Boolean Satisfiability Based Planning and Analysis of Planning

Sections 11.5 and 11.6

Outline

- Background and overview
- The Satplan algorithm
- Converting planning problems into Boolean formulas
- Solving Booelan formulas
- Summary and analysis of planning approaches

Additional reference used for the slides:
Weld, D.S. (1999). Recent advances in AI planning.
AI Magazine, 20(2), 93-122.

What is a satisfiability problem?

- *SAT*: propositional satisfiability problem
- given a Boolean formula in CNF, find an interpretation that makes it true.
- *CNF*: conjunctive normal form, conjunction of disjunctions
- *interpretation*: assignment of truth values to literals (propositions)

SAT Example

$$(A \vee B) \wedge (\neg A \vee C)$$

Possible interpretations are:

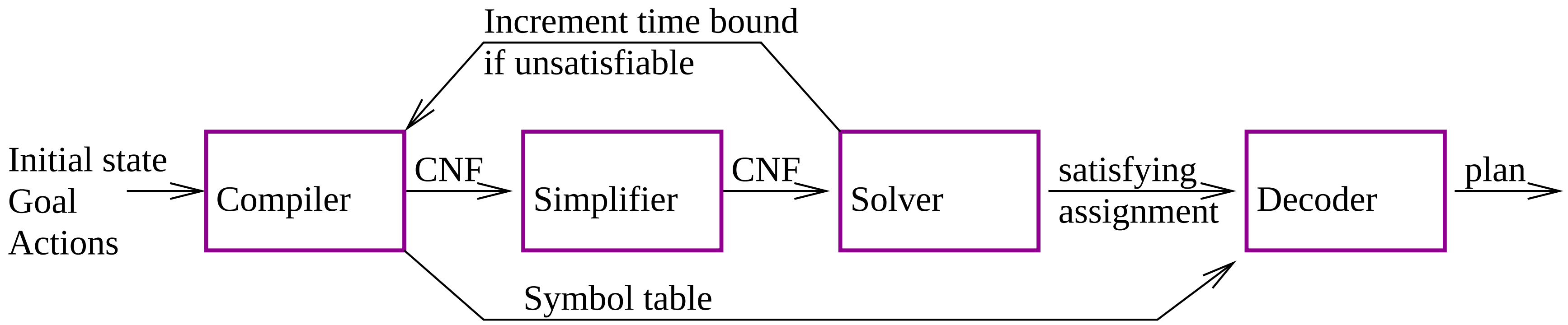
A : T, B : T, C: T
A : T, B : F, C: T
A : F, B : T, C: T
A : F, B : T, C: F

But not one that assigns F to both A and B.

SAT formulas for planning

If these are the initial conditions, these are the desired goals, which action(s) would be executed at time 0, at time 1, and so on.
An assignment would, for example, assign F to dollying at time 0, but T to wrapping at time 0.

SAT-based planning architecture



The SATPLAN algorithm

```
function SATPLAN (problem, $T_{max}$)
returns a solution, or failure
  inputs: problem, a planning problem
          $T_{max}$, an upper limit for plan length

  for $T = 0$ to $T_{max}$ do
    cnf, mapping ← TRANSLATE-TO-SAT(problem, $T$)
    assignment ← SAT-SOLVER(cnf)
    if assignment is not null then
      return EXTRACT-SOLUTION(assignment,mapping)
  return failure
```

Building CNF formulas for planning problems

- Code the initial conditions:
  $garb^0 \land cleanhands^0 \land quiet^0 \land \neg dinner^0 \land \neg present^0$
- Guess a time when the goal conditions will be true, and code the goal propositions:
  $\neg garb^2 \land dinner^2 \land present^2$

Building CNF formulas for planning problems

- Code the preconditions and effects for each action. In order for the action to be executed at time t, its preconditions must be true at time t, and the effects will take place at time $t+1$. This must be done for every time step and for every action:
  $cook^0 \rightarrow cleanhands^0 \wedge dinner^1$
  $cook^1 \rightarrow cleanhands^1 \wedge dinner^2$
  $wrap^0 \rightarrow quiet^0 \wedge present^1$
  $\ldots$

