Automated Planning & Artificial Intelligence

SAT Planning

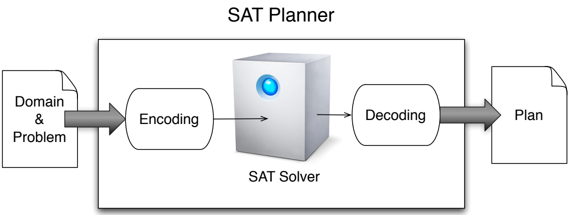
Humbert Fiorino

Humbert.Fiorino@imag.fr
http://membres-lig.imag.fr/fiorino

Laboratory of Informatics of Grenoble – MAGMA team

March 2011

An alternative approach



- Rather than directly solving a planning problem, it could be a good idea to transform it into another well-known and tackled problem
- This is a traditional strategy in science : "reduction", "confluence" = to look for relations or equivalences between statements
- One possible reduction for planning problems is towards SAT problems

Satisfiability Problem

- Let x_i represent propositional variables that can assume only values *true* or *false*
- Clause = a disjunction of propositional variables or their negation :

$$(x_1 \vee x_3 \vee \bar{x}_4)$$

- Formula in conjunctive normal form (CNF) = a conjunction of clauses :

$$(x_1 \vee x_3 \vee \bar{x}_4) \wedge (x_4) \wedge (x_2 \vee \bar{x}_3)$$

- SAT Problem = given a formula in CNF, find if there is an assignment of values to the propositional variables so that the formula evaluates to *true*

SAT Problem Encoding

- Propositional variables are represented by an integer : 1 means "x_1 is *true*" and -1 otherwise
- A clause is a list of integers : $[1, 3, -4]$ stands for $(x_1 \vee x_3 \vee \bar{x}_4)$
- A SAT problem is a list of lists of integers : $[[1, 3, -4], [4], [2, -3]]$ represents $(x_1 \vee x_3 \vee \bar{x}_4) \wedge (x_4) \wedge (x_2 \vee \bar{x}_3)$
- If x_i value is *true*, all clauses containing it can be removed and $-x_i$ can be removed from the clauses
- unitPropagate($[[1, 3, -4], [4], [2, -3]]$,4) $= [[1, 3], [2, -3]]$

unitPropagate(Γ,x)

```
if isEmpty(Γ) then
    return [];
else
    c ← getFirst(Γ);
    Γ ← Γ − {c};
    if contains(c,x) then
        return unitPropagate(Γ,x);
    else
        if contains(c,−x) then
            return remove(−x,c) · unitPropagate(Γ,x);
        else
            return c · unitPropagate(Γ,x);
```

DP(Γ, μ)

```
if isEmpty(Γ) then
    return [];
else
    if contains(Γ, []) then
        return [];
    else
        x ← getFirst(getFirst(Γ));                    // A basic strategy
        μ′ ← DP(unitPropagate(Γ,x), add(x, μ));
        if isEmpty(μ′) then
            return DP(unitPropagate(Γ,−x), add(−x, μ));
        else
            return μ′;
```

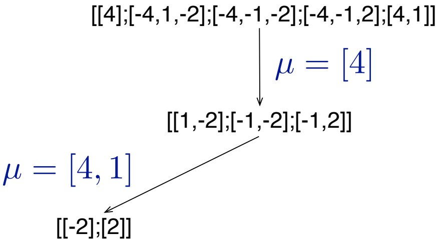
DP search space

[[4];[-4,1,-2];[-4,-1,-2];[-4,-1,2];[4,1]]

DP search space

$$
\begin{array}{c}
[[4];[-4,1,-2];[-4,-1,-2];[-4,-1,2];[4,1]] \\
\Big\downarrow \; \mu = [4] \\
[[1,-2];[-1,-2];[-1,2]]
\end{array}
$$

DP search space



DP search space

[[4];[-4,1,-2];[-4,-1,-2];[-4,-1,2];[4,1]]

$\downarrow \mu = [4]$

[[1,-2];[-1,-2];[-1,2]]

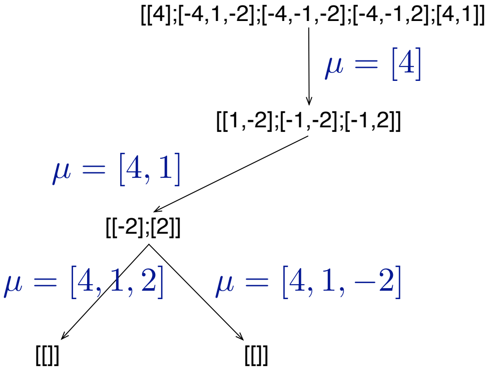
$\downarrow \mu = [4, 1]$

[[-2];[2]]

