33. A. R. Lingard and E. B. Richards. Planning parallel actions. *Artificial Intelligence*, 99:261–324, 1998.
34. K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publ., 1993.
35. J. S. Penberthy and D. S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the 3'rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114. Morgan Kaufmann, 1992.
36. M. Peot and D. Smith. Conditional nonlinear planning. In *Proceedings of the 1'st International Conference on Artificial Intelligence Planning Systems (AIPS'92)*, pages 189–197. Morgan Kaufmann, 1992.
37. R. K. Ranjan, A. Aziz, R. K. Brayton, B. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. In *IEEE/ACM Proceedings International Workshop on Logic Synthesis*, 1995.
38. M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the 10'th International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 1039–1046. Morgan Kaufmann, 1987.
39. P. Stone and M. M. Veloso. Towards collaborative and adversarial learning: A case study in robotic soccer. *International Journal of Human-Computer Studies (IJHCS)*, 1998.
40. R. S. Sutton and Barto A. G. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

41. M. Veloso, J. Carbonell, A. Pérez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1), 1995.
42. M. M. Veloso, M. E. Pollack, and M. T. Cox. Rationale-based monitoring for planning in dynamic environments. In *Proceedings of the 4'th International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, pages 171–179. AAAI Press, 1998.
43. D. Weld. Recent advances in AI planning. *Artificial Intelligence Magazine*, 1999. (in press).
44. D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:197–227, 1994.