Planning as Satisfiability

- Planning as propositional satisfiability

* Based on slides by Alan Fern, Stuart Russell and Dana Nau

Architecture of a SAT-based Planner

Diagram:
- Problem Description: Init State, Goal, Actions
- Compiler (encoding) → CNF
- Simplifier (polynomial inference) → CNF
  - Increment plan length
  - If unsatisfiable
- Solver (SAT engine/s)
  - Satisfying model
- Decoder → Plan
Propositional Satisfiability

• A formula is **satisfiable** if it is true in some model
  - e.g. $A \lor B, C$

• A formula is unsatisfiable if it is true in no models
  - e.g. $A \land \neg A$

• Testing satisfiability of CNF formulas is a famous NP-complete problem

Propositional Satisfiability

• Many problems (such as planning) can be naturally encoded as instances of satisfiability

• Thus there has been much work on developing powerful satisfiability solvers
  - these solvers work amazingly well in practice (we will touch on some later)
Encoding Planning as Satisfiability:
Basic Idea

- Bounded planning problem \((P,n)\):
  - \(P\) is a planning problem; \(n\) is a positive integer
  - Find a solution for \(P\) of length \(n\)
- Create a propositional formula that represents:
  - Initial state
  - Goal
  - Action Dynamics
  for \(n\) time steps
- We will define the formula for \((P,n)\) such that:
  1) any model (i.e. satisfying truth assignment) of the formula represent a solution to \((P,n)\)
  2) if \((P,n)\) has a solution then the formula is satisfiable

Encoding Planning Problems

- We can encode \((P,n)\) so that we consider either layered plans or totally ordered plans
  - an advantage of considering layered plans is that fewer time steps are necessary (i.e. smaller \(n\) translates into smaller formulas)
  - for simplicity we first consider totally-ordered plans
- Encode \((P,n)\) as a formula \(\Phi\) such that
  \(\langle a_0, a_1, ..., a_{n-1} \rangle\) is a solution for \((P,n)\)
  if and only if
  \(\Phi\) can be satisfied in a way that makes the fluents \(a_0, ..., a_{n-1}\) true
- \(\Phi\) will be conjunction of many other formulas …
Formulas in $\Phi$

- Formula describing the **initial state**: (let $E$ be the set of possible facts in the planning problem)
  \[
  \bigwedge\{e_0 \mid e \in s_0\} \land \bigwedge\{-e_0 \mid e \in E - s_0\}
  \]
  Describes the complete initial state (both positive and negative fact)
  
  - E.g. $\text{on}(A,B,0) \land \neg\text{on}(B,A,0)$

- Formula describing the **goal**: ($G$ is set of goal facts)
  \[
  \bigwedge\{e_n \mid e \in G\}
  \]
  says that the goal facts must be true in the final state at timestep $n$
  
  - E.g. $\text{on}(B,A,n)$

- Is this enough?
  
  - Of course not. The formulas say nothing about actions.

---

Formulas in $\Phi$

- For every action $a$ and timestep $i$, formula describing what fluents must be true if $a$ were the $i$th step of the plan:
  
  - $a_i \Rightarrow \bigwedge\{e_i \mid e \in \text{Precond}(a)\}$, $a$’s preconditions must be true
  - $a_i \Rightarrow \bigwedge\{e_{i+1} \mid e \in \text{ADD}(a)\}$, $a$’s ADD effects must be true in $i+1$
  - $a_i \Rightarrow \bigwedge\{-e_{i+1} \mid e \in \text{DEL}(a)\}$, $a$’s DEL effects must be false in $i+1$

- **Complete exclusion** axiom:
  
  - For all actions $a$ and $b$ and timesteps $i$, formulas saying $a$ and $b$ can’t occur at the same time
    
    $\neg a_i \lor \neg b_i$
  
  - this guarantees there can be only one action at a time

- Is this enough?
  
  - The formulas say nothing about what happens to facts if they are not effected by an action
  
  - This is known as the **frame problem**
Frame Axioms

- **Frame axioms:**
  - Formulas describing what doesn’t change between steps $i$ and $i+1$

- Several ways to write these (your book shows another way)
  - Here I show an alternative that typically works best in practice

- **Explanatory frame axioms**
  - One axiom for every possible fact $e$ at every timestep $i$
  - Says that if $e$ changes truth value between $s_i$ and $s_{i+1}$, then the action at step $i$ must be responsible:

$$
\neg e_i \land e_{i+1} \Rightarrow V\{a_i / e \text{ in ADD}(a)\}
$$

If $e$ became true then some action must have added it

$$
e_i \land \neg e_{i+1} \Rightarrow V\{a_i / e \text{ in DEL}(a)\}
$$

If $e$ became false then some action must have deleted it

Example

- Planning domain:
  - one robot $r1$
  - two adjacent locations $l1$, $l2$
  - one operator (move the robot)

- Encode $(P,n)$ where $n = 1$

  - Initial state:  \{at($r1,l1$)\}
    Encoding:  at($r1,l1,0$) $\land$ $\neg$at($r1,l2,0$)

  - Goal:  \{at($r1,l2$)\}
    Encoding:  at($r1,l2,1$)

  - Action Schema: see next slide
Example (continued)

- Schema: move(r, l, l')
  PRE: at(r,l)
  ADD: at(r,l')
  DEL: at(r,l)

