Solving the Entailment Problem in the Fluent Calculus using Binary Decision Diagrams

Steffen Hölldobler and Hans–Peter Störr
Artificial Intelligence Institute
Department of Computer Science
Dresden University of Technology

Abstract

The paper is an exercise in formal program development. It is rigorously shown how planning problems encoded as entailment problems in the fluent calculus can be mapped onto satisfiability problems for propositional formulas, which in turn can be mapped to the problem to find models using binary decision diagrams. Preliminary experimental results of an implementation are discussed.

Keywords: Reasoning about Actions, Planning, Fluent Calculus, Binary Decision Diagrams.

Introduction

In recent years propositional methods have seen a surprising revival in the field of Intellectics, i.e., Artificial Intelligence and Cognitive Science. Greedy satisfiability testing and its variants (Selman, Levesque, & Mitchell 1992) and the various procedures for answer set computing (e.g. (Niemelä & Simons 1997)) are just two examples. Another propositional method, model checking using binary decision diagrams (BDDs), has found widespread use in applications of computer aided design such as formal verification and logic synthesis (Bryant 1986). In many cases techniques based on BDDs have significantly improved the performance of algorithms and enabled the solution of new classes of problems (see e.g. (Burch *et al.* 1992; 1994)). But only recently researchers have started to investigate whether BDDs may also help to increase the efficiency of algorithms solving typical problems in Intellectics like, for example, planning problems (Cimatti *et al.* 1997; Cimatti, Roveri, & Traverso 1998). In this paper we report on some preliminary results in an attempt to utilize BDDs for reasoning about situations, actions and causality.

Without any doubt intelligent agents must be able to reason about the current state of the world, their ability to perform actions and the causes of performing certain actions. In particular, they must be able to plan their actions ahead in order to achieve their goals. There are a variety of approaches to planning starting with (Green 1969) up to (Blum & Furst 1997). In the preparation for a planning contest held at AIPS98, the language PDDL (Planning Domain Definition Language) for expressing planning problems was proposed (Ghallab *et al.* 1998). While this language allows for a convenient specification of planning problems, we are unaware of a formal semantics. In a separate paper, a formal transformation from the `:adl` fragment of PDDL into the fluent calculus is specified (Störr in preparation). The fluent calculus itself is a formal calculus for reasoning about situations, actions and causality which admits a well–defined (standard) semantics (Hölldobler & Schneeberger 1990; Thielscher 1998a).

In this paper we consider a restricted fragment of the fluent calculus, which allows for the specification of planning problems as entailment problems in the spirit of (McCarthy 1963). We formally define a transformation mapping these planning problems onto satisfiability problems in propositional logic and, thus, we prove the decidability of the abovementioned fragment of the fluent calculus. Thereafter, we describe how the shortest plan solving the planning problem can be extracted from the propositional encoding. We briefly present some preliminary findings of an implementation using BDDs and conclude by discussing our results.

Foundations

Logics

Let Σ_V, Σ_F and Σ_P denote disjunct sets of *variables*, *function symbols* and *predicate symbols* respectively. Σ_V is countably infinite, whereas Σ_F and Σ_P are finite. The set of (*first order*) *formulas* is denoted by $\mathcal{L}(\Sigma_V \cup \Sigma_F \cup \Sigma_P)$; we abbreviate this set by $\mathcal{L}$ if the sets Σ_V, Σ_F and Σ_P can be determined from the context. σ denotes a substitution and $X\sigma$ the instance of the syntactic object X under σ .

Table 1 depicts some notational conventions in the sense that, for example, whenever we use F , we implicitly assume $F \in \mathcal{L}$. Σ_{Fl}, $\Sigma_{V,St}$, $\Sigma_{V,Fl}$, $\Sigma_{V,Sit}$ and constructor state terms are defined later. All symbols are possibly indexed. Free variables occurring in formulas are assumed to be universally quantified.

The *entailment problem* $\mathcal{F} \models F$ consists of a set $\mathcal{F}$ of formulas and a formula F and is the question whether $\mathcal{F}$ entails F .

Symbol	f	$F, G, \ldots$	$\mathcal{F}$	$i, j, \ldots$	z
Element of	Σ_{Fl}	$\mathcal{L}$	$2^{\mathcal{L}}$	$\mathbb{N}_0$	$\Sigma_{V,St}$
Symbol	g	s	t		
Element of	$\Sigma_{V,Fl}$	$\Sigma_{V,Sit}$	constructor state terms		

Table 1: Notational conventions.

Planning

In this paper we consider planning problems having the following properties:

- The set of states is characterized by a finite set of propositional fluents, i.e., a set of propositional variables, which can take values out of $\{\top, \bot\}$.
- The actions are deterministic and their preconditions as well as effects depend only on the state they are executed in.
- The goal of the planning problem is a property which depends solely on the reached state.

This class of problems corresponds to the problems from track 1 and 2 of the planning competition held at AIPS98. There, planning problems were formulated within a language called PDDL (Ghallab *et al.* 1998). As mentioned in the introduction, we have defined a translation from PDDL into the fluent calculus elsewhere. For the purpose of this paper we simply assume a fluent calculus specification of the planning problems.

The Fluent Calculus

The fluent calculus is a calculus for reasoning about situations, actions and causality. It is based on the idea to consider states as multi–sets of fluents and to represent such states on the term level. The latter is done with the help of a binary function symbol $\circ$, which is associative, commutative and has a constant $\emptyset$ as unit element but is not idempotent (Hölldobler & Schneeberger 1990; Thielscher 1998a). We consider a restricted version of the fluent calculus specified in the sequel.

