Part II

# Classical Representation for Planning



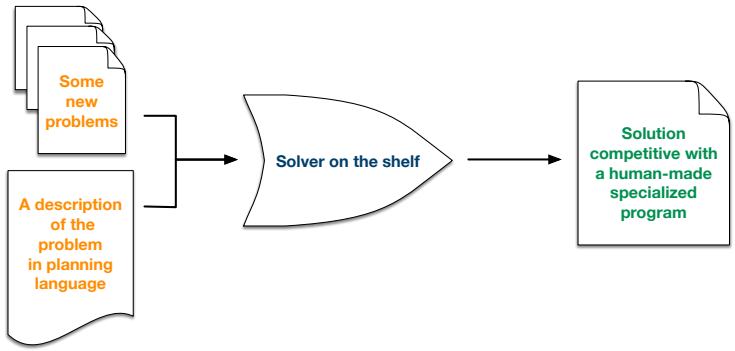
## Outline of Part II





## Reminder: AI Planning Ambition

**Ambition:** Write one program that can solve all search problem.




**Problem:** How represent a planning problem ?



## Many Classical Planning Representations

- There is three different ways to represent classical planning problems:
  1. Set theoric representation, each state of the world is a set of propositions and each action is a syntactic expression specifying which propositions belong to the state in order for the action to be applicable and which propositions the action will add or remove to change the state of the world.
  2. Classical representation, the states and the actions are like the ones described for set theoric representation except that first order literals and logical connectives are used instead propositions.
  3. State variable representation, each state is represented by a tuple of value $n$ state variables $\{x_1, \ldots, x_n\}$ and each action is represented by a partial function that map this tuple into some other tuple of values of the $n$ states.
- Each of them is equivalent in expressive power.

# I. Set-Theoretic Representation

## Planning Domains, Problems and Solutions

A set theoric representation relies on a finite set of proposition symbols based on logical fromalism that are intented to represent various propositions about the world. We need to define the basic notion of

- Planning Domain
- Planning Problem
- Planning Solution



## Planning Domains Definition

Let $L = \{p_1, \ldots, p_n\}$ be a finite set of proposition symbols. A set theoric planning domain on $L$ is a restricted state transition system $\Sigma = (S, A, \gamma)$ such that:

- $S \subseteq 2^L$, i.e., each state $s$ is a subset of $L$. If $p \in s$ then $p$ holds in $s$. Otherwise $p$ does not hold in $s$ (Closed World Assumption).
- Each action $a \in A$ is a triple of subset of $L$ written $a = (\text{precond}(a), \text{effect}^-(a), \text{effect}^+(a))$ and $\text{effect}^-(a)$ $\text{effect}^+(a)$ are disjoint.
- $S$ has the property that if $s \in S$, then, for every action $a$ that is applicable to $s$, the set ($s$ - $\text{effect}^-$(a)) $\cup$ $\text{effect}^+(a) \in S$.
- The state transition function is $\gamma(s, a) = (s$ - $\text{effect}^-$(a)) $\cup$ $\text{effect}^+(a)$ if $a \in A$ is applicable to $s \in S$.



## Planning Problem Definition

A set theoric planning problem is a triple $\mathcal{P} = (\Sigma, s_0, g)$ where

- $s_0$, the initial state, is a member of $S$
- $g \subseteq L$ is a set of propositions called goal propositions that give the requirements that a state must be satisfy in order to be a goal state. The set of goal states is $S_g = \{s \in S \mid g \subseteq s\}$.

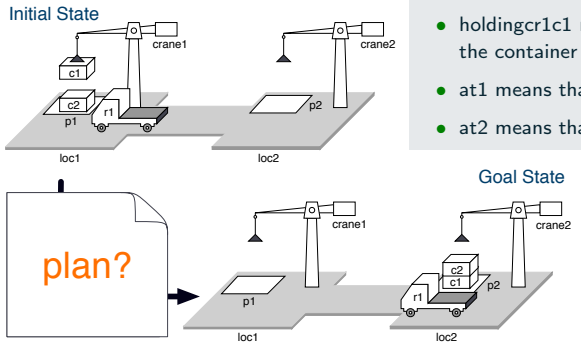
## Planning Problem Example

Here is one possible set theoric representation of the domain described in the robot docker example.

