Building a Manufacturing
Robot Software System
ENPM809B

University of Maryland

Task-level Planning

Zeid Kootbally
zeidk@umd.edu

March 12, 2018

Table of Contents

Prerequisites

- Planner (`popf-tif-clp`) already compiled and `popf3-clp` generated.
- `export PATH=$PATH:<absolute path to popf3-clp>`

Expectations

At the end of this lecture, you should be able to:

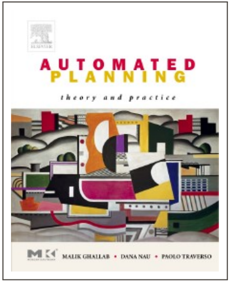
1. Understand and use the different planning components.
2. Build PDDL domain and problem files.
3. Generate a plan.

Assignment for Next 2 Weeks: Planning with qual1b

- Create domain and problem files for ARIAC.
- Generate a plan.
- Parse the plan and execute the actions.
- After part drops: Generate a new plan (initial state must be updated).

Related Reading

- Book:
  - Malik Ghallab, Dana Nau, and Paolo Traverso
    Automated Planning -- Theory and Practice
    Morgan Kaufmann Publishers, 2004
    ISBN 1-55860-856-7
- Website:
  - http://projects.laas.fr/planning/aptp/index.html



What is Planning (in AI)?

- Planning:

What is Planning (in AI)?

- Planning:
  - Explicit deliberation process that chooses and organizes actions by anticipating their outcomes and that aims at achieving some pre-stated objectives.

What is Planning (in AI)?

- Planning:
  - Explicit deliberation process that chooses and organizes actions by anticipating their outcomes and that aims at achieving some pre-stated objectives.

What is Planning (in AI)?

- Planning:
  - Explicit deliberation process that chooses and organizes actions by anticipating their outcomes and that aims at achieving some pre-stated objectives.

What is Planning (in AI)?

- Planning:
  - Explicit deliberation process that **chooses and organizes actions** by anticipating their outcomes and that aims at achieving some pre-stated objectives.

What is Planning (in AI)?

- Planning:
  - Explicit deliberation process that chooses and organizes actions by anticipating their outcomes and that aims at achieving some pre-stated objectives.

What is Planning (in AI)?

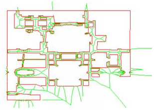
- Planning:
  - Explicit deliberation process that chooses and organizes actions by anticipating their outcomes and that aims at achieving some pre-stated objectives.

What is Planning (in AI)?

- Planning:
  - Explicit deliberation process that chooses and organizes actions by anticipating their outcomes and that aims at achieving some pre-stated objectives.
- AI Planning: computational study of this deliberation process.

What is Planning (in AI)?

- Area of research in artificial intelligence for over three decades.
- Planning techniques have been applied in a variety of tasks:
    - Space exploration
    - Manufacturing
    - Robot navigation
    - Autonomous driving



Task-level Specifications

- Task-level robot system is one that can be instructed in terms of task-level goals:
  - “Grasp part A and place it inside box B”.
- This type of specification contrasts sharply with a complete specification of each motion of the robot and not simply a description of a desired goal.
- Task-level specifications is that they are independent of the robot performing the task, whereas a motion specification is wedded to a specific robot.
- See [LPJMO89] for further reading.

Types of Planning

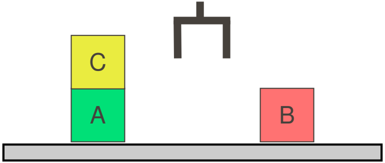
- Domain-specific: Specific representations and problems adapted to each problem.
  - Path planning.
  - Motion planning.
- Domain-independent:
  - Uses generic representations and techniques.
  - One solver can be used to solve any problem.
  - Saves money and time if you are a big company:
    - No need for a team to write a solver.
    - Can get things up and running quicker.

Planning – The Problem

- To plan, a planner needs:
  - An initial state S_i.
  - A goal state S_g.
  - Some actions A defined in a domain.
- A planner takes all of these and generates a plan.
  - A series of actions from A that will turn S_i into S_g.
  - What to do and when to do it.
  - Performs search to consider the different plans available until S_g is found.

The Blocks World Domain

- One of the most famous planning domains in artificial intelligence.
- Consists of:
  - A table.
  - A variable number of blocks.
  - An arm that can move the blocks.
- The goal is to build one or more vertical stacks of blocks.
- The rules:
  - Only one block may be moved at a time: it may either be placed on the table or placed atop another block
  - Any blocks that are, at a given time, under another block cannot be moved.



