Compilation of Planning to SAT

Yiqiao Wang
Motivation

- **Propositional SAT**: Given a Boolean formula
  - e.g., \( (P \lor Q) \land (\neg Q \lor R \lor S) \land (\neg R \lor \neg P) \), does there exist a model?
  - i.e., an assignment of truth values to the propositions that makes the formula true?

- **Lots of research on algorithms for solving it**
  - This was the very first problem shown to be NP-complete

- **IDEA:**
  - Translate classical planning problems into satisfiability problems, and solving them using highly optimized SAT solvers
Outline

- Architecture of SAT-based planning
- SAT-based planning approach
- Encoding planning problems as SAT problems
- Making encodings more efficient
- Extracting a plan
- Satisfiability algorithms
  - Systematic SAT Solvers: Davis-Putnam-Logemann-Loveland
  - Stochastic SAT Solvers: GSAT
- Discussion
Architecture of SAT-based planning system

Init State Goal Actions → Compiler → Simplifier → Solver → Decoder → Plan

Symbol Table

Increment time bound if Unsatisfiable

SAF CNF Satisfying Assignment
Architecture of SAT-based planning system Cont.

- **Compiler**
  - take a planning problem as input, guess a plan length, and generate a propositional logic formula, which if satisfied, implies the existence of a solution plan

- **Symbol table**
  - record the correspondence between propositional variables and the planning instance

- **Simplifier**
  - use fast techniques such as unit clause propagation and pure literal elimination to shrink the CNF formula

- **Solver**
  - use systematic or stochastic methods to find a satisfying assignment. If the formula is unsatisfiable, then the compiler generates a new encoding reflecting a longer plan length

- **Decoder**
  - translate the result of solver into a solution plan.
Outline

✓ Architecture of SAT-based planning
  ➢ SAT-based planning approach
  ➢ Encoding planning problems as SAT problems
  ➢ Making encodings more efficient
  ➢ Extracting a plan
  ➢ Satisfiability algorithms
    • Systematic SAT Solvers: Davis-Putnam-Logemann-Loveland
    • Stochastic SAT Solvers: GSAT
  ➢ Discussion
Planning Problem Definition

Define what a planning problem is

- **Initial State**
  Describes the facts that hold and do not hold in initial state
- **Goal State**
  Describes the facts that much hold in goal state
- **Transition function** \( \gamma: S \times A \rightarrow S \)
  - \( S \): Sets of states
  - \( A \): Set of actions
  - \( \gamma \) is encoded in terms of actions’ preconditions and effects, and exclusion axioms

Bounded planning problem \((P,n)\):

- \( P \) is a planning problem; \( n \) is a positive integer
- Find a solution for \( P \) of length \( \leq n \)
  - \( <a_0, a_1, \ldots, a_{n-1}> \) is a solution for \((P,n)\),
  - Plan length not known in advance \( \Rightarrow \) the approach needs to repeat for different tentative lengths
SAT-based Planning Approach

- Do iterative deepening:
  - 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
Fluents

If $\pi = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ is a solution for $(P,n)$, then it generates the following states:

$s_0, s_1 = \gamma(s_0, a_0), s_2 = \gamma(s_1, a_1), \ldots, s_n = \gamma(s_{n-1}, a_{n-1})$

**Fluents**: propositions that describe what’s true in each $s_i$

- $\text{at}(r1,\text{loc}_1,i)$ is a fluent that’s true iff $\text{at}(r1,\text{loc}_1)$ is in $s_i$

- We’ll use $l_i$ to denote the fluent for literal $l$ in state $s_i$
  - e.g., if $l = \text{at}(r1,\text{loc}_1)$
    then $l_i = \text{at}(r1,\text{loc}_1,i)$

- $a_i$ is a fluent saying that $a$ is the $i$’th step of $\pi$
  - e.g., if $a = \text{move}(r1,\text{loc}_2,\text{loc}_1)$
    then $a_i = \text{move}(r1,\text{loc}_2,\text{loc}_1,i)$
Outline

 ✓ Architecture of SAT-based planning
 ✓ SAT-based planning approach
   ➢ Encoding planning problems as SAT problems
   ➢ Making encodings more efficient
   ➢ Extracting a plan
   ➢ Satisfiability algorithms
     ● Systematic SAT Solvers: Davis-Putnam-Logemann-Loveland
     ● Stochastic SAT Solvers: GSAT
   ➢ Discussion
What is in $\Phi$?

