Linguagem PDDL

Caetano Colin Torres
João Vitor de Souza Costa

Conteúdo da Apresentação

01 Introdução

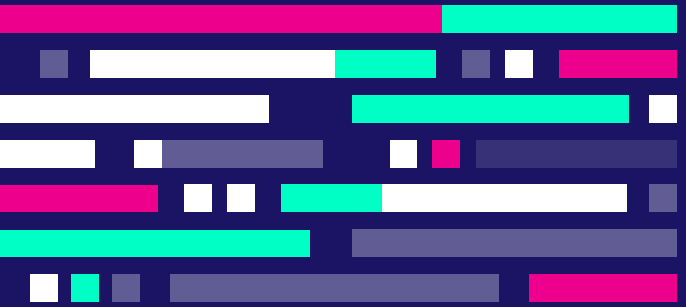
02 Exemplos de Uso

03 Aplicações e Sintaxe

01

Introdução

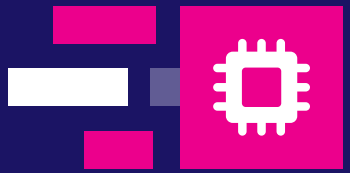
Sobre o PDDL



- PDDL é uma das poucas linguagens feitas com o propósito de criar um padrão para o planejamento de Inteligência Artificial. Seu desenvolvimento iniciou em 1998 e foi aprimorando ao longo dos anos.
- O uso mais popular da PDDL usado hoje é o PDDL2.1, que é uma extensão do PDDL para expressar domínios temporais. PDDL 3 adiciona restrições de trajetórias e preferências ao PDDL 2.1, e PDDL+ permite a modelagem de domínios discretos/contínuos em PDDL.

http://planning.domains/

Definindo Conceitos



Planejamento Automatizado

O Planejamento automatizado é uma área da IA que estuda o processo de deliberação por meio da computação. É um processo que escolhe e organiza ações, antecipando os resultados esperados, visando alcançar objetivos pré-definidos.

02

Aplicações E Sintaxe

Aplicações

PDDL

é usado para Planejamento de IA. O planejamento automatizado tem diversas aplicações práticas.

- Avaliação de Custos de Planos
- Coordenação de encontros Sociais
- Máquinas Autônomas
- Comportamentos Autônomos

Componentes do PDDL

Antes de tudo, como modelamos um “mundo” em PDDL?

Um **mundo** é descrito por um conjunto de estados. Cada um contendo uma lista **fatos** e/ou **objetos**. Um mundo começa com um **estado inicial**, que é regido por um conjunto de regras e restrições que limitam quais **ações** podem ser tomadas em cada estado. E cada ação geralmente representa a transição para outro estado

Componentes do PDDL

O que devemos saber em relação a um "mundo" em PDDL?

- **Objetos**: Coisas que nos interessam dentro do mundo;
- **Predicados**: Fatos que nos interessam (propriedades de objetos) que podem ser *true/false*;
- **Especificação de Objetivo**: O estado do mundo em que queremos estar no final, ou seja, coisas que queremos que sejam *true* no final;
- **Ações/Operadores**: Maneiras de mudar o estado do mundo, ou seja, coisas que acontecem que mudam os fatos;
- **Estado Inicial**: Estado do mundo em que nós começamos (o que é *true* no começo);

Como são os programas PDDL?

Programas em PDDL são divididos em domínio e problemas.

- **Domínio**: Define o que é possível no mundo, como que tipos de objetos podem existir e que ações estes objetos podem tomar;
- **Problemas**: Definem problemas concretos que devem ser resolvidos. Em arquivos de problema, são definidos que objetos existem e suas propriedades.

Domínio é como a definição de classes, enquanto problemas são como instanciação de objetos.

Sintaxe do PDDL

Os arquivos PDDL tem a extensão “.pddl”.

