Artificial Intelligence

CSC348 Unit 4: Reasoning, change and planning

Syedur Rahman
Lecturer, CSE Department
North South University
syedur.rahman@wolfson.oxon.org

Artificial Intelligence: Lecture Notes

The lecture notes from the introductory lecture and this unit will be available shortly from the following URL:

- http://www.geocities.com/syedatnsu/

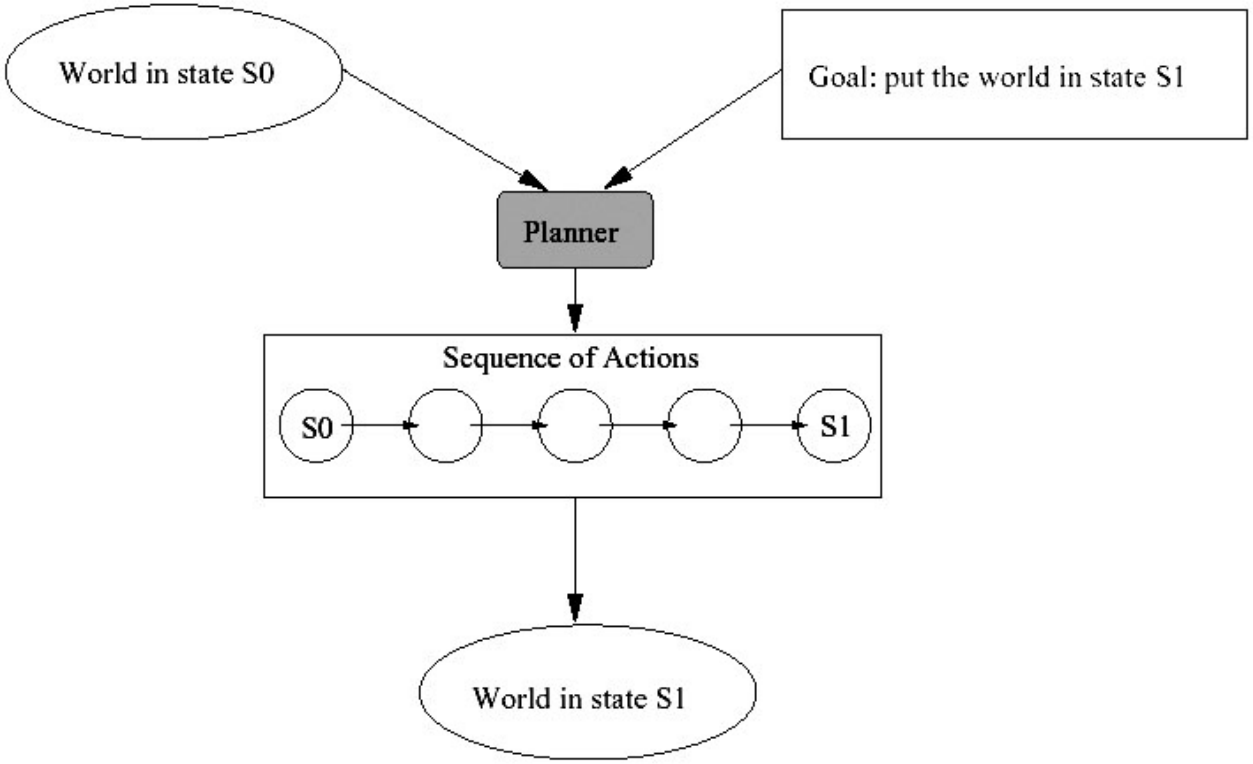
Acknowledgements

These lecture notes contain material from the following sources

- *Logical Programming and Artificial Intelligence* by S. Kapetanakis, 2004
- *Artificial Intelligence: A modern approach* by S. Russell and P. Norvig, International Edition, 2nd edition
- *Intelligent Systems* by S.Clark, 2005