Classical Representations

- propositional representation
  - world state is set of propositions
  - action consists of precondition propositions, propositions to be added and removed
- STRIPS representation
  - like propositional representation, but first-order literals instead of propositions.
    - A literal is an atom that is either positive or negative, e.g. an atom or a negated atom.
    - An atom is a first order predicate logic.
- state-variable representation
  - state is tuple of state variables $\{x1, \ldots, xn\}$
  - action is partial function over states

STRIPS Representation

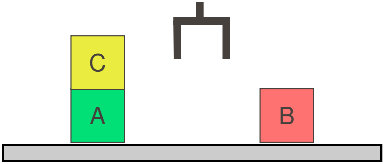
- STRIPS (Stanford Research Institute Problem Solver) is an automated planner developed by Richard Fikes and Nils Nilsson in 1971 at SRI International.
- The same name was later used to refer to the formal language of the inputs to this planner.
- One of the early systems developed in AI for planning.
- A STRIPS instance is composed of:
  - An initial state;
  - The specification of the goal states – situations which the planner is trying to reach;
  - A set of actions. For each action, the following are included:
    - preconditions (what must be established before the action is performed);
    - effects (what is established after the action is performed).

The Planning Domain Definition Language (PDDL)

- Planning has its own language for expressing problems.
- PDDL is an attempt to standardize AI planning languages.
- PDDL introduced in 1998 by D. McDermott *et al.*:
  - Lisp-like syntax.
  - Different versions: 1.2, 2.1, 2.2, 3.0, 3.1.
  - Variants: PDDL+, MAPL, NDDL, OPT, PDDL, etc.
- Defined in two parts:
  - Domain: abstract predicate definition, actions.
  - Problem: initial state and goal state.
- Standard language means:
  - Write one model, run any planner to solve it.
  - Fair benchmarking between planners, each one uses exactly the same files.

Objects in the STRIPS Representation

- Objects in the Block World domain are robots, tables, and blocks.
  - $\mathtt{robots}\{\mathtt{r1}, \mathtt{r2}, \ldots\}$
    - Can manipulate blocks.
  - $\mathtt{tables}\{\mathtt{t1}, \mathtt{t2}, \ldots\}$
    - Can hold blocks.
  - $\mathtt{blocks}\{\mathtt{a}, \mathtt{b}, \ldots\}$
    - Objects that are manipulated by the robot and can be placed on a table or on top of another block.



Objects in PDDL

```
(define (domain bwdomain)
   (:requirements :strips :typing)
    (:types
        robot ;Can manipulate blocks
        table ;Can hold blocks
        block ;Are manipulated by the robot
    )
    ...
)
```

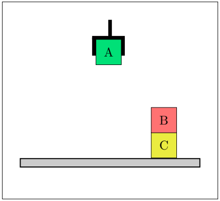
Predicates in the STRIPS Representation

- STRIPS representation is based on first order predicate logic (atom).
- Predicates are used to define the relations between objects in the domain.
  - $\underline{\texttt{on}(x, y)}$: An object $\underline{x}$ is on top of an object $\underline{y}$.
  - $\underline{\texttt{clear}(x)}$: The block $\underline{x}$ has nothing on top of it.
  - $\underline{\texttt{ontable}(x, y)}$: The block $\underline{x}$ is on table $\underline{y}$.
  - $\underline{\texttt{holding}(x, y)}$: The robot $\underline{x}$ is holding block $\underline{y}$.
  - $\underline{\texttt{handempty}(x)}$: The robot $\underline{x}$ is not holding anything.

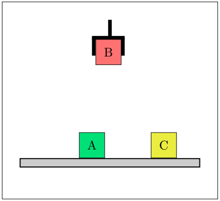
States in the STRIP Representation

- State: a set s of <u>ground atoms</u> representing what is currently true.
- A <u>ground atom</u> is an atom that does not contain any free variables (all its objects are real objects).

- `clear(b)`
- `on(b, c)`
- `holding(r1, a)`
- `ontable(c, t1)`



- `clear(a), clear(c)`
- `holding(r1, b)`
- `ontable(a, t1)`
- `ontable(c, t1)`



- <u>Closed World Assumption</u>: Atoms not listed in a state are assumed to be false.

