FCPlanner: A Planning Strategy for First-Order MDPs

Eldar Karabaev
Institute for Theoretical Computer Science
Dresden University of Technology
Dresden, Germany
karabaev@tcs.inf.tu-dresden.de

Olga Skvortsova*
Institute for Artificial Intelligence
Dresden University of Technology
Dresden, Germany
skvortsova@inf.tu-dresden.de

Introduction

FCPLANNER (Fluent Calculus Planner) is a planning system that is based on the first-order value iteration algorithm (FOVIA) (Großmann, Hölldobler, & Skvortsova 2002) for solving first-order MDPs. Following the idea of symbolic dynamic programming (SDP) within the Situation Calculus by Boutilier and colleagues (Boutilier, Reiter, & Price 2001), FOVIA addresses the well-known scalability problem of the classical dynamic programming algorithms by employing the abstraction technique, i.e., a state space is divided into clusters, called *abstract states*, and the value functions are computed for them thereafter. The dynamics of an MDP is formalized in the probabilistic Fluent Calculus (pFC) that allows for introducing stochastic actions. Our approach constructs a first-order representation of value functions and policies by exploiting the logical structure of the MDP. Thus, FOVIA can be seen as a symbolic (logical) counterpart of classical value iteration algorithm (Bellman 1957).

Abstract States

We formalize abstract states symbolically, within the Fluent Calculus (FC) (Hölldobler & Schneeberger 1990). Fluent Calculus, much like Situation Calculus, is a logical approach to modelling dynamically changing systems based on first-order logic. One could indeed argue that Fluent Calculus and Situation Calculus have very much in common. But the latter has the following disadvantage: Knowledge of the current state is represented indirectly via the initial conditions and the actions which the agent has performed up to a point. As a consequence, each time a condition is evaluated in an agent program, the entire history of actions is involved in the computation. This requires ever increasing computational effort as the agent proceeds, so that this concept does not scale up well to long-term agent control (Thielscher 2004). Fluent Calculus overcomes the aforementioned unfolding problem by providing the crucial concept of an explicit state representation. The information on what is true in the current state of the world is effortlessly extracted from the state description without tracing back to the initial state. Therefore we have opted for Fluent Calculus as logical formalism underlying our automated symbolic dynamic programming approach.

In FC, functions whose values vary from state to state are called *fluents* and are denoted by function symbols. For example, the fluent $on(X, table)$ denotes the presence of a block X on the table. A *state* is a multiset of fluents represented as a term, called *fluent term*, using a constant 1 denoting the empty multiset and a binary function symbol $\circ$ denoting multiset union that is associative, commutative and admits unit element. For example, a state in which the block a is on the block b and b is on the table is specified by $on(a, b) \circ on(b, table)$. Constants are denoted by small letters, variables by capital ones and substitutions by θ or σ.

Abstract states are characterized by means of conditions that must hold in each ground instance thereof and, thus, they represent sets of real-world states. Informally, abstract states can be specified by stating that particular fluent terms do or do not hold. We refer to such abstract states as *CN-states*, where C stands for conjunction and N for negation, respectively.

Formally, let $\mathcal{L}$ be a set of fluent terms. A *CN-state* is a pair $(P, \mathcal{N})$, where $P \in \mathcal{L}$, $\mathcal{N} \in 2^{\mathcal{L}}$. Let $\cdot^M$ be a mapping from fluent terms to multisets of fluents, which can be formally defined as follows: $1^M = \dot{\{\}}$ or $F^M = \dot{\{F\}}$, if F is a fluent, or $(F \circ G)^M = F^M \dot{\cup} G^M$, where F, G are fluent terms and $\dot{\cup}$ is a multiset union. Let $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ be an interpretation, whose domain Δ is the set of all finite multisets of ground fluents and every *CN-state* $Z = (P, \mathcal{N})$ is mapped onto

$$Z^{\mathcal{I}} = \{d \in \Delta \mid \exists\theta.\, (P\theta)^M \,\dot{\subseteq}\, d \wedge \forall N \in \mathcal{N}.\forall\sigma.((N\theta)\sigma)^M \,\dot{\not\subseteq}\, d\}\,,$$

where $\dot{\subseteq}$ is a submultiset relation.

In other words, the P-part of a state Z describes properties that a real-world state should satisfy, whereas $\mathcal{N}$-part specifies the properties that are not allowed to fulfil. For example, the *CN-state* $Z = (on(X, table) \circ red(X), \{on(Y, X)\})$ represents all states in which there exists a red object that is on the table and clear, viz., none of other objects covers it.

Thus, the real-world state

$$z = \dot{\{}on(a, table), red(a), on(b, table), green(b)\dot{\}}$$

*Supported by the research training group GRK 334/3 (DFG).
Corresponding author.

is specified by Z. Whereas,

$$z' = \dot{\{}on(a, table), red(a), on(b, a)\dot{\}}$$

is not.