Formally, the fluent calculus is an order–sorted calculus with sorts ACTION, SIT, FLUENT, and STATE and ordering constraint FLUENT $<$ STATE. The set Σ_V of variables is the union of the disjunct sets $\Sigma_{V,A}$, $\Sigma_{V,Sit}$, $\Sigma_{V,Fl}$ and $\Sigma_{V,St}$, i.e., it consists of a countable set of variables for each sort. The set Σ_F of function symbols is the union $\Sigma_A \cup \Sigma_{Sit} \cup \Sigma_{Fl} \cup \Sigma_{St}$, where Σ_A is a set of function symbols denoting action names, $\Sigma_{Sit} = \{S_0, do\}$ is the set of function symbols denoting situations, Σ_{Fl} is a set of constant symbols denoting fluent names, $\Sigma_{St} = \{\emptyset, \circ, state\}$ is the set of function symbols denoting states. All sets are mutually disjoint and finite. The mentioned function symbols are sorted as follows:

$$
\begin{array}{lcl}
S_0 & : & \text{SIT}, \\
do & : & \text{ACTION} \times \text{SIT} \rightarrow \text{SIT}, \\
\emptyset & : & \text{STATE}, \\
\circ & : & \text{STATE} \times \text{STATE} \rightarrow \text{STATE}, \\
state & : & \text{SIT} \rightarrow \text{STATE}.
\end{array}
$$

The set Σ_P of predicate symbols consists only of the equality symbol with sort $=:$ STATE $\times$ STATE. We will often make use of a macro *holds* of sort FLUENT $\times$ SIT defined as

$$holds(f, s) \stackrel{def}{=} (\exists z)\ state(s) = f \circ z$$

and a macro *set* of sort STATE defined as

$$set(z) \stackrel{def}{=} \neg(\exists f, \hat{z})\ z = f \circ f \circ \hat{z}\ .$$

The language $\mathcal{L}_{FC}$ of the fluent calculus is the set of all well–formed and well–sorted first–order formulas over the given alphabet. State terms of the form $\emptyset \circ f_1 \circ \ldots \circ f_m$, where $f_1, \ldots, f_m \in$ FLUENT are pairwise distinct, $m \geq 0$, are called *constructor state terms*.

The axioms $\mathcal{F}$ of the fluent calculus considered in this paper are the union $\mathcal{F}_{un} \cup \mathcal{F}_{mset} \cup \mathcal{F}_{S_0} \cup \mathcal{F}_{ms} \cup \mathcal{F}_{su}$.

- $\mathcal{F}_{un}$ is a set of *unique name assumption for fluents* combined with a domain closure axiom for fluents:

$$\mathcal{F}_{un} = \{\neg f_1 = f_2 \mid f_1, f_2 \in \Sigma_{Fl} \text{ and } f_1 \neq f_2\} \cup \{(\forall g) \bigvee_{f \in \Sigma_{Fl}} g = f\}\ .$$

- $\mathcal{F}_{mset}$ is a set of axioms ensuring that the sort STATE denotes multisets of fluents.[1] It consists of the following formulae:
  - the standard axioms of equality
  - the axioms AC1 for $\circ$ and $\emptyset$:

$$
\begin{aligned}
z \circ \emptyset &= z \\
z_1 \circ z_2 &= z_2 \circ z_1 \\
(z_1 \circ z_2) \circ z_3 &= z_1 \circ (z_2 \circ z_3)
\end{aligned}
$$

  - an axiom that guarantees that fluents and $\emptyset$ are the only irreducible elements of $\circ$:[2]

$$[(\exists g)\ z = g \vee z = \emptyset \leftrightarrow (\forall z', z'')\ z = z' \circ z'' \rightarrow z' = \emptyset \vee z'' = \emptyset]$$

  - a property known as Levi's lemma from monoid theory:

$$z_1 \circ z_2 = z_3 \circ z_4 \rightarrow (\exists z_a, z_b, z_c, z_d) \left[\begin{array}{l} z_1 = z_a \circ z_b \wedge z_2 = z_c \circ z_d \wedge \\ z_3 = z_a \circ z_c \wedge z_4 = z_b \circ z_d \end{array} \right]$$

  - an induction axiom:

$$(\forall P)\ \left[P(\emptyset) \wedge (\forall g, z)\ [P(z) \rightarrow P(g \circ z)] \rightarrow (\forall z)\ P(z)\right]\ .$$

[1] More specifically, these axioms ensure that in every model $\mathcal{M}$ we have that STATE$^{\mathcal{M}}$ with operations $\emptyset^{\mathcal{M}}$ and $\circ^{\mathcal{M}}$ is isomorphic to the set of finite multisets over FLUENT$^{\mathcal{M}}$ with operations $\dot{\emptyset}$ and $\dot{\cup}$ denoting the empty multiset and multiset union (Störr & Thielscher 2000).

[2] Please note that $(\exists g)\ z = g$ is true iff z is contained in the sort FLUENT, which is a subsort of the sort STATE.

- $\mathcal{F}_{S_0}$ contains a single axiom $\Phi_I(state(s_0))$ of the form $state(s_0) = t$ describing the initial state, where t is a constructor state term.
- $\mathcal{F}_{ms}$ contains an axiom specifying that in each state each fluent may occur at most once:

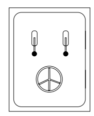
$$\mathcal{F}_{ms} = \{(\forall s,\ z)\ \neg(\exists g)\ state(s) = g \circ g \circ z\}$$

- $\mathcal{F}_{su}$ is the set of *state update axioms* of the form

$$\Delta_p(s) \wedge \bigwedge_{g \in \vartheta^-} holds(g,s) \wedge \bigwedge_{g \in \vartheta^+} \neg holds(g,s)$$
$$\rightarrow\ state(do(a,s)) \circ \vartheta^- = state(s) \circ \vartheta^+, \quad (1)$$

where ϑ^- and ϑ^+ are constructor state terms denoting the negative and positive direct effects of an action a under condition $\Delta_p(s) \in \mathcal{L}_{FC}$ respectively. $\Delta_p(s)$ is a Boolean combination of formulas of the form $holds(f,s)$. In the following we will denote the antecedent of (1) with $\Delta(s)$.