Operators

- A planning operator in a STRIPS planning domain is a triple $\mathtt{o} = (\mathtt{name(o)}, \mathtt{precond(o)}, \mathtt{effects(o)})$ where:
  - $\mathtt{name(o)}$ is the name of the operator $\mathtt{name(o)}$ is a syntactic expression of the form $\mathtt{n}(\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{k})$, where $\mathtt{n}$ is a (unique) symbol and $\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{k}$ are all the variables that appear in $\mathtt{o}$.
  - The preconditions $\mathtt{precond(o)}$ and effects $\mathtt{effects(o)}$ of the operator are sets of literals.
- An action in a STRIPS planning domain is a ground instance of a planning operator.

STRIPS Operators: Block World Example

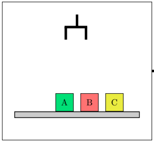
- pickup(x, y, z): Robot x picks up block y off table z.
- putdown(x, y, z): Robot x puts down block y on table z.
- stack(x, y, z): Robot x stacks block y on top of block z.
- unstack(x, y, z): Robot x unstacks block y from block z.

STRIPS Operators: pickup

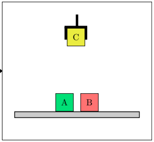
- $\mathtt{pickup}(x, y, z)$: Robot x picks up block y off table z.
  - $\mathtt{precond}: \mathtt{ontable}(y, z), \mathtt{clear}(x), \mathtt{handempty}(x)$
  - $\mathtt{effects}: \neg\mathtt{ontable}(y, z), \neg\mathtt{clear}(y), \neg\mathtt{handempty}(x), \mathtt{holding}(x, y)$
- Effects addlist: $\mathtt{holding}(x, y)$
- Effects deletelist: $\neg\mathtt{ontable}(y, z), \neg\mathtt{clear}(y), \neg\mathtt{handempty}(y)$

STRIPS Actions: pickup

- Action: a ground instance (via substitution) of an operator.
- Operator $\texttt{pickup}(\texttt{x}, \texttt{y}, \texttt{z})$
  - $\texttt{precond} : \texttt{ontable}(\texttt{y}, \texttt{z}), \texttt{clear}(\texttt{x}), \texttt{handempty}(\texttt{x})$
  - $\texttt{effects} : \neg\texttt{ontable}(\texttt{y}, \texttt{z}), \neg\texttt{clear}(\texttt{y}), \neg\texttt{handempty}(\texttt{x}), \texttt{holding}(\texttt{x}, \texttt{y})$
- Action $\texttt{pickup}(\texttt{r1}, \texttt{c}, \texttt{t1})$



$\texttt{pickup}(\texttt{r1}, \texttt{c}, \texttt{t1})$
precond: $\texttt{ontable}(\texttt{c}, \texttt{t1}), \texttt{clear}(\texttt{c}), \texttt{handempty}(\texttt{r1})$
effects: $\neg\texttt{ontable}(\texttt{c}, \texttt{t1}), \neg\texttt{clear}(\texttt{c}), \neg\texttt{handempty}(\texttt{r1}), \texttt{holding}(\texttt{r1}, \texttt{c})$



PDDL Operators: Block World Example

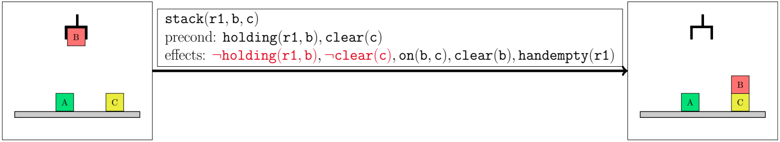
```
;;Robot r picking up a block x off the table t
(:action pickup
  :parameters (?r - robot ?b - block ?t - table)
  :precondition (and
    (ontable ?b ?t)
    (clear ?b)
    (handempty ?r))
  :effect (and
    (holding ?r ?b)
    (not (ontable ?b ?t))
    (not (clear ?b))
    (not (handempty ?r)))
)
```

STRIPS Operators: stack

- $\mathtt{stack}(x, y, z)$: Robot x stacks block y on top of block z.
  - $\mathtt{precond}: \mathtt{holding}(x, y), \mathtt{clear}(z)$
  - $\mathtt{effects}: \neg\mathtt{holding}(x, y), \neg\mathtt{clear}(z), \mathtt{on}(y, z), \mathtt{clear}(y), \mathtt{handempty}(x)$
- Effects addlist: $\mathtt{on}(y, z), \mathtt{clear}(y), \mathtt{handempty}(x)$
- Effects deletelist: $\neg\mathtt{holding}(x, y), \neg\mathtt{clear}(z)$

