Constraint-Based Planning (SATPlan) (Pre Lecture)

Dr. Neil T. Dantam

CSCI-498/598 RPM, Colorado School of Mines

Spring 2018

SAT Problem

Given: A Boolean formula:

- Variables $P = p_1 \ldots p_n$
- Formula $\phi : \mathbb{B}^n \mapsto \mathbb{B}$

Find: Is $\phi(P)$ satisfiable?

- $\exists P, \ (\phi(P) = 1)$
- What is P?

Solution: Davis-Putnam-Logeman-Loveland (DPLL) Backtracking Search

NP-complete, but modern DPLL solvers are unreasonably effective.

SATPlan Outline

1. Ground first-order logic domain (PDDL) to propositional logic
2. Encode planning problem as boolean formula:
    - Unroll for fixed steps, n.
    - One boolean variable per state/action per step
3. If `SAT`: return action variable assignments
4. Else: increment n and repeat

Example: State Variables

First-Order Logic

Objects: A, B, C

Predicate: `ontable` $(?x)$

Propositional Logic

- `ontable-A`
- `ontable-B`
- `ontable-C`

Unrolled

For $n = 3$:

- `ontable-A-0`
- `ontable-B-0`
- `ontable-C-0`
- `ontable-A-1`
- `ontable-B-1`
- `ontable-C-1`
- `ontable-A-2`
- `ontable-B-2`
- `ontable-C-2`

Exercise: State Variables

First-Order Logic

Objects: A, B

Predicate: $\texttt{on}\,(?x, ?y)$

Propositional Logic

Unrolled

Example: Action Variables

First-Order Logic

Objects: A, B, C

Predicate: `pick-up` $(?x)$

Propositional Logic

- `pick-up-A`
- `pick-up-B`
- `pick-up-C`

Unrolled

For $n = 3$:

- `pick-up-A-0`
- `pick-up-B-0`
- `pick-up-C-0`
- `pick-up-A-1`
- `pick-up-B-1`
- `pick-up-C-1`
- `pick-up-A-2`
- `pick-up-B-2`
- `pick-up-C-2`

Exercise: Action Variables

First-Order Logic

Objects: A, B

Predicate: $\texttt{stack}(?x, ?y)$

Propositional Logic

Unrolled

SATPlan Encoding

Start and Accept

Start: State state holds at step 0

Goal: Goal expression holds at step n

SATPlan Encoding

Transition Function

Operator Encoding: Selected operator's preconditions and effects must hold:

$$o_i^{[k]} \implies \left(\overbrace{\mathrm{pre}(o_i)^{[k]}}^{\text{precondition at step } k} \quad \wedge \quad \overbrace{\mathrm{eff}(o_i)^{[k+1]}}^{\text{effect at step } k+1} \right)$$

Operator Exclusion: One operator per step:

$$o_i^{[k]} \implies \left(\neg o_0{}^{[k]} \wedge \neg o_{(i-1)}{}^{[k]} \wedge \neg o_{(i+1)}{}^{[k]} \wedge \neg o_m{}^{[k]}\right)$$

Frame Axioms: Each proposition p is unchanged unless set by an effect:

$$\left(p^{[k]} = p^{[k+1]}\right) \vee \overbrace{\left(o_j{}^{[k]} \vee \ldots \vee o_\ell{}^{[k]}\right)}^{\text{operators changing } p}$$

Example: Start State

PDDL Facts

```
(define
 (problem sussman-anomaly)
  (:domain blocks)
  (:objects a b c)
  (:init (on c a)
         (ontable a)
         (ontable b)
         (clear c)
         (clear b)
         (handempty))
  (:goal (and (on b c)
              (on a b))))
```

Start State

$$
\begin{aligned}
& \texttt{on-c-a-0} \\
\wedge\ & \texttt{ontable-a-0} \\
\wedge\ & \texttt{ontable-b-0} \\
\wedge\ & \texttt{clear-c-0} \\
\wedge\ & \texttt{clear-b-0} \\
\wedge\ & \texttt{handempty-0} \\
\wedge\ & \neg\texttt{on-a-a-0} \\
\wedge\ & \neg\texttt{on-a-b-0} \\
\wedge\ & \neg\texttt{on-a-c-0} \\
\wedge\ & \neg\ldots
\end{aligned}
$$

Exercise: Start State

PDDL Facts

```
(define
 (problem sussman-anomaly-alt)
  (:domain blocks)
  (:objects a b c)
  (:init (ontable c)
         (ontable a)
         (on b a)
         (clear c)
         (clear b)
         (handempty))
  (:goal (and (on c b)
              (on b a))))
```

Start State

Example: Goal

PDDL Facts

```
(define
 (problem sussman-anomaly)
  (:domain blocks)
  (:objects a b c)
  (:init (on c a)
         (ontable a)
         (ontable b)
         (clear c)
         (clear b)
         (handempty))
  (:goal (and (on b c)
              (on a b))))
```

Goal

$\texttt{on-a-b-k} \wedge \texttt{on-b-c-k}$

Exercise: Goal

PDDL Facts

Goal

```
(define
 (problem sussman-anomaly-alt)
  (:domain blocks)
  (:objects a b c)
  (:init (ontable c)
         (ontable a)
         (on b a)
         (clear c)
         (clear b)
         (handempty))
  (:goal (and (on c b)
              (on b a))))
```