To exemplify the axiomatization consider a simple safe with two levers, the left of which has to be up and the right of which has to be down in order to open the safe.



Such a safe can be modeled with the help of the fluents l and r representing the fact that the left and the right lever are up respectively, and the fluent o representing the fact that the door is open. There are three actions: tl and tr toggle the left and right lever respectively, and op opens the safe iff the left lever is up and the right one is down. Initially, the left and the right lever are down. Is there a sequence of actions such that the safe is open after its execution? To model this scenario the axioms are instantiated as follows:

$$\mathcal{F}_{S_0} = \{state(s_0) = \emptyset\},$$
$$\mathcal{F}_{un} = \left\{ \begin{array}{l} l \neq r,\ l \neq o,\ r \neq o, \\ (\forall g)\ \ [g = l \vee g = r \vee g = o] \end{array} \right\},$$
$$\mathcal{F}_{su} =$$
$$\left\{ \begin{array}{l} holds(l,s) \rightarrow state(do(tl,s)) \circ l = state(s), \\ \neg holds(l,s) \rightarrow state(do(tl,s)) = state(s) \circ l, \\ holds(r,s) \rightarrow state(do(tr,s)) \circ r = state(s), \\ \neg holds(r,s) \rightarrow state(do(tr,s)) = state(s) \circ r, \\ holds(l,s) \wedge \neg holds(r,s) \wedge \neg holds(o,s) \rightarrow \\ \quad state(do(tr,s)) = state(s) \circ o \end{array} \right\}.$$

$\mathcal{F}_{mset}$ and $\mathcal{F}_{un}$ remain unchanged. The planning problem itself is represented by the entailment problem

$$\mathcal{F} \models (\exists s)\ holds(o,s), \quad (2)$$

which is equivalent to

$$\mathcal{F} \models (\exists z)\ (\exists s)\ z = state(s) \wedge (\exists \hat{z})\ z = o \circ \hat{z}\ .$$

The binding for s generated in a proof of (2) encodes the sequence of actions solving the planning problem.

In this paper we consider entailment problems representing propositional planning problems, i.e., entailment problems of the form

$$\mathcal{F} \models (\exists z)\,[(\exists s)\ z = state(s) \wedge \Phi_G(z)]\,, \quad (3)$$

where $\Phi_G(z)$ is a Boolean combination of terms of the form $(\exists z')$: $z = z' \circ f$ for some fluent f. In other words, we are asking if there is a situation, in which some Boolean combination of fluents holds. One should observe that $\Phi_G(z)$ is independent of $\mathcal{F}_{su} \cup \mathcal{F}_{S_0}$ because it does not contain an expression of sort SIT.

In this paper we consider a fluent calculus FC, which is restricted wrt. the general calculus as follows:

- we allow only (finitely many) constants as fluents,
- states are effectively sets of fluents due to $\mathcal{F}_{ms}$,
- the initial state is completely specified,
- the state update axioms specify only deterministic actions without ramifications or other constraints.

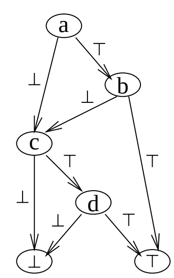
The first and the second restriction imply that there are only finitely many different states uniquely characterized by the set of fluents, which hold in each state. As we will show in this paper these four restrictions are sufficient conditions to ensure that the entailment problem is decidable.

For notational convenience we abbreviate a term $do(a_n, do(\ldots(do(a_1, S_0)\ldots))$ to $a_n \ldots a_1 S_0$. Furthermore, $\mathcal{F}_{eq} = \mathcal{F}_{mset} \cup \mathcal{F}_{ms} \cup \mathcal{F}_{un}$.

Binary Decision Diagrams

The idea of BDDs is similar to decision trees: a Boolean function is represented as a rooted acyclic directed graph. The difference to decision trees is that there is a fixed order of the occurrences of variables in each branch of the diagram, and that isomorphic substructures of the diagram are represented only once.[3] This can lead to exponential savings in space in comparison to representations like decision trees or disjunctive or conjunctive normal form.

We will introduce BDDs via an example. A formal treatment of BDDs is out of the scope of this paper and we refer the interested reader to the literature (see e.g. (Bryant 1986)). Consider the propositional logic formula $(a \wedge b) \vee (c \wedge d)$. Using the variable ordering $a < b < c < d$ a BDD representation of this formula is given in the figure to the right.



For a given valuation of the propositional variables a, b, c and d the value of the Boolean function represented by the BDD is obtained by traversing the diagram starting from the root and taking at each node the edge labeled with the value of the variable occurring in the node. The label of the terminal node defines the value of the function under the current valuation. For example $\langle a \mapsto \bot,\ b \mapsto \bot,\ c \mapsto \top, d \mapsto \bot\rangle$ leads to a node labeled $\bot$, i.e., the value of the formula is $\bot$ wrt. this valuation.

[3] Thus, the BDD is *ordered* and *reduced*, also called ROBDD. These properties are so useful that they are required in almost all BDD applications, so many authors include these properties into the definition of BDDs.

