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. It is rigorously shown how planning problems encoded as a class of entailment problems in the fluent calculus can be mapped onto satisfiability problems for propositional formulas, which in turn can be mapped to the problem of finding models using binary decision diagrams (BDDs). The mapping is shown to be sound and complete. First experimental results of an implementation are presented and discussed.

1 Introduction

In recent years propositional methods have seen a surprising revival in the field of Intellectics. Greedy satisfiability testing and its variants [17] and the various procedures for answer set computing (e.g. [16, 9]) are just two examples. 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 [5, 6, 8]. This comes to a surprise because model checking using BDDs has significantly improved the performance of algorithms and enabled the solution of new classes of problems in areas like formal verification and logic synthesis (see e.g. [3, 4]). Can we adopt this technology for, say, problems occurring in reasoning about situations, actions and causality? Can we enrich these techniques by exploiting the experiences made in the state of the art implementations of propositional logic calculi and systems mentioned at the beginning of this paragraph?

This paper reports on an attempt to find answers for these and related questions in the context of the fluent calculus. The fluent calculus is a formal system for reasoning about situations, actions and causality which admits a well–defined semantics as given in [11] and [20]. In Section 2 a restricted fragment of the fluent calculus is considered, which allows for the specification of planning problems as entailment problems in the spirit of [14]. In Section 3 a transformation is formally defined which maps these entailment problems onto satisfiability problems in propositional logic. The mapping is shown to be sound and complete. Thus, the decidability of the abovementioned fragment of the fluent calculus is established. In Section 4 it is shown how the shortest plan solving the given planning problem can be extracted from the propositional encoding. Finally, in Section 5 first and promising findings of an implementation using BDDs are presented. In

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

Table 1. Notational conventions.

this implementation, the propositional logic formulas are represented by reduced and ordered BDDs and techniques from model checking are applied to search for models. A discussion of the achieved results in Section 6 concludes the paper. Due to lack of space most of the proofs had to be omitted. They can be found in detail in [12].

2 Foundations

In this section some notions and notations concerning logics, planning problems, the fluent calculus and binary decision diagrams are presented.

2.1 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 z, we implicitly assume $z \in \Sigma_{V,St}$. The sets Σ_{Fl}, $\Sigma_{V,St}$, $\Sigma_{V,Fl}$, $\Sigma_{V,Sit}$ as well as constructor state terms are defined in Section 2.3. All symbols are possibly indexed.

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.

2.2 Planning

In this paper we consider planning problems having the following properties: (i) The set of states is characterized by a set of propositional fluents, i.e., a set of propositional variables, which can take values out of the set $\{\top, \bot\}$ of truth values. (ii) The actions are deterministic and their preconditions as well as effects depend only on the state they are executed in. (iii) The goal of the planning problem is a property which depends solely on the reached state. This class of problems corresponds roughly to the problems from track 1 and 2 of the planning competition held at the 4th International Conference on Artificial Planning Systems (AIPS98). There, planning problems were formulated within a language called PDDL [10]. Unfortunately, PDDL lacks a formal semantics. As shown in [18] this can be rectified by a translation from PDDL into the fluent calculus.

As an example of such a problem consider the so–called GRIPPER class: *A robot equipped with two grippers G_1 and G_2 can move between two rooms A and B. Initially the robot is in room A together with a number of balls $B_1, \ldots, B_n$. The task is to transport these balls into room B.* The problems differ wrt the number of balls and are then called GRIPPER–1, GRIPPER–2.

2.3 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 [11, 19, 20] but is not idempotent. In this paper we consider a restricted version of the calculus as specified in this section.

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 $\Sigma_A \cup \Sigma_{Sit} \cup \Sigma_{Fl} \cup \Sigma_{St} \cup \Sigma_O$, 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:

S_0 : SIT $\quad$ $\circ$: STATE $\times$ STATE $\to$ STATE $\quad$ $\emptyset$: STATE
do : ACTION $\times$ SIT $\to$ SIT $\quad$ $state$: SIT $\to$ STATE

The set Σ_P of predicate symbols contains only the equality $=$ with sort STATE $\times$ STATE. The macros *holds* and *set* with sorts FLUENT $\times$ SIT and STATE respectively are often used:

$$holds(f, s) \stackrel{def}{=} (\exists \hat{z})\ state(s) = f \circ \hat{z} \qquad (1)$$
$$set(z) \stackrel{def}{=} \neg(\exists f, \hat{z})\ z = f \circ f \circ \hat{z}$$