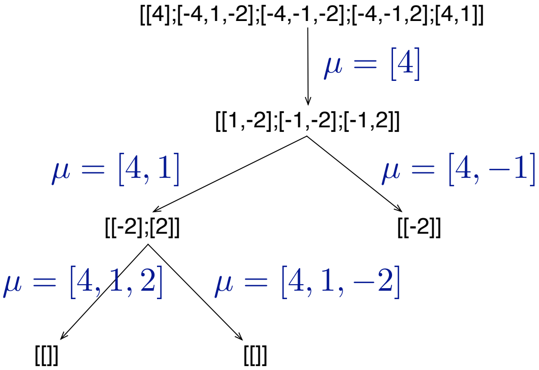
$\downarrow \mu = [4, 1, 2]$

[[]]

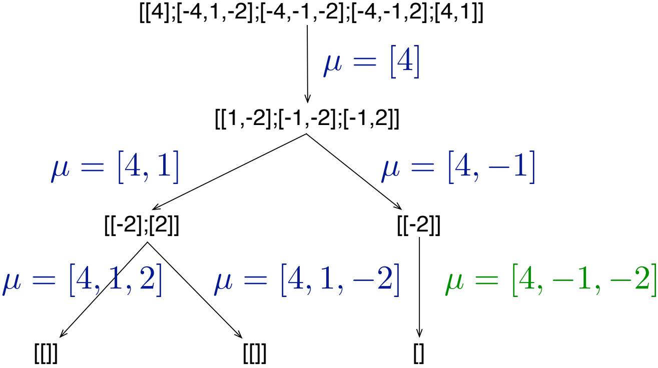
DP search space



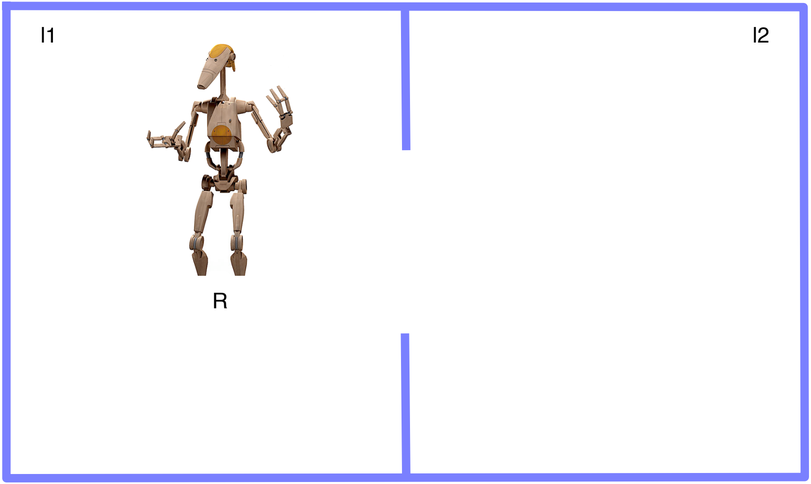
DP search space



DP search space

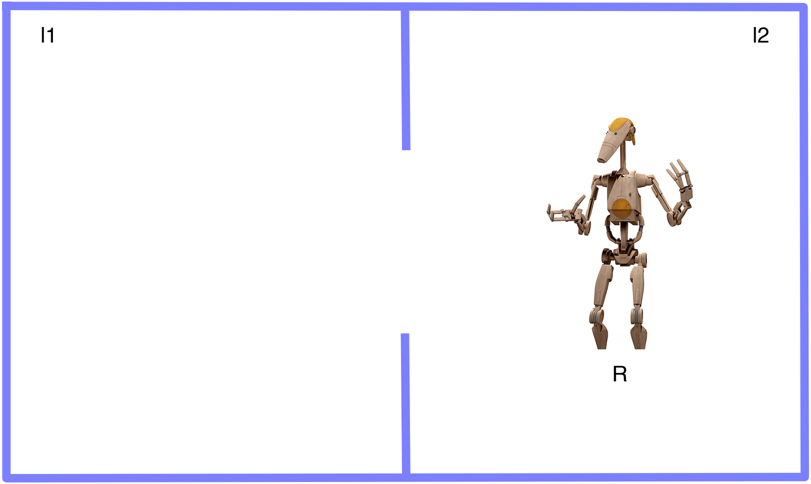


An example



Only one operator : MOVE(x, from, to)