**Example (Set of propositions)**

$L = \{$onrobotc1, holdingcr1c1, at1, at2$\}$ where

- ontopc1p1 means that the container c1 is on top of p1
- onrobotc1 means that the container c1 is on the robot
- holdingcr1c1 means that crane cr1 is holding the container c1
- at1 means that the robot is at loc1
- at2 means that the robot is at loc2...





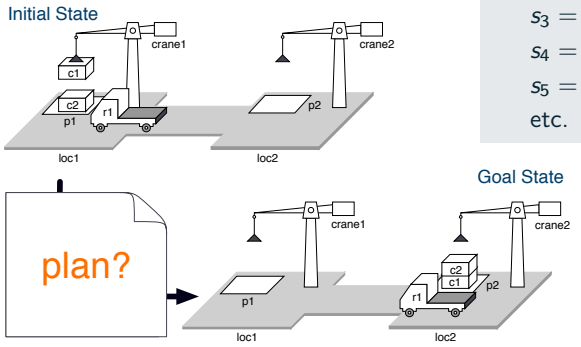

## Planning Problem Example

Here is one possible set theoric representation of the domain described in the robot docker example.

**Example (Set of states)**

$S = \{s_0, \ldots, s_5\}$ where

- $s_0 = \{holdingcr1c1, at1\}$ ;
  $s_1 = \{holdingcr1c1, at2\}$ ;
  $s_2 = \{onrobotc1, at1\}$
  $s_3 = \{holdingcr2c1, at1\}$ ;
  $s_4 = \{onrobot, at1\}$ ;
  $s_5 = \{onrobot, at2\}$,
  etc.





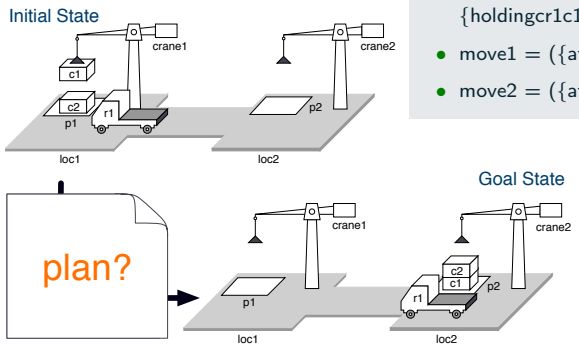

## Planning Problem Example

Here is one possible set theoric representation of the domain described in the robot docker example.

**Example (Set of actions)**

$A = \{$take, put, load, unload, move1, move2$\}$ where

- loadc1p1 = ({ontopc1p1}, {ontopc1p1}, {holdingcr1c1})
- unloadc1p1 = ({holdingcr1c1}, {holdingcr1c1}, {ontopc1p1})
- move1 = ({at2}, {at2}, {at1})
- move2 = ({at1}, {at1}, {at2})






## Plan Definition

**Definition (Plan)**

A plan is any sequence of action $\pi = \langle a_1, \ldots, a_k \rangle$, where $k \geq 0$. The length of the plan $|\pi| = k$, the number of actions. If $\pi_1 = \langle a_1, \ldots, a_k \rangle$ and $\pi_2 = \langle a'_1, \ldots, a'_j \rangle$ are plans, then their concatenation is a plan $\pi_1 \cdot \pi_2 = \langle a_1, \ldots, a_k, a'_1, \ldots, a'_j \rangle$.

