Department of Electronics, Information, and Bioengeneering
Politecnico di Milano

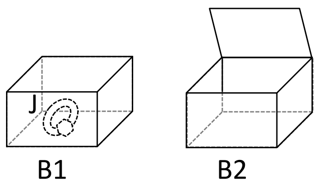
Artificial Intelligence 2019-20

Marco Colombetti

5. Planning

5.1 Situation Calculus

In this section we use a very simple example, in which a number of boxes (B1, B2, ...) can be open or closed, and can contain certain objects (e.g., a jewel, J). We may represent an actual situation, like



by the following ground literals:[1]

Box(B1)	Box(B2)	Object(J)
Closed(B1)	¬Closed(B2)[2]	In(J,B1)

However, if we want to represent the effect of opening box B1, we cannot extend the initial representation by adding

¬Closed(B1)

because this would make the representation inconsistent. This shows that representing change in FOL is not trivial.

Situations (or "states")

To deal with change we need to describe different situations within the same logical system. One possibility is to adopt *Situation Calculus*, first proposed by John McCarthy and Patrick Hayes in 1969.[3] We first distinguish between those facts that cannot change in time (like being a box or being an object) and those that can change (like being open or being in). For every type of fact that *can* change, we specify that the fact holds *in the context of a given situation*.

[1] In FOL, an *atom* (or *atomic sentence*) is a formula of the form $P(t_1,...,t_n)$, where P is an n-ary predicate symbol and the t_i are terms (i.e., individual constants, variables, or functional terms). A *literal* is an atom or the negation of an atom. A sentence is *ground* if it contains no variables.

[2] Of course, instead of taking "closed" as the primitive concept and representing "open" as "not closed," we could take "open" as primitive and represent "closed" as "not open." We may as well treat both concepts as primitive, using two predicates and introducing an axiom to states that something is open if, and only if, it is not closed.

[3] McCarthy, J., and P. J. Hayes (1969). Some philosophical problems from the standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, eds., *Machine Intelligence 4*, 463-502. Edinburgh University Press, Edinburgh.

A way of doing this is by adding a new argument to those predicates (usually called *fluents*) that represent facts that are subject to change. So, if we decide to use the constant S0 to denote the situation depicted above we can state that:

Box(B1) Box(B2) Object(J)

Closed(B1,S0) ¬Closed(B2,S0) In(J,B1,S0)

This way we *reify situations*, that is, we treat situations as objects of the domain of the interpretation of the logical language. This allows us to refer to situations by constants (like S0) or more complex functional terms (as we shall see later).

Actions

Suppose we want to describe the situation obtained by opening B1, starting from S0. How are we going to represent the action of opening B1? In FOL we can do so by the functional term

open(B1)

Analogously to the case of situations, this amounts to *reifying actions* (because the term open(B1) denotes an object in the domain of the interpretation of our logical language). Now, to represent the situation obtained by opening B1 starting from S0 we can use the functional term

result(open(B1),S0)

Then, to state that in this new situation box B1 is open we use the sentence

¬Closed(B1,result(open(B1),S0))

which can be read as: "B1 is not closed in then situation resulting from opening B1 in S0." Note that the second argument of predicate Open (i.e., the functional term result(open(B1),S0)) denotes a situation.

The preconditions and effects of the actions of opening and closing a generic box x, starting from a generic situation s, can now be specified using the following logical axioms (in which the variables x and s are to be considered as universally quantified):

Box(x) ∧ ¬Open(x,s) → Open(x,result(open(x),s))

Box(x) ∧ Open(x,s) → ¬Open(x,result(close(x),s))

(This representation is similar to the one originally proposed by McCarthy and Hayes, and rather different from the one adopted in Section 10.4.2 of the textbook.) Each of these axioms has the form of a conditional, whose antecedent specifies *sufficient conditions* for the execution of an action of the form open(x) or close(x), respectively, and whose consequent specifies the effect of executing such an action. We can now use all the axioms (including those representing the facts of S0) to prove, for example, that the following statement holds:

Open(B1,result(open(B1),S0))

However, it is not possible to prove that

Open(B2,result(open(B1),S0))

even if this is intuitively true, as box B2 has not been touched. The reason is that we described the change introduced in a situation by executing an action of the form open(x), but we did not specify *what is left unchanged* when such an action is performed. This may seem superfluous because we, as human beings, implicitly assume that every fact that is not an effect of an action is left unchanged by a performance of the action. This "principle of inertia" is so natural for us, that it is difficult to understand why a logical system may be unable to derive such consequences. The point is that nothing in FOL corresponds to such a principle: as far as logical deduction is concerned, only what is explicitly stated in the axioms can be exploited. This is indeed a difficulty, which is known as the *frame problem* (where the term "frame" denotes the part of the environment that is not affected by an action). To be able to derive a fact like Open(B2,result(open(B1),S0)) we have to introduce new axioms to specify every fact that is left unchanged by the performance of open(B1). Such axioms are called *frame axioms*. Obviously, being obliged to use additional axioms to specify everything that does not change as an effect of an action is extremely unsatisfactory. It is at least in part for this reason that alternatives to the Situation Calculus have been developed since the mid 1970s.

To complete our representation we need to specify axioms concerning the effects (an the frame) of two more actions: take(y,x), for taking object y from box x, put(y,x), for putting object y into box x. This is left as an exercise to the reader. (Hint: assume that the objects and the boxes can be manipulated by a robotic arm with a gripper, which can hold an object or alternatively be "free.")

Let us now see how a goal is represented in the Situation Calculus. Call K the set of axioms specifying: (i) the initial state, S0; and (ii), the effects and frames of all actions. A planning problem is now specified by asking a reasoner to prove whether there exists a situation in which certain facts hold. For example, supposing that we are interested in having the jewel in box B2, we can try to establish whether

$$K \models \exists u\, In(J,B2,u)$$

After a reasoner has proved this entailment, a plan to reach the goal is typically extracted from the proof. For example, we can use resolution reasoning to prove that $K \land \neg\exists u\, In(J,B2,u)$ is unsatisfiable. In our case, this result would be achieved with the following substitution:

```
u = result(put(J,B2),
           result(take(J,B1),
                  result(open(B1),
                         S0)
                  )
           )
```

From this functional term, working inside-out, it is possible to extract the plan:

```
open(B1);  take(J,B1);  put(J,B2)
```

5.2 Classical Planning

For the STRIPS system and PDDL, the reader is referred to Chapter 10 of the textbook. Here I add a few complementary notes.

State representation and database semantics

In STRIPS-like systems, a situation (now called a "state") is represented as a finite set (or conjunction) of *facts*. Every fact is a *ground atom* (or *ground positive literal*) of FOL:

Box(B1)	Box(B2)	Object(J)
Closed(B1)	Open(B2)	In(J,B1)

Note that states are not explicitly mentioned, and that we need to use both predicate Closed and predicate Open because *negative literals are not allowed in states*.

A state is like a tiny database: new facts can be added and existing facts can be deleted. The interpretation of states follows so-called *database semantics*, that is, FOL semantics plus the following additional assumptions:

- Unique Name Assumption (UNA): different individual constants always name different objects;
- Domain Closure Assumption (DCA): the domain of the interpretation does not contain any "anonymous" object (i.e., every existing object must be denoted by an individual constant);
- Closed World Assumption (CWA): a fact that is not part of the representation of a state does not hold in that state.

It is important to understand that database semantics presupposes that the representation of a state is *complete*, in that: (i) every existing object is denoted by a constant; and (ii), every fact that holds in a state is explicitly represented. Of course, these assumptions are not always met in an actual application.

