Introduction to PDDL

Dr. Adrián Domínguez Díaz

Automated Planning

Index

- **The PDDL Language**
- "Gripper" domain
- PDDL Editors
- PDDL Planners
- Exercises

The PDDL Language

- Language designed to code planning problems
  - We will model the problem to be solved with a high level of abstraction

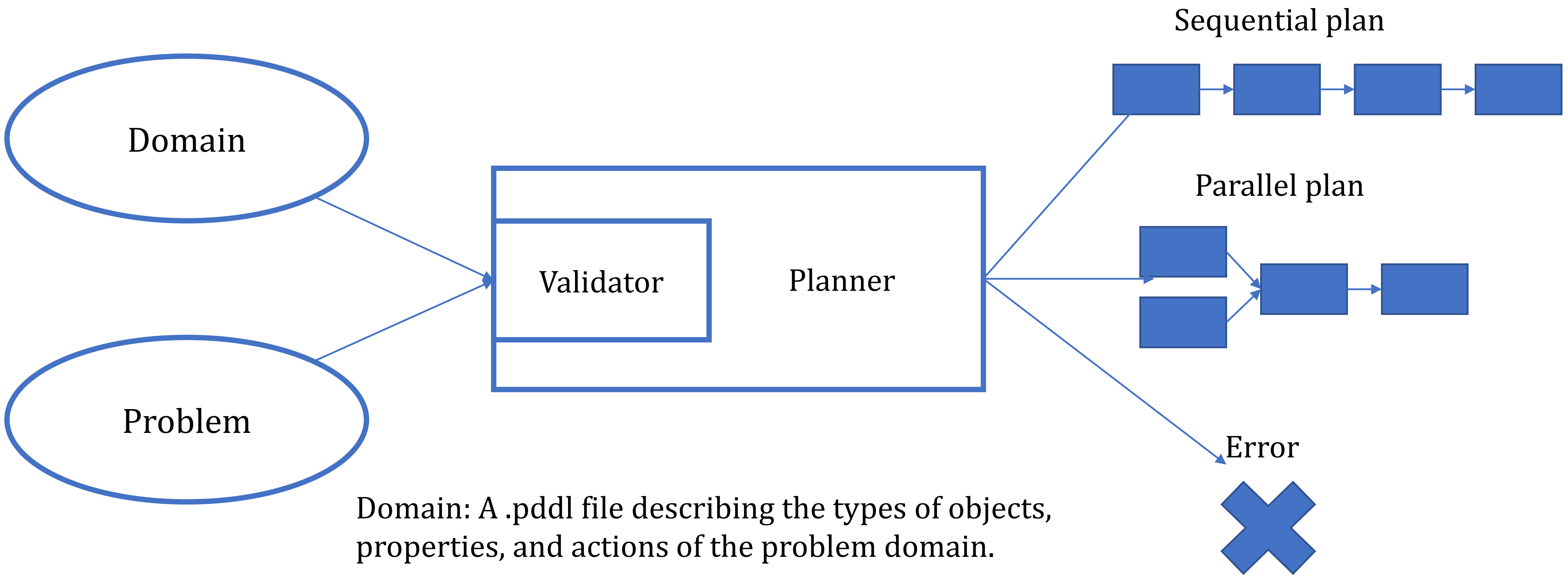
- PDDL Components:
  - Objects: Objects we want to represent
  - Predicates: Boolean properties of objects
  - Initial state: The initial state in which the problem begins
  - Goal: The final state we want to achieve
  - Actions: Actions that change the state of the problem

Features of PDDL 1.2

- PDDL 1.2 allows to define classic planning problems
  - It has the same expressiveness as classical representations
  - Classical Representation: First-Order Logic

- Inherits and extends previous planning languages
  - STRIPS: Language based on first-order logic (predicates).
  - ADL: It adds universal and existential quantifiers, among others.
  - Extensions: There are many, they add more expressiveness to PDDL.
  - Planners usually support STRIPS and some occasional extensions.

PDDL Inputs and Outputs



Domain: A .pddl file describing the types of objects, properties, and actions of the problem domain.

Problem: .pddl file with description of the objects of the problem, initial state and goal to be achieved.