The state produced by applying $\pi$ to a state $s$ is the state that is produced by applying the action of $\pi$ in the order given. We will denote this by extending the state transition function $\gamma$ as follows:

$$\gamma(s, \pi) = \begin{cases} s & \text{if } k = 0 \\ \gamma(\gamma(s, a_1), \langle a_2, \ldots, a_k \rangle) & \text{if } k > \text{ and } a_1 \text{ is applicable to } s \\ \text{undefined} & \text{otherwise} \end{cases}$$

## Plan Solution Definition

**Definition (Plan Solution)**

Let $\mathcal{P} = (\Sigma, s_0, g)$ be a planning problem. A plan $\pi$ is a solution for $\mathcal{P}$ if $g \subseteq \gamma(s_0, \pi)$.

A solution can have two proprieties:

1. A solution plan $\pi$ is redundant if there is a proper subsequence of $\pi$ that is also a solution of $\mathcal{P}$.
2. A solution plan $\pi$ is minimal if no other solution plan for $\mathcal{P}$ contains fewer actions that $\pi$.



## Plan Solution Example

**Example**

In the planning domain described previously, suppose the initial state is $s_0 = \{ontopc1p1, at1\}$ and $g = \{onrobotc1, at2\}$. Let

- $\pi_1 = \langle move2, move2 \rangle$
- $\pi_2 = \langle loadc1p1, unloadc1p1 \rangle$
- $\pi_3 = \langle loadc1p1, move2, move1, unloadc1p1, loadc1p1, move2, \rangle$
- $\pi_4 = \langle loadc1p1, move2, move1, move2 \rangle$
- $\pi_5 = \langle loadc1p1, move2, \rangle$

Then $\pi_1$ is not a solution because it is not applicable to $s_0$; $\pi_2$ is not a solution because although it is applicable to $s_0$, the resulting state is not a goal state; $\pi_3$ is a redundant solution; $\pi_4$ and $\pi_5$ are solutions but only is minimal.



## Properties of the Set Theoric Representation

1. Readability. On advantage of the set theoric representation is that it provides a more concise and readable representation of the state transition system than we would get by enumerating all of the states and transition explicitly.
2. Computation. A propositions in a state $s$ is assumed to persist in $\gamma(s, a)$ unless explicitly mentioned in the effects of $a$. The effects are defined with two subsets: effect$^-(a)$ and effect$^+(a)$. Hence, the transition function $\gamma$ and the applicability conditions of actions rely on very early computable set operations: if precond$(a) \subseteq s$;, then $\gamma(s, a) = (s-$ effect$^-(a)) \cup$ effect$^+(a)$.
3. Expressibility. A significant problem is that not every state transition system $\Sigma$ has a set theoric representation.



# II. Classical Representation

## Classical Representation

The classical representation scheme generalize the set theoric representation scheme using notation derived from first order logic.

- States are represented as set of logicals atoms that ere true or false within some interpretation.
- Actions are represented by **p** lanning operators that change the truth values of theses atoms.



## States Representation

The classical planning language is built on a first order language $\mathcal{L}$.

**Definition (State)**

A state is a set of ground atoms of $\mathcal{L}$. $\mathcal{L}$ has no function symbols. Thus the set $S$ of all possible states is guaranteed to be finite. As in the set of theoric representation scheme, an atom $p$ holds in $s$ iff $p \in s$. If $g$ is a set of literals, we will say that $s$ satisfies $g$ (denoted $s \models g$) when there is a substitution $\sigma$ such that every positive literal of $\sigma(g)$ is in $s$ and no negated literal of $\sigma(g)$ is in $s$.



## States Representation Example

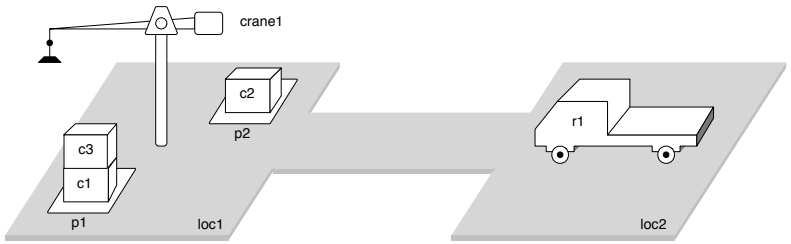



**Figure 1:** Initial state $s_0 = \{$ attached(p1, loc1), attached(p2, loc1); in(c1, p1, in(c3, p1), top(c3, p1), on(c3, c1), on(c1, pallet) in(c2, p2), top(c2, p2), on(c2,pallet), belong(crane1, loc1), empty(crane1), adjacent(loc1, loc2), adjacent(loc2, loc1), at(r1, loc2), occupied(loc2), unloaded(r1)$\}$.