5.3 The planning process

In Classical Planning, plan formation is based on heuristic search. A wide variety of methods can be applied to solving PDDL problems. We shall discuss a few of them using the box example. More precisely:

- a number of boxes may be open or closed and may contain one or more objects
- a gripper, attached to a robotic arm, can:
  - open a box (if the box is closed)
  - close a box (if the box is open).
  - take an object from a box (if the box is open, the object is in the box, and the gripper is free)
  - put an object in a box (if the box is open and the gripper holds the object)

We shall consider a very simple initial state S0, in which the gripper is free and there are only two boxes (B1, B2) and one object (J), configured as follows:

Open(B1) Closed(B2) In(J,B2) Free

Note that we need neither a Box(–) predicate, because we can identify boxes with those things that are either open or closed, nor an Object(–) predicate, because objects are exactly those things that are either in a box or held by the gripper.

Below is a possible representation of the action schemes (using a syntax that is different from the one adopted by the textbook, but should be easily understandable):

```
action open(x)
        prec   Closed(x)
        eff    Open(x) ∧ ¬Closed(x)
action close(x)
        prec   Open(x)
        eff    Closed(x) ∧ ¬Open(x)
action take(y,x)
        prec   In(y,x) ∧ Open(x) ∧ Free
        eff    Holds(y) ∧ ¬In(y,x) ∧ ¬Free
action put(y,x)
        prec   Holds(y) ∧ Open(x)
        eff    In(y,x) ∧ Free ∧ ¬Holds(y)
```

Now suppose that our goal is to move the jewel into the other box:

In(J,B2)

Forward planning

We can exploit the PDDL representation to create an SSP (State Space Problem), to which we can then apply all familiar search strategies. We first specify finite domains for all variables:

x ∈ {B1,B2}	(x represents a box)
y ∈ {J}	(y represents an object)

From the names and parameters of the actions in the action schemes we can now generate a finite set of possible actions:

open(B1)	close(B1)	take(J,B1)	put(J,B1)
open(B2)	close(B2)	take(J,B2)	put(J,B2)

For every action, we compute the applicability conditions (from the preconditions of the action schemes) and the results (from the effects of the action schemes). We assume all costs to be equal to 1. Moreover, the initial state and the goal are given. This completes the specification of an SSP.

In *forward planning*, we can use any search strategy to compute a search tree starting from the initial state and trying to reach a goal. However, the branching factor of the search tree may be very high, unless powerful heuristics are exploited. For example, with 100 different boxes (part of which are open, and the other ones closed) the actions that may be performed in the initial state are 100 + *N*, where *N* is the number of objects contained in open boxes. This is due to the fact that forward search tend to try all actions that are *possible*, rather than all actions that may be *relevant*.

Backward planning

With backward planning, the planning process starts from the initial, or 0th level, goal,

```
In(J,B2)
```

Once it has been verified that the goal is not satisfied in the initial state, the action schemes are examined to establish which action could have such fact as one of its effects. The only possibility is the action put(J,B2):

```
In(J,B2)
  | put(J,B2)
```

In order for this action to be execute, its preconditions must be satisfied:

```
In(J,B2)
  | put(J,B2)
Holds(J) ∧ Open(B2)
```

The conjunction is now taken as a 1st level goal and the planning procedure is recursively invoked:[4]

```
In(J,B2)
  | put(J,B2)
Holds(J) ∧ Open(B2)
  | take(J,x)
In(J,x) ∧ Open(x) ∧ Free
```

The first 2nd level subgoal, In(J,x), is satisfied in the initial state, provided variable x is assigned value B1; therefore the subgoal can be eliminated from the list of 2nd level subgoals:

In(J,B2)

  | put(J,B2)

Holds(J) ∧ Open(B2)

  | take(J,B1)

~~In(J,x)~~ ∧ Open(B1) ∧ Free

The two remaining 2nd level subgoals, Open(B1) and Free, can be eliminated for the same reason:

In(J,B2)

  | put(J,B2)

Holds(J) ∧ Open(B2)

  | take(J,B1)

~~In(J,x) ∧ Open(B1) ∧ Free~~

[4] This is a gross simplification of an actual process of backward planning, but it will suffice to give an idea.