PDDL Syntax: Domain

```
(define (domain <name>)
        ; Required Extensions
        (:requirements <requirement_name>)

        ; Types of objects in the domain
        (:types
                <type_name_1> ... <type_name_n> - object
                <sub_name_1> ... <sub_name_n> - <type_name_1>
        )
        ; Predicates, which define Boolean properties of objects
        (:predicates
                (<predicate_name> <argument_1> ... <argument_n>)
        )
        ; Actions, which allow you to change the value of predicates
        <PDDL list of actions>
)
```

PDDL Syntax: Actions

```
(:action <action_name>

        ; Parameters (objects) that the action receives
        :parameters (<argument_1> ... <argument_n>)

        ; Logical expression that must be met in order to execute the action
        :precondition (<logical_expression>)

        ; A logical expression indicating the effects of action on the value of predicates
        ; Positive predicates will be activated and negative predicates will be deactivated
        :effect (<logical_expression>)
)
```

PDDL Syntax: Problem

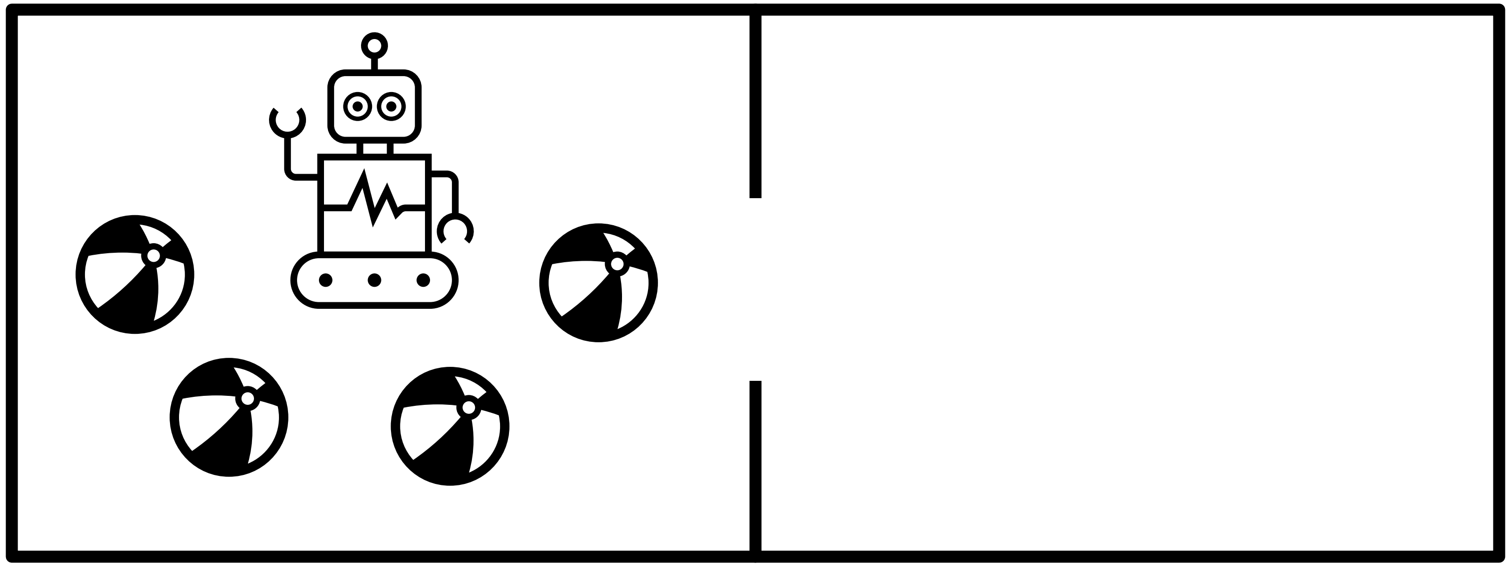
```
(:define (problem <problem name>)

        ; Domain to which the problem belongs
        (:domain <domain name>)

        ; Objects of each type existing in the problem to be solved
        (:objects <objects list>)

        ; Instantiated predicates about objects that are true at the start of the problem
        (:init <predicate list>)

        ; Boolean expression, instantiated on objects,
        ; that needs to be achieved to fix the problem
        (:goal logical_expression)
)
```

Index

- The PDDL Language
- **"Gripper" domain**
- PDDL Editors
- PDDL Planners
- Exercises

Gripper: description

- Robot that can move between adjoining rooms, as well as pick up or drop a ball with each of its two arms. There are two adjoining rooms. In one is the robot and four balls. You want the robot to take the balls to the other room.



Gripper: Domain and Problem

- Domain:
  - Object Types: Robot, Arm, Room, Ball.
  - Predicates: Am I in X room? Is the ball X in room Y? Do I have my X arm free? Do I have X Arm Y in my grip?
  - Actions: move to another room, catch the ball with a certain arm, drop the ball with a certain arm.
- Problem:
  - Items: Two rooms, two arms, four balls.
  - Initial state: In the first room is the robot and all the balls. Both arms of the robot are free.
  - Goal: We want all the balls to be in the second room.