STRIPS Actions: stack

- Action: a ground instance (via substitution) of an operator.
- Operator $\mathtt{stack}(\mathtt{x}, \mathtt{y}, \mathtt{z})$
  - $\mathtt{precond} : \mathtt{holding}(\mathtt{x}, \mathtt{y}), \mathtt{clear}(\mathtt{z})$
  - $\mathtt{effects} : \neg\mathtt{holding}(\mathtt{x}, \mathtt{y}), \neg\mathtt{clear}(\mathtt{z}), \mathtt{on}(\mathtt{y}, \mathtt{z}), \mathtt{clear}(\mathtt{y}), \mathtt{handempty}(\mathtt{x})$
- Action $\mathtt{stack}(\mathtt{r1}, \mathtt{b}, \mathtt{c})$



PDDL Operators: Block World Example

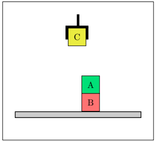
```
;;Robot r stacking a block x atop a block y
(:action stack
  :parameters (?r - robot ?b1 ?b2 - block)
  :precondition (and
    (holding ?r ?b1)
    (clear ?b2))
  :effect (and
    (on ?b1 ?b2)
    (clear ?b1)
    (handempty ?r)
    (not (holding ?r ?b1))
    (not (clear ?b2)))
)
```

Notation

- Let S be a set of literals.
  - S^{+}={atoms that appear positively in S}
  - S^{-}={atoms that appear negatively in S}
- Let a be an operator or action.
  - $\texttt{precond}^{+}(a)$={atoms that appear positively in $\texttt{precond}(a)$}
  - $\texttt{precond}^{-}(a)$={atoms that appear negatively in $\texttt{precond}(a)$}
  - $\texttt{effects}^{+}(a)$={atoms that appear positively in $\texttt{effects}(a)$}
  - $\texttt{effects}^{-}(a)$={atoms that appear negatively in $\texttt{effects}(a)$}
- Example:
  - $\texttt{stack}(x, y, z)$
    - $\texttt{precond} : \texttt{holding}(x, y), \texttt{clear}(z)$
    - $\texttt{effects} : \neg\texttt{holding}(x, y), \neg\texttt{clear}(z), \texttt{on}(y, z), \texttt{clear}(y), \texttt{handempty}(x)$
  - $\texttt{precond}^{+}(a) = \{\texttt{holding}(x, y), \texttt{clear}(z)\}$
  - $\texttt{effects}^{+}(a) = \{\texttt{on}(y, z), \texttt{clear}(y), \texttt{handempty}(x)\}$
  - $\texttt{effects}^{-}(a) = \{\neg\texttt{holding}(x, y), \neg\texttt{clear}(z)\}$

Executability

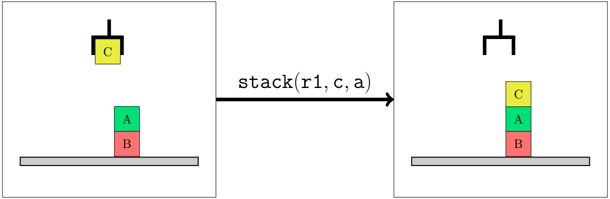
- An action $\mathtt{a}$ is executable in a state $\mathtt{s}$ if $\mathtt{s}$ satisfies $\mathtt{precond(a)}$, i.e.,
  - if $\mathtt{precond}^+(\mathtt{a}) \subseteq \mathtt{s}$ and $\mathtt{precond}^-(\mathtt{a}) \cap \mathtt{s} = \varnothing$.
- An operator $\mathtt{o}$ is applicable to $\mathtt{s}$ if there's a ground instance $\mathtt{a}$ of $\mathtt{o}$ that is executable in $\mathtt{s}$.
- Example:



  - $\mathtt{s} = \{\mathtt{ontable(b)}, \mathtt{on(a,b)}, \mathtt{holding(r1,c)}, \mathtt{clear(a)}\}$
  - $\mathtt{o} = \mathtt{stack(x,y,z)}$
  - $\mathtt{a} = \mathtt{stack(r1,c,a)}$
  - $\mathtt{stack(x,y,z)}$
    - $\mathtt{precond}: \mathtt{holding(x,y)}, \mathtt{clear(z)}$
    - $\mathtt{effects}: \neg\mathtt{holding(x,y)}, \neg\mathtt{clear(z)}, \mathtt{on(y,z)}, \mathtt{clear(y)}, \mathtt{handempty(x)}$
  - $\mathtt{stack(r1,c,a)}$
    - $\mathtt{precond}: \mathtt{holding(r1,c)}, \mathtt{clear(a)}$
    - $\mathtt{effects}: \neg\mathtt{holding(r1,c)}, \neg\mathtt{clear(a)}, \mathtt{on(c,a)}, \mathtt{clear(c)}, \mathtt{handempty(r1)}$

Result of Performing and Action

- If a is executable in s, the result of performing it is a new state given by:
  - $\gamma(\mathtt{s},\mathtt{a}) = (\mathtt{s} - \mathtt{effects}^{-}(\mathtt{a})) \cup \mathtt{effects}^{+}(\mathtt{a})$
  - Delete the negative effects and add the positive ones.
  - Example:



  - $\mathtt{s} = \{\mathtt{ontable(b)}, \mathtt{on(a,b)}, \mathtt{holding(r1,c)}, \mathtt{clear(a)}\}$
  - $\mathtt{a} = \mathtt{stack(r1,c,a)}$
    - $\mathtt{precond} : \mathtt{holding(r1,c)}, \mathtt{clear(a)}$
    - $\mathtt{effects} : \neg\mathtt{holding(r1,c)}, \neg\mathtt{clear(a)}, \mathtt{on(c,a)}, \mathtt{clear(c)}, \mathtt{handempty(r1)}$
    - $\gamma(\mathtt{s},\mathtt{a}) = \{\mathtt{ontable(b)}, \mathtt{on(a,b)}, \cancel{\mathtt{holding(r1,c)}}, \cancel{\mathtt{clear(a)}}, \mathtt{on(c,a)}, \mathtt{clear(c)}, \mathtt{handempty(r1)}\}$

Planning Domain

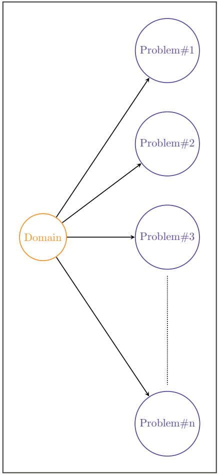
```
(define (domain bwdomain)
  (:requirements :strips :typing)
  (:types...)
  (:predicates
    ...
  )
  (:action pickup
    :parameters(...)
    :precondition(...)
    :effect(and
      ...)
  )
  (:action putdown
    :parameters(...)
    :precondition(...)
    :effect(and
      ...)
  )
   ...
)
```

Planning Problem

```
(define (problem bwproblem)
  (:domain bwdomain)
   ;;ground atoms used in the problem
   (:objects
   ...
   )

   ;;initial state
   (:init
        ...
   )

   ;;goal state
   (:goal (and
     ...
    ))
  )
```

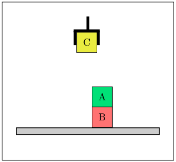


Exercise #1: Domain and Problem Files for the Block World Domain

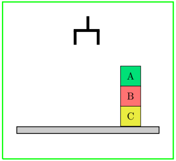
1. Write putdown and unstack actions using the STRIPS representation (examples in slides 28 and 31).

Exercise #1: Domain and Problem Files for the Block World Domain

1. Write `putdown` and `unstack` actions using the STRIPS representation (examples in slides 28 and 31).
2. Write the initial and goal states (what predicates are true in these states).
    - Initial state:



    - Goal state:



Exercise #1: Domain and Problem Files for the Block World Domain

3. Write the `putdown` action in bwdomain-exercise.pddl.
4. Write the `unstack` action in bwdomain-exercise.pddl.
5. Write the initial state in bwproblem-exercise.pddl.
6. Write the goal state in bwproblem-exercise.pddl.
7. Run the planner with these two files as follows:

```
popf3-clp bwdomain-exercise.pddl bwproblem-exercise.pddl
```

Numeric-valued Fluents in PDDL

- Sometimes, predicates are not enough to describe a planning problem.
- Many real-world problems involve continuous resources, such as fuel, money, time, space, object count: quantitative value.
- Numeric-valued fluents have been introduced in PDDL 2.1.
  - Numeric action preconditions.
  - Numeric action effects.