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, $m \geq 0$, are pairwise distinct, 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 finite multisets of fluents[19]. It consists of the following formulas:
  - the standard axioms of equality
  - the axioms AC1 for $\circ$ and $\emptyset$:

$$(\forall z)\ z \circ \emptyset = z$$
$$(\forall z_1,\ z_2)\ z_1 \circ z_2 = z_2 \circ z_1$$
$$(\forall z_1,\ z_2,\ z_3)\ (z_1 \circ z_2) \circ z_3 = z_1 \circ (z_2 \circ z_3)$$

  - an axiom that guarantees that fluents and $\emptyset$ are the only irreducible elements of sort STATE wrt. $\circ$:

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

  - a property called Levi's axiom after a lemma used in the theory of trace monoids:

$$(\forall z_1, z_2, z_3, z_4)\ z_1 \circ z_2 = z_3 \circ z_4 \ \rightarrow\ (\exists z_a, z_b, z_c, z_d)$$
$$[z_1 = z_a \circ z_b \ \wedge\ z_2 = z_c \circ z_d \ \wedge\ z_3 = z_a \circ z_c \ \wedge\ z_4 = z_b \circ z_d]$$

  - an induction axiom:

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

– $\mathcal{F}_{S_0}$ 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 a set of *state update axioms* of the form

$$(\forall) \left[\begin{array}{l} \Delta_p(s) \ \wedge \bigwedge\limits_{g \in \vartheta^-} holds(g, s) \ \wedge \bigwedge\limits_{g \in \vartheta^+} \neg holds(g, s) \\ \rightarrow\ state(do(a, s)) \circ \vartheta^- = state(s) \circ \vartheta^+ \end{array}\right] , \qquad (2)$$

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, $s \in \Sigma_{V,S}$ and $(\forall)$ denotes the universal closure. $\Delta_p(s)$ is a boolean combination of formulas of the form $holds(f, s)$. In the following $\Delta(s)$ will be used to denote the antecedent of (2).

To exemplify $\mathcal{F}_{su}$ and $\mathcal{F}_{S_0}$ consider the GRIPPER class. There are three actions: (i) The robot may *move* from one room to the other. (ii) The robot may *pick* up a ball if it is in the same room as the ball and one of its grippers

is empty. (iii) the robot may *drop* a ball if it is carrying one. These actions are specified by the state update axioms:

$$\begin{aligned}\mathcal{F}_{su} = \{\ & holds(at\text{-}robby(r_1), s) \wedge \neg holds(at\text{-}robby(r_2), s) \\ & \rightarrow state(do(move(r_1, r_2)), s) \circ at\text{-}robby(r_1) = state(s) \circ at\text{-}robby(r_2) \ , \\ & holds(at(b, r), s) \wedge holds(at\text{-}robby(r), s) \wedge holds(free(g), s) \\ & \quad \wedge \neg holds(carry(b, g), s) \\ & \rightarrow state(do(pick(b, r, g)), s) \circ at(b, r) \circ free(g) = state(s) \circ carry(b, g) \ , \\ & holds(carry(b, g), s) \wedge holds(at\text{-}robby(r), s) \wedge \neg holds(at(b, r), s) \\ & \quad \wedge \neg holds(free(g), s) \\ & \rightarrow state(do(drop(b, r, g)), s) \circ carry(b, g) = state(s) \circ at(b, r) \circ free(g)\ \}\end{aligned}$$

The initial state of a GRIPPER class problem is specified by

$$\mathcal{F}_{S_0} = \{state(S_0) = at(B_1, A) \circ \ldots \circ at(B_n, A) \circ free(G_1) \circ free(G_2) \circ at\text{-}robby(A)\},$$

where n is instantiated to some number, $at(x, y)$ denotes that ball x is in room y, $free(x)$ that gripper x is free and $at\text{-}robby(x)$ that the robot is in room x.