## Planning Operator Definition

The planning operators define the transition function $\gamma$ of the state transition system.

**Definition (Planning Operator)**

A planning operator is a triple $o = (\text{name}(o), \text{precond}(o), \text{effects}(o))$ whose elements are follows:

- name($o$), the name of the operator, is a syntactic expression of the form $n(x_1, \ldots, x_k)$ where $n$ is a symbol called an operator symbol (n is unique in $\mathcal{L}$) and $x_1, \ldots, x_k$ are all variable symbols that appear anywhere in $o$.
- precond($o$) and effects($o$), the preconditions and effects of $o$, respectively are generalizations of the preconditions and the effects of the set theory action, i.e., instead of being sets of proposition they are sets of literals.

## Planning Operator Example

**Example (Take operator)**

The planning operator take($k$,$l$,$c$,$d$,$p$) can be defined as follow:

;; crane $k$ at location $l$ takes $c$ off of $d$ in pile $p$

take($k$,$l$,$c$,$d$,$p$)

**precond:** belong($k$,$l$), attached($p$,$l$), empty($k$), top($k$), on($c$,$d$)

**effects:** holding($k$,$c$), ¬empty($k$), ¬in($c$,$p$), ¬top($c$,$p$), ¬on($c$,$d$), top($d$,$p$)



## Action Definition

**Definition (Action)**

An action is any ground instance of planning operator. If $a$ is an action and $s$ is a state such that $\text{precond}^+(a) \subseteq s$ and $\text{precond}^-(a) \cap s = \emptyset$, then $a$ is applicable to $s$, and the result of applying $a$ to $s$ is the state:

$$\gamma(s,a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$$

Thus, like in set theoric planning, state transitions can easily be computed using set operations.



## Action Example

**Example**

The action take($crane1$,$loc1$,$c3$,$c1$,$p1$) is applicable to the state $s_0$ of the figure 42. The result is the state $s_5 = \gamma(s_0,$ take($crane1$,$loc1$,$c3$,$c1$,$p1$)) shown by the figure below.

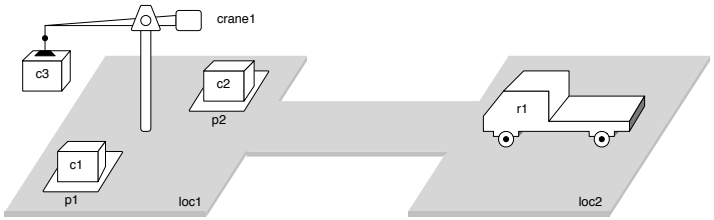



**Figure 2:** $s_5 = \{$ attached(p1, loc1), in(c1, p1), top(c1,p1), on(c1, pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), holding(crane1,c3), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc2), occupied(loc2), unloaded(r1)$\}$.



## Classical Planning Domains Definition

**Definition (Classical Planning Domain)**

Let $\mathcal{L}$ be a first order language that has finitely many predicate symbols and constraint symbols. A classical planning domain in $\mathcal{L}$ is a restricted state transition system $\Sigma = (S, A, \gamma)$ such that:

- $S \subseteq 2^{\text{all ground atoms of}\mathcal{L}}$
- $A = \{$all ground instances of the operators in $\mathcal{O}\}$ where $\mathcal{O}$ is a set of operators as defined earlier
- $\gamma(s,a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$ if $a \in A$ is applicable to $s \in S$ and otherwise $\gamma(s,a)$ is undefined
- $S$ is closed under $\gamma$, i.e., if $s \in S$, then for every action $a$ that is applicable to $s$, $\gamma(s,a) \in S$.