Bryant has shown in (Bryant 1986) that, given a fixed variable order, every Boolean function is represented by exactly one BDD. Moreover, propositional satisfiability, validity and equivalence problems are decidable over BDDs in linear or constant time. Of course, the complexity of the mentioned problems does not go away: the effort has been moved to the construction of the BDDs. But as Bryant has shown as well, there are efficient algorithms for logical operations, substitutions, restrictions etc. on BDDs, whose cost is in most cases proportional to the size of its operands. BDDs may be used as a theorem prover, i.e., by construction of a BDD corresponding to a logical formula, and check the BDD for interesting properties, but more often they are used as an implementation tool for algorithms which are semantically based on Boolean functions or, equivalently, propositional formulas, or, via the characteristic functions, sets. In the implementation these formulas or sets are always represented as BDDs. The use of BDDs in this paper follows this spirit.

Mapping FC onto Propositional Logic

The envisioned implementation will recursively generate sets of states which are reachable from an initial state by applying actions until one of these states satisfies the goal condition. This two–step behavior is already reflected in (3): The first conjunct expresses the fact that we are looking for a state z such that z is obtained from $state(S_0)$ by applying state update axioms, whereas the second conjunct expresses the fact that in z certain fluents should or should not hold. We start with the first step and are aiming at finding a propositional logic characterization of $\mathcal{F} \models (\exists s)\ z = state(s)$. We will define a relation $T(z, z')$ which holds iff the state z' is a successor state of z wrt the state update axioms.[4] Moreover, this transformation will enable us to encode the reasoning process into propositional logic.

Note that after expanding the macro *holds* the precondition $\Delta(s)$ of each state update axiom

$$\Delta(s) \rightarrow state(do(a,s)) \circ \vartheta^- = state(s) \circ \vartheta^+ \qquad (4)$$

in $\mathcal{F}_{su}$ effectively depends on $state(s)$, and this term contains the only occurrences of terms of sort SIT occurring in the precondition of (4). To explicitly express this dependence we will write $\Delta(state(s))$ instead of $\Delta(s)$. Making use of this notation the expression $\Delta(z)$ denotes the formula Δ, where each occurrence of $state(s)$ has been replaced by z.

For each state update axiom $\phi(a)$ of the form (4) we define

$$T_{\phi(a)}(z, z') = [\Delta(z) \wedge z' \circ \vartheta^- = z \circ \vartheta^+], \qquad (5)$$

and for the set $\mathcal{F}_{su}$ of state update axioms we define

$$T(z, z') = \bigvee_{\phi(a) \in \mathcal{F}_{su}} T_{\phi(a)}(z, z'). \qquad (6)$$

This definition is motivated by the following result:

[4]This corresponds to the transition relation in finite state systems.

Lemma 1 *Let t and t' be two constructor state terms and $\mathcal{F} \models \mathrm{state}(s) = t$. $\mathcal{F} \models \mathrm{state}(\mathrm{do}(a,s)) = t'$ iff $\mathcal{F}_{eq} \models T_{\phi(a)}(t, t')$ for some $\phi(a) \in \mathcal{F}_{su}$.*

Sketch of Proof For two constructor state terms t and t' with $\mathcal{F} \models state(s) = t$ we have that $\mathcal{F} \models state(do(a,s)) = t'$ iff there is a state update axiom $\phi(a)$ which gives a reason for this. The latter fact is equivalent to $\mathcal{F}_{eq} \models T_{\phi(a)}(t, t')$. [5] □

A binding of the form z/t, where t is a constructor state term is called *constructor state binding*. A substitution consisting only of constructor state bindings is called *constructor state substitution*. In the sequel, σ will always denote a constructor state substitution.

Our task is to encode entailment problems in FC into satisfiability problems in propositional logic. On the first glance this seems to be impossible, because there are infinitely many terms of the sorts SIT and FLUENT, whereas the set of valuations of a finite propositional program is finite. Fortunately, however, we are primarily interested in logic consequences of the form $(\exists s)\ state(s) = z$ in which the only free variable z is of type STATE. From axiom $\mathcal{F}_{ms}$ we know that the values for z may contain each fluent at most once. Since there are only finitely many fluents, the set of possible bindings for z is also finite.

More precisely, we want to show that whenever σ is an answer substitution for the entailment problem

$$\mathcal{F} \models ((\exists s)\ [z = state(s) \wedge \Phi_G(z)])\,\sigma,$$

then there exists a propositional valuation $\mathcal{B}_S(\sigma)$ which is a model for an appropriately generated propositional logic formula $\mathcal{B}((\exists s)\ [z = state(s) \wedge \Phi_G(z)])$. The mapping $\mathcal{B}$ depends on state terms, $\Phi_I(z)$, $T(z, z')$ and $\Phi_G(z)$ and is recursively defined in the sequel.

The basic idea underlying $\mathcal{B}_S$ is as follows. Suppose $\Sigma_{Fl} = \{f_1, \ldots, f_m\}$. Each variable z occurring in a constructor state binding $z/t \in \sigma$ is represented by m propositional variables $z_{f_1}, \ldots, z_{f_m}$ such that $\mathcal{B}_S(\sigma)(z_{f_i}) = \top$ iff f_i occurs in t. The precise definition of $\mathcal{B}_S$ is motivated by lemma 2.

We turn now to the formal definitions:

Ground constructor substitutions Let z/t be a constructor state binding and σ a constructor state substitution. Then $\mathcal{B}_S(z/t)$ is the valuation defined by

$$\mathcal{B}_S(z/t)(z_f) = \top \text{ iff } f \text{ occurs in } t,$$

for all $f \in \Sigma_{Fl}$, and

$$\mathcal{B}_S(\sigma) = \bigcup_{z/t \in \sigma} \mathcal{B}_S(z/t).$$

[5]The proof is omitted for space reasons. The full proof can be found in (Hölldobler & Störr 1999).