Reasoning problems themselves are specified as entailment problems in the fluent calculus. For the GRIPPER class we obtain the entailment problem

$$\mathcal{F} \models (\exists s)\ holds(at(B_1, B), s) \wedge \ldots \wedge holds(at(B_n, B), s).$$

Expanding abbreviation (1) this can be reformulated as

$$\mathcal{F} \models (\exists s)\ \big[(\exists z)\ state(s) = at(B_1, B) \circ z \wedge \ldots \wedge (\exists z)\ state(s) = at(B_n, B) \circ z\big],$$

which itself is equivalent to

$$\begin{aligned}\mathcal{F} \models (\exists z)\ \big[&(\exists s)\ state(s) = z \wedge \\ &(\exists z')\ z = at(B_1, B) \wedge \ldots \wedge (\exists z')\ z = at(B_n, B) \circ z'\big].\end{aligned}$$

In general, reasoning in FC amounts to solving an entailment problem of the form

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

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, one is looking for 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.

The fluent calculus FC considered in this paper is restricted wrt the general calculus as follows: (i) Only constants are allowed as as fluents. (ii) States are effectively sets of fluents due to $\mathcal{F}_{ms}$. (iii) The initial state is completely specified. (iv) The state update axioms specify only deterministic actions without ramifications or other constraints. The first restriction implies that the set of fluents is finite if Σ_{Fl} is finite. The second restriction implies that there are only finitely many different states uniquely characterized by the set of fluents, which hold in each state, if Σ_{Fl} is finite. As will be shown in this paper these restrictions are sufficient conditions to ensure that the entailment problem (3) in FC is decidable.

2.4 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.[1] This can lead to exponential savings in space in comparison to representations like decision trees or disjunctive or conjunctive normal form.

Bryant has shown in [2] 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 constructing a BDD corresponding to a logical formula, and checking 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.

3 Mapping the Fluent Calculus 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. Starting with the first step and aiming at finding a propositional logic characterization of $\mathcal{F} \models (\exists s)\ z = state(s)$ a relation $\mathbf{T}(z, z')$ is defined which holds iff the state z' is a successor state of z wrt the state update axioms.[2] Moreover, this transformation allows for an encoding of the reasoning process into propositional logic.

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

$$(\forall)\ [\Delta(s) \rightarrow state(do(a, s)) \circ \vartheta^- = state(s) \circ \vartheta^+ \in \mathcal{F}_{su}] \tag{4}$$

[1] 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.

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

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

$$\mathbf{T}_{\phi(a)}(z, z') \stackrel{def}{=} \Delta(z) \wedge z' \circ \vartheta^- = z \circ \vartheta^+, \tag{5}$$

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

$$\mathbf{T}(z, z') \stackrel{def}{=} \bigvee_{\phi(a) \in \mathcal{F}_{su}} \mathbf{T}_{\phi(a)}(z, z'). \tag{6}$$

This definition is motivated by the following result.

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 $\mathcal{F}_{un} \cup \mathcal{F}_{mset} \models \mathbf{T}_{\phi(a)}(t, t')$ for some $\phi(a) \in \mathcal{F}_{ua}$.*

A binding of the form z/t, where t is a constructor ground 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.

The task to encode entailment problems in the fluent calculus into satisfiability problems in propositional logic seems to be impossible on the first glance, because there are infinitely many terms of the sorts SIT and STATE, whereas the set of valuations of a finite propositional program is finite. Fortunately, however, one is 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 $\mathcal{F}_{ms}$ one learns that the values for z may contain each fluent at most once. Because there are only finitely many fluents in FC, the set of possible bindings for z is also finite.

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

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

then there exists a propositional valuation $\mathcal{B}_S(\sigma)$ such that $\mathcal{B}_S(\sigma)$ is a model for an appropriately generated propositional logic formula. This formula is obtained by giving an equivalent representation of the entailed formula in terms of $\Phi_I(z)$, $T(z, z')$ and $\Phi_G(z)$ and specifying a mapping $\mathcal{B}$ which maps this representation to a propositional formula.