Encoding: (for actions move(r1,l1,l2) and move(r1,l2,l1) at time step 0)

move(r1,l1,l2,0) ⇒ at(r1,l1,0)
move(r1,l1,l2,0) ⇒ at(r1,l2,1)
move(r1,l1,l2,0) ⇒ ¬at(r1,l1,1)

move(r1,l2,l1,0) ⇒ at(r1,l2,0)
move(r1,l2,l1,0) ⇒ at(r1,l1,1)
move(r1,l2,l1,0) ⇒ ¬at(r1,l2,1)

Example (continued)

- Schema: move(r, l, l')
  PRE: at(r,l)
  ADD: at(r,l')
  DEL: at(r,l)

- Complete-exclusion axiom:
  ¬move(r1,l1,l2,0) ∨ ¬move(r1,l2,l1,0)

- Explanatory frame axioms:
  ¬at(r1,l1,0) ∧ at(r1,l1,1) ⇒ move(r1,l2,l1,0)
  ¬at(r1,l2,0) ∧ at(r1,l2,1) ⇒ move(r1,l1,l2,0)
  at(r1,l1,0) ∧ ¬at(r1,l1,1) ⇒ move(r1,l1,l2,0)
  at(r1,l2,0) ∧ ¬at(r1,l2,1) ⇒ move(r1,l2,l1,0)
Complete Formula for \((P,1)\)

\[
\begin{align*}
&[ \text{at}(r1,l1,0) \land \neg \text{at}(r1,l2,0) ] \land \\
&\text{at}(r1,l2,1) \land \\
&[ \text{move}(r1,l1,l2,0) \Rightarrow \text{at}(r1,l1,0) ] \land \\
&[ \text{move}(r1,l1,l2,0) \Rightarrow \text{at}(r1,l2,1) ] \land \\
&[ \text{move}(r1,l1,l2,0) \Rightarrow \neg \text{at}(r1,l1,1) ] \land \\
&[ \text{move}(r1,l2,l1,0) \Rightarrow \text{at}(r1,l2,0) ] \land \\
&[ \text{move}(r1,l2,l1,0) \Rightarrow \text{at}(r1,l1,1) ] \land \\
&[ \text{move}(r1,l2,l1,0) \Rightarrow \neg \text{at}(r1,l2,1) ] \land \\
&[ \neg \text{move}(r1,l1,l2,0) \lor \neg \text{move}(r1,l2,l1,0) ] \land \\
&[ \neg \text{at}(r1,l1,0) \land \text{at}(r1,l1,1) \Rightarrow \text{move}(r1,l1,l2,0) ] \land \\
&[ \neg \text{at}(r1,l2,0) \land \text{at}(r1,l2,1) \Rightarrow \text{move}(r1,l1,l2,0) ] \land \\
&[ \text{at}(r1,l1,0) \land \neg \text{at}(r1,l1,1) \Rightarrow \text{move}(r1,l1,l2,0) ] \land \\
&[ \text{at}(r1,l2,0) \land \neg \text{at}(r1,l2,1) \Rightarrow \text{move}(r1,l2,l1,0) ]
\end{align*}
\]

Convert to CNF and give to SAT solver.

Extracting a Plan

- Suppose we find an assignment of truth values that satisfies \(\Phi\).
  - This means \(P\) has a solution of length \(n\)
- For \(i=0,\ldots,n-1\), there will be exactly one action \(a_i\) such that \(a_i = true\)
  - This is the \(i\)th action of the plan.
- Example (from the previous slides):
  - \(\Phi\) can be satisfied with \(\text{move}(r1,l1,l2,0) = true\)
  - Thus \(\langle \text{move}(r1,l1,l2,0) \rangle\) is a solution for \((P,0)\)
    - It's the only solution - no other way to satisfy \(\Phi\)
Supporting Layered Plans

- Complete exclusion axiom:
  - For all actions \( a \) and \( b \) and time steps \( i \) include the formula \( \neg a_i \lor \neg b_i \)
  - this guaranteed that there could be only one action at a time

- Partial exclusion axiom:
  - For any pair of incompatible actions (recall from Graphplan) \( a \) and \( b \) and each time step \( i \) include the formula \( \neg a_i \lor \neg b_i \)
  - This encoding will allowed for more than one action to be taken at a time step resulting in layered plans
  - This is advantageous because fewer time steps are required (i.e. shorter formulas)

Planning Benchmark Test Set

- Extension of Graphplan test set
- blocks world - up to 18 blocks, \( 10^{19} \) states
- logistics - complex, highly-parallel transportation domain.
  - Logistics.d:
    - 2,165 possible actions per time slot
    - \( 10^{16} \) legal configurations (\( 2^{2000} \) states)
    - optimal solution contains 74 distinct actions over 14 time slots

- Problems of this size never previously handled by general-purpose planning systems
Scaling Up Logistics Planning

What SATPLAN Shows

- General propositional reasoning can compete with state of the art specialized planning systems
  - New, highly tuned variations of DP surprising powerful
  - Radically new stochastic approaches to SAT can provide very low exponential scaling

- Why does it work?
  - More flexible than forward or backward chaining
  - Randomized algorithms less likely to get trapped along bad paths
Discussion

• How well does this work?
  ▶ Created an initial splash but by itself, not very practical without help in choosing good encoding

• However combining SatPlan with planning graphs can overcome this problem