State terms Let $\oplus$ denote the exclusive or. For each $f \in \Sigma_{Fl}$ we define $\mathcal{B}_f$:

$$\begin{array}{lcl} \mathcal{B}_f(\emptyset) & = & \perp \\ \mathcal{B}_f(f') & = & \top \text{ iff } f = f' \\ \mathcal{B}_f(z) & = & z_f \\ \mathcal{B}_f(t_1 \circ t_2) & = & \mathcal{B}_f(t_1) \oplus \mathcal{B}_f(t_2) \end{array}$$

Goal formulas Recall that each goal formula $\Phi_G(z)$ is a Boolean combination of formulas of the form $(\exists z')\ z' = z \circ f$. Each Boolean operation can be represented by means of the operators $\wedge$ and $\neg$. Thus, we define

$$\begin{array}{lcl} \mathcal{B}_G((\exists z')\ z = z' \circ f) & = & z_f, \\ \mathcal{B}_G(\neg F) & = & \neg \mathcal{B}_G(F), \\ \mathcal{B}_G(F \wedge G) & = & \mathcal{B}_G(F) \wedge \mathcal{B}_G(G). \end{array}$$

F formulas In the proof of the main Theorem 3 we need to apply $\mathcal{B}$ to formulas of a certain form involving $T(z, z')$. Let $\mathbf{F}$ be the set of formulas defined by

$z = t \in \mathbf{F}$,
if $F(z) \in \mathbf{F}$ and $T(z, z')$ as defined in (5) and (6), then $(\exists z)\ [set(z)\ \wedge\ F(z) \wedge T(z, z)] \in \mathbf{F}$.

For this class of formulas we define

$$\mathcal{B}_F(z = t) = \bigwedge\nolimits_{f \in \mathcal{F}_{Fl}} (z_f \leftrightarrow \mathcal{B}_f(t)) \quad ,$$

$$\mathcal{B}_F((\exists z)\ [set(z)\ \wedge\ F(z) \wedge T(z, z')]) = (\exists (z_f)_{f \in \Sigma_{Fl}})\ [\mathcal{B}_F(F(z)) \wedge \mathcal{B}_T(T(z, z'))],$$

where

$$\begin{array}{lcl} \mathcal{B}_T(T(z, z')) & = & \bigvee\nolimits_{\Phi(a) \in \mathcal{F}_{su}} \mathcal{B}_T(T_{\Phi(a)}(z, z')), \\ \mathcal{B}_T(T_{\Phi(a)}(z, z')) & \equiv & \mathcal{B}_T(\Delta(z) \wedge z' \circ \vartheta^- = z \circ \vartheta^+) \\ & = & \mathcal{B}_G(\Delta(z)) \wedge \bigwedge\nolimits_{f \in \Sigma_{Fl}} \\ & & (\mathcal{B}_f(z' \circ \vartheta^-) \leftrightarrow \mathcal{B}_f(z \circ \vartheta^+)), \\ (\exists (z_f)_{f \in \Sigma_{Fl}}) F & = & (\exists z_{f_1}) \ldots (\exists z_{f_m}) F, \\ (\exists z_f) F & = & F[z_f/\top] \vee F[z_f/\perp], \\ \Sigma_{Fl} & = & \{f_1, \ldots, f_m\}, \end{array}$$

F denotes a propositional logic formula, $F[z_f/\top]$ and $F[z_f/\perp]$ denote the formulas obtained from F by replacing all occurrences of z_f in F by $\top$ and $\perp$ respectively.

Initial state Recall that the initial state is characterized by a formula $state(s_0) = t$.

$$\mathcal{B}_I(state(s_0) = t) = \bigwedge_{f \text{ occurs in } t} z_f \ \wedge \bigwedge_{f \text{ does not occur in } t} \neg z_f$$

In the sequel we will omit the index associated with $\mathcal{B}$ if it can be determined from the context to which class of syntactic objects $\mathcal{B}$ is applied.

Lemma 2 *Let F be either $\Phi_I(z)$, $\Phi_G(z)$ or an $\mathbf{F}$ formula and σ a constructor state substitution such that $F\sigma$ does not contain any free variables. Then,*

$$\mathcal{F}_{eq} \models F\sigma \quad \textit{iff} \quad \mathcal{B}(\sigma) \models \mathcal{B}(F).$$

The proof is done by induction over the structure of F . It is omitted for space reasons; the full proof is presented in (Hölldobler & Störr 1999).

Lemma 2 provides a technique for transforming a restricted subset of fluent calculus formulas (which includes $T(z, z')$) into satisfiability–equivalent propositional formulas. This is the base to transform planning problem specified in FC into propositional logic. The steps of this transformation are described in the proof of the following main theorem.

Theorem 3 *The entailment problem in FC can be mapped onto the satisfiability problem in propositional logic.*

Proof Consider the entailment problem in the FC:

$$\mathcal{F} \models (\exists s, z)\ [z = state(s) \wedge \Phi_G(z)].$$

Because of $\mathcal{F}_{ms}$ this holds iff there is a constructor state substitution σ such that

$$\mathcal{F} \models (\exists s)\ [z\sigma = state(s) \wedge \Phi_G(z\sigma)],$$

or, equivalently:

$$\{\sigma \mid \mathcal{F} \models (\exists s)\ [z\sigma = state(s) \wedge \Phi_G(z\sigma)]\} \neq \emptyset. \qquad (7)$$