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 substitution $\sigma = \{z/t\}$ is represented by m propositional variables $z_{f_1}, \ldots, z_{f_m}$ such that in the propositional valuation $v = \mathcal{B}(\sigma)$ one obtains $v(z_{f_i}) = \top$ iff f_i occurs in t. A formula F is represented by a propositional formula $\mathcal{B}(F)$ such that a ground constructor substitution σ is an answer substitution for $\mathcal{F}_{un} \cup \mathcal{F}_{mset} \models F\sigma$ iff the valuation $\mathcal{B}_S(\sigma)$ fulfills $\mathcal{B}(F)$. We turn now to a formal definition:

Ground constructor substitutions: Let z/t be a binding of a constructor state substitution. Then $\mathcal{B}_S(z/t)$ is the valuation v defined by $v(z_f) = \top$ iff f occurs in t, for all $f \in \Sigma_{Fl}$. Let σ be a constructor state substitution. Then

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

Constructor state terms: Let $\oplus$ denote the exclusive or. For each $f \in \Sigma_{Fl}$ define: [3]

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

Goal formulas: Recall that each goal formula $\Phi_G(z)$ is a boolean combination of formulas of the form $(\exists z')\ z' = z \circ f$. Define:

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

$\mathbf{F}$ *formulas:* In the proof of the Theorem 1 $\mathcal{B}$ has to be applied to formulas of the following form. Let $\mathbf{F}$ be the set of formulas defined by (i) $z = t \in \mathbf{F}$ and (ii) if $F(z) \in \mathbf{F}$ and $\mathbf{T}(z, z')$ as defined in (6), then $(\exists z)\ [set(z) \wedge F(z) \wedge \mathbf{T}(z, z)] \in \mathbf{F}$. For this class of formulas define:

$$\begin{array}{c} \mathcal{B}_F(z = t) = \bigwedge_{f \in \mathcal{F}_{Fl}} (z_f \leftrightarrow \mathcal{B}_f(t)) \\ \mathcal{B}_F((\exists z)\ [set(z)\ \wedge\ F(z) \wedge \mathbf{T}(z, z')]) = (\exists (z_f)_{f \in \Sigma_{Fl}})\ [\mathcal{B}_F(F(z)) \wedge \mathcal{B}_T(\mathbf{T}(z, z'))], \end{array}$$

where

$$\begin{array}{rl} \mathcal{B}_T(\mathbf{T}(z, z')) & = \bigvee_{\phi(a) \in \mathcal{F}_{su}} \mathcal{B}_T(\mathbf{T}_{\phi(a)}(z, z')), \\ \mathcal{B}_T(\mathbf{T}_{\phi(a)}(z, z')) & = \mathcal{B}_G(\Delta(z)) \wedge \bigwedge_{f \in \Sigma_{Fl}} (\mathcal{B}_f(z' \circ \vartheta^-) \leftrightarrow \mathcal{B}_f(z \circ \vartheta^+)), \\ (\exists (z_f)_{f \in \Sigma_{Fl}}) F & = (\exists z_{f_1}) \ldots (\exists z_{f_m}) F \text{ and} \\ (\exists z_f) F & = F[z_f/\top] \vee F[z_f/\bot] \end{array}$$

assuming that $\mathcal{B}_T(\mathbf{T}_{\phi(a)}(z, z'))$ is defined as in (5) and $\Sigma_{Fl} = \{f_1, \ldots, f_m\}$. Furthermore, in the last equation F denotes a propositional logic formula and $F[z_f/\top]$ and $F[z_f/\bot]$ denote the formulas obtained from F by replacing all occurrences of z_f in F by $\top$ and $\bot$ respectively.

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

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

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.

[3] Thus, $\mathcal{B}(\sigma) \models \mathcal{B}_f(t\sigma)$ is true iff fluent f occurs an odd number of times in $t\sigma$.

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.* $\mathcal{F}_{un} \cup \mathcal{F}_{mset} \models F\sigma$ *iff* $\mathcal{B}(\sigma) \models \mathcal{B}(F)$.

Thus, Lemma 2 provides a way to transform a restricted subset of fluent calculus formulas (which includes $\mathbf{T}(z, z')$) into satisfiability-equivalent propositional formulas. This is the base to transform entailment problems in FC into satisfiability problems in propositional logic. The steps of this transformation are described in the proof of the following theorem.