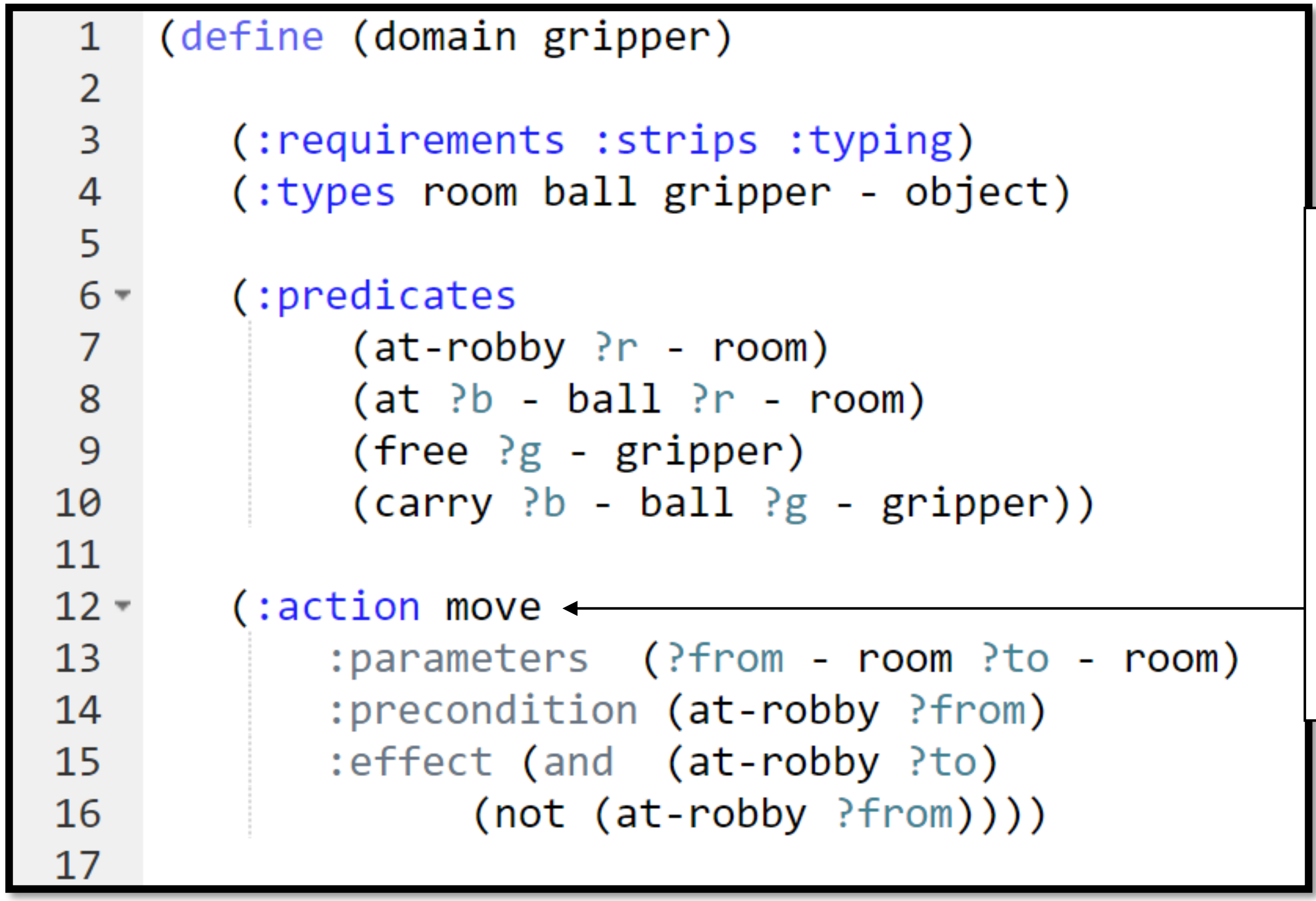
Gripper: Domain and Problem

- Domain:
  - Object Types: `~~robot~~* gripper room ball - object`
    - Since the AI goes inside the robot, it will be implicit in predicates and actions
  - Predicates: `(at-robby ?r - room) (at-ball ?b – ball ?r - room)`
    `(free ?g – gripper)  (carry ?g - gripper ?b - ball)`
  - Actions: `move, pick-up, drop`