Because conjunction can be mapped onto set intersection and $\Phi_G(z\sigma)$ does not depend on $\mathcal{F}_{su}$ and $\mathcal{F}_{S_0}$ (7) is equivalent to

$$\{\sigma \mid \mathcal{F} \models (\exists s)\ z\sigma = state(s)\} \cap \mathcal{G} \neq \emptyset, \qquad (8)$$

where $\mathcal{G} = \{\sigma \mid \mathcal{F}_{eq} \models \Phi_G(z\sigma)\}$. Let m be the number of fluent constants. Because of axiom $\mathcal{F}_{ms}$ we have at most 2^m different states and because preconditions and effects of actions depend only on the current state, the length of the shortest plan can be at most 2^m (such that every state is visited once). Thus (8) is equivalent to

$$\left\{\sigma \mid \mathcal{F} \models \bigvee_{n=0}^{2^m} (\exists (a_i)_{1 \leq i \leq n})\ z\sigma = state(a_n \ldots a_1 S_0)\right\} \cap \mathcal{G} \neq \emptyset, \qquad (9)$$

where $(a_i)_{1 \leq i \leq n}$ denotes a sequence of actions of length n . Because disjunction can be mapped onto set union (9) is equivalent to

$$\bigcup_{n=0}^{2^m} \mathcal{Z}_n \cap \mathcal{G} \neq \emptyset, \qquad (10)$$

where

$$\mathcal{Z}_n = \{\sigma \mid \mathcal{F} \models (\exists (a_i)_{1 \leq i \leq n})\ z\sigma = state(a_n \ldots a_1 S_0)\}. \qquad (11)$$

Because $state(S_0)$ depends only on $\mathcal{F}_{S_0} = \{\Phi_I(state(s_0))\}$ and by Lemma 1 equation (11) can be computed recursively by

$$\begin{array}{lcll} \mathcal{Z}_0 & = & \{\sigma \mid \mathcal{F}_{eq} \models \Phi_I(z\sigma)\}, & (12) \\ \mathcal{Z}_{n+1} & = & \{\sigma \mid \mathcal{F}_{eq} \models T(z\hat{\sigma}, z\sigma), \hat{\sigma} \in \mathcal{Z}_n\}. & (13) \end{array}$$

With

$$\begin{aligned} \mathbf{Z}_0(z) &= \Phi_I(z), & (14)\\ \mathbf{Z}_{n+1}(z) &= (\exists \hat{z})\ [set(\hat{z}) \wedge \mathbf{Z}_n(\hat{z}) \wedge T(\hat{z}, z)], & (15) \end{aligned}$$

(12) and (13) can be equivalently combined to

$$\mathcal{Z}_n = \{\sigma \mid \mathcal{F}_{eq} \models \mathbf{Z}_n(z\sigma)\}, n \geq 0. \qquad (16)$$

From Lemma 2 we conclude that (16) is equivalent to

$$\mathcal{Z}_n = \{\sigma \mid \mathcal{B}(\sigma) \models \mathcal{B}(\mathbf{Z}_n(z))\}, n \geq 0. \qquad (17)$$

Finally, an application of Lemma 2 to $\mathcal{G}$ guarantees that (10) is equivalent to

$$\bigcup_{n=0}^{2^m} \mathcal{Z}_n \cap \{\sigma \mid \mathcal{B}(\sigma) \models \mathcal{B}(\Phi_G(z))\} \neq \emptyset,$$

where $\mathcal{Z}_n$ is specified in (17). This, however, is equivalent to

$$\Big\{\sigma \mid \mathcal{B}(\sigma) \models \Big(\bigvee_{n=0}^{2^m} \mathcal{B}(\mathbf{Z}_n)\Big) \wedge \mathcal{B}(\Phi_G(z))\Big\} \neq \emptyset \ , \qquad (18)$$

where $\mathbf{Z}_n$ is specified in (14) and (15). □

The following corollary is an immediate consequence of Theorem 3 and the decidability of propositional logic.

Corollary 4 *The entailment problem in FC is decidable.*

Plan Extraction

In practical applications of planning it is not only relevant whether a plan solving the problem exists, but in most cases one would like to know how such a plan looks like. As it turns out, it is possible to extend the decision procedure presented in the previous section such that a plan can be recovered. In fact, the extended algorithm returns always the shortest plan.

The main idea for extracting the plan is the following: In the decision procedure presented in the previous section we calculate the sets $\mathcal{Z}_i$ which characterize the states reachable from the initial state after i actions. Thus, if

$$\mathcal{Z}_i \cap \{\sigma \mid \mathcal{B}(\sigma) \models \mathcal{B}(\Phi_G(z))\} \neq \emptyset,$$

i.e., if the $\mathcal{Z}_i$ contains a goal state, then there must be a plan of length i. We can now reconstruct such a plan step by step by taking a substitution σ (characterizing a state $z\sigma$) from the intersection, computing the intersection of the set of states from which this state may be reached and $\mathcal{Z}_{i-1}$, and repeating this process until eventually initial state is encountered. Thus, we find a sequence $\sigma_0, \ldots, \sigma_n$ of substitutions representing the states $z\sigma_0, \ldots, z\sigma_n$, where the first one is the initial state specified by $\mathcal{F}_{S_0}$, the last one fulfills the goal $\Phi_G(z\sigma_n)$ and $z\sigma_{i+1}$, $0 \leq i < n$, is reachable from the previous state z_i by executing an action. The final step is to find actions which transform each $z\sigma_i$ to $z\sigma_{i+1}$ by finding a state update axiom $\phi(a)$ such that $\mathcal{F}_{un} \cup \mathcal{F}_{mset} \models T_{\phi(a)}(z\sigma_i, z\sigma_{i+1})$.