Intuitively, *CN-states* can be represented as first-order formulae. The above-given *CN-state* Z corresponds to the following formula:

$$\exists X.on(X, table) \wedge red(X) \wedge \forall Y.\neg on(Y, X) \ .$$

Please note that *CN-states* should be thought of as incomplete state descriptions, i.e., the properties that are not listed in either P- or $\mathcal{N}$-part can hold or not.

Stochastic Actions

The technique for introducing stochastic actions within the probabilistic Fluent Calculus is to decompose a stochastic action into deterministic primitives under nature's control, referred to as *nature's choices*. We use a relation symbol *choice*$/2$ to model nature's choice. Consider the action $putdown(T, B)$ of putting a block T down onto a block B from the blocksworld scenario:

$$\begin{aligned} &\textit{choice}\,(putdown(T, B), A) \leftrightarrow \\ &\qquad (A = putdown_1(T, B) \vee A = putdown_2(T, B)), \end{aligned}$$

where $putdown_1(T, B)$ and $putdown_2(T, B)$ define two nature's choices for action $putdown(T, B)$. The nature's choice $putdown_1(T, B)$ states the successful putting of the block T down onto B. Whereas, $putdown_2(T, B)$ defines the failure execution of the $putdown$-action which results in the block T falling down on the table.

For each of nature's choices $a_j(\overline{X})$ associated with an action $a(\overline{X})$ with parameters $\overline{X}$ we define the probability $\textit{prob}\,(a_j(\overline{X}), a(\overline{X}), Z)$. It denotes the probability with which one of nature's choices $a_j(\overline{X})$ is chosen in a *CN-state* Z. For example,

$$\textit{prob}\,(putdown_1(T, B), putdown(T, B), Z) = .7$$

states that the probability for the successful execution of the $putdown$ action in Z is .7.

FOVIA is an iterative approximation algorithm for constructing optimal policies. The difference to classical case is that it produces a first-order representation of optimal policies by utilizing the logical structure of MDP. The algorithm itself can be found in (Großmann, Hölldobler, & Skvortsova 2002).

Preprocessing

In order to convert a PPDDL goal description into a goal state space that is used as an input of our FOVIA algorithm, we have designed a procedure for translating first-order formulae into a set of *CN-states*.

Since a state space is considered as a disjunction of *CN-states*, we first convert a FO formula into DNF. We start with pushing all quantifiers in front of the formula and convert the quantifier-free part into DNF thereof. In order to check whether a disjunct can be directly converted into a *CN-state*, we have to examine its variables. If a disjunct contains no 'bad' variables then it can be directly converted into a respective *CN-state*. Otherwise, the formula itself needs an additional treatment.

The procedure of marking variables as 'bad' works as follows: If a variable occurring within a positive literal is bounded universally then it is marked as 'bad'. Intuitively, based on the semantics of *CN-states*, the variables that occur in the P-part of a *CN-state* are considered existentially bounded. Each 'bad' variable is eliminated via groundization.

For example, in the following formula

$$\forall X.\exists Y.red(X) \wedge blue(Y)$$

the variable X will be marked as 'bad'.

Assume that we have only two blocks a and b in the domain. After eliminating X (and slight simplification), we obtain:

$$red(a) \wedge red(b) \wedge \exists Y.blue(Y) \ .$$

The variable Y will not be marked as 'bad', hence, it will not be grounded. Similarly, the negative literals are checked for 'bad' variables. The same technique for eliminating 'bad' variables is applied for action descriptions.

Although our approach relies on partial groundization of state and action descriptions, there are domains, e.g., colored blocksworld, where most variables are marked as 'good', and hence, need not be grounded.

Regression of Abstract States

The classical as well as first-order value iteration algorithms are intimately related to regression of states. The crucial difference of the symbolic value iteration is that the regression is performed on the abstract states instead of the single states themselves.

Given a *CN-state* Z and an action description A, our regression procedure produces the set of all possible predecessor *CN-states* Z_i such that Z is reachable from each of Z_i by executing A. In FOVIA, actions are specified by preconditions that are represented as *CN-states* and STRIPS style effects Q^+ and Q^-.