- Arquivo de Domínio: O Arquivo de domínio estabelece o contexto do mundo modelado. Ele determina quais são os detalhes que os estados podem incluir (predicados), e o que podemos fazer para mover-se entre os estados dentro do mundo (ações).

```
(define (domain <domain name>)
  (:predicates
    <predicate-list>
  )

  (:action
    <action-details>
  )
)
```

<domain-name> - Nome do Mundo

Sintaxe do PDDL

- Arquivo de Problema: O arquivo de problema representa uma instância do mundo que o domínio foi estabelecido. Ele determina o que é true no começo do plano (Estado Inicial), e o que nós queremos que seja true no final do plano (Objetivo).

```
(define (problem <title>)
      (:domain <domain-name>)
      (:objects
      <object-list>
      )

      (:init
            <predicates>
      )
      (:goal
            <predicates>
      )
)
```

<title> - título do arquivo de problema
<domain-name> - refere-se ao nome do arquivo de domínio estabelecido

Sintaxe do PDDL

- Definição de tipos: Tipos são definidos com a sintaxe

<tipo1> <tipo2> - <tipo-pai>

- Variáveis: Durante o código PDDL, podemos encontrar variáveis (geralmente como parâmetros de ações). Estas começam com "?"

```
(:types
    location locatable - object
    warehouse - location
    container - locatable
)

(:predicates
    (at ?var1 - location ?var2 - locatable)
)
```

Estamos dizendo que location e locatable são objetos, warehouse é uma location, e container é locatable.

?var1 é uma variável do tipo location, e ?var2 é uma variável do tipo locatable

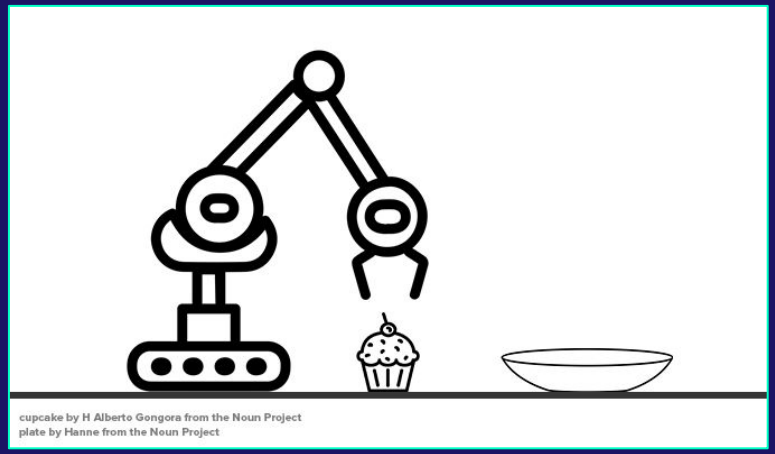
03

Exemplos de Uso

Exemplos de Uso

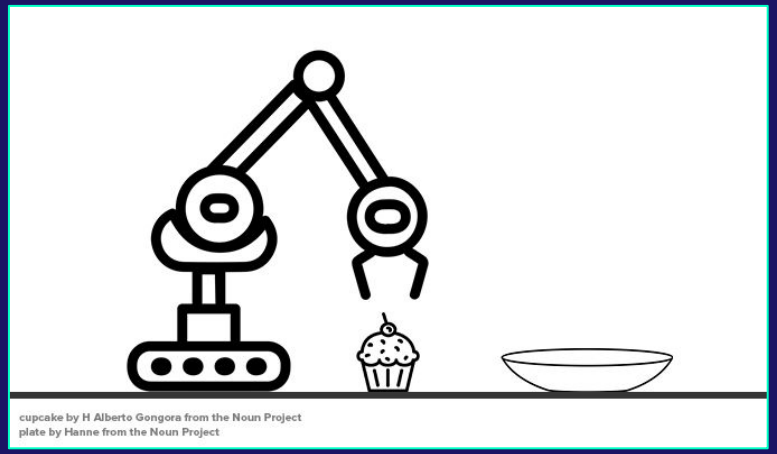
Para explorar os exemplos a seguir mais a fundo, basta acessar o repositório https://github.com/caetanoct/AI-assignment no GitHub onde realizamos alguns exemplos.

Exemplos – Let’s eat



Temos um braço robótico, um *cupcake*, um prato e uma pessoa. A pessoa quer comer o *cupcake*, mas só pode comê-lo se ele estiver no prato. Vamos modelar esse problema em PDDL, a começar pelo domínio.

