Deterministic Planning

1. *Classical planning*

- To scale up planning, a factored representation is usually used.
- Planning domain definition language (PDDL):
  - **State**: a conjunction of fluents
    * Fluents are state variables, representing variable that can change over time
    * Fluents are ground boolean variable
    * Example: At(Truck, Melbourne)
    * Data semantics are used:
      · Closed world assumption: fluents not mentioned are false
      · Unique name assumption: $Truck_1$ and $Truck_2$ are distinct
    * The following are not allowed:
      · $At(x, y) \leftarrow$ because it is not grounded
      · $\neg Poor \leftarrow$ because it is a negation
      · $At(Father(Fred), Sydney) \leftarrow$ because PDDL does not allow functions
    * Think of states as a set of fluents, manipulated using set operations
  - **Action**: PDDL specifies the result of an action in terms of what changes. What is left remains unchanged.
    * A set of action is specified by an **action schema**. A schema consists of an **action name**, a lists of all **variables** used, a **precondition** and an **effect**
    * The precondition and effects are conjunctions of literals
    * The result of executing action a in state s is a state s' which is a set of fluents formed as follows:
      · Start from s
      · Remove fluents that appear in the action's effect as negative literals: **delete list**

- Add fluents that appear in the action's effect as positive literals: **add list**

  $$RESULT(s, a) = (s - DEL(a)) \cap ADD(a)$$

- A set of action schemas defines a planning **domain**
- A specific problem is defined by adding an initial state and a goal
- The **initial state** is a conjunction of ground atoms
- The **goal state** is a conjunction of literals
- PDDL does not allow quantifiers

- Planning as state-space search
  - **Forward search**:
    - Difficulties for forward search includes:
      - State space can be very large - exponential with the number of state variables
      - Action space can be very large
    - Forward search is hopeless without good heuristics
  - **Backward relevant-state search**:
    - Only consider actions that are relevant to the goal, or current state
    - There is a set of relevant states to consider at each step
    - In backward search, we regress from a state description to a predecessor state description
      - Distinguish between state and description: in a state, every variable is assigned a value(True/False). For a ground fluents, there are 2^n ground states. For n ground fluents, there are 3^n descriptions, each fluent can be positive, negative or not mentioned.
      - Example: $\neg Poor \wedge Famous$ describes states where *Poor* is false and *Famous* is true, but other unmentioned fluents can have any values
    - Given a goal g and action a, regression from g over a gives description g':

      $$g' = (g - Add(a) \cap Precond(a))$$

- Heuristics for planning
    - **Ignore ore-conditions heuristic**: drops all pre-conditions
        - Solves relaxed problem ← Admissible
        - Any single goal fluent achievable in one step
        - Number of steps is roughly number of unsatisfied goals, except some actions can satisfy multiple goals, some may undo some goals.
    - **Ignore delete lists heuristic**
        - Ignoring delete lists allows monotonic progress towards goal
    - **Problem decomposition**
        - Divide goal into subgoals, solve subgoals, then combine them
        - If each subproblem uses an admissible heuristic, taking max is admissible
        - Assume subgoal independent: sum the cost of solving each subgoal
            - Solution optimistic when there is negative interaction: action in a one subplan deletes goal in another subplan (thus true cost > sum of cost of subplans) ← admissible
            - Solution pessimistic when there is positive interaction: action in one subplan achieves goals in another subplan ← not admissible
            - If admissible, sum is better than max
            - Admissible if an optimal solution is reuired

2. *SATPLAN*

- One way to do planning is to transform the planning problem into a **Boolean satisfiability (SAT)** problem, and solve with a SAT solver
- Solving SAT requires finding an assignment to variables that will make a Boolean formula true, or declare that no assignment exists
- SAT solvers usually takes input in **conjunctive normal form (CNF)** formulas:
    - A CNF formula is a conjunction of clauses

- A clause is a disjunction of literals
- A literals is a variable or its negation
- Any boolean formula can be converted to CNF

- Translat PDDL into SAT
  - Propositionalize the actions: replace each action schema with set of ground actions by substituting constraints for each variable
  - Define initial state: asset F^0 for every fluent F in initial state and $\neg F^0$ for every fluent not in the initial state
  - Propositionalize the goal: each goal is a conjunction constructs a disjunction over all possible ground conjunctions obtained by replacing the variable with constraints
  - Add successor-state axioms: for each fluent F, add axiom of the form:

    $$F^{t+1} \Leftrightarrow ActionCauseF^t \vee (F^t \wedge \neg ActionCauseNotF^t)$$

  - Add preconditoin axioms: for each ground action A, add axiom $A^t \Rightarrow PRE(A)^t$ (if a action is taken at time t, its preconds must have been true)
  - Add action exclusion axioms: say that every action is distinct from every other actions, i.e. only one action is allowed at each time step.
    * To do this, for every pair of actions A_i^t and A_j^t: add mutual exclusion constraint $\neg A_i^t \vee A_j^t$
    * If we want to allow parallel actions, add mutual exclusion only if pair of action really interfere with each other

Algorithm 1: SAT plan

function SATPLAN(init, transition, goal, T_{max}) returns action or failure

Input : init, transition, goal consisitute a description of the problem; T_{max}, an upper limit for plan length

Output: *solution/failure*

for $t = 0;\ i < T_{max};\ t = i + 1$ **do**
    cnf ← TRANSLATE-TO-SAT(init, transition, goal, t);
    model ← SAT-SOLVER(cnf) **if** *model is not null* **then**
        return EXTRACT-SOLUTION(model)
    end
end
return failure

- The number of steps required is not known in advance: try every value for T up to T_{max}
- Instead of Boolean SAT, can also encode problem as constraint satisfaction problem (CSP). Similar, but variable need not be binary
- PlanSAT and Bounded PlanSAT asks whether there is a solution of length k or less. Both problems are decidable for classical planning as the number of states is finite (there is no function allowed)
  - If function symbols are added, number of states becomes infinite and PlanSAT becomes semidecidable (if there exist a solution, return the solution but may not terminate when no solution exists)