## Classical Planning Problems Definition

**Definition (Classical Planning Problem)**

A classical planning problem is a triple $\mathcal{P} = (\mathcal{O}, s_0, g)$ where:

- $\mathcal{O}$ is the set of planning operators
- $s_0$, the initial state, is any state in $S$
- $g$, the goal, is any set of ground literals
- $S_g = \{s \in S \mid s \text{ satisfies } g\}$



## Plan Example

**Example**

Consider the following plan:

$$\pi_1 = \langle \text{ take(crane1, loc1,c3,c1,p1), move(r1,loc2,loc1), load(crane1,loc1,c3,r1) } \rangle$$

This plan is applicable to the state $s_0$ shown in figure 42 producing the state $s_6$. We verify that

$$g_1 = \{\text{loaded(r1,c3), at(r1,loc1)}\}$$

is included in $s_6$.



## Action Example

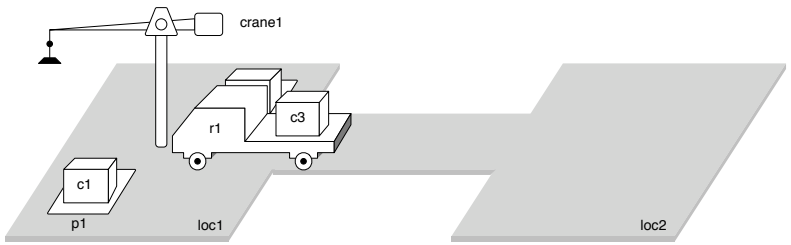



**Figure 3:** $s_6 = \{$ attached(p1, loc1), in(c1, p1), top(c1,p1), on(c1, pallet), attached(p2,loc1), in(c2,p2), top(c2,p2), on(c2,pallet), belong(crane1,loc1), empty(crane1), adjacent(loc1,loc2), adjacent(loc2,loc1), at(r1,loc1), occupied(loc1), loaded(r1)$\}$.



## Extending the Classical Representation

Classical planning formalism is very restricted, extensions to it are needed in order to describe interesting domains. The most important extensions are :

- Typing variables
- Conditional Planning Operators
- Quantified expression
- Disjunctive preconditions
- Axiomatic Inference
- etc.

A planning langage, called PDDL, has been developed to express all these extensions (PDDL stands for Planning Domain Description Langage).

# III. State Variable Representation

## State Variables

- State variables representation are équivalent to the previous ones
- The main motivation here is to rely on functions instead of logical relations
- Consider as example the relation at(r1, $l$) that hold in a state $s$ if and only if the robot r1 is in a location location $l$:
  - The robot can be at only one location at the same time, e.g, we cannot have in the same state at(r1, loc1) and at(r1, loc2)
  - It cound be avantageaous to reprensent this relation using a function that map the set of states into the set of locations
    - $rloc_{r1} : S \rightarrow locations$
    - $rloc_{r1}(s)$ gives the unique location of r1 in a state $s$.
    - $rloc_{r1}$ is a state variable



## Operators and Actions

The definition of operators and actions differ slightly.

**Definition (Planning Operator)**

A planning operator is a triple $o = (\text{name}(o), \text{precond}(o), \text{effects}(o))$ whose elements are follows:

- name($o$), the name of the operator, is a syntactic expression of the form $n(x_1, \ldots, x_k)$ where $n$ is a symbol called an operator symbol (n is unique in $\mathcal{L}$) and $x_1, \ldots, x_k$ are all variable symbols that appear anywhere in $o$.
- precond($o$) is a set of expressions on state variables and relations.
- effects($o$) is the set of assignement of values to state variables of the form $x(t_1, \ldots, t_k) \leftarrow t_{k+1}$, where each $t_i$ is a terme in the apppropriate range.