- Problem:
  - Objects: `rooma roomb – room; left right – gripper; ball1 … - ball;`
  - Initial state: `(at-robby rooma) (at-ball ball1 rooma)… (free left)…`
  - Goal: `(at-ball ball1 roomb), …, at-ball(ball4,roomb)`

- To test it in PDDL: http://editor.planning.domains/

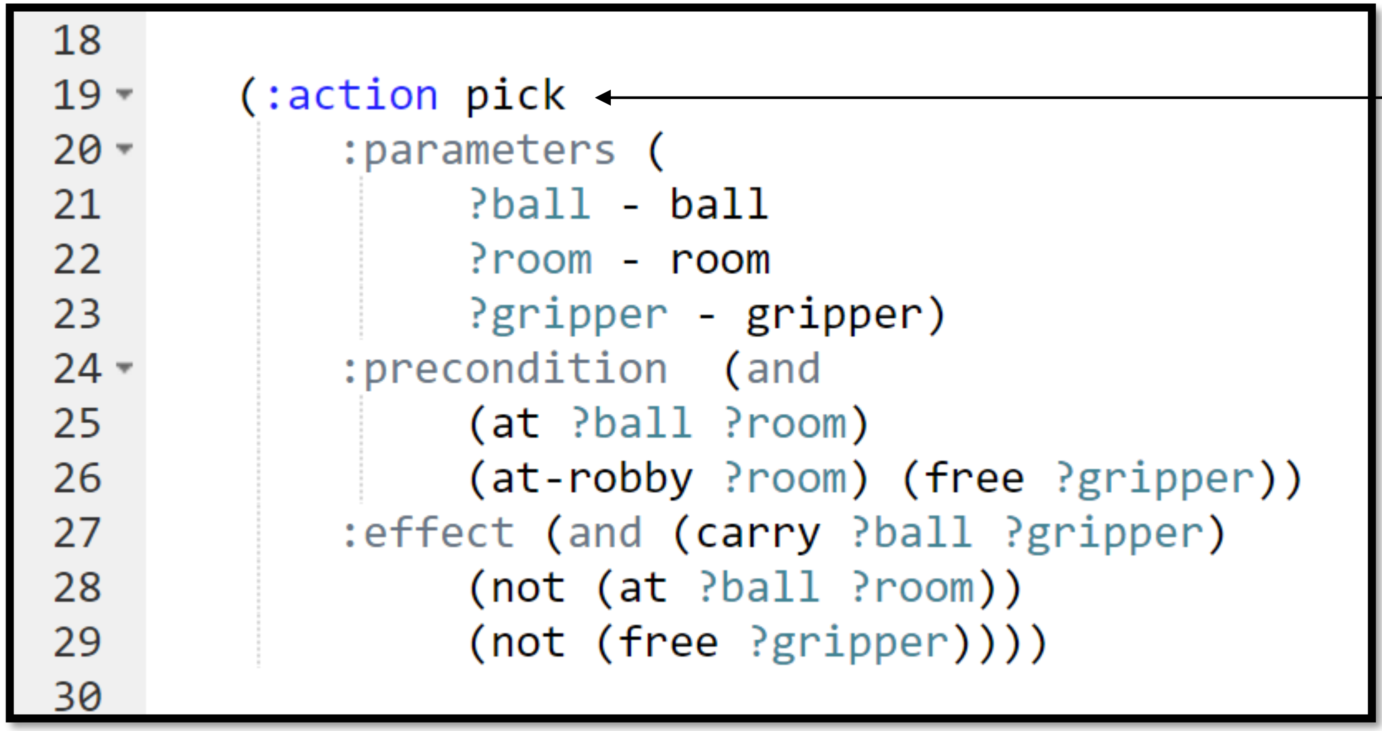
Gripper: domain in PDDL (1/3)



```
(define (domain gripper)

   (:requirements :strips :typing)
   (:types room ball gripper - object)

   (:predicates
        (at-robby ?r - room)
        (at ?b - ball ?r - room)
        (free ?g - gripper)
        (carry ?b - ball ?g - gripper))

   (:action move
       :parameters  (?from - room ?to - room)
       :precondition (at-robby ?from)
       :effect (and  (at-robby ?to)
             (not (at-robby ?from))))

```

- Action: move
- Parameters:
  - Origin room
  - Destination room
- Precondition:
  - Robot at the origin
- Effects:
  - Robot is at the destination
  - Robot is no longer at origin