Numeric Preconditions

- In principle, PDDL numeric preconditions comprise:
  - A comparison operator: $>, >=, =, <=, <$ with...
  - A left- and right-hand side written using constants, the values of functions, and the operators $+, -, /, *$
- Examples in PDDL:

```
;;(quantity-of-parts-in-tray ?partstray) > 0
(> (quantity-of-parts-in-partstray ?partstray)0)


;;(quantity-of-parts-in-kit ?kit) < (2 x (capacity-of-parts-in-kit ?kit))
(< (quantity-of-parts-in-kit ?kit) (* 2 (capacity-of-parts-in-kit ?kit)))
```

Numeric Effects

- In principle, PDDL numeric effects comprise:
  - A variable to update.
  - A syntax to update the variable:
    - `assign`: $=$
    - `increase`: $+=$
    - `decrease`: $-=$
    - `scaleup`: $*=$
    - `scaledown`: $/=$
- Examples in PDDL:

```
;;increase (quantity-of-parts-in-kit ?part ?kit) by 1
(increase (quantity-of-parts-in-kit ?kit) 1)

;;(quantity-of-parts-in-kit ?part ?kit)=(capacity-of-parts-in-kit ?part ?kit)
(assign (quantity-of-parts-in-kit ?part ?kit) (capacity-of-parts-in-kit
    ?part ?kit))
```

Numeric-valued Fluents in the Domain File

- To use numeric-value fluents in PDDL:

```
(define (domain bwdomain)
  (:requirements :strips :typing :fluents)
   (:types ...)
   (:predicates ...)
   (:functions
     ;flag set to 1 when grasp is set and to 0 when not
      (grasp-set-flag)
      ;quantity of parts in a tray
      (quantity-of-parts-in-tray ?part - Part ?tray -
          PartsTray)
      ;capacity of parts in a kit
      (capacity-of-parts-in-kit ?part - Part ?kit -
          KitTray)
   )
   (:action ...)
   ...
)
```

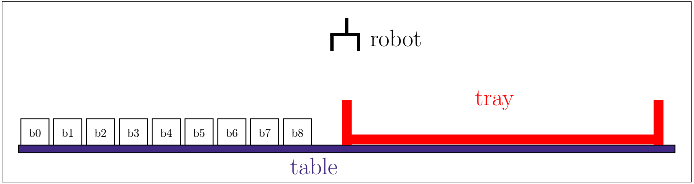
Numeric-valued Fluents in the Problem File

- Fluents can be set in the initial and/or goal states with comparison operators.

```
(:init
  (= (capacity-of-parts-in-kit part_rod_type kit_agv1) 4)
   ...
)

(:goal (and
 (= (quantity-of-parts-in-kit part_rod_type kit_agv1)
     (capacity-of-parts-in-kit part_rod_type kit_agv1))
   ...
   )
)
```

Exercise #2: Numeric-valued Fluents in PDDL



- Initial state:
  - robot x 1
  - table x 1
  - block x 9
  - tray x 1
- Goal state:
  - 7 blocks in the tray
- Requirements:
  - Only 2 actions: `pickup` and `putdown`
  - Use numeric-valued fluents

Assignment for Next 2 Weeks: Planning with qual1b

- Create domain and problem files for ARIAC.
  - Identify the object types (robot, bins, trays, ...).
  - Identify and write predicates and numeric-valued fluents.
  - Write actions (task-level commands).
  - Build part of the initial state from sensors.
    - Number of parts of a specific type by querying the bins.
  - Write the goal state (kit order).
- Generate a plan.
- Parse the plan and execute the actions.
- After part drops: Generate a new plan (initial state must be updated).
- Tips:
  - Use types of parts to pickup instead of instances of parts.
  - Combine with numeric-valued fluents to tell the planner how many parts of a specific type needs to be in the kit.
    - Type: `piston_rod_part`
    - `(= (capacity_part_in_tray piston_rod_part)3)`

References I

[LPJMO89] T. Lozano-Perez, J. L. Jones, E. Mazer, and P. A. O'Donnell, Task-level planning of pick-and-place robot motions, Computer **22** (1989), no. 3, 21–29.