- **Formula describing the initial state:**
  \[ \land \{ l_0 \mid l \in s_0 \} \land \land \{ \neg l_0 \mid l \in L - s_0 \} \]

- **Formula describing the goal state:**
  \[ \land \{ l_n \mid l \in g^+ \} \land \land \{ \neg l_n \mid l \in g^- \} \]

- **Formulas describing the preconditions and effects of actions:**
  For every action $a$ in $A$, formulas describing what changes $a$ would make if it were the $i$'th step of the plan:
  - $a_i \Rightarrow \land \{ p_i \mid p \in \text{Precond}(a) \} \land \land \{ e_{i+1} \mid e \in \text{Effects}(a) \}$

- **Formulas describing Complete exclusion:**
  - For all actions $a$ and $b$, formulas saying they cannot occur at the same time
    \[ \neg a_i \lor \neg b_i \]
  - this guarantees there can be only one action at a time

- **Formulas providing a solution to the Frame Problem**
Example

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

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

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

  - Goal: \(\{\text{at}(r1,l2)\}\)
    Encoding: \(\text{at}(r1,l2,1) \land \neg\text{at}(r1,l1,1)\)
Example (continued)

- **Action:** \texttt{move(r,l1,l2)}
  - precond: \texttt{at(r,l1)}
  - effects: \texttt{at(r,l2), \neg at(r,l1)}

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

- **Complete-exclusion axiom:**
  \[
  \neg \text{move}(r1,l1,l2,0) \lor \neg \text{move}(r1,l2,l1,0)
  \]

- **Explanatory frame axioms:**
  \[
  \begin{align*}
  \neg \text{at}(r1,l1,0) \land \text{at}(r1,l1,1) & \Rightarrow \text{move}(r1,l2,l1,0) \\
  \neg \text{at}(r1,l2,0) \land \text{at}(r1,l2,1) & \Rightarrow \text{move}(r1,l1,l2,0) \\
  \text{at}(r1,l1,0) \land \neg \text{at}(r1,l1,1) & \Rightarrow \text{move}(r1,l1,l2,0) \\
  \text{at}(r1,l2,0) \land \neg \text{at}(r1,l2,1) & \Rightarrow \text{move}(r1,l2,l1,0)
  \end{align*}
  \]
What are these “Explanatory Frame Axioms” and the “Complete Exclusion Axioms”? 
The Frame Problem

The Frame Problem:
Describing what does not change between steps $i$ and $i+1$

Two Common Solutions:
1. Classical Frame Axioms
2. Explanatory frame axioms
1. Classical Frame Axioms

- **Classical frame axioms** (McCarthy & Hayes 1969)
  - State which fluents are unaffected by a given action
  - For each action $a$, for each fluent not in effects($a$), and for each step $i$, we have: $f_i \land a_i \Rightarrow f_{i+1}$
  - Problem: if no action occurs at step $i$ nothing can be inferred about propositions at level $i+1$
  - Sol: at-least-one axiom: at least one action occurs
  - If more than one action occurs at a step, either one can be selected.
2. Explanatory frame axioms

- **Explanatory frame axioms** (Haas 1987)

  - Enumerate the set of actions that could have occurred in order to account for a state change.
  - Says that if \( f \) changes between \( s_i \) and \( s_{i+1} \), then the action at step \( i \) must be responsible:

    \[
    (\neg f_i \land f_{i+1} \Rightarrow \forall \{a_i | f \in \text{effects}^+(a)\}) \land (f_i \land \neg f_{i+1} \Rightarrow \forall \{a_i | l \in \text{effects}^-(a)\})
    \]