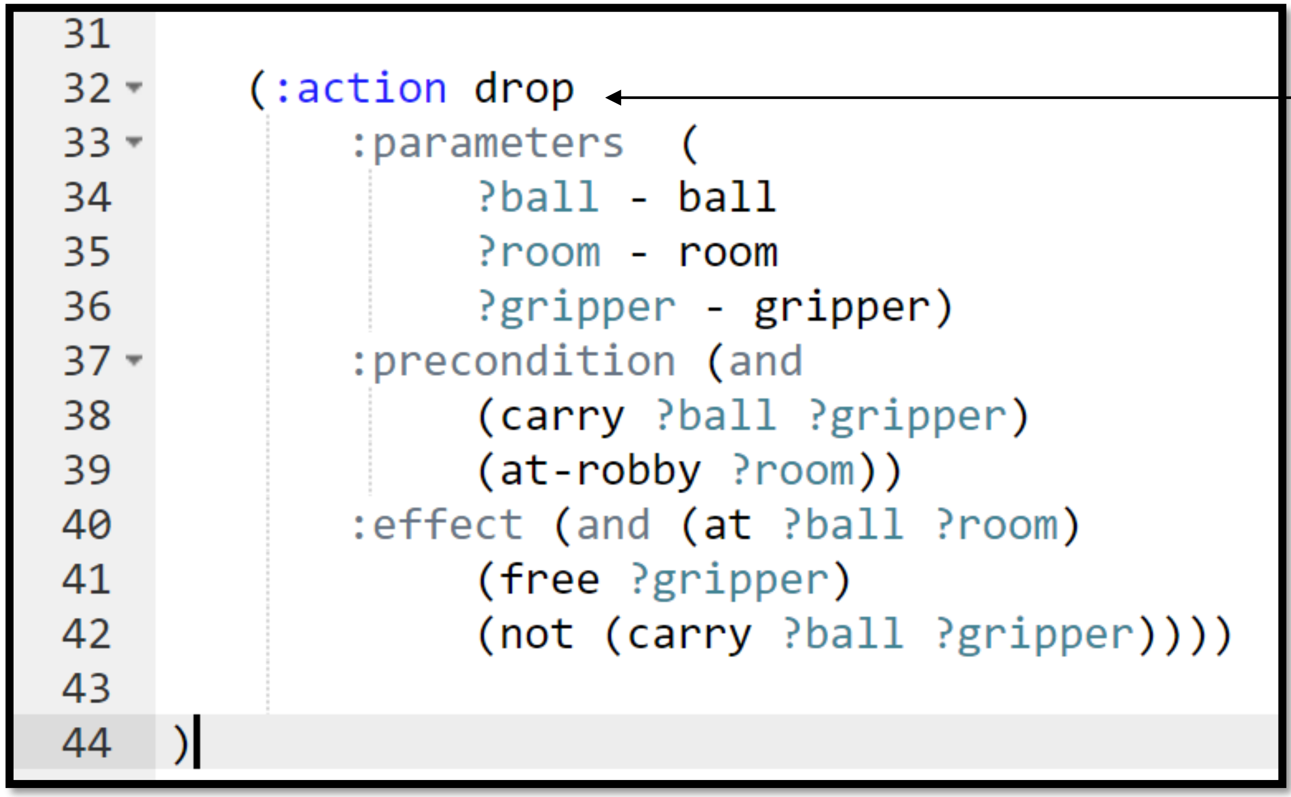
Gripper: domain in PDDL (2/3)



```

  (:action pick
      :parameters (
          ?ball - ball
          ?room - room
          ?gripper - gripper)
      :precondition  (and
          (at ?ball ?room)
          (at-robby ?room) (free ?gripper))
      :effect (and (carry ?ball ?gripper)
          (not (at ?ball ?room))
          (not (free ?gripper))))

```

- Action: Pick
- Parameters:
  - Ball
  - Room
  - Arm (of the robot)
- Precondition:
  - The ball is in the room
  - The robot is in room
  - The arm is free
- Effects:
  - The arm carries the ball
  - The arm is no longer free
  - The ball is no longer at the room