Algorithm 5 *Let $\mathcal{Z}_i$, $0 \leq i \leq 2^m$, be the sets computed by equation (17) such that (18) is fulfilled. Take the smallest n such that*

$$\mathcal{Z}_n \cap \{\sigma \mid \mathcal{B}(\sigma) \models \mathcal{B}(\Phi_G(z))\} \neq \emptyset \qquad (19)$$

and compute a sequence $\sigma_0, \ldots, \sigma_n$ of substitutions and a sequence $a_1, \ldots, a_n$ of actions such that

$$\begin{aligned} \sigma_n &\in \mathcal{Z}_n \cap \{\sigma \mid \mathcal{B}(\sigma) \models \mathcal{B}(\Phi_G(z))\}\\ \sigma_{i-1} &\in \mathcal{Z}_{i-1} \cap \{\sigma \mid \mathcal{B}(\sigma) \models \mathcal{B}(T(z, z\sigma_i))\}\\ a_i &\quad \text{such that} \quad \mathcal{B}(T_{\phi(a_i)}(z\sigma_{i-1}, z\sigma_i)) = \top \end{aligned}$$

Theorem 6 *Algorithm 5 is correct and complete.*

Sketch of Proof We have to prove that the algorithm returns a result if there is a plan and that the result is indeed a plan of shortest length. The proof is by induction on the steps of the algorithm. That it is indeed the shortest plan is guaranteed by the fact that Algorithm 5 selects the smallest n satisfying equation (19).[6] □

An Implementation using BDDs — Preliminary Results

Given the theoretical results presented in the previous two sections, we can now use BDDs as the actual inference engine for the planning process. The implementation closely follows the structure of the constructions in the proofs. Starting from a fluent calculus description of the planning problem or, likewise, from a PDDL-description, the inference engine constructs for each action a the BDD–representation for $\mathcal{B}(T_{\phi(a)}(z, z'))$ and computes their disjunction $\mathcal{B}(T(z, z'))$. The BDDs of the formulas $\mathcal{B}(\mathbf{Z}_i(z))$ are computed iteratively by the following equations, which follow directly from (14) and (15):

$$\begin{aligned} \mathcal{B}(\mathbf{Z}_0(z)) &= \mathcal{B}(\Phi_I(z))\\ \mathcal{B}(\mathbf{Z}_{n+1}(z)) &= \mathcal{B}((\exists \hat{z})\ [set(\hat{z}) \wedge \mathbf{Z}_n(\hat{z}) \wedge T(\hat{z}, z)]) \end{aligned}$$

Finally, a satisfiability test corresponding to (18) decides whether the planning problem has a solution. Similarly, algorithm 5 can be implemented using BDDs.

There are a number of possible optimizations for this algorithm and we have just started to experiment with these: First, it suffices to compute $\mathcal{B}(\mathbf{Z}_i(z))$ until it becomes stationary, oscillates or until $\mathcal{B}(\mathbf{Z}_i(z)) \wedge \mathcal{B}(\Phi_G(z\sigma)) = \top$. Second, a variety of techniques developed for similar algorithms may be applied. We developed a domain independent variable ordering principle called *sort ordering*, which, depending on the problem, resulted in improvements of sometimes two orders of magnitude in computation speed. Partitioning of the transition relation $\mathcal{B}(T(\hat{z}, z))$ did not change calculation times much, but resulted in sometimes quite large reductions of memory usage on some problems (up to 64 fold). *Frontier simplification* (Clarke, Grunberg, & Long 1994; Burch *et al.* 1992) has resulted in moderate

[6] The full proof can be found in (Hölldobler & Störr 1999).

improvements in terms of calculation time. Other techniques such as ignoring fluents and actions which seem irrelevant will be tested in the near future.

The implementation was tested using the problems of the planning contest at AIPS98 and we have received mixed results so far. In the problem class called *Gripper* our planner performed extremely well: it was able to solve even the most difficult problems, whereas the planners which have participated in the competition were only able to solve but the simplest problems. In other problem classes, however, our implementation did not outperform existing planning algorithms yet. Nevertheless, having just started to investigate optimization techniques it is too early to relate our BDD implementation of planning to existing planners.

A more detailed discussion of our findings can be found in a companion paper (Hölldobler & Störr 2000).

Discussion

In this paper we have formally specified a mapping from entailment problems in the fluent calculus FC to satisfiability problems in propositional logic, and we have reported on preliminary findings of an implementation using BDDs. At present, our algorithm is closely related to model checking algorithms (Burch *et al.* 1992) which perform symbolic breadth first search in the state space. It generates a series $(\mathbf{Z}_i)_{i=0,\ldots}$ of propositional formulas represented as BDDs, which encode the set of states reachable after the execution of i actions (or a suitable subset which contains all states which don't occur at lower levels of the search tree), until there is a goal state among the states encoded. But we anticipate deviations from that pattern by imposing restrictions on fluents and actions taken into account (similar to abstraction in planning, e.g. (Knoblock 1994)).

The algorithm is similar to Graphplan (Blum & Furst 1997) in that it builds up a data structure for each level, which describes the states reachable after the execution of n actions. Unlike Graphplan, that gives only an upper bound of the the set of states reachable by its mutex mechanism, our algorithm computes an exact symbolic representation of this set. Consequently, the plan extraction process is deterministic and no backtracking is needed.

Our approach always generates shortest plans, and is able to prove that there is no plan if there isn't one. In contrast to algorithms based on planning as satisfiability (Kautz & Selman 1996) and Graphplan the algorithm presented here is not limited to the generation of polynomial length plans. On the other hand, each step of the algorithm may take exponential space, because the maximum size of BDDs is $O(2^n)$ for n propositional variables. However, the experimental results achieved so far indicate that in practice the BDDs are much smaller than the theoretical limit.