Theorem 1. *Each entailment problem* (3) *in FC can be mapped onto a propositional satisfiability problem SAT, such that* (3) *is solvable iff SAT is solvable.*

Proof. Consider the entailment problem in the fluent calculus:

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

By $\mathcal{F}_{ms}$ this holds iff there is a constructor state substitution σ such that $\mathcal{F} \models (\exists)\ z\sigma = state(s) \wedge \Phi_G(z\sigma)]$, or, equivalently:

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

Because conjunction can be mapped onto set intersection (7) is equivalent to

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

where $\mathcal{G} = \{\sigma \mid \mathcal{F}_{mset} \cup \mathcal{F}_{un} \models \Phi_G(z\sigma)\}$. Let m be the number of fluent constants. Because of axiom $\mathcal{F}_{ms}$ there are 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

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

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

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

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

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

$$\mathcal{Z}_0 = \{\sigma \mid \mathcal{F}_{mset} \cup \mathcal{F}_{un} \models \Phi_I(z\sigma)\}, \tag{12}$$
$$\mathcal{Z}_n = \{\sigma \mid \mathcal{F}_{mset} \cup \mathcal{F}_{un} \models \mathbf{T}(z\hat{\sigma}, z\sigma), \hat{\sigma} \in \mathcal{Z}_{n-1}\},\ n > 0. \tag{13}$$

With
$$\mathbf{Z}_0(z) = \Phi_I(z), \tag{14}$$
$$\mathbf{Z}_n(z) = (\exists \hat{z})\ [set(\hat{z}) \wedge \mathbf{Z}_{n-1}(\hat{z}) \wedge \mathbf{T}(\hat{z}, z)],\ n > 0, \tag{15}$$

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

$$\mathcal{Z}_n = \{\sigma \mid \mathcal{F}_{mset} \cup \mathcal{F}_{un} \models \mathbf{Z}_n(z\sigma)\}, n \geq 0. \tag{16}$$

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

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

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

$$\bigcup_{n=0}^{2^m} \mathcal{Z}_n \cap \{\sigma \mid \mathcal{B}(\sigma) \models \mathcal{B}(\Phi_G(z\sigma))\} \neq \emptyset, \tag{18}$$

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

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

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

The following corollary is an immediate consequence of Theorem 1 and the decidability of propositional logic.

Corollary 1. *The entailment problem* (3) *in FC is decidable.*

4 Plan Extraction

In practical applications it is not only relevant whether a sequence of actions (or 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. Very pleasantly, the extended algorithm returns always the shortest plan.

The main idea for extracting the plan is the following: The sets $\mathcal{Z}_i$ constructed in the proof of Theorem 1 characterize the states reachable from the initial state after i actions. Thus, if $\mathcal{Z}_i \cap \mathcal{G} \neq \emptyset$, i.e., if $\mathcal{Z}_i$ contains a goal state, then there must be a plan of length i. The plan can now be reconstructed 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 the 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 $z\sigma_i$ 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 $\mathcal{F}_{un} \cup \mathcal{F}_{mset} \models \mathbf{T}_{\phi(a)}(z\sigma_i, z\sigma_{i+1})$.

In the implementation of the algorithm all sets and formulas are represented by in their BDD representation $\mathcal{B}$. Please note that it suffices to compute the sets $(\mathcal{Z}_i)_{i=0,1,\ldots}$ until either a solution is found or it can be determined that all

reachable states have been visited (such that the sequence becomes stationary or cyclic).

Algorithm 1. Let $\mathcal{Z}_i$, $i = 0 \leq i \leq 2^m$ be the sets computed by equation (17). If (18) is not fulfilled return "unsolvable", else take the smallest n such that

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

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

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

Then $s = a_n \ldots a_1 S_0$ corresponds to a shortest plan wrt the goal Φ_G.

Theorem 2. *Algorithm 1 is correct and complete.*

In other words, the algorithm always proves either that there is no plan or returns a shortest plan solving the problem.

5 An Implementation using BDDs — First Results

The theoretical results presented in the previous two sections can be applied to use a BDD implementation as the inference engine for solving entailment problems (3) in FC and computing plans. The implementation closely follows the structure of the constructions used in the proofs. Starting from a fluent calculus specification of the entailment problem, the inference engine constructs for each action a the BDD–representations for $\mathcal{B}(\mathbf{T}_{\phi(a)}(z, z'))$, and computes their disjunction $\mathcal{B}(\mathbf{T}(z, z'))$ by (6). The BDDs of the formulas $\mathcal{B}(\mathbf{Z}_i(z))$ are computed iteratively by application of (14) and (15) translated by $\mathcal{B}$. Thus, BDD representations of the sets $\mathcal{Z}_i$ are computed iteratively until either an i is reached such that $\mathcal{Z}_i = \mathcal{Z}_{i+1}$ or $\mathcal{Z}_i \cap \{\sigma \mid \mathcal{B}(\sigma) \models \mathcal{B}(\Phi_G(z\sigma))\} \neq \emptyset$. Similarly, Algorithm 1 can be implemented using BDDs.