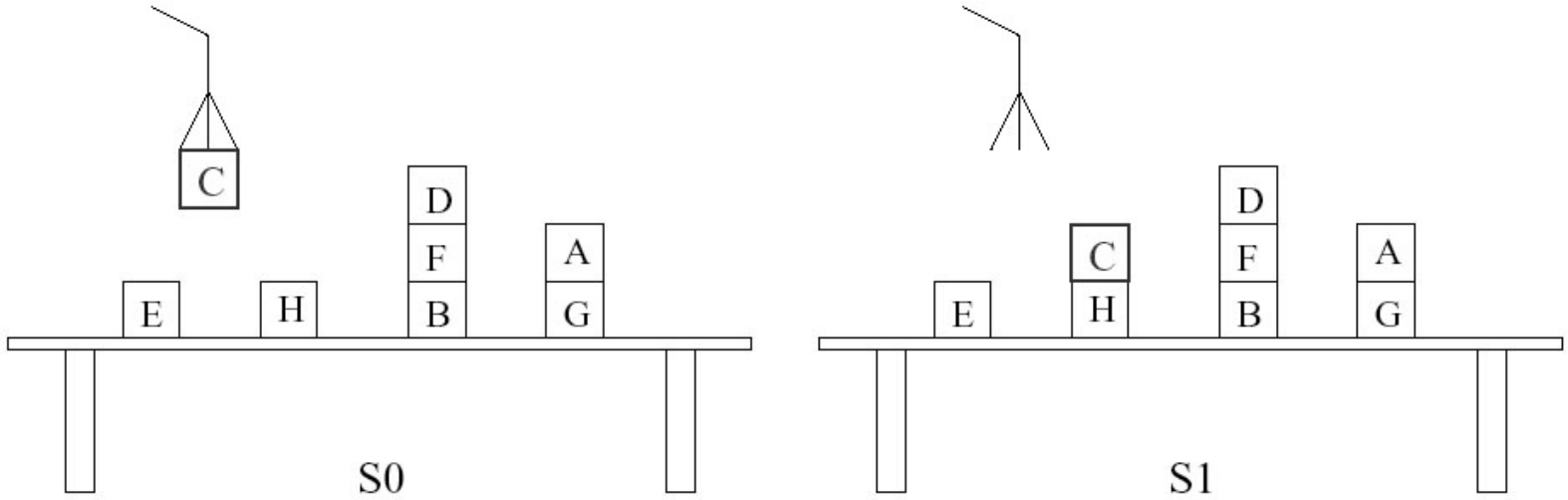
Planning intuitively

The goal in planning is to find a sequence of actions that, when executed from some starting state s_0, will achieve a set of goals or objectives.



Situation Calculus

The situation calculus is one method of representing change in the world. The language that we will use is first-order logic. It works by maintaining an up-to-date version of the state of the world, i.e. by tracking changes in the world. Example: the dynamic blocks world:



situation calculus: result(*putdown*(H), s_0) = s_1

The Result Function

The *result* function is used to describe the situation that results from performing an action in a situation:

result(α, s) stands for the situation that exists when action α is executed in situation s.

Predicates: new and old

Predicates that change the situation in the world need an extra argument to say in what situation they were true. In the blocks world:

1. *held*(*x*) becomes *held*(*x*, *s*) and
2. *on*(*x*, *table*) becomes *on*(*x*, *table*, *s*)

Predicates that do not change between situations do not need the extra argument so *block*(*x*) remains unchanged.

We also have two new predicates to represent the actions:

1. pickup(*a*) denotes the action of picking up block *a*.
2. putdown(*b*) denotes the action of putting the block that is currently held on top of *b*.

Introduction to STRIPS

The language of STRIPS is propositional or first-order logic. The state of the world is represented by a conjunction of literals, (but first-order literals must be ground and function-free)

The Closed World Assumption (CWA) holds: everything that is not mentioned in a state is assumed to be false.

The current state is represented as a conjunction of literals.

The goal is represented in STRIPS as a conjunction of positive (non-negated) literals. The goal is satisfied if the current state contains all (positive) literals in the goal.

An action in is represented by a set of preconditions that must hold before it can be executed and a set of effects that happen when it is executed.

More about STRIPS

For readability, it is common to separate the effects of an action in STRIPS into:

1. an add list which contains the positive literals
2. a delete list which contains the negative literals

Result of an action:

1. the positive literals in the effect are added to the state
2. any negative literals in the effect that match existing positive literals in the state make the positive literals disappear
3. everything else remains as it is

Exceptions:

1. positive literals already in the state are not added again.
2. negative literals that match with nothing in the state are ignored.

Goals in STRIPS **can** contain variables. They are assumed to be existentially quantified.