  Note that $cook^0 \rightarrow cleanhands^0 \wedge dinner^1$ will be translated into CNF as
  $\neg cook^0 \vee (cleanhands^0 \wedge dinner^1) =$
  $(\neg cook^0 \vee cleanhands^0) \wedge (\neg cook^0 \vee dinner^1)$

Building CNF formulas for planning problems

- The conditions under which a proposition does not change from time t to time $t+1$ must also be specified. Otherwise, only changed propositions can be proven, those that don't cannot be proven for subsequent times. These are called *frame axioms*.
- *Full (classical) frame axioms* say that if a proposition p was true at time t, and an action that does not affect p is executed, then p is true at time $t+1$.

  $garbage^0 \wedge cook^0 \rightarrow garbage^1$

  $\ldots$

  This must be done for every time, proposition and action that does not affect the proposition.

Building CNF formulas for planning problems

- *Explanatory frame axioms* state which actions could have caused a proposition to change:
  $garbage^0 \land \neg garbage^1 \rightarrow dolly^0 \lor carry^0$
  $\ldots$
- Full frame axioms also require the *at-least-one axioms* so ensure that an action is executed at each time step. Otherwise, there might be times where no action is executed and propositions cannot be proven for subsequent time steps.
  $cook^0 \lor wrap^0 \lor dolly^0 \lor carry^0$
  $cook^1 \lor wrap^1 \lor dolly^1 \lor carry^1$

Building CNF formulas for planning problems

- $A \rightarrow (P \wedge E)$ axioms combined with full frame axioms ensure that two actions occurring at time t lead to an identical world state at time $t + 1$. They explicitly force the propositions unaffected by an executing action to remain unchanged. Therefore, if is turns out that more than one action is executed at a time step, one will be selected. As a result, actions cannot be executed in parallel.

Building CNF formulas for planning problems

- Explanatory frame actions allow parallel actions, so one must make sure that conflicting actions are not executed in parallel.
  Such axioms are called *conflict exclusion* constraints. Two actions are conflicting if one's precondition is the negation of the other's effect. For each such action pair α, β, add clauses of the form $\neg\alpha^t \vee \neg\beta^t$:
  $\neg cook^0 \vee \neg carry^0$
  $\ldots$

Building CNF formulas for planning problems

- Sometimes *complete exclusion* axioms are used to ensure that only one action occurs at each time step, guaranteeing a totally ordered plan. Such axioms add clauses of the form $\neg\alpha^t \vee \neg\beta^t$ for each action pair α, β.

Dealing with action schemas

- If actions have parameters, all possible instantiations must be written. For instance, the action schema $fly(p, a_1, a_2)$ becomes
  $fly - plane1 - CMX - MSP^t$
  $fly - plane1 - MSP - CMX^t$
  $fly - plane2 - JFK - PIT^t$
  ...
- If there are T times, A actions, O objects, and the maximum arity of actions is P, then there are $T \times A \times O^P$ instantiations.
- Notice that the number of instantiated actions is exponential in the maximum arity of the actions.

Dealing with action schemas

- With 12 planes and 30 airports, there are $12 \times 30 \times 30 = 10,800$ fly actions at each time step. There are $10,800^2 - 10,800 = 116,629,200$ pairs for each time step, and with 10 time steps, there are 1.2 billion clauses in the complete action exclusion axioms.

Dealing with action schemas

- The number of action instantiations can be decreased by using *symbol splitting*. Each action literal is split into n literals each stating a parameter of the action. For instance, the action $fly - plane1 - CMX - MSP^t$ is represented as:
  $fly_1 - plane1^t$
  $fly_2 - CMX^t$
  $fly_3 - MSP^t$
  $\ldots$
- With symbol splitting, if there are T times, A actions, O objects, and the maximum arity of actions is P, then there are $T \times A \times P \times O$ instantiations.

Dealing with action schemas

- Notice that the number of instantiated actions is no more exponential in the maximum arity of the actions.
- If all the parameters are needed in a clause, then the clause size does not change. But irrelevant parameters can be left out resulting in a decrease in the size.