This approach is an implicit[4] breadth first search. In each single step the whole breadth of the search tree in depth i is searched. The sets $\mathcal{Z}_i$ can get quite complex and their BDDs quite large. Even more so, the size of the BDD for $\mathcal{B}(\mathbf{T}(z, z'))$, can quickly become too large to be handled in a graceful manner. Thus, a number of techniques were invented to limit a potential explosion in its size. In the sequel some of these techniques and their effects are sketched using examples from [15].

[4] It is called implicit because the calculated sets of states are never explicitly enumerated, but represented as a whole by a BDD, whose size depends more on the structure of the set, than on its actual size.

Variable Order It is well known that the variable order used in a BDD has a large influence on the size of the BDD. Unfortunately it is still a difficult problem to find even an near optimal variable order.[5] Often, a good variable order is found by empiric knowledge and experimentation. In experiments it has turned out that fluents, which directly influence each other, should be grouped together. We have developed a variable ordering called *sort order*, which employs this idea by grouping fluents by their arguments, since fluents sharing arguments are likely to influence each other. In planning problems that use sorts to restrict the considered argument values for fluents, arguments belonging to large sorts are preferred in this ordering. Due to lack of space a precise definition of this heuristic can not be given here. On some problems this ordering lead to improvements of orders of magnitude in the size of the BDDs if compared to a simple lexicographical ordering, but this depends on the domain of the problem (of course).

Partitioning of the Transition Relation The maximal size of a BDD is exponential in the number of propositional variables it contains. Thus, the BDD representing $\mathcal{B}(\mathbf{T}(z, z'))$, which contains twice as many propositional variables as the BDDs representing the $\mathcal{Z}_i$, is prone to get very large. A way to reduce this problem is to divide the disjunction $\mathbf{T}(z, z')$ into several parts $\mathbf{T}_1(z, z')$, $\ldots$, $\mathbf{T}_n(z, z')$, which correspond to subsets of the state update actions. In experiments, partitioning led to a reduction in the size of the BDDs in most of the tested problems.

On the other hand, such a decrease in the size of the BDDs does not necessarily lead to a decrease in computation time. In each step, the results of applying the parts of the transition relation to the set of states reached so far have to be put together, and this takes time. Nevertheless, partitioning may be useful even if the computation time increases. In the experiments, one problem (MPRIME-X-1) could only be solved after a partitioning of the transition relation; otherwise, the memory exceeded before a solution was found.

Frontier Simplification explores the fact, that the algorithm for solving the entailment problem in the fluent calculus works also if the following two conditions are enforced for all $i \geq 0$: (i) $\mathcal{Z}_i$ represents all states which may be reached by executing i actions, but not by executing less than i actions. (ii) $\mathcal{Z}_i$ does not represent any states which cannot be reached by executing at most i actions. The sets $\mathcal{Z}_i$ can be chosen freely within these limitations. Hence, it is desirable that the algorithm chooses the $\mathcal{Z}_i$ such that their BDD representations are as small as possible. In our experiments, frontier simplification has sometimes lead to moderate improvements both in computation time and memory requirements.

To the end of this section the experimental results on the GRIPPER class are discussed. These problems were quite hard for the systems taking part in the AIPS98 competition. The difficulty is rooted in the combinatorial explosion of

[5] The problem to find the optimal variable order is NP-complete.

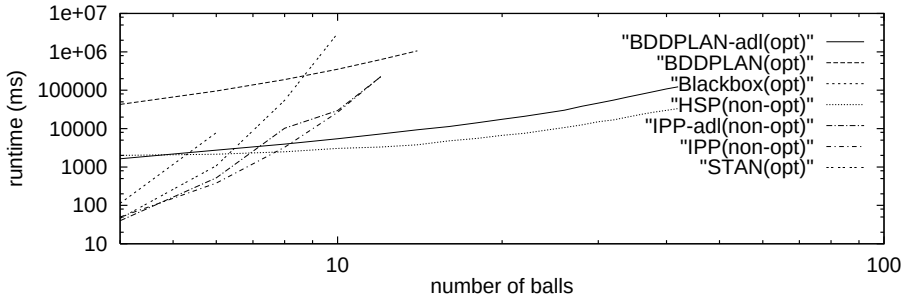

Fig. 1. Runtimes of different planners on GRIPPER class problems (in milliseconds) with different numbers of balls. Planners marked with `opt` provided optimal (i.e., shortest) plans, planners marked with `-adl` work on the sorted version of the domains, the others on the STRIPS-version.

