Research Review:

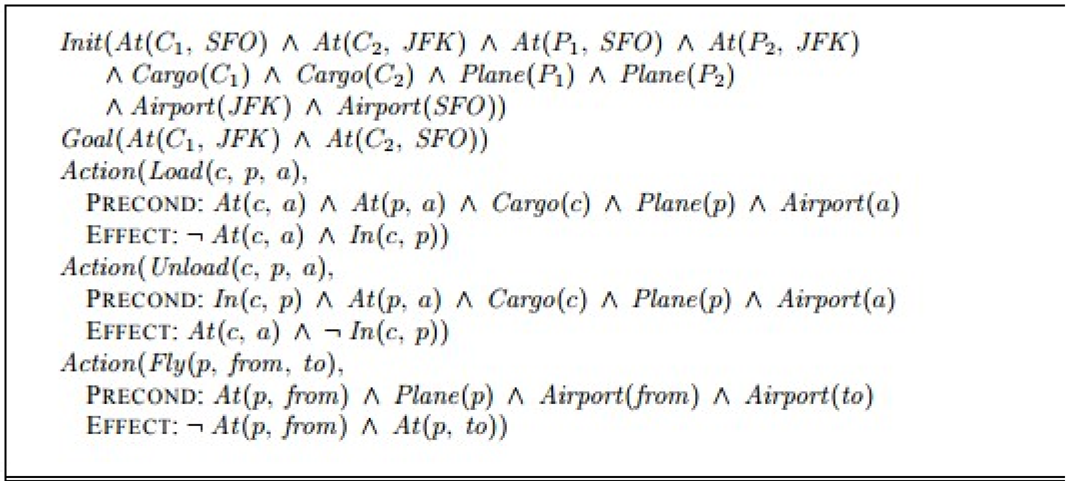
Introduction:

AI Planning, is a branch of artificial intelligence that concerns the realization of strategies or action sequences, typically for execution by intelligent agents, autonomous robots or vehicles.

STRIPS

The Standford Research Institute Problem Solver is an automated planning technique that works by executing a domain and problem to find a goal. With STRIPS, you first describe the world. You do this by providing objects, actions, preconditions, and effects. These are all the types of things you can do in the game world.

Once the world is described, you then provide a problem set. A problem consists of an initial state and a goal condition. STRIPS can then search all possible states, starting from the initial one, executing various actions, until it reaches the goal.



$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\quad \land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$

Figure 11.2 A STRIPS problem involving transportation of air cargo between airports.

ADL:

Action description language (ADL) is an automated planning and scheduling system in particular for robots. It is considered an advancement of STRIPS

As an example consider the problem of air freight transport, where certain goods must be transported from an airport to another airport by plane and where airplanes need to be loaded and unloaded.

The necessary actions would be *loading*, *unloading* and *flying*; over the descriptors one could express `In(c, p)` and `At(x, A)` whether a freight *c* is in an airplane *p* and whether an object *x* is at an airport *A*.

The actions could be defined then as follows:

```
Action (
  Load (c: Freight, p: Airplane, A: Airport)
  Precondition: At(c, A) ^ At(p, A)
  Effect: ¬At(c, A) ^ In(c, p)
)

Action (
  Unload (c: Freight, p: Airplane, A: Airport)
  Precondition: In(c, p) ^ At(p, A)
  Effect: At(c, A) ^ ¬In(c, p)
)

Action (
  Fly (p: Airplane, from: Airport, to: Airport)
  Precondition: At(p, from)
  Effect: ¬At(p, from) ^ At(p, to)
)
```

STRIPS VS ADL:

1. The STRIPS language only allows positive literals in the states, while ADL can support both positive and negative literals. For example, a valid sentence in STRIPS could be Rich ∧ Beautiful. The same sentence could be expressed in ADL as ¬Poor ∧ ¬Ugly

2. In STRIPS the unmentioned literals are false. This is called the closed-world assumption In ADL the unmentioned literals are unknown. This is known as the Open World Assumption.

3. In STRIPS we only can find ground literals in goals. For instance, Rich ∧ Beautiful. In ADL we can find quantified variables in goals. For example, ∃x At (P1, x) ∧ At(P2, x) is the goal of having P1 and P2 in the same place in the example of the blocks

4. In STRIPS the goals are conjunctions, e.g., (Rich ∧ Beautiful). In ADL, goals may involve conjunctions and disjunctions (Rich ∧ (Beautiful ∨ Smart)).

5. In STRIPS the effects are conjunctions, but in ADL conditional effects are allowed: when P:E means E is an effect only if P is satisfied

6. The STRIPS language does not support equality. In ADL, the equality predicate (x = y) is built in.

7. STRIPS does not have support for types, while in ADL it is supported (for example, the variable p : Person).

PLANNING GRAPH:

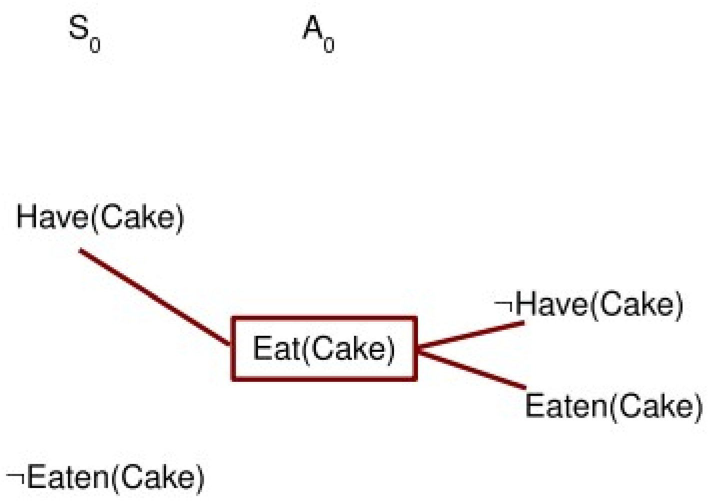
Planning graph is organised into levels Level S0: initial state, consisting of nodes representing each fluent that holds in S0 Level A0: each ground action that might be applicable in S0 Then alternate Si and Ai Si contains fluents which could hold at time i, (may be both P and ¬P); literals may show up too early but never too late Ai contains actions which could have their preconditions satisfied at i

Initial state: $Have(Cake)$

Goal: $Have(Cake) \land Eaten(Cake)$

$Eat(Cake)$:
- PRECOND: $Have(Cake)$
- EFFECT: $\neg Have(Cake) \land Eaten(Cake)$

$Bake(Cake)$:
- PRECOND: $\neg Have(Cake)$
- EFFECT: $Have(Cake)$



GRAPHPLAN:

Graphplan is an algorithm for automated planning developed by Avrim Blum and Merrick Furst in 1995. Graphplan takes as input a planning problem expressed in STRIPS and produces, if one is possible, a sequence of operations for reaching a goal state.

The name *graph*plan is due to the use of a novel *planning graph* to reduce the amount of search needed to find the solution from straightforward exploration of the *state space graph*.

Making the Plan Graph

• Start with initial conditions

• Add actions with satisfied preconditions • Add all effects of actions at previous levels

• Add maintenance actions

References:

https://machinelearnings.co/historical-intro-to-ai-planning-languages-92ce9321b538

https://en.wikipedia.org/wiki/Action_description_language

https://en.wikipedia.org/wiki/Graphplan

http://www.cs.nott.ac.uk/~psznza/G52PAS/lecture12.pdf