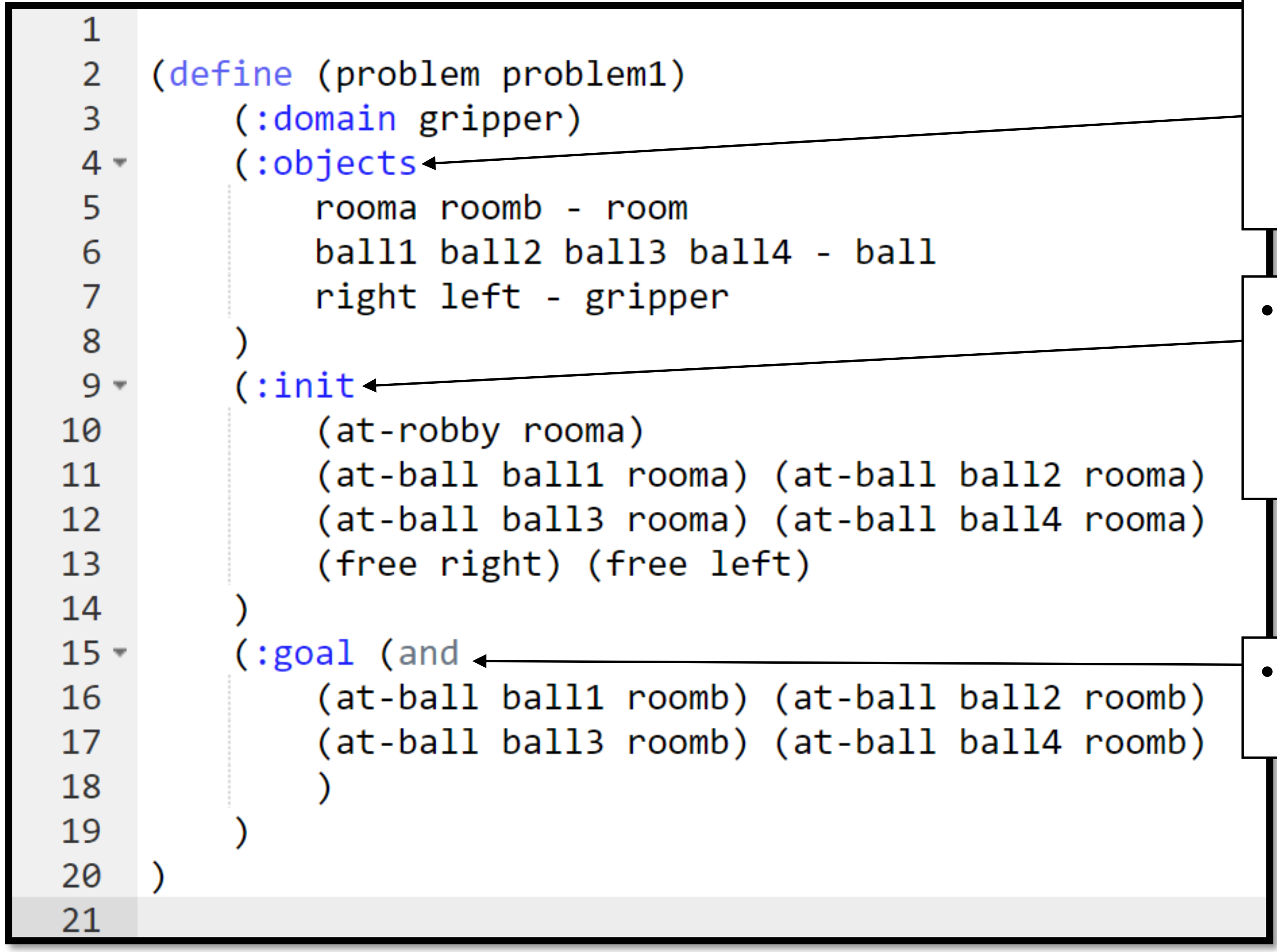
Gripper: domain in PDDL (3/3)



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

)
```

- Action: drop
- Parameters:
  - Ball
  - Room
  - Arm (of the robot)
- Precondition:
  - The arm carries the ball
  - The robot is in room
- Effects:
  - The ball is in the room
  - The arm is free
  - The arm no longer carries the ball

Gripper: problem in PDDL