alternatives due to the existence of two grippers. In Fig. 1 the runtimes of these planners[6] are compared to our system, BDDPLAN.[7] Only one planner (HSP) was able to solve all of the problems of this class, but it generated only suboptimal plans by using only one of the two grippers, whereas BDDPLAN generates the shortest possible plan by design.

6 Discussion

We have formally specified a mapping from entailment problems in a restricted fluent calculus to satisfiability problems in propositional logic, which is sound and complete, and we have reported some first experimental results of an implementation using BDDs. We are still in the process of investigating how optimization techniques well–known in the area of model checking using BDDs can be adapted such that they increase the efficiency of the implementation.

The mapping is tailored to a specific class of fluent calculus formulas. 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 [20, 21] into the 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 [7].

Although the problems considered in this paper admitted only a single initial state (i.e, $\mathcal{Z}_0$ is unitary), the algorithm itself is not restricted to this case. If

[6] See `http://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html`.

[7] The runtime of BDDPLAN is measured on a different machine, so it is only accurate up to a constant factor.

the initial situation is incompletely specified then there are several initial states, which leads to a set $\mathcal{Z}_0$ containing more than one element.

At present, our algorithm is closely related to model checking algorithms [3] 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 represent sets of answer substitutions that encode logical consequences of the fluent calculus specification describing the set of reachable states. This process is executed until a goal state is found or until unsatisfiability of the problem can be determined. The approach has the advantage that it always generates shortest plans, and is able to prove that there is no plan if there isn't one. Unlike planning algorithms based on planning as satisfiability [13] and Graphplan [1] the algorithm presented here is not limited to the generation of polynomial length plans. On the other hand, each time step may take space exponential space, since 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 a 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. Unlike the approach taken in [6], which explores new possibilities in the generation of plans for non–deterministic domains using BDDs, we can avoid the encoding of the actions with propositional variables in order to reduce the BDD sizes.

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 [8] 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.

To sum up, our BDD based implementation shows some promising initial results but it is too early to completely evaluate it yet.

Acknowledgment

We benefited from discussions with Sven–Erik Bornscheuer and Enno Sandner.

References

[1] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

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

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

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

[5] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. "planning via model checking: A decision procedure for ar. In S. Steel and R. Alami, editors, *Proceedings of the Fourth European Conference on Planning (ECP97)*, LNAI 1348 , pages 130–142, Toulouse, France, Sept. 1997. Springer-Verlag.

[6] Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Automatic OBDD-based generation of universal plans on non–deterministic domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI98)*, Madison, Wisconsin, July 26-30 1998.

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

[8] Stefan Edelkamp and Malte Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *ECP'99*, LNAI, pages 135–147, Durham, 1999. Springer.

[9] T. Eiter, N. Leone, C. Mateis, G. Pfeier, and F. Scarnello. The KR system DLV: Progress report, comparisons and benchmarks. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann Publishers, 1998.

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

[11] S. Hölldobler and J. Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990.

[12] S. Hölldobler and H.-P. Störr. 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, 1999.

[13] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1194–1201, Portland, Oregon, 1996. AAAI-Press.

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

[15] Drew McDermott. Planning problem repository. `ftp://ftp.cs.yale.edu/pub/mcdermott/domains/`, 1999.

[16] I. Niemelä and P. Simons. 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*, pages 420–429, 1997.

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

[18] H.-P. Störr. *Planen mit binären Entscheidungsdiagrammen.* PhD thesis, Dresden University of Technology, Department of Computer Science, 2000. (in German; to appear).

[19] H.-P. Störr and M. Thielscher. A new equational foundation for the fluent calculus. In *Proceedings CL 2000.*

[20] Michael Thielscher. Introduction to the Fluent Calculus. *Electronic Transactions on Artificial Intelligence*, 2(3–4):179–192, 1998.

[21] Michael Thielscher. Reasoning about actions: Steady versus stabilizing state constraints. *Artificial Intelligence*, 104:339–355, 1998.