Example: Operator Encoding

`pick-up`

```
(:action pick-up
        :parameters (?x)
        :precondition (and (clear ?x)
                           (ontable ?x)
                           (handempty))
        :effect (and (not (ontable ?x))
                     (not (clear ?x))
                     (not (handempty))
                     (holding ?x)))
```

pick-up-a-0

$$\texttt{pick-up-a-0} \implies \Big(\texttt{clear-a-0} \land \texttt{ontable-a-0} \land \texttt{handempty-0} \land \neg\texttt{ontable-a-1} \land \neg\texttt{clear-a-1} \land \neg\texttt{handempty-1} \land \texttt{holding-a-1} \Big)$$

Exercise: Operator Encoding

`put-down`

```
(:action put-down
        :parameters (?x)
        :precondition (holding ?x)
        :effect (and (not (holding ?x))
                     (clear ?x)
                     (handempty)
                     (ontable ?x)))
```

put-down-a-1

Exercise: Operator Encoding

unstack

```
(:action unstack
        :parameters (?x ?y)
        :precondition (and (on ?x ?y)
                           (clear ?x)
                           (handempty))
        :effect (and (holding ?x)
                     (clear ?y)
                     (not (clear ?x))
                     (not (handempty))
                     (not (on ?x ?y))))
```

unstack-b-c-0

Example: Operator Exclusion

pick-up-a-0

```
(define (domain blocks)
  (:predicates (ontable ?x) (clear ?x) (handempty) (holding ?x))
  (:action pick-up  :parameters (?x)
           :precondition (and (clear ?x) (ontable ?x) (handempty))
           :effect (and (not (ontable ?x))(not (clear ?x))
                        (not (handempty)) (holding ?x)))
  (:action put-down  :parameters (?x)
           :precondition (holding ?x)
           :effect (and (not (holding ?x)) (clear ?x)
                        (handempty)(ontable ?x))))
```

$$o_i^{[k]} \implies \left(\neg o_0^{[k]} \wedge \neg o_{(i-1)}^{[k]} \wedge \neg o_{(i+1)}^{[k]} \wedge \neg o_m^{[k]}\right)$$

$$\texttt{pick-up-a-0} \implies$$
$$(\ \neg\texttt{pick-up-b-0} \wedge \neg\texttt{pick-up-c-0}$$
$$\wedge\neg\texttt{put-down-a-0} \wedge \neg\texttt{put-down-b-0} \wedge \neg\texttt{put-down-c-0})$$

Exercise: Operator Exclusion

put-down-b-1

```
(define (domain blocks)
  (:predicates (ontable ?x) (clear ?x) (handempty) (holding ?x))
  (:action pick-up  :parameters (?x)
           :precondition (and (clear ?x) (ontable ?x) (handempty))
           :effect (and (not (ontable ?x))(not (clear ?x))
                        (not (handempty)) (holding ?x)))
  (:action put-down  :parameters (?x)
           :precondition (holding ?x)
           :effect (and (not (holding ?x)) (clear ?x)
                        (handempty)(ontable ?x))))
```

$$o_i^{[k]} \implies \left(\neg o_0^{[k]} \wedge \neg o_{(i-1)}^{[k]} \wedge \neg o_{(i+1)}^{[k]} \wedge \neg o_m^{[k]} \right)$$

Example: Frame Axioms

```
(define (domain blocks)
  (:predicates (ontable ?x) (clear ?x) (handempty) (holding ?x))
  (:action pick-up  :parameters (?x)
           :precondition (and (clear ?x) (ontable ?x) (handempty))
           :effect (and (not (ontable ?x))(not (clear ?x))
                        (not (handempty)) (holding ?x)))
  (:action put-down  :parameters (?x)
           :precondition (holding ?x)
           :effect (and (not (holding ?x)) (clear ?x)
                        (handempty)(ontable ?x))))
```

$$\left(p^{[k]} = p^{[k+1]}\right) \vee \left({o_j}^{[k]} \vee \ldots \vee {o_\ell}^{[k]}\right)$$

$$(\texttt{ontable-a-0} = \texttt{ontable-a-1}) \vee \texttt{pick-up-a-0} \vee \texttt{put-down-a-0}$$

Exercise: Frame Axioms

```
(define (domain blocks)
  (:predicates (ontable ?x) (clear ?x) (handempty) (holding ?x))
  (:action pick-up  :parameters (?x)
           :precondition (and (clear ?x) (ontable ?x) (handempty))
           :effect (and (not (ontable ?x))(not (clear ?x))
                        (not (handempty)) (holding ?x)))
  (:action put-down  :parameters (?x)
           :precondition (holding ?x)
           :effect (and (not (holding ?x)) (clear ?x)
                        (handempty)(ontable ?x))))
```

$$\left(p^{[k]} = p^{[k+1]}\right) \vee \left({o_j}^{[k]} \vee \ldots \vee {o_\ell}^{[k]}\right)$$

SATPlan Extensions and Implementations

Operator Exclusion: Relax this constraint:

$$o_i^{[k]} \implies \left(\neg o_0^{[k]} \wedge \neg o_{(i-1)}^{[k]} \wedge \neg o_{(i+1)}^{[k]} \wedge \neg o_m^{[k]}\right)$$

Blackbox: H. Kautz and B. Selman. **Unifying SAT-based and graph-based planning**. International Joint Conference on Artificial Intelligence 1999.

Madagascar: J. Rintanen. **Madagascar: Scalable planning with SAT**. 8th International Planning Competition. 2014.

TMKit: N. Dantam, Z. Kingston, S. Chaudhuri, and L. Kavraki. **Incremental Task and Motion Planning: A Constraint-Based Approach**. Robotics: Science and Systems. 2016.