```
(define (problem problem1)
    (:domain gripper)
    (:objects
        rooma roomb - room
        ball1 ball2 ball3 ball4 - ball
        right left - gripper
    )
    (:init
        (at-robby rooma)
        (at-ball ball1 rooma) (at-ball ball2 rooma)
        (at-ball ball3 rooma) (at-ball ball4 rooma)
        (free right) (free left)
    )
    (:goal (and
        (at-ball ball1 roomb) (at-ball ball2 roomb)
        (at-ball ball3 roomb) (at-ball ball4 roomb)
        )
    )
)
```

- Objects
  - Two rooms
  - Four balls
  - Two arms (left and right)

- Initial State (list of preds. → ~~and~~):
  - The robot is in room A
  - All four balls are in room A
  - Left and right arms are free

- Goal (logical expression → and)
  - The four balls are in room B

Gripper: Interesting facts about PDDL

- "Data types" have been used in this implementation
    - This is a feature of ADL, highly supported and used in PDDL
    - But it can be implemented without it, using predicates instead

- A pure STRIPS deployment can be performed, without types
    - In this case, we need three predicates: ROOM, BALL, and GRIPPER
    - They indicate whether a particular object is of a certain type
    - It is necessary to add them as a precondition in the actions
    - They need to be added to the initialization of the issues
    - You can download this version here: domain y problem

- STRIPS and ADL were the languages that gave rise to PDDL
    - There are :requirements that describe what each language supports: STRIPS / ADL

Gripper: Interesting facts about PDDL

- Most classic planners (PDDL 1.2) support:
  - STRIPS Features and some ADL Features
  - Just because a certain feature is supported, doesn't mean it always works well
  - Especially when we combine "unusual" features

- The planning.wiki indicates, for each version of PDDL
  - What features are typically supported by planners
  - What features are typically used by the community
  - When a function is not used, what is usually done in its place? (example)

- Still, we must prepare to deal with every planner
  - Changing the coding of our domain/problem until it works

Gripper: Interesting facts about PDDL

- PDDL is generally used for:
  - Benchmark: Compare planner performance on benchmark problems
  - Solve very specific planning problems with such planners
    - Usually related to the control of robots

- Lacks common features in general-purpose languages
  - There is no context, actions receive as a parameter all the data they use
  - No support for data types: only predicates (bool) and numbers
  - No overhead, different actions are created for different types of data
  - There are no arrays, you create as many independent objects as you need
  - Loops have a very limited use that depends on the planner
  - Etc., etc. → It's better to accept it and adapt than to try to force it.

Index