## Planning Operator Example

**Example (Move operator)**

The planning operator move($r$,$l$,$m$) can be defined as follow:

;; robot $r$ at location $l$ move to an adjacent location $m$

move($r$,$l$,$m$)

**precond:** rloc($r$) = $l$ , adjacent($l$,$m$)

**effects:** rloc($r$) $\leftarrow$ $m$

## Domains and Problems (1/2)

By extention planning domains and problems are defined as follows:

**Definition (Planning Domains)**

A Planning domain is a restricted state-transition system $\Sigma = (S, A, \gamma)$ such that:

- $S \subseteq \Pi_{x \in X D_x}$, where $D_x$ is the tange of the ground state variable $x$; a stat $s$ is noted $s = \{(x = c) \mid x \in X\}$, where $w \in D_x$.
- $A = \{allgroundinstanceofoperators\}$; an action $q$ is applicable to a state $s$ iff every expression $(x = c)$ in precond$(a)$ is also in $s$.
- $\gamma(s, a) = \{(x = c) \mid x \in X\}$, where $c$ is a specified assignement $w \leftarrow c$ in effect$(a)$, otherwise $(x = c) \in s$.
- $S$ is closed under $\gamma$.



## Domains and Problems (2/2)

**Definition (Planning Problem)**

A Planning problem is a triple $\mathcal{P} = (\Sigma, s_0, g)$, where $s_0$ is an initial state in $S$ and the goal $g$ is a set of expressions on the state variables in $X$.



# IV. An introduction to PDDL

## What is PDDL ?

PDDL = **P**lanning **D**omain **D**escription **L**anguage

- standard encoding language for "classical" planning tasks

Components of a PDDL planning task:

- **Objects:** Things in the world that interest us.
- **Predicates:** Properties of objects that we are interested in; can be true or false.
- **Initial state:** The state of the world that we start in.
- **Goal specification:** Things that we want to be true.
- **Actions/Operators:** Ways of changing the state of the world.

## How to Put the Pieces Together

Planning tasks specified in PDDL are separated into two files:

1. A **domain file** for predicates and actions.
2. A **problem file** for objects, initial state and goal specification.



## Domain Files

Domain files look like this:

```
(define (domain <domain name>)
   <PDDL code for predicates>
   <PDDL code for first action>
   [...]
   <PDDL code for last action>
)
```

`<domain name>` is a string that identifies the planning domain.

Examples on the web: Logistics, Depots, Gripper, Blocksworld, etc.



## Problem Files

Problem files look like this:

```
(define (problem <problem name>)
   (:domain <domain name>)
   <PDDL code for objects>
   <PDDL code for initial state>
   <PDDL code for goal specification>
)
```

`<problem name>` is a string that identifies the planning task, e.g. gripper with 4 balls to move.

`<domain name>` must match the domain name in the corresponding domain file.

Examples on the web: Logistics, Depots, Gripper, Blocksworld, etc.



## Running Example: Gripper task with four balls

There is a robot that can move between two rooms and pick up or drop balls with either of his two arms. Initially, all balls and the robot are in the first room. We want the balls to be in the second room.

- **Objects:** The two rooms, four balls and two robot arms.
- **Predicates:** Is $x$ a room? Is $x$ a ball? Is ball $x$ inside room $y$? Is robot arm $x$ empty? [...]
- **Initial state:** All balls and the robot are in the first room. All robot arms are empty. [...]
- **Goal specification:** All balls must be in the second room.
- **Actions/Operators:** The robot can move between rooms, pick up a ball or drop a ball.