Still, the size of the encountered BDDs is the main problem limiting the scalability of the algorithm and is an topic of further research. Since the maximum size of BDDs is exponential in the number of propositional variables, the reduction of this number is a foremost concern. In contrast to SATPLAN-like approaches, it is not necessary to unfold the time steps of the plan, since all time steps are treated separately. Moreover we are able to omit the variables encoding actions easily, since we are not restricted to a clausal form of the formulas we are working with, and the actions can be reconstructed from the sequence of states. The encoding we use at present is "naive" in the sense that each fluent corresponds to a single propositional variable. We assume that the use of domain dependent properties of fluents provides a large space for improvements, as discussed in (Edelkamp & Helmert 1999) for the BDD based planning system *Mips*, which is used to explore automated generation of efficient state encodings for STRIPS/ADL/PDDL planning problems and the implementation of heuristic search algorithms with BDDs.

Depending on the task, it seems to be inevitable to encode the actions in the case of nondeterministic domains, as in the work of (Cimatti, Roveri, & Traverso 1998). Their system generates so–called *universal* plans, which consist of a state–action table that contains for each state the action, which leads to the goal in the shortest way. This approach opens new possibilities in generation of plans for non–deterministic domains. However, considering the case of deterministic domains, we conjecture, that this approach is limited to less complex reasoning problems in comparison to state–only encodings, because, additional to the states before and after the execution, the executed actions have to be encoded into the transition relation as well. This leads to a considerable increase in the number of propositional variables and, consequently, in the maximal size of the BDDs. But we have not yet performed direct comparisons to bolster this conjecture.

Our translation is tailored to a specific class of fluent calculus formulas, which is just large enough to specify the considered class of planning problems. However, it seems likely that there is a more general way to translate the formulas of a larger fragment of the fluent calculus while keeping the restriction to propositional fluents, such that we could introduce recent work on the fluent calculus like ramification (Thielscher 1998a; 1998b) into our planner without modifying the translation and the proofs. The concept of ramification within the fluent calculus involves a limited use of constructs of second order logic, namely a calculation of the transitive closure of a relation over states, but this does not seem to pose a difficult problem as the set of states is finite and there are algorithms to compute this transitive closure using BDDs (Clarke, Grunberg, & Long 1994).

A recent result on a decidable fragment of a (second–order) situation calculus (Ternovskaia 1999) also indicates that limited forms of second–order formulas do not lead to undecidablity of the entailment problem, which is an obvious precondition for envisioning a BDD–based implementation of an algorithm solving these problems. On the other hand, as soon as a finite set of unary function symbols or two binary function symbols are used in defining fluents, then the entailment problem in the fluent and the situation calculus becomes undecidable (Hölldobler 1999), and hence these extended fragments can no longer mapped onto propositional satisfiability problems.

Acknowledgement

We benefitted a lot from discussions with Sven–Erik Bornscheuer and Enno Sandner.

References

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

Bryant, R. E. 1986. Graph–based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 8(C-35):677–691.

Burch, J.; Clarke, E.; McMillan, K.; and Dill, D. 1992. Symbolic model checking: 10^{20} states and beyond. *Information and Computation* 98(2):142–170.

Burch, J. R.; Clarke, E. M.; Long, D. E.; McMillan, K. L.; and Dill, D. L. 1994. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits* 13(4):401–424.

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. "Planning via model checking: A decision procedure for AR. In Steel, S., and Alami, R., eds., *Proceedings of the Fourth European Conference on Planning (ECP97)*, LNAI 1348, 130–142.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Automatic OBDD-based generation of universal plans on non–deterministic domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI98)*.

Clarke, E.; Grunberg, O.; and Long, D. 1994. Model checking. In *Proceedings of the International Summer School on Deductive Program Design*.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *ECP'99*, LNAI, 135–147. Durham: Springer.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. The planning domain definition language. ftp://ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz.

Green, C. 1969. Application of theorem proving to problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 219–239. Morgan Kaufmann.

Hölldobler, S., and Schneeberger, J. 1990. A new deductive approach to planning. *New Generation Computing* 8:225–244. A short version appeared in the Proceedings of the German Workshop on Artificial Intelligence, Informatik Fachberichte *216*, pages 63-73, 1989.

Hölldobler, S., and Störr, H.-P. 1999. Solving the entailment problem in the fluent calculus using binary decision diagrams. Technical Report WV-99-05, AI Institute, Computer Science Department, Dresden University of Technology. http://pikas.inf.tu-dresden.de/publikationen/TR/1999/wv-99-05.ps.

Hölldobler, S., and Störr, H.-P. 2000. Bdd-based reasoning in the fluent calculus – first results. 8th. Intl. Workshop on Non-Monotonic Reasoning (NMR'2000).

Hölldobler, S. 1999. The undecidability of the entailment problem in the fluent and the situation calculus. Technical Report WV-99-02, Artificial Intelligence Institute, Computer Science Department, Dresden University of Technology.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2).

McCarthy, J. 1963. Situations and actions and causal laws. Stanford Artificial Intelligence Project: Memo 2.

Niemelä, I., and Simons, P. 1997. Smodels — an implementation of the well–founded and stable model semantics. In *Proceedings of the 4th International Conderence on Logic Programming and Non–monotonic Reasoning*, 420–429.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, 440–446.

Störr, H.-P., and Thielscher, M. 2000. A new equational foundation for the fluent calculus. (submitted).

Störr, H.-P. 2000. *Planning with BDDs*. Ph.D. Dissertation, Dresden. (in preparation).

Ternovskaia, E. 1999. Automata theory for reasoning about actions. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 153–158. Stockholm, Sweden: Morgan Kaufmann Publishers, San Francisco.

Thielscher, M. 1998a. Introduction to the fluent calculus. *Electronic Transactions on Artificial Intelligence* 2(3-4):179–192.

Thielscher, M. 1998b. Reasoning about actions: Steady versus stabilizing state constraints. *Artificial Intelligence* 104:339–355.