An example



Only one operator : MOVE(x, from, to)

Initial state

$$s_0 \wedge \{ \bigwedge_{f \notin s_0} \neg f \}$$

- at(R, l1, 0) $\wedge$ $\neg$ at(R, l2, 0)

Goal

$$g \wedge \{ \bigwedge_{f \notin g} \neg f \}$$

- at(R, l2, 1) $\wedge$ $\neg$ at(R, l1, 1)

Actions

$$a_i \rightarrow \{\bigwedge \text{precond}(a_i)\} \wedge \{\bigwedge \text{effect}^+(a_i)\} \wedge \{\bigwedge \neg\text{effect}^-(a_i)\}$$

- Remember that $A \rightarrow B \equiv \neg A \vee B$
- MOVE(R, l1, l2, 0) $\rightarrow$ at(R, l1, 0) $\wedge$ at(R, l2, 1) $\wedge$ $\neg$ at(R, l1, 1)
- MOVE(R, l2, l1, 0) $\rightarrow$ at(R, l2, 0) $\wedge$ at(R, l1, 1) $\wedge$ $\neg$ at(R, l2, 1)

State transitions

$$\neg f_i \wedge f_{i+1} \rightarrow \{ \bigvee_{f_{i+1} \in \mathrm{effect}^+(a_i)} a_i \}$$

$$f_i \wedge \neg f_{i+1} \rightarrow \{ \bigvee_{f_{i+1} \in \mathrm{effect}^-(a_i)} a_i \}$$

- ¬ at(R, l1, 0) ∧ at(R, l1, 1) → MOVE(R, l2, l1, 0)
- ¬ at(R, l2, 0) ∧ at(R, l2, 1) → MOVE(R, l1, l2, 0)
- at(R, l1, 0) ∧ ¬ at(R, l1, 1) → MOVE(R, l1, l2, 0)
- at(R, l2, 0) ∧ ¬ at(R, l2, 1) → MOVE(R, l2, l1, 0)

Action disjunction

$$\neg a_i \vee \neg b_i$$

- ¬ MOVE(R, l1, l2, 0) ∨ ¬ MOVE(R, l2, l1, 0)

- $\neg$ MOVE(R, l1, l2, 0) $\vee$ $\neg$ MOVE(R, l2, l1, 0)
- $\neg$ at(R, l1, 0) $\wedge$ at(R, l1, 1) $\rightarrow$ MOVE(R, l2, l1, 0)
- $\neg$ at(R, l2, 0) $\wedge$ at(R, l2, 1) $\rightarrow$ MOVE(R, l1, l2, 0)
- at(R, l1, 0) $\wedge$ $\neg$ at(R, l1, 1) $\rightarrow$ MOVE(R, l1, l2, 0)
- at(R, l2, 0) $\wedge$ $\neg$ at(R, l2, 1) $\rightarrow$ MOVE(R, l2, l1, 0)
- MOVE(R, l1, l2, 0) $\rightarrow$ at(R, l1, 0) $\wedge$ at(R, l2, 1) $\wedge$ $\neg$ at(R, l1, 1)
- MOVE(R, l2, l1, 0) $\rightarrow$ at(R, l2, 0) $\wedge$ at(R, l1, 1) $\wedge$ $\neg$ at(R, l2, 1)
- at(R, l2, 1) $\wedge$ $\neg$ at(R, l1, 1)
- at(R, l1, 0) $\wedge$ $\neg$ at(R, l2, 0)

- $\neg a \lor \neg b$
- $\neg c \land d \rightarrow b$
- $\neg e \land f \rightarrow a$
- $c \land \neg d \rightarrow a$
- $e \land \neg f \rightarrow b$
- $a \rightarrow c \land f \land \neg d$
- $b \rightarrow e \land d \land \neg f$
- $f \land \neg d$
- $c \land \neg e$

- $\neg a \vee \neg b$
- $c \vee \neg d \vee b$
- $e \vee \neg f \vee a$
- $\neg c \vee d \vee a$
- $\neg e \vee f \vee b$
- $\neg a \vee c$
- $\neg a \vee f$
- $\neg a \vee \neg d$
- $\neg b \vee e$
- $\neg b \vee d$
- $\neg b \vee \neg f$
- f
- $\neg d$
- c
- $\neg e$

- $\neg$ a $\vee$ $\neg$ b
- a
- $\neg$ b

Summary