## Gripper task: Objects

**Objects:**

- Rooms: `rooma`, `roomb`
- Balls: `ball1`, `ball2`, `ball3`, `ball4`
- Robot arms: `left`, `right`

**In PDDL:**

```
(:objects rooma roomb - room
          ball1 ball2 ball3 ball4 - ball
          left right - gripper)
```



## Gripper task: Predicates

**Predicates:**

- `at-robby(`$x$`)` - true iff $x$ is a room and the robot is in $x$
- `at-ball(`$x$`, `$y$`)` - true iff $x$ is a ball, $y$ is a room, and $x$ is in $y$
- `free(`$x$`)` - true iff $x$ is a gripper and $x$ does not hold a ball
- `carry(`$x$`, `$y$`)` - true iff $x$ is a gripper, $y$ is a ball, and $x$ holds $y$

**In PDDL:**

```
(:predicates (at-robby ?x - room)
             (at-ball ?x -room ?y - ball)
             (free ?x - gripper) (carry ?x - gripper ?y - ball))
```



## Gripper task: Initial state

**Initial state:**

- `free(left)` and `free(right)` are true.
- `at-robby(rooma)`, `at-ball(ball1, rooma)`, ... are true.
- Everything else is false.

**In PDDL:**

```
(:init (free left) (free right)
       (at-robby rooma)
       (at-ball ball1 rooma) (at-ball ball2 rooma)
       (at-ball ball3 rooma) (at-ball ball4 rooma))
```



## Gripper task: Goal specification

**Goal specification:**

- `at-ball(ball1, roomb)`, ..., `at-ball(ball4, roomb)` must be true.
- Everything else we don't care about.

**In PDDL:**

```
(:goal (and (at-ball ball1 roomb)
            (at-ball ball2 roomb)
            (at-ball ball3 roomb)
            (at-ball ball4 roomb))
```

## Gripper task: Movement operator

### Action/Operator:

- **Description:** The robot can move from $x$ to $y$.
- **Precondition:** `at-robby(`$x$`)` are true.
- **Effect:** `at-robby(`$y$`)` becomes true and `at-robby(`$x$`)` becomes false. Everything else doesn't change.

### In PDDL:

```
(:action move
   :parameters (?x - room ?y - room)
   :precondition (and (at-robby ?x))
   :effect       (and (at-robby ?y) (not (at-robby ?x))))
```



## Gripper task: Pick-up operator

### Action/Operator:

- **Description:** The robot can pick up $x$ in $y$ with $z$.
- **Precondition:** `at-ball(`$x$`, `$y$`)`, `at-robby(`$y$`)` and `free(`$z$`)` are true.
- **Effect:** `carry(`$z$`, `$x$`)` becomes true, `at-ball(`$x$`, `$y$`)` and `free(`$z$`)` become false. Everything else doesn't change.

### In PDDL:

```
(:action pick-up
   :parameters (?x - ball ?y - room ?z - gripper)
   :precondition (and (at-ball ?x ?y) (at-robby ?y) (free ?z))
   :effect       (and (carry ?z ?x)
                      (not (at-ball ?x ?y)) (not (free ?z))))
```



## Gripper task: Drop operator

### Action/Operator:

- **Description:** The robot can drop $x$ in $y$ from $z$.
- **Precondition:** `at-robby(`$y$`)`, `carry(`$z$`, `$x$`)` are true.
- **Effect:** `at-ball(`$x$`, `$y$`)` and `free(`$z$`)` becomes true, and `carry(`$z$`, `$x$`)` become false. Everything else doesn't change.

### In PDDL:

```
(:action drop
   :parameters (?x - ball ?y -room ?z - gripper)
   :precondition (and (carry ?z ?x) (at-robby ?y))
   :effect       (and (at-ball ?x ?y) (free ?z)
                      (not (carry ?z ?x))))
```



## A Note on Action Effects

Action effects can be more complicated than seen so far.

They can be universally quantified:

```
(forall (?v1 ... ?vn)
        <effect>)
```

They can be conditional:

```
(when <condition>
      <effect>)
```

They can have cost.

They can have duration and time constrainst....

## Further readings

## Further readings

V. Lifschitz
**On the semantics of STRIPS.**
Reasoning about actions and plans 1-9, Morgan Kaufmann, 1987

B. Nebel
**On the compatibility and expressive power of propositional planning formalism.**
Journal of Artificial Intelligence Research 12:271-315, 2000

D. McDermott
**PDDL, the Planning Domain Definition Language.**
Technical report. Yale Center for Computational Vision and Control, 1998