  - Example:

    - \( \neg \text{at}(r1,l1,0) \land \text{at}(r1,l1,1) \Rightarrow \text{move}(r1,l2,l1,0) \)
    - \( \neg \text{at}(r1,l2,0) \land \text{at}(r1,l2,1) \Rightarrow \text{move}(r1,l1,l2,0) \)
    - \( \text{at}(r1,l1,0) \land \neg \text{at}(r1,l1,1) \Rightarrow \text{move}(r1,l1,l2,0) \)
    - \( \text{at}(r1,l2,0) \land \neg \text{at}(r1,l2,1) \Rightarrow \text{move}(r1,l2,l1,0) \)
Explanatory frame axioms (cont)

- Allows parallelism
  - Two actions can be executed in parallel if
    - Their preconditions are satisfied at time \( t \)
    - Their effects do not conflict
  - Gives shorter plans – smaller encoding

- Uncontrolled parallelism is problematic
  - Can create valid plans without valid solution
    - Action \( \alpha \) has precondition \( X \) and effect \( Y \)
    - Action \( \beta \) has precondition \( \neg Y \) and effect \( \neg X \)
Explanatory frame axioms (cont)

Need Exclusion Axioms

- **Complete** exclusion axioms – *totally ordered plan*
  - Only one action occurs at a time
    \[ \neg \alpha_t \lor \neg \beta_t \]

- **Conflict** exclusion axioms – *partially ordered plan*
  - Two actions conflict if one’s precondition is inconsistent with the other’s effect
  - Conflict exclusion should be used whenever possible
Outline

- Architecture of SAT-based planning
- SAT-based planning approach
- Encoding planning problems as SAT problems
  - Making encodings more efficient
  - Extracting a plan
  - Satisfiability algorithms
    - Systematic SAT Solvers: Davis-Putnam-Logemann-Loveland
    - Stochastic SAT Solvers: GSAT
- Discussion
Architecture of SAT-based planning system

Init State → Compiler → Simplifier → Solver → Decoder → Plan

Symbol Table

Increment time bound if Unsatisfiable CNF CNF Satisfying Assignment
Space of Encodings

- **Want a compiler to quickly produce a small SAT encoding**
  - Number of variables
  - Number of clauses
  - Total number of literals summed over all clauses

- **Two factors determine these sizes:**
  - Encoding
    - Choice of **Action Representation**
      - Regular, simple split, overloaded split, or bitwise
      - Tradeoff between the number of variables and the number of clauses in the formula
    - Choice of **Frame Axioms**: classical or explanatory
  - Optimizations being used
Action Encoding

- **Regular**
  - Each ground action is represented by a different logical variable

- **Simple Operator Splitting**
  - Replace each n-ary action proposition with n unary propositions
  - Advantage: instances of each action share the same variable
    - move2(l1,i) is used to represent move(r1,l1,l2,i), can be reused to represent move(r2,l1,l2,i) – represent cases where starting location is the same

- **Overloaded Operator Splitting**
  - Allowing different actions to share the same variable

- **Bitwise**
  - Propositional variables are represented using bits
# Action Encoding

[Ernst et al, IJCAI 1997]

<table>
<thead>
<tr>
<th>Representation</th>
<th>One Propositional Variable per</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>fully-instantiated action</td>
<td>move(r1,l1,l2,i)</td>
</tr>
<tr>
<td></td>
<td>(n</td>
<td>F</td>
</tr>
<tr>
<td>Simply-split</td>
<td>fully-instantiated action’s argument</td>
<td>move1(r1,i) (\land) move1(l1,i) (\land) move1(l2,i)</td>
</tr>
<tr>
<td></td>
<td>(n</td>
<td>F</td>
</tr>
<tr>
<td>Overloaded-split</td>
<td>fully-instantiated argument</td>
<td>Act(move, i) (\land) Act1(r1, i) (\land) Act2(l1, i) (\land) Act3(l2, i)</td>
</tr>
<tr>
<td></td>
<td>(n</td>
<td>F</td>
</tr>
<tr>
<td>Bitwise</td>
<td>Binary encodings of actions</td>
<td>Bit1</td>
</tr>
<tr>
<td></td>
<td>(n</td>
<td>F</td>
</tr>
</tbody>
</table>

