# SAT-based planners

Dra. Mª Dolores Rodríguez Moreno

# Objectives

## Specific Objectives

- Encoding planning problems as satisfiability problems

## Source

- Stuart Russell & Peter Norvig (2009). Artificial Intelligence: A Modern Approach. (3rd Edition). Ed. Pearsons
- License https://creativecommons.org/licenses/by-nc-sa/2.0/

# Outline

- **Motivation**
- Overall approach
- Notation
- Example: GoTo
- SAT planners
- Conclusions

# Motivation

- Propositional satisfiability: given a boolean formula
  - e.g., $(P \vee Q) \wedge (\neg Q \vee R \vee S) \wedge (\neg R \vee \neg P)$,

  does there exist a model (an assignment of truth values to the propositions) that makes the formula true?
- This was the very first problem shown to be NP-complete
- Algorithms are known for solving all but a small subset in average-case polynomial time
- Therefore, try translating classical planning problems into satisfiability problems, and solving them that way

# Outline

- Motivation
- **Overall approach**
- Notation
- Example: GoTo
- SAT planners
- Conclusions

# Overall Approach

- A *bounded planning problem* is a pair $(P,n)$:
  - $P$ is a planning problem; $n$ is a positive integer
  - Any solution for $P$ of length $n$ is a solution for $(P,n)$
- Planning algorithm:
- Do iterative deepening like we did with Graphplan:
  - for $n = 0, 1, 2, \ldots,$
    - encode $(P,n)$ as a satisfiability problem $\Phi$
    - if $\Phi$ is satisfiable, then
      - From the set of truth values that satisfies $\Phi$, a solution plan can be constructed, so return it and exit

# Outline

- Motivation
- Overall approach
- **Notation**
- Example: GoTo
- SAT planners
- Conclusions

# Notation: propositions

- For satisfiability problems we need to use propositional logic
- Need to encode ground atoms into propositions
  - For set-theoretic planning we encoded predicates into propositions by rewriting them as shown here:
    - Predicate: at(r1,loc1)
    - Proposition: at-r1-loc1
- For planning as satisfiability we'll do the same thing
  - But we won't bother to do a syntactic rewrite
  - Just use at(r1,loc1) itself as the proposition
- Also, we'll write plans starting at $a_0$ rather than $a_1$
  - $\pi = \langle a_0, a_1, \ldots, a_{n-1} \rangle$

# Notation: fluents

- Proposition saying a particular predicate is true in a particular state
  - at(r1,loc1,$i$) is a fluent that's true iff at(r1,loc1) is in $s_i$
  - We'll use $l_i$ to denote the fluent for literal $l$ in state $s_i$
    - e.g., if $l$ = at(r1,loc1)
      then $l_i$ = at(r1,loc1,$i$)
  - $a_i$ is a fluent saying that $a$ is the $i$'th step of $\pi$
    - e.g., if $a$ = GoTo(r1,loc2,loc1)
      then $a_i$ = GoTo(r1,loc2,loc1,$i$)

# Outline

- Motivation
- Overall approach
- Notation
- **Example: GoTo**
- SAT planners
- Conclusions

# Example (I)

- Planning domain:
  - one robot r1
  - two adjacent locations l1, l2
  - one planning operator (to move the robot from one location to another)
  - Initial state: at(r1,l1)
  - Goal state: at(r1,l2)
- Encode $(P,n)$ where $n = 1$
- Formulas in $\Phi$ implies 5 steps:

# Example (II)

1. Formula describing the Initial state as a conjunction of propositions true:
   at(r1,l1)
   Encoding: $at(r1,l1,0) \wedge \neg at(r1,l2,0)$

2. Formula describing the Goal as a conjunction of propositions true:
   at(r1,l2)
   Encoding: $at(r1,l2,1) \wedge \neg at(r1,l1,1)$

3. For every action a in A and for i = 1, …, n, a formula describing what changes a in time 0 to n-1:
   Operator: GoTo (see next slide)

# Example (III)

- Operator: GoTo $(r,l,l')$
  precond: at$(r,l)$
  effects: at$(r,l')$, $\neg$at$(r,l)$

Preconditions true in t=0 and the effects true in t=1

Encoding:

$$
\begin{aligned}
&\text{GoTo(r1,l1,l2,0)} \Rightarrow \text{at(r1,l1,0)} \wedge \text{at(r1,l2,1)} \wedge \neg\text{at(r1,l1,1)}\\
&\text{GoTo(r1,l2,l1,0)} \Rightarrow \text{at(r1,l2,0)} \wedge \text{at(r1,l1,1)} \wedge \neg\text{at(r1,l2,1)}\\
&\left.\begin{aligned}&\text{GoTo(r1,l1,l1,0)} \Rightarrow \text{at(r1,l1,0)} \wedge \text{at(r1,l1,1)} \wedge \neg\text{at(r1,l1,1)}\\
&\text{GoTo(r1,l2,l2,0)} \Rightarrow \text{at(r1,l2,0)} \wedge \text{at(r1,l2,1)} \wedge \neg\text{at(r1,l2,1)}\end{aligned}\right\}\ \text{contradictions (easy to detect)}\\
&\left.\begin{aligned}&\text{GoTo(l1,r1,l2,0)} \Rightarrow \ldots\\
&\text{GoTo(l2,l1,r1,0)} \Rightarrow \ldots\\
&\text{GoTo(l1,l2,r1,0)} \Rightarrow \ldots\\
&\text{GoTo(l2,l1,r1,0)} \Rightarrow \ldots\end{aligned}\right\}\ \text{nonsensical, and we can avoid generating them if we use data types like we did for state-variable representation}
\end{aligned}
$$

# Example (IV)

4. Complete-exclusion axiom:
   - For every pair of actions *a* and *b*, and for $i = 0, \ldots, n-1$, a formula saying they can't both be the *i*'th step of the plan, this guarantees there can be only one action at a time

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

$$\neg \text{GoTo(r1,l1,l2,0)} \vee \neg \text{GoTo(r1,l2,l1,0)}$$

# Example (V)

5. Explanatory frame axioms:
   - Formulas describing what *doesn't* change between steps $i$ and $i$+1
   - One way: explanatory frame axioms
   - For $i$ = 0, …, $n$–1, an axiom for every literal $l$
     - Says that if $l$ changes between $s_i$ and $s_{i+1}$, then the action at step $i$ must be responsible:

$$(\neg l_i \wedge l_{i+1} \Rightarrow \bigvee_{a \text{ in } A}\{a_i \,/\, l \in \text{effects}^+(a)\})$$
$$\wedge\ (l_i \wedge \neg l_{i+1} \Rightarrow \bigvee_{a \text{ in } A}\{a_i \,/\, l \in \text{effects}^-(a)\})$$

$\neg$at(r1,l1,0) $\wedge$ at(r1,l1,1) $\Rightarrow$ GoTo (r1,l2,l1,0)
$\neg$at(r1,l2,0) $\wedge$ at(r1,l2,1) $\Rightarrow$ GoTo(r1,l1,l2,0)
at(r1,l1,0) $\wedge$ $\neg$at(r1,l1,1) $\Rightarrow$ GoTo(r1,l1,l2,0)
at(r1,l2,0) $\wedge$ $\neg$at(r1,l2,1) $\Rightarrow$ GoTo(r1,l2,l1,0)

- $\Phi$ is the conjunct of all of these

# Extracting a Plan

- Suppose we find an assignment of truth values that satisfies it
  - This means $P$ has a solution of length $n$
- For $i$=1,…,$n$, there will be exactly one action $a$ such that $a_i = true$
  - This is the $i$th action of the plan
- The formula on the previous slide
  - It can be satisfied with GoTo(r1,l1,l2,0) = *true*
    - Thus ⟨GoTo(r1,l1,l2,0)⟩ is a solution for ($P$,1)
  - It's the only solution - no other way to satisfy it

# Planning

- How to find an assignment of truth values that satisfies $\Phi$?
  - Use a satisfiability algorithm
- Example: the *DPLL* algorithm
  - First need to put $\Phi$ into conjunctive normal form
    e.g., $\Phi = D \wedge (\neg D \vee A \vee \neg B) \wedge (\neg D \vee \neg A \vee \neg B) \wedge (\neg D \vee \neg A \vee B) \wedge A$
  - Write $\Phi$ as a set of *clauses* (disjuncts of literals)
    $\Phi = \{\{D\}, \{\neg D, A, \neg B\}, \{\neg D, \neg A, \neg B\}, \{\neg D, \neg A, B\}, \{A\}\}$
  - Some special cases:
    - If $\Phi = \varnothing$ then $\Phi$ is always *true*
    - If $\Phi = \{\ldots, \varnothing, \ldots\}$ then $\Phi$ is always *false* (hence unsatisfiable)
    - If $\Phi$ contains a *unit clause*, *l*, then *l* must be true in order to satisfy $\Phi$

# Outline

- Motivation
- Overall approach
- Notation
- Example: GoTo
- **SAT planners**
- Conclusions

## SAT planners

- SATPLAN: builds a GP, translates manually the graph constraints to {} axioms, then uses a SAT and if no solution is found → length increases
- BLACKBOX: combines GP and SATPLAN
- LPSAT: uses a backtrack random algorithm with a new formalism (LCNF) that combines propositional logic with a set of metric constraints
- LPG
- …

# Outline

- Motivation
- Overall approach
- Notation
- Example: GoTo
- SAT planners
- **Conclusions**

# Conclusion

- The philosophy of the algorithm is:
  - The planning problem is translated to CNF
  - Guess the length which aims to achieve the goal
  - A set of propositional clauses is generated to check the satisfiability
  - Apply algorithms for propositional logic (DPLL, WALSAT, LMTS-style)