Dealing with action schemas

- The downside of symbol splitting is that parallel actions cannot be allowed. For instance the two parallel actions $fly-plane1-CMX-MSP^0$, and $fly-plane2-MSP-JFK^0$ would be represented as
  $fly_1-plane1^0 \wedge fly_2-CMX^0 \wedge fly_3-MSP^0 \wedge$
  $fly_1-plane2^0 \wedge fly_2-MSP^0 \wedge fly_3-JFK^0$

  We know $plane1$ and $plane2$ flew, but it is no longer possible to determine the origin and destination for each.

  We need to go back to using complete action exclusion axioms.

Solving SAT problems

- *Systematic solvers* perform a backtracking search in the space of possible assignments
- *Stochastic solvers* perform a random search.
- It is possible to simplify formulas before processing
  - If there are *unit clauses*, i.e., clauses with one literal, the literal should be assigned true.
  - If there are *pure literals*, i.e., those can be assigned true because such an assignment cannot make the clause false.

Unit clauses and pure literals

- Consider the following CNF formula
  $(A \vee B \vee \neg E) \wedge (B \vee \neg C \vee D) \wedge (\neg A) \wedge (B \vee C \vee E) \wedge (\neg D \vee \neg E)$
- It becomes
  $(B \vee \neg E) \wedge (B \vee \neg C \vee D) \;\wedge\; (B \vee C \vee E) \wedge (\neg D \vee \neg E)$
  after the unit clause $(\neg A)$ causes $\neg A$ to be assigned true.
- It reduces to
  $(\neg D \vee \neg E)$
  after the pure literal B is assigned true.

The DPLL Algorithm

```
function DPLL (clauses, symbols, model)
returns true or false
   inputs: clauses, the set of clauses in the CNF representation
           symbols, a list of the proposition symbols in the formula
           model, an assignment of truth values to the propositions

   if every clause in clauses is true in model then return true
   if some clause in clauses is false in model then return false
   P, value ← FIND-PURE-SYMBOL (symbols, clauses, model)
   if P is non-null then return
      DPLL(clauses, symbols - P, EXTEND ( P, value, model))
   P, value ← FIND-UNIT-CLAUSE (clauses, model)
   if P is non-null then return
      DPLL(clauses, symbols - P, EXTEND ( P, value, model))
   P ← FIRST(symbols);    rest ← REST(symbols)
   return DPLL(clauses, rest, EXTEND ( P, true, model)) or
          DPLL(clauses, rest, EXTEND ( P, false, model))
```

The GSAT Algorithm

```
function GSAT(clauses, max-restarts, max-flips)
returns a satisying model, or failure
  inputs: clauses, the set of clauses in the CNF representation
          max-restarts, the number of restarts
          max-flips, the number of flips allowed before giving up

  for $i$ = 1 to max-restarts do
    model $\leftarrow$ a randomly generated truth assignment
    for $i$ = 1 to max-flips do
      if every clause in clauses is true in model then return model
      else
        $V \leftarrow$ a variable whose change gives the largest increase in the
              number of satisfied clauses; break ties randomly
        model $\leftarrow$ model with the assignment of $V$ flipped

  return failure
```

The WALKSAT Algorithm

```
function WALKSAT (clauses, p, max-flips)
returns a satisying model, or failure
  inputs: clauses, the set of clauses in the CNF representation
          p, the probability of choosing to do a “random walk” move
,              typically around 0.5
          max-flips, the number of flips allowed before giving up

  model ← a randomly generated truth assignment
  for i = 1 to max-flips do
    if every clause in clauses is true in model then return model
    clause ← a randomly selected clause from clauses
                that is false in model
    with probability p flip the value in model of a randomly selected
      symbol from clause
    else flip whichever symbol in clause
       maximizes the number of satisfied clauses
  return failure
```

Analysis of planning

- d objects with ternary relations $\rightarrow 2^{d^3}$ propositions
- divide and conquer
  - negative interactions
    - POP: causal links
    - Graphplan: mutexes
    - Satplan: mutexes in logic
  - serializable subgoals?

    $\rightarrow$ no need to backtrack, but cannot always find the optimal plan