At this point, it is possible to eliminate the 1st level subgoal Holds(J), because we know how to achieve it from the initial state:

In(J,B2)
    | put(J,B2)
~~Holds(J)~~ ∧ Open(B2)
    | take(J,B1)
~~In(J,x) ∧ Open(B1) ∧ Free~~

We are now left with the 1st level subgoal Open(B2), which can be achieved applying the same backward planning procedure:

In(J,B2)
    | put(J,B2)
~~Holds(J)~~   ∧   Open(B2)
    | take(J,B1) | open(B2)
~~In(J,x) ∧ Open(B1) ∧ Free~~   ~~Closed(B2)~~

As a result, we have a *partially ordered plan*, in which two actions, take(J,B1) and open(B2), have to be performed before a third action, put(J,B2), can be executed. This plan admits of two possible linear realisations:

take(J,B1); open(B2); put(J,B2)
open(B2); take(J,B1); put(J,B2)

5.4 The planning process: SATPlan

As we have seen, in the Situation Calculus a planning problem is solved by proving that K ⊨ g, where K is the set of all axioms representing the initial state and the preconditions and effects of actions (plus the necessary frame axioms), and g is a sentence representing the goal.

SATPlan (see Sections 7.7.4 and 10.4.1 of the textbook) exploits logic in a different way: instead of proving that K ⊨ g (i.e., that K ∧ ¬g is *unsatisfiable*) and then *extracting a plan from the proof*, it proves that K ∧ g is *satisfiable* by building a model of it and then *extracts a plan from the model*.

A model satisfying K ∧ g is searched incrementally, by tentatively setting the length t of a plan to 0, 1, 2, ... until a predefined maximum length is reached. For every value of t a different propositional representation of K ∧ g is generated, and the attempt to build a model of this representation is carried out (for example using the DPLL algorithm). For t = 0 (which means a plan of length 0, implying that the goal is immediately satisfied) only the initial state and the goal are relevant.

We shall now considering a simplified version of our previous example, with only one box (B) that is initially closed and contains J. The goal is that the gripper holds the jewel. With t = 0 the goal is represented by the proposition symbol:

Holds-J^0

Note that to apply SATplan the goal is not negated, because one is trying to prove satisfiability, not unsatisfiability! As far as the initial state is concerned, both positive and negative facts must be represented, because logical semantics is applied (and not database semantics). As a consequence, the symbols Open and Free become unnecessary:

Closed-B^0
In-J-B^0
¬Holds-J^0

Let us now apply DPLL to this representation:

goal: Holds-J^0

init: Closed-B^0

In-J-B^0

$\neg\text{Holds-J}^0$

We start an attempt to build a model by assigning *true* to Holds-J^0 (and leaving all other propositional symbols undefined, for the time being):

Model = $\{\text{Holds-J}^0\}$

Applying the DPLL algorithm the representation becomes:

~~Holds-J^0~~

Closed-B^0

In-J-B^0

false

The fourth clause shows that no model exists for t = 0 (i.e., the goal cannot be reached without executing any action). We then increment t to 1 and generate a new propositional representation. The representation of the initial state does not change:

goal: Holds-J^1

init: Closed-B^0

In-J-B^0

$\neg\text{Holds-J}^0$

To represent the execution of an action, we shall include in the model a fact that holds at the step at which the action is performed. For example, to say that the action of opening B is executed at t = 0 we shall include in the model the sentence

open-B^0

Of course, we must make sure that at t = 0 the preconditions of opening the box hold in the model. This is guaranteed by suitable *precondition axioms*, which state that if an action of a certain type is executed at t, then certain facts (i.e., the action's preconditions) necessarily hold at the same t. In our example (assuming for simplicity's sake that we only have actions to open the box and to take the jewel) the precondition axioms are:

$\text{open-B}^0 \rightarrow \text{Closed-B}^0$

$\text{take-J-B}^0 \rightarrow \neg\text{Holds-J}^0 \wedge \text{In-J-B}^0 \wedge \neg\text{Closed-B}^0$