- The PDDL Language
- "Gripper" domain
- **PDDL Editors**
- PDDL Planners
- Exercises

PDDL Editors

- Visual Studio Code with PDDL extension
  - Recommended editor, excellent PDDL support
  - View extension documentation for installation and use
  - By default, it uses the services of planning.domains

- Web editor with SIW-THEN-BFSf (2014) planner:
  - http://editor.planning.domains
  - It allows you to call the planner and shows us your output
  - Recommended only for quick initial tests.

Index

- The PDDL Language
- "Gripper" domain
- PDDL Editors
- **PDDL Planners**
- Exercises

PDDL Planners

- Installation and use of on-premises planners
  - Usually developed in C++, complicated to compile
    - Some give source code that needs to be compiled
    - Others give executables for specific versions of Linux
  - Some recent planners are packaged using Singularity
    - Docker-like container system
  - All of them are used in a similar way if they come from ICAPS
    - Set by the International Planning Competition (IPC) of the ICAPS congress

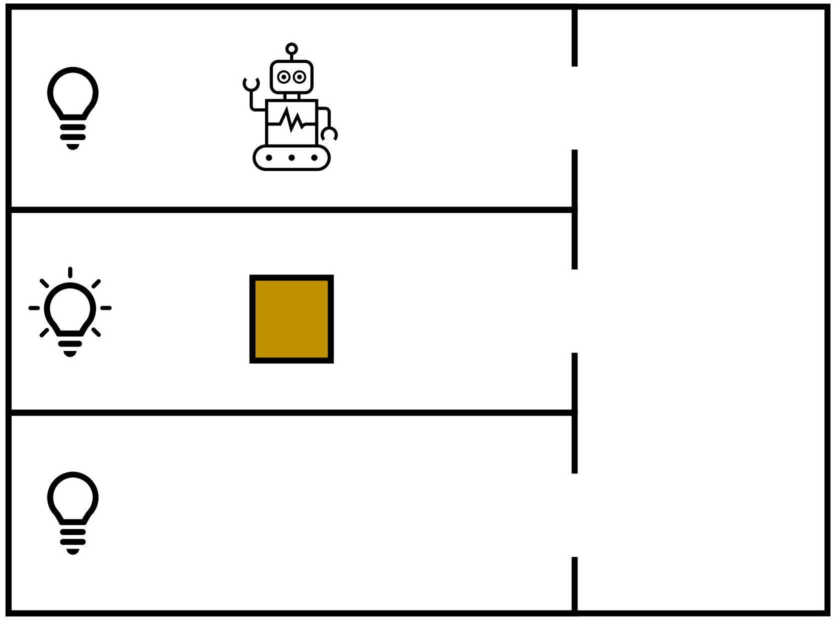
- Some examples
  - SGPlan, LAMA, LPG-TD, OPTIC, FastDownward, etc.
  - Many planners are published at the IPS competition websites

Index

- The PDDL Language
- "Gripper" domain
- PDDL Editors
- PDDL Planners
- **Exercises**

Exercises – Shakey's World Domain

- There's a short robot that can move between rooms, move boxes from room to room, get on or off a box, and turn the light on/off in a room. Since he is short, to turn the light on/off in the room, he needs to get on a box that is there first.



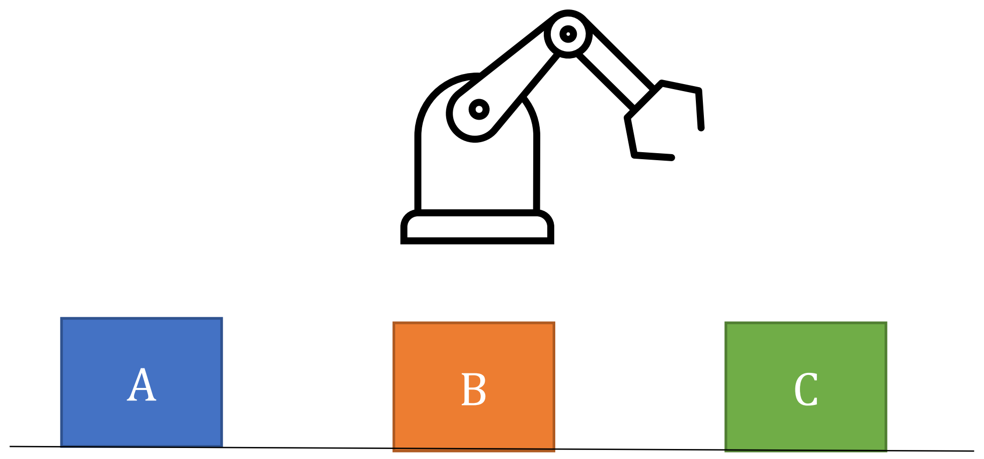
Exercises – Shakey's World Domain

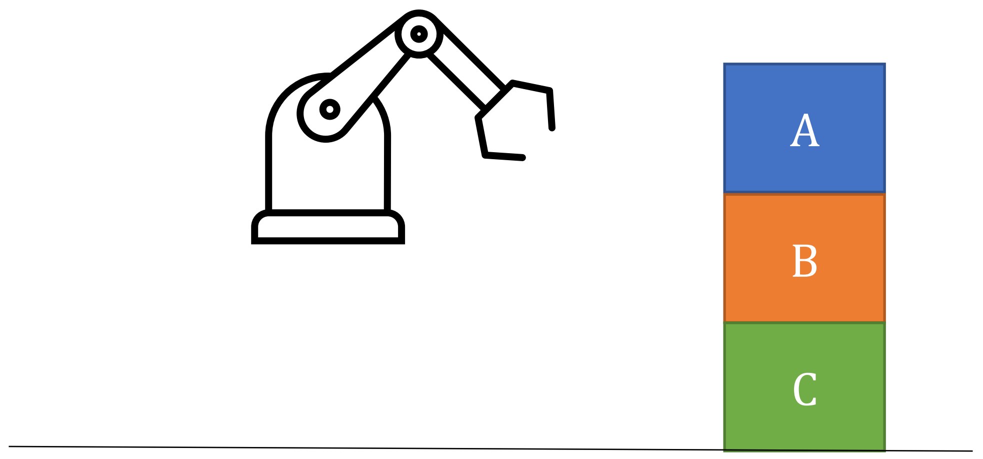
- Domain:
  - Object Types: Room, Box, Light
  - Predicates: robot location?, box location?, light location?, robot on the floor?, robot in box X?, light on?
  - Actions: go to room, move box to room, climb up to box, go down the box, turn on light, turn off light.

- Problem:
  - Items: Three rooms, one box, three lights.
  - Initial state: Box in one room, robot in another, lights out.
  - Goal: Turn on all three lights.

Exercises – Blocks-World Domain

- There's a robotic arm that can move around and pick up and drop blocks. Initially, all the blocks are on an infinite ribbon. We want the blocks to sit on top of each other in a specific order.





Exercises – Blocks-World Domain

- Domain:
  - Item Types: Arm, Box
  - Predicates: Is box X on the tape, does box X have another box Y on top of it? Doesn't the X box have anything on it? Is the arm free? Do you have your arm to the X box?
  - Actions: pick up tape box, leave box on tape, pick up box from top of another, leave box on top of another

- Problem:
  - Items: One arm, boxes A, B, and C.
  - Initial state: All boxes on the tape. Free arm.
  - Goal: Blocks stacked in ABC order (top to bottom).