Classical Planning
via State-space search

Mostly from slides by

Malcolm Ryan

with slight modifications (in green).
Originals here:
http://www.cse.unsw.edu.au/~cs3431/wiki/slides/2008/malcolmr-planning/CS3431-1.ppt

Textbook

- Chapter 10 is entirely on classical planning.

What is planning?

- Planning is an AI approach to control
- Deliberation about action
- Key ideas
  - We have a **model** of the world
  - Model describes **states** and **actions**
  - Give the planner a **goal** and it outputs a **plan**
  - Aim for **domain independence**
- Planning is search

Classical planning

Classical planning is the name given to early planning systems (before about 1995)

Most of these systems are based on the Fikes & Nilsson's STRIPS notation for actions

Includes both **state-space** and **plan-space** planning algorithms.

The Model

Planning is performed based on a given model of the world.

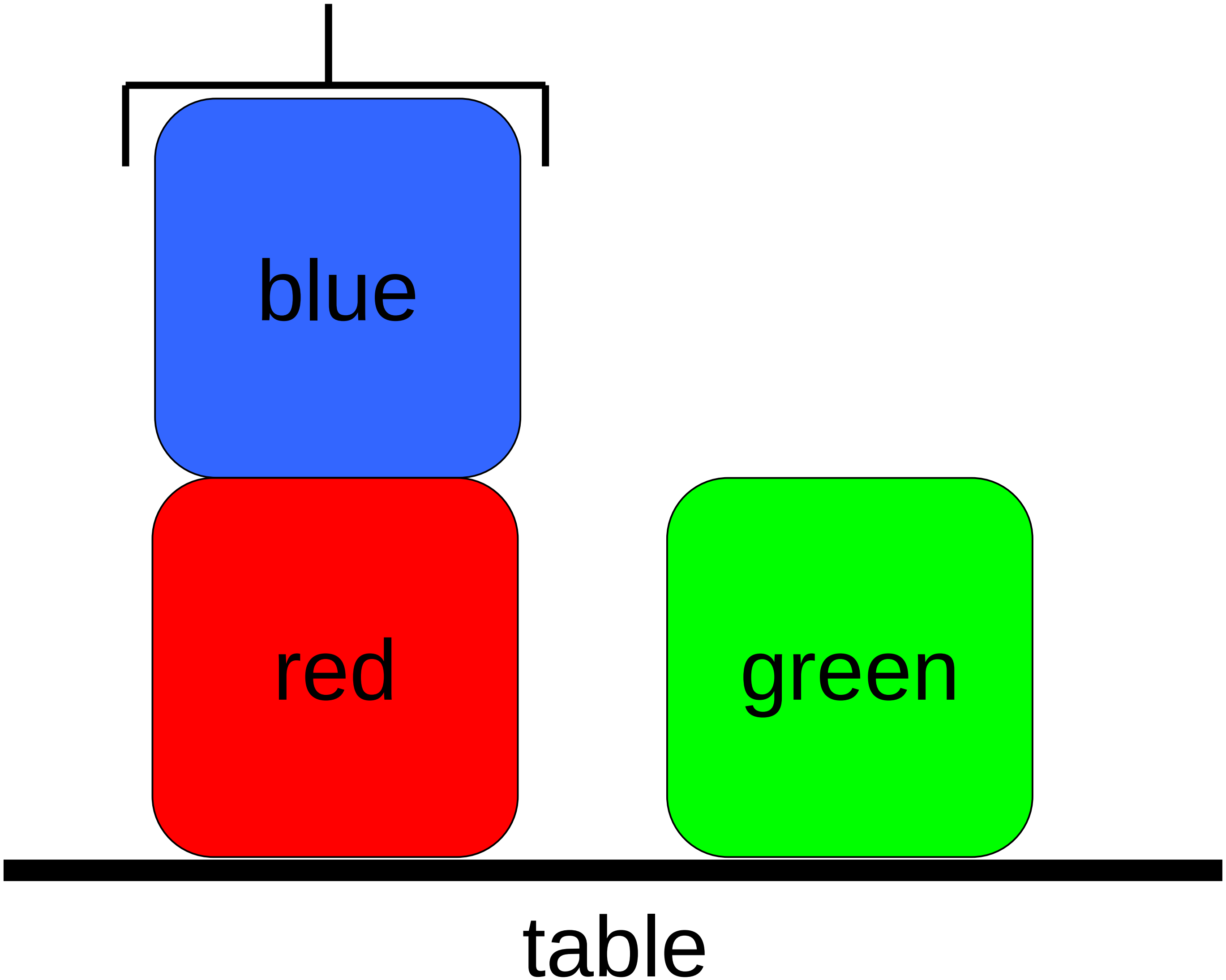
A model Σ includes:

- A set of states, *S*
- A set of actions, *A*
- A transition function, $\gamma: S \times A \rightarrow S$

Restrictions on the Model

1. S is **finite**
2. Σ is **fully observable**
3. Σ is **deterministic**
4. Σ is **static** (no external events)
5. S has a **factored representation**
6. Goals are restricted to **reachability**.
7. Plans are **ordered sequences** of actions
8. Actions have **no duration**
9. Planning is done **offline**

Example: Blocks World

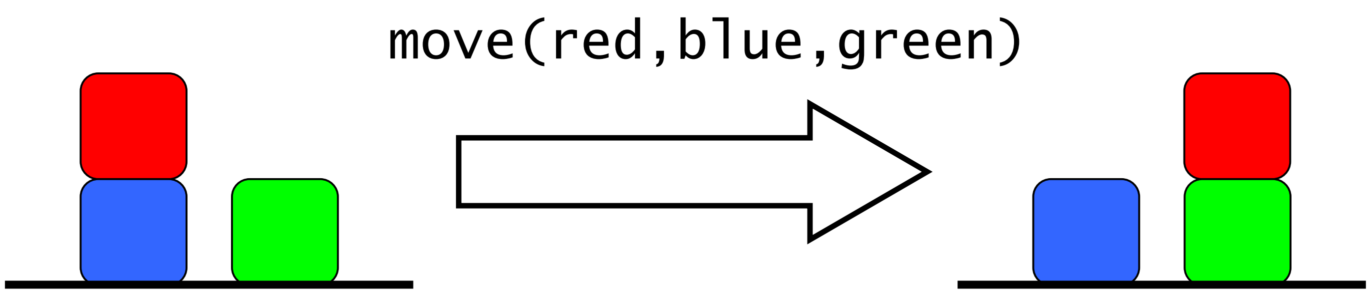


Example: Blocks World

S = the set of all different configurations of the blocks

A = the set of "move" actions

γ describes the outcomes of actions



States, actions and goals

States, actions and goals are described in the language of **symbolic logic**.

Predicates denote particular features of the world:

Eg, in the blocks world:

- **`on(`*`block1, block2`*`)`**
- **`on_table(`*`block`*`)`**
- **`clear(`*`block`*`)`**

Representing States

States are described by conjunctions of ground predicates (possibly negated).

$\texttt{on(blue, red)} \wedge \neg\, \texttt{on(green, red)}$

The **closed world assumption** (CWA) is employed to remove negative literals:

`on(blue, red)`

That is, everything that is not explicitly stated to be true is assumed to be false.

The state description is **complete**.

Representing Goals

The goal is the specification of the task

A goal is a usually conjunction of predicates:

`on(red, green)` $\wedge$ **`on_table(green)`**

The CWA does **not** apply.

So the above goal could be satisfied by:

`on(red, green)` $\wedge$ **`on_table(green)`** $\wedge$
`on(blue, red)` $\wedge$ **`clear(blue)`** $\wedge$ …

Representing Actions

Actions are described in terms of **preconditions** and **effects**.

Preconditions are predicates that must be true **before** the action can be applied.

Effects are predicates that are made true (or false) **after** the action has executed.

Sets of similar actions can be expressed as a **schema** (roughly, actions with variables).

