Boolean Satisfiability Based Planning (SATPLAN)

Section 10.4.1

Nilufer Onder
Department of Computer Science
Michigan Technological University
Outline

- Background and overview
- The Satplan algorithm
- Converting planning problems into Boolean formulas
- Solving Boolean formulas
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)
\[(A \lor B) \land (\neg A \lor C)\]

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.
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

- Initial state
- Goal
- Actions

Compiler

CNF

Simplifier

CNF

Solver

satisfying assignment

Decoder

plan

Increment time bound if unsatisfiable

Symbol table
The SATPLAN algorithm

```plaintext
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:
  \[ \text{garb}^0 \land \text{cleanhands}^0 \land \text{quiet}^0 \land \neg\text{dinner}^0 \land \neg\text{present}^0 \]

- Guess a time when the goal conditions will be true, and code the goal propositions:
  \[ \neg\text{garb}^2 \land \text{dinner}^2 \land \text{present}^2 \]
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:

- \( \text{cook}^0 \rightarrow \text{cleanhands}^0 \land \text{dinner}^1 \)
- \( \text{cook}^1 \rightarrow \text{cleanhands}^1 \land \text{dinner}^2 \)
- \( \text{wrap}^0 \rightarrow \text{quiet}^0 \land \text{present}^1 \)

\ldots

Note that \( \text{cook}^0 \rightarrow \text{cleanhands}^0 \land \text{dinner}^1 \) will be translated into CNF as

\[
\neg \text{cook}^0 \lor (\text{cleanhands}^0 \land \text{dinner}^1) = \\
(\neg \text{cook}^0 \lor \text{cleanhands}^0) \land (\neg \text{cook}^0 \lor \text{dinner}^1)
\]
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 $t + 1$.

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

... 

This must be done for every time, proposition and action that does not affect the proposition.
Explanatory frame axioms state which actions could have caused a proposition to change:

\[ \text{garbage}^0 \land \neg \text{garbage}^1 \rightarrow \text{dolly}^0 \lor \text{carry}^0 \]

\[ \ldots \]

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.

\[ \text{cook}^0 \lor \text{wrap}^0 \lor \text{dolly}^0 \lor \text{carry}^0 \]

\[ \text{cook}^1 \lor \text{wrap}^1 \lor \text{dolly}^1 \lor \text{carry}^1 \]
$A \rightarrow (P \land E)$ 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.
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 $\alpha, \beta$, add clauses of the form $\neg\alpha^t \lor \neg\beta^t$:

$\neg\text{cook}^0 \lor \neg\text{carry}^0$

...
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 $\neg \alpha^t \lor \neg \beta^t$ for each action pair $\alpha, \beta$. 
Dealing with action schemas

If actions have parameters, all possible instantiations must be written. For instance, the action schema $\text{fly}(p, a_1, a_2)$ becomes

$\text{fly} - \text{plane}1 - \text{CMX} - \text{MSP}_t$

$\text{fly} - \text{plane}1 - \text{MSP} - \text{CMX}_t$

$\text{fly} - \text{plane}2 - \text{JFK} - \text{PIT}_t$

$\ldots$

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.
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 

  \[
  \text{fly} - \text{plane1} - \text{CMX} - \text{MSP}^t
  \]

  is represented as:

  \[
  \begin{align*}
  \text{fly} & - \text{plane1}^t \\
  \text{fly} & - \text{CMX}^t \\
  \text{fly} & - \text{MSP}^t \\
  \ldots
  \end{align*}
  \]

- 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 $fly - plane_1 - CMX - MSP^0$, and $fly - plane_2 - MSP - JFK^0$ would be represented as

$fly_1 - plane_1^0 \land fly_2 - CMX^0 \land fly_3 - MSP^0 \land fly_1 - plane_2^0 \land fly_2 - MSP^0 \land fly_3 - JFK^0$

We know $plane_1$ and $plane_2$ 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.
Consider the following CNF formula

\[(A \lor B \lor \neg E) \land (B \lor \neg C \lor D) \land (\neg A) \land (B \lor C \lor E) \land (\neg D \lor \neg E)\]

It becomes

\[(B \lor \neg E) \land (B \lor \neg C \lor D) \land (B \lor C \lor E) \land (\neg D \lor \neg E)\]

after the unit clause \((\neg A)\) causes \(\neg A\) to be assigned true.

It reduces to

\[(\neg D \lor \neg E)\]

after the pure literal \(B\) is assigned true.

The DPLL (Davis Putnam Logemann Loveland) algorithm uses these operations to simplify formulas.
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 satisfying 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 ← 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
            $V$ ← a variable whose change gives the largest increase in the
            number of satisfied clauses; break ties randomly
            model ← model with the assignment of $V$ flipped
        fi
    fi
end
return failure
The WALKSAT Algorithm

function \textsc{WalkSat}(\textit{clauses}, p, \textit{max-flips})
returns a satisfying model, or \textit{failure}
inputs: \textit{clauses}, the set of clauses in the CNF representation
        \textit{p}, the probability of choosing to do a “random walk” move
        \textit{max-flips}, the number of flips allowed before giving up

\textit{model} ← a randomly generated truth assignment
for \textit{i} = 1 to \textit{max-flips} do
  if every clause in \textit{clauses} is true in \textit{model} then return \textit{model}
  \textit{clause} ← a randomly selected clause from \textit{clauses}
  that is false in \textit{model}
  with probability \textit{p} flip the value in \textit{model} of a randomly selected
  symbol from \textit{clause}
  else flip whichever symbol in \textit{clause}
  maximizes the number of satisfied clauses
return \textit{failure}
The Satplan approach demonstrates how a planning problem can be transformed into a Boolean satisfiability problem.

The choice of the SAT solver is important. SATPLAN used CHAFF and SIEGE.

Handling negative interactions
- POP: causal links
- Graphplan: mutexes
- Satplan: mutexes in logic
Sources for the slides

- AIMA textbook (3rd edition)
- AIMA slides (http://aima.cs.berkeley.edu/)