Blocks World Example

Predicates:

$block(x) \equiv x$ is a block

$clear(x) \equiv x$ is clear, i.e. there is nothing on top of x

$on(x, y) \equiv x$ is on top of y

Actions:

$Action(Move(b, x, y),$

PRECOND: $on(b, x) \wedge clear(b) \wedge clear(y) \wedge block(b)$
$\wedge block(y) \wedge (b \neq x) \wedge (b \neq y) \wedge (x \neq y)$

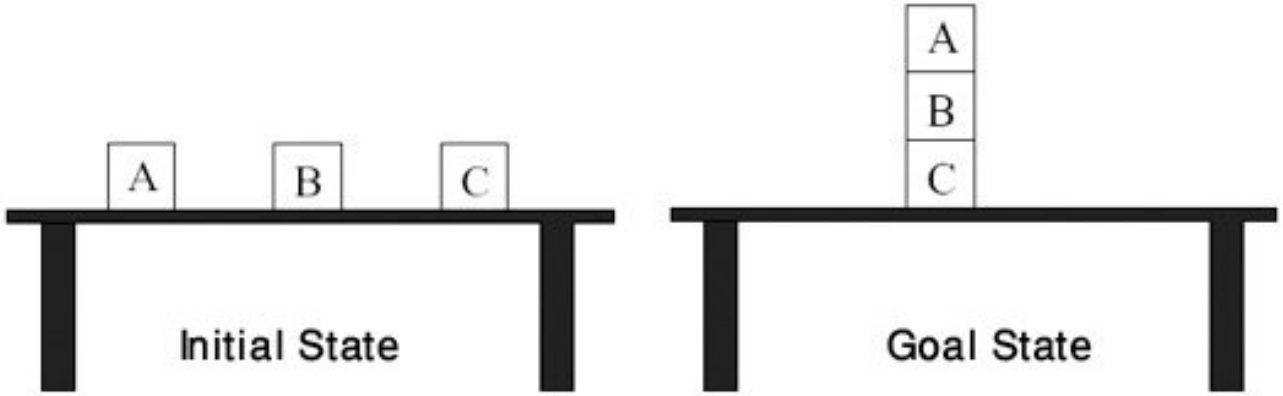
EFFECT: $on(b, y) \wedge clear(x) \wedge \neg on(b, x) \wedge \neg clear(y))$

$Action(MoveToTable(b, x),$

PRECOND: $on(b, x) \wedge clear(b) \wedge (b \neq x) \wedge block(b)$

EFFECT: $on(b, \text{table}) \wedge clear(x) \wedge \neg on(b, x))$

The next page shows the complete description of a planning problem in the blocks world using STRIPS



Initial state: $Init(on(a, table) \wedge on(b, table) \wedge on(c, table) \wedge block(a) \wedge block(b) \wedge block(c) \wedge clear(a) \wedge clear(b) \wedge clear(c))$

Goal state: $Goal(on(a, b), on(b, c))$

$Action(move(b, x, y),$

PRECOND: $on(b, x) \wedge clear(b) \wedge clear(y) \wedge block(b) \wedge (b \neq x) \wedge (b \neq y) \wedge (x \neq y)$

EFFECT: $on(b, y) \wedge clear(x) \wedge \neg on(b, x) \wedge \neg clear(y))$

$Action(movetotable(b, x),$

PRECOND: $on(b, x) \wedge clear(b) \wedge block(b) \wedge (b \neq x)$

EFFECT: $on(b, table) \wedge clear(x) \wedge \neg on(b, x))$

Planning and Search

The planning problem can be seen as a search problem.

We can move from one state of the problem to another in both a *forward* and *backward* direction because the actions are defined in terms of both *preconditions* and *effects*.

- Forward search: *progression planning*
- Backward search: *regression planning*

Progression Planning

Basic Algorithm:

1. start at the initial state.
2. an action is applicable in a state if its preconditions are satisfied
3. generate successor states from all applicable actions
4. check whether we have reached the goal i.e. whether the current state satisfies the goal of the planning problem
5. typically, the step cost in a STRIPS planner is just 1

But, one can instantly see that forward planning is very inefficient. In fact, it suffers from all the caveats of the underlying search algorithm.

Regression Planning

A better way to solve a planning problem is through backward state-space search, i.e. by starting at the goal and working our way back to the initial state.

Advantage: we need only consider moves that achieve part of the goal! As we're using STRIPS, there is no problem in finding the predecessors of a state.

Example: the goal is to have a column A->B->C->table

- it is relevant to find a way to put B on top of C
- it is also relevant to find a way to put A on top of B
- it is not relevant to find a way to put A on C
- it is not relevant to find a way to put C on A

Another property that we must insist on is *consistency* i.e. the actions we select do not undo any of the desired (goal) literals.

Regression Planning

Simplified algorithm:

- select a literal from the goal description
- select an action which contains the selected literal in its add list
- add the preconditions of the selected action to the current state and remove literals in the add list
- return success when the start state is (a subset of) the current list of literals