We now illustrate the regression procedure with an example from the blocksworld scenario. Here, we present one regression step through action $putdown(Top, Bottom)$ that has two nature's choices, given below:

$$\begin{aligned} &putdown_1(Top, Bottom) \\ &\quad \text{Pre} : (holding(Top), \{on(X, Bottom)\}) \\ &\quad \text{Eff} : Q^+ = on(Top, Bottom) \\ &\qquad\quad Q^- = holding(Top) \\ &putdown_2(Top, Bottom) \\ &\quad \text{Pre} : (holding(Top), \{on(X, Bottom)\}) \\ &\quad \text{Eff} : Q^+ = on(Top, table) \\ &\qquad\quad Q^- = holding(Top) \ . \end{aligned}$$

The regression of the *CN-state* Z:

$$Z = (on(B_0, B_1) \circ on(B_1, table) \circ on(B_2, table), \emptyset)$$

yields the following predecessor states Z_i:

$$Z_1 = (holding(B_2) \circ on(B_0, B_1) \circ on(B_1, table), \emptyset)$$
$$Z_2 = (holding(B_2) \circ on(B_0, B_1) \circ on(B_1, table) \circ on(B_3, table), \{on(B_4, B_3)\})$$
$$Z_3 = (holding(B_0) \circ on(B_1, table) \circ on(B_2, table), \{on(B_3, B_1)\}) ,$$

where Z_1 represents all real-world states, where a gripper holds a block B_2, a block B_0 is on B_1 and B_1 is on the table; Z_2 asserts the same information as Z_1 and additionally states that some block B_3 is on the table and there is no such block B_4 that is on B_3; and Z_3 is interpreted as the set of all real-world states, where a gripper holds a block B_0, blocks B_1 and B_2 are on the table, and there is no such block B_3 that is on B_1.

The regression procedure can be outlined as follows. We first check whether the Q^- effects and the P-part of a *CN-state* Z are consistent wrt. each other. If the answer is no, then the regression procedure stops delivering the empty set of predecessor *CN-states*. Otherwise, a predecessor state is constructed as follows: The Q^+ effects are subtracted from the P-part of the *CN-state* Z and the result is joined with the P-part of the action preconditions forming the P-part of a predecessor *CN-state*. Analogously, the $\mathcal{N}$-part of a predecessor *CN-state* is built by subtracting the Q^- effects from the $\mathcal{N}$-part of Z and joining the result with the $\mathcal{N}$-part of the action preconditions. If the resulting predecessor state is consistent then it is added to the set of the Z's predecessor states. We describe the consistency check in more detail in the section on optimizations.

The operations over fluent terms and sets of fluent terms, e.g., aforementioned subtraction and union, are based on solving the submultiset matching problem that usually has multiple solutions (Große *et al.* 1992). This implies that the regression procedure may deliver multiple predecessor states. Recalling our running example, both *CN-states* Z_1 and Z_3 were obtained as a result of the regression of Z through a single nature's choice $putdown_1$.

Some Optimizations

In general, a state description may contain two kinds of inconsistencies. The inconsistency of the first kind takes place when some element of the $\mathcal{N}$-part contradicts with the P-part. For example, in a state description $(red(a), \{red(X)\})$ the P-part asserts that the block a is red, whereas the $\mathcal{N}$-part prohibits any block X of being red. In this case, the consistency test will include a simple syntactic check.

The second kind of inconsistencies is referred to as domain-dependent. For example, the state description $(empty \circ holding(a), \emptyset)$ is formally consistent (wrt. the previous kind of inconsistency). And only after having learned that the domain contains a single gripper, this *CN-state* is turned to be inconsistent. In this case, the consistency test uses additional domain axioms which, e.g., state that the combination of fluents $empty$ and $holding(X)$ is forbidden.

The state space that represents a value function after some iteration step of FOVIA algorithm may contain redundancies. For example, consider a state space that consists of two abstract states $Z_1 = (holding(a), \emptyset)$ and $Z_2 = (holding(X), \emptyset)$ that are both assigned the same value, say, of 10. The *CN-state* Z_1 represents the set of all real-world states that do satisfy the fact $holding(a)$. At the same time, the *CN-state* Z_2 describes all real-world states represented by Z_1 plus additional states, where X is instantiated by a constant different from a. Since the values associated with Z_1 and Z_2 are the same, Z_1 can be painlessly removed without loss of information. In FCPLANNER, we employ the automated normalization procedure that, given a state space, delivers an equivalent one that contains no redundancies (Skvortsova 2003). The technique employs the notion of a subsumption relation that enables to determine which states are redundant and can be removed from the state space therefore.

References

Bellman, R. E. 1957. *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic Dynamic Programming for First-Order MDPs. In Nebel, B., ed., *Proceedings of the Seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, 690–700. Morgan Kaufmann.

Große, G.; Hölldobler, S.; Schneeberger, J.; Sigmund, U.; and Thielscher, M. 1992. Equational logic programming, actions, and change. 177–191. MIT Press.

Großmann, A.; Hölldobler, S.; and Skvortsova, O. 2002. Symbolic Dynamic Programming within the Fluent Calculus. In Ishii, N., ed., *Proceedings of the IASTED International Conference on Artificial and Computational Intelligence*, 378–383. Tokyo, Japan: ACTA Press.

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

Skvortsova, O. 2003. Towards Automated Symbolic Dynamic Programming. Master's thesis, TU Dresden.

Thielscher, M. 2004. FLUX: A logic programming method for reasoning agents. *Theory and practive of Logic Programming*.