These two axioms translate into the following clauses:

$\neg\text{open-B}^0 \vee \text{Closed-B}^0$

$\neg\text{take-J-B}^0 \vee \neg\text{Holds-J}^0$

$\neg\text{take-J-B}^0 \vee \text{In-J-B}^0$

$\neg\text{take-J-B}^0 \vee \neg\text{Closed-B}^0$

Now we have to represent the effects of the actions. We have seen before that STRIPS solves the frame problem of Situation Calculus thanks to a database-like treatment of states. But we cannot adopt this solution here, because we are using pure logic (although at the propositional level). Another possible solution of the frame problem is based on a change of viewpoint in the representation: instead of focusing on actions and stating their effects on fluents, we focus on fluents and use *fluent axioms* to state how they are affected by actions. The general pattern of an effect axiom for a fluent F is (see the textbook, p. 267):

$\text{F}^{t+1} \leftrightarrow \text{ActionCausesF}^t \vee (\text{F}^t \wedge \neg\text{ActionCausesNotF}^t)$

Remembering that in our example the only possible actions are open-B and take-J-B, the fluent axioms are:

$\text{Closed-B}^1 \leftrightarrow \text{Closed-B}^0 \wedge \neg\text{open-B}^0$

$\text{In-J-B}^1 \leftrightarrow \text{In-J-B}^0 \wedge \neg\text{take-J-B}^0$

$\text{Holds-J}^1 \leftrightarrow \text{take-J-B}^0 \vee \text{Holds-J}^0$

These three biconditional axioms are equivalent to the six conditional axioms:

$\text{Closed-B}^1 \rightarrow \text{Closed-B}^0 \wedge \neg\text{open-B}^0$

$\text{Closed-B}^0 \wedge \neg\text{open-B}^0 \rightarrow \text{Closed-B}^1$

$\text{In-J-B}^1 \rightarrow \text{In-J-B}^0 \wedge \neg\text{take-J-B}^0$

$\text{In-J-B}^0 \wedge \neg\text{take-J-B}^0 \rightarrow \text{In-J-B}^1$

$\text{Holds-J}^1 \rightarrow \text{take-J-B}^0 \vee \text{Holds-J}^0$

$\text{take-J-B}^0 \vee \text{Holds-J}^0 \rightarrow \text{Holds-J}^1$

which in turn translate into the following clauses:

$\neg\text{Closed-B}^1 \vee \text{Closed-B}^0$

$\neg\text{Closed-B}^1 \vee \neg\text{open-B}^0$

$\neg\text{Closed-B}^0 \vee \text{open-B}^0 \vee \text{Closed-B}^1$

$\neg\text{In-J-B}^1 \vee \text{In-J-B}^0$

$\neg\text{In-J-B}^1 \vee \neg\text{take-J-B}^0$

$\neg\text{In-J-B}^0 \vee \text{take-J-B}^0 \vee \text{In-J-B}^1$

$\neg\text{Holds-J}^1 \vee \text{take-J-B}^0 \vee \text{Holds-J}^0$

$\neg\text{take-J-B}^0 \vee \text{Holds-J}^1$

$\neg\text{Holds-J}^0 \vee \text{Holds-J}^1$

Finally, we have to add the relevant *action exclusion axioms*, which state that certain actions cannot be performed together at any step of the plan:

$\neg\text{open-B}^0 \vee \neg\text{take-J-B}^0$

Here is the complete representation:

goal:	Holds-J^1
init:	Closed-B^0
	In-J-B^0
	$\neg\text{Holds-J}^0$
prec:	$\neg\text{open-B}^0 \vee \text{Closed-B}^0$
	$\neg\text{take-J-B}^0 \vee \neg\text{Holds-J}^0$
	$\neg\text{take-J-B}^0 \vee \text{In-J-B}^0$
	$\neg\text{take-J-B}^0 \vee \neg\text{Closed-B}^0$
fluent:	$\neg\text{Closed-B}^1 \vee \text{Closed-B}^0$
	$\neg\text{Closed-B}^1 \vee \neg\text{open-B}^0$
	$\neg\text{Closed-B}^0 \vee \text{open-B}^0 \vee \text{Closed-B}^1$
	$\neg\text{In-J-B}^1 \vee \text{In-J-B}^0$
	$\neg\text{In-J-B}^1 \vee \neg\text{take-J-B}^0$
	$\neg\text{In-J-B}^0 \vee \text{take-J-B}^0 \vee \text{In-J-B}^1$
	$\neg\text{Holds-J}^1 \vee \text{take-J-B}^0 \vee \text{Holds-J}^0$
	$\neg\text{take-J-B}^0 \vee \text{Holds-J}^1$
	$\neg\text{Holds-J}^0 \vee \text{Holds-J}^1$
excl:	$\neg\text{open-B}^0 \vee \neg\text{take-J-B}^0$

Again, DPLL will fail to find a model satisfying this representation, because the goal cannot be achieved at $t = 1$ (i.e., performing only one action). It will therefore be necessary to build a representation for $t = 2$, with the goal

Holds-J^2

This time DPLL will succeed, returning a model that contains, among other things, two facts that specify a successful plan:

open-B^0

take-J-B^1

To keep this presentation short, we shall only search for a one-action plan that achieves the box open at t = 1. Here is the representation:

goal: $\neg\text{Closed-B}^1$

init: Closed-B^0

In-J-B^0

$\neg\text{Holds-J}^0$

prec: $\neg\text{open-B}^0 \lor \text{Closed-B}^0$

$\neg\text{take-J-B}^0 \lor \neg\text{Holds-J}^0$

$\neg\text{take-J-B}^0 \lor \text{In-J-B}^0$

$\neg\text{take-J-B}^0 \lor \neg\text{Closed-B}^0$

fluent: $\neg\text{Closed-B}^1 \lor \text{Closed-B}^0$

$\neg\text{Closed-B}^1 \lor \neg\text{open-B}^0$

$\neg\text{Closed-B}^0 \lor \text{open-B}^0 \lor \text{Closed-B}^1$

$\neg\text{In-J-B}^1 \lor \text{In-J-B}^0$

$\neg\text{In-J-B}^1 \lor \neg\text{take-J-B}^0$

$\neg\text{In-J-B}^0 \lor \text{take-J-B}^0 \lor \text{In-J-B}^1$

$\neg\text{Holds-J}^1 \lor \text{take-J-B}^0 \lor \text{Holds-J}^0$

$\neg\text{take-J-B}^0 \lor \text{Holds-J}^1$

$\neg\text{Holds-J}^0 \lor \text{Holds-J}^1$

excl: $\neg\text{open-B}^0 \lor \neg\text{take-J-B}^0$

Below are the steps of the DPLL algorithm:

Step 1. Unit clause: $\neg\text{Closed-B}^1$

Model = $\{\neg\text{Closed-B}^1\}$

Closed-B^0

In-J-B^0

$\neg\text{Holds-J}^0$

$\neg\text{open-B}^0 \lor \text{Closed-B}^0$

$\neg\text{take-J-B}^0 \lor \neg\text{Holds-J}^0$

$\neg\text{take-J-B}^0 \lor \text{In-J-B}^0$

$\neg\text{take-J-B}^0 \lor \neg\text{Closed-B}^0$

$\neg\text{Closed-B}^0 \lor \text{open-B}^0$

$\neg\text{In-J-B}^1 \lor \text{In-J-B}^0$

$\neg\text{In-J-B}^1 \lor \neg\text{take-J-B}^0$

$\neg\text{In-J-B}^0 \lor \text{take-J-B}^0 \lor \text{In-J-B}^1$

$\neg\text{Holds-J}^1 \lor \text{take-J-B}^0 \lor \text{Holds-J}^0$

$\neg\text{take-J-B}^0 \lor \text{Holds-J}^1$

$\neg\text{Holds-J}^0 \lor \text{Holds-J}^1$

$\neg\text{open-B}^0 \lor \neg\text{take-J-B}^0$

Step 2. Unit clause: Closed-B^0

Model = $\{\neg\text{Closed-B}^1, \text{Closed-B}^0\}$

In-J-B^0

$\neg\text{Holds-J}^0$

$\neg\text{take-J-B}^0 \lor \neg\text{Holds-J}^0$

$\neg\text{take-J-B}^0 \lor \text{In-J-B}^0$

$\neg\text{take-J-B}^0$

open-B^0

$\neg\text{In-J-B}^1 \lor \text{In-J-B}^0$

$\neg\text{In-J-B}^1 \lor \neg\text{take-J-B}^0$

$\neg\text{In-J-B}^0 \lor \text{take-J-B}^0 \lor \text{In-J-B}^1$

$\neg\text{Holds-J}^1 \lor \text{take-J-B}^0 \lor \text{Holds-J}^0$

$\neg\text{take-J-B}^0 \lor \text{Holds-J}^1$

$\neg\text{Holds-J}^0 \lor \text{Holds-J}^1$

$\neg\text{open-B}^0 \lor \neg\text{take-J-B}^0$

Step 3. Unit clause: In-J-B^0

Model = $\{\neg\text{Closed-B}^1, \text{Closed-B}^0, \text{In-J-B}^0\}$

$\neg\text{Holds-J}^0$

$\neg\text{take-J-B}^0 \lor \neg\text{Holds-J}^0$

$\neg\text{take-J-B}^0$

open-B^0

$\neg\text{In-J-B}^1 \lor \neg\text{take-J-B}^0$

$\text{take-J-B}^0 \lor \text{In-J-B}^1$

$\neg\text{Holds-J}^1 \lor \text{take-J-B}^0 \lor \text{Holds-J}^0$

$\neg\text{take-J-B}^0 \lor \text{Holds-J}^1$

$\neg\text{Holds-J}^0 \lor \text{Holds-J}^1$

$\neg\text{open-B}^0 \lor \neg\text{take-J-B}^0$

Step 4. Unit clause: $\neg\text{Holds-J}^0$

Model = $\{\neg\text{Closed-B}^1, \text{Closed-B}^0, \text{In-J-B}^0, \neg\text{Holds-J}^0\}$

$\neg\text{take-J-B}^0$

open-B^0

$\neg\text{In-J-B}^1 \lor \neg\text{take-J-B}^0$

$\text{take-J-B}^0 \lor \text{In-J-B}^1$

$\neg\text{Holds-J}^1 \lor \text{take-J-B}^0$

$\neg\text{take-J-B}^0 \lor \text{Holds-J}^1$

$\neg\text{open-B}^0 \lor \neg\text{take-J-B}^0$

Step 5. Unit clause: $\neg$take-J-B^0
Model = {$\neg$Closed-B^1, Closed-B^0, In-J-B^0, $\neg$Holds-J^0, $\neg$take-J-B^0}
open-B^0
In-J-B^1
$\neg$Holds-J^1

Step 6. Unit clause: open-B^0
Model = {$\neg$Closed-B^1, Closed-B^0, In-J-B^0, $\neg$Holds-J^0, $\neg$take-J-B^0,open-B^0}
In-J-B^1
$\neg$Holds-J^1

Step 7. Unit clause: In-J-B^1
Model = {$\neg$Closed-B^1, Closed-B^0, In-J-B^0, $\neg$Holds-J^0, $\neg$take-J-B^0, In-J-B^1}
$\neg$Holds-J^1

Step 8. Unit clause: $\neg$Holds-J^1
Model = {$\neg$Closed-B^1, Closed-B^0, In-J-B^0, $\neg$Holds-J^0, $\neg$take-J-B^0, open-B^0, In-J-B^1, $\neg$Holds-J^1}

The plan is then extracted from the model and consists of the action performed at $t = 0$:
open-B^0