N – number of steps; |F| - number of fluents; |O| - number of operators; A_0 – maximum arity of predicates
Comparisons of Different Encodings

- Regular explanatory and simple splitting explanatory encodings are the smallest
  - Explanatory frame axioms are smaller
    - State only what changes, not what does not change
  - Regular explanatory encodings allow for parallel actions
    - Shorter plans
    - Conflict exclusion axioms are a subset of complete exclusion axioms.
Architecture of SAT-based planning system

Increment time bound if Unsatisfiable

Init State
Goal
Actions

Compiler → CNF

Simplifier

CNF

Solver

Decoding
Symbol Table

Satisfying Assignment

Plan
Optimizations

Optimize the CNF formula produced by a compiler

1. Compile-time optimization
   - Shrink the size of CNF formula that SAT-compiler generates

2. Adding domain-specific information (e.g., control knowledge)
   - Precondition |= action conflicts, effects |= derived effects
   - State invariant:
     • A truck is at only one location
   - Optimality: disallowing unnecessary subplans
     • Do not return a package to its original location
   - Simplifying assumptions: not logically entailed
     • Once trucks are loaded they should immediately move
Outline

- Architecture of SAT-based planning
- SAT-based planning approach
- Encoding planning problems as SAT problems
- Making encodings more efficient

- Extracting a plan

- Satisfiability algorithms
  - Systematic SAT Solvers: Davis-Putnam-Logemann-Loveland
  - Stochastic SAT Solvers: GSAT

- Discussion
Extracting a Plan

- Suppose we find an assignment of truth values that satisfies $\Phi$.
  - This means $P$ has a solution of length $n$

- For $i=1,\ldots,n$, there will be exactly one action $a$ 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 $\Phi$
Outline

✓ Architecture of SAT-based planning
✓ SAT-based planning approach
✓ Encoding planning problems as SAT problems
✓ Making encodings more efficient
✓ Extracting a plan

➢ Satisfiability algorithms
  ● Systematic SAT Solvers: Davis-Putnam-Logemann-Loveland
  ● Stochastic SAT Solvers: GSAT

➢ Discussion
SAT Algorithms

➢ How to find an assignment of truth values that satisfies $\Phi$?
  • Use a satisfiability algorithm

➢ Systematic Search
  • E.g., DP (Davis Putnam Logemann Loveland)
    backtrack search + unit propagation

➢ Local Search
  • E.g., GSAT (Selman), Walksat (Selman, Kautz & Cohen)
    greedy local search + noise to escape minima
Outline

✓ Architecture of SAT-based planning
✓ SAT-based planning approach
✓ Encoding planning problems as SAT problems
✓ Making encodings more efficient
✓ Extracting a plan
✓ Satisfiability algorithms
  ● Systematic SAT Solvers: Davis-Putnam-Logemann-Loveland
  ● Stochastic SAT Solvers: GSAT

➢ Discussion
Discussion

- Recall the overall approach:
  - 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 \(\Phi\), extract a solution plan and return it

- By itself, not very practical (takes too much memory and time)

- But it can be combined with other techniques
  - e.g., planning graphs
  - Blackbox: combines planning-graph expansion and satisfiability checking
Conclusion

- **What SATPLAN shows**
  - General SAT solvers can compete with state of the art specialized planning systems, in fact today’s SAT-based planners are among the fastest!!!

- **Why SATPLAN works**
  - More flexible than forward or backward chaining
  - Randomized algorithms less likely to get trapped on bad paths
I have reused slides from the following two sources:

- Open-Loop Planning as Satisfiability by Henry Kautz
- Aussagenlogische Erfüllbarkeitstechniken SATPlan by Ulrich Scholz