STRIPS operators

An early but still widely used form of action description is as “STRIPS operators”.

Three parts:

Precondition	A conjunction of predicates
Add-list	The set of predicates made **true**
Delete-list	The set of predicates made **false**

Blocks World Action Schema

`move(`*`block`*`, `*`from`*`, `*`to`*`)`

Pre:

`on(`*`block, from`*`), clear(block),`
`clear(`*`to`*`)`

Add:

`on(`*`block, to`*`), `*`clear(from`*`)`

Del:

`on(`*`block, from`*`), clear(`*`to`*`)`

Blocks World Actions

Note that this action schema defines many actions:

`move(red, blue, green)`

`move(red, green, blue)`

`etc…`

We also need to define:

move_to_table(*block, from*)

move_from_table(*block, to*)

Representing Plans

A plan is simply a **sequence** of actions.eg

π = `move_from_table(red, blue),`

`move(red, blue, green),`

`move_to_table(red, green)`

We require that every action in the sequence is **applicable**, i.e. its precondition is true before it is executed.

Reasoning with STRIPS

An action *a* is **applicable** in state *s* if its precondition is satisfied, ie:

$pre^+(a) \subseteq s$

$pre^-(a) \cap s = \emptyset$

The result of executing *a* in *s* is given by:

$\gamma(s,a) = (s - del(a)) \cup add(a)$

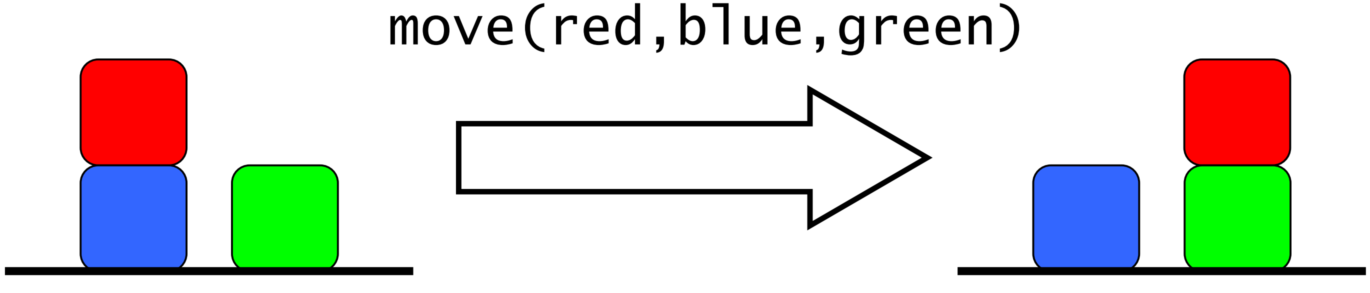
This is called **progressing** *s* through *a*

Progression example

Taking the earlier example:

s = **`on(red, blue), on_table(blue), clear(red), on_table(green), clear(green)`**

a = `move(red, blue, green)`



Progression example

1. Check action is applicable:

   `on(red, blue), clear(red), clear(green)`

2. Delete predicates from delete-list:

   `on(red, blue), on_table(blue),`
   `clear(red), on_table(green),`
   `clear(green)`

3. Add predicates from add-list:

   `on_table(blue), clear(red),`
   `on_table(green), on(red, green),`
   `clear(blue)`

Progression example 2

Consider instead the action

a = `move_from_table(blue, green)`

This has precondition:

pre(a) = **`on_table(blue)`**, **`clear(blue)`**, **`clear(green)`**

This action cannot be executed as **`clear(blue)`** is not in s.

i.e. it is not applicable

Reasoning with STRIPS

We can also **regress** states.

If we want to achieve goal *g*, using action *a*, what needs to be true beforehand?

An action *a* is **relevant** for *g,* if:

$g \cap \mathrm{add}(a) \neq \emptyset$

$\mathrm{g} \cap \mathrm{del}(a) = \emptyset$

The result of regressing *g* through *a* is:

$\gamma^{-1}(g,a) = (g - add(a)) \cup pre(a)$

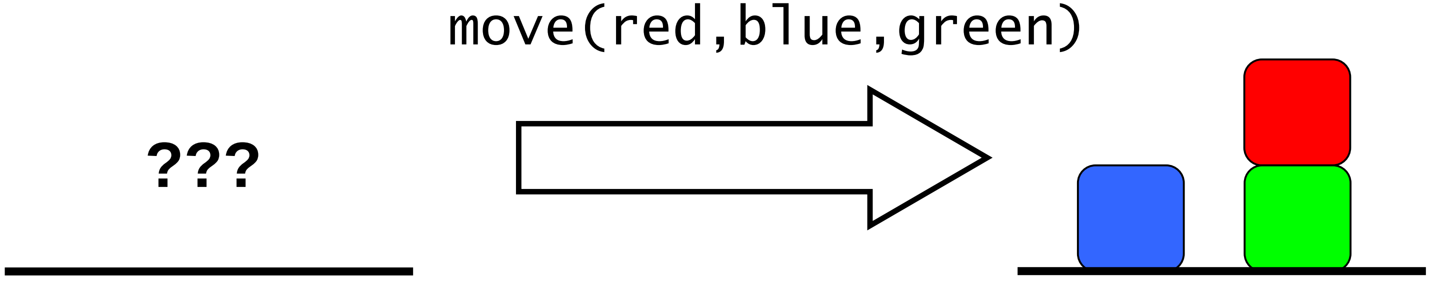
Regression Example

Take the goal:

g = **`on(red, green), on_table(blue)`**

Regress through action:

a = `move(red, blue, green)`



Regression Example

1. Check action is relevant:

   $g \cap \text{add}(a) = \{\texttt{on(red, green)}\} \neq \varnothing$

   $g \cap \text{del}(a) = \varnothing$

2. Remove predicates from add list:

   on(red, green), on_table(blue)

3. Add preconditions:

   on_table(blue), on(red, blue), clear(red), clear(green)

Regression example 2

Consider instead the action

a = `move_to_table(red, blue)`

This has effects:

add(a) = **`on_table(red), clear(blue)`**

This action is not relevant as it does not achieve any of the goal predicates, ie:

$g \cap \mathrm{add}(a) = \emptyset$

Regression Example 3

Consider instead the goal

g = **`clear(blue)`**, **`clear(green)`**

Now a = `move(red, blue, green)` achieves **`clear(blue)`** but is not relevant, as it conflicts with the goal:

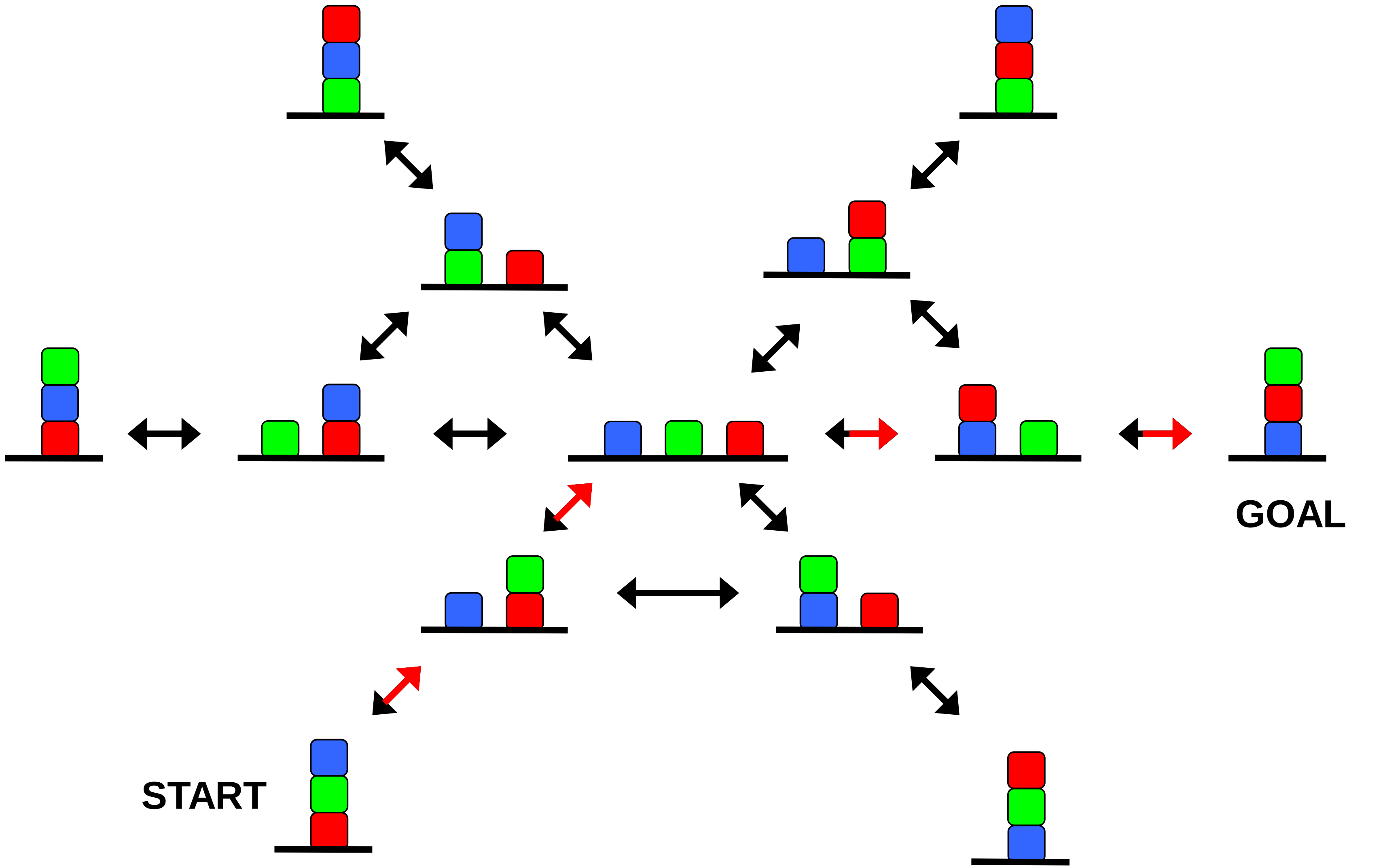
$g \cap \mathrm{del}(a) = \{\texttt{clear(green)}\} \neq \emptyset$

Planning as state-space search

Imagine a directed graph in which nodes represent states and edges represent actions.

An edge joins two nodes if there is an action that takes you from one state to the other.

Graph of state space



Forward/Backward chaining

Planning can be done as **forward** or **backward chaining**.

Forward chaining starts at the initial state and searches for a path to the goal using progression.

Backward chaining starts at the goal and searches for a path to the initial state using regression.

Forward Search

Forward-search(s, g)

if s satisfies g **then** return empty plan

$applicable = \{a \mid a$ is applicable in $s\}$

if $applicable = \varnothing$ **then** fail

choose action $a \in applicable$

$s' = \gamma(s,a)$

π' = Forward-search(s', g)

return $a.\pi'$

Backward Search

Backward-search(s, g)

if s satisfies g **then** return empty plan

relevant = $\{a \mid a$ is relevant to $g\}$

if *relevant* $= \emptyset$ **then** fail

choose action $a \in$ *relevant*

$g' = \gamma^{-1}(g,a)$

π' = Backward-search(s, g')

return π'.a

Heuristics for planning

Remember A*: we want an (under) estimate of the distance to a goal. One approach: do a quick plan (for the node to be evaluated) under relaxed constraints. For example:

- ***Ignore preconditions:*** How many steps would it take if actions had no preconditions?
- ***Ignore delete lists:*** How many steps would it take if actions had no delete lists?

Are these admissible? Do you see any issues?