Exemplos – Let’s eat



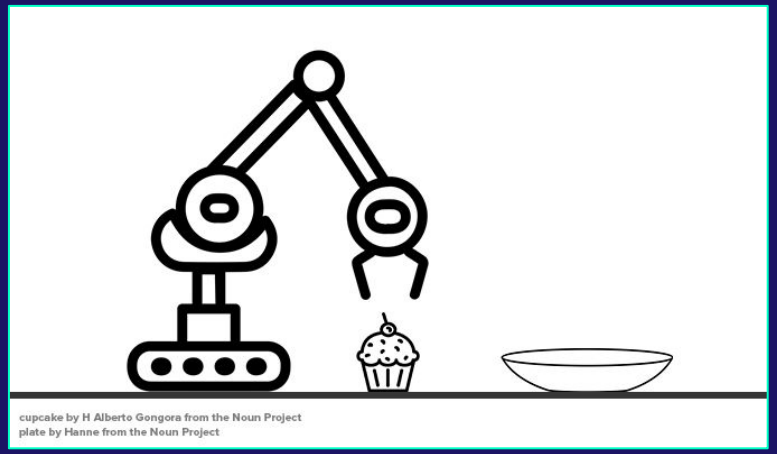
Começamos definindo o nome do domínio, os tipos de objetos que podem existir e bibliotecas externas.

```
(define (domain letseat)

    (:requirements :typing)

    (:types
        location locatable eater - object
        bot cupcake - locatable
        person - eater
    )
```

Exemplos – Let’s eat



Também devemos definir alguns predicados. Onde está o *cupcake*? O Braço mecânico está vazio? De onde o *cupcake* pode ser comido?

```
(:predicates
    (on ?obj - locatable ?loc - location)
    (holding ?arm - bot ?cupcake - cupcake)
    (arm-empty)
    (path ?loc1 - location ?loc2 - location)
    (used-to-eat ?loc - location)
    (eaten ?person - eater ?cupcake - cupcake)
)
```

Exemplos - Let's eat

Também devemos definir algumas ações/operadores. Deve ser possível pegar e largar o cupcake e mover o braço robótico. A pessoa também deve ser capaz de comer o cupcake.

```
(:action pick-up
  :parameters (?arm - bot ?cupcake - cupcake ?loc - location)
  :precondition (and
      ; Note how we use the same variable loc
      ; in both lines below. This is to make
      ; sure it's looking at the same location.
      (on ?arm ?loc)
      (on ?cupcake ?loc)
      (arm-empty)
  )
  :effect (and
      (not (on ?cupcake ?loc))
      (holding ?arm ?cupcake)
      (not (arm-empty))
  )
)
```

Exemplos - Let's eat

Também devemos definir algumas ações/operadores. Deve ser possível pegar e largar o cupcake e mover o braço robótico. A pessoa também deve ser capaz de comer o cupcake.

```
(:action drop
  :parameters (?arm - bot ?cupcake - cupcake ?loc - location)
  :precondition (and
      (on ?arm ?loc)
      (holding ?arm ?cupcake)
  )
  :effect (and
      (on ?cupcake ?loc)
      (arm-empty)
      (not (holding ?arm ?cupcake))
  )
)
```

Exemplos – Let’s eat

Também devemos definir algumas ações/operadores. Deve ser possível pegar e largar o cupcake e mover o braço robótico. A pessoa também deve ser capaz de comer o cupcake.

```
(:action move
  :parameters (?arm - bot ?from - location ?to - location)
  :precondition (and
      (on ?arm ?from)
      (path ?from ?to)
  )
  :effect (and
      (not (on ?arm ?from))
      (on ?arm ?to)
  )
)
```

Exemplos - Let's eat

Também devemos definir algumas ações/operadores. Deve ser possível pegar e largar o cupcake e mover o braço robótico. A pessoa também deve ser capaz de comer o cupcake.

```
(:action eat
  :parameters (?person - eater ?cupcake - cupcake ?loc - location)
  :precondition (and
      (on ?cupcake ?loc)
      (used-to-eat ?loc)
  )
  :effect (and
      (not (on ?cupcake ?loc))
      (eaten ?person ?cupcake)
  )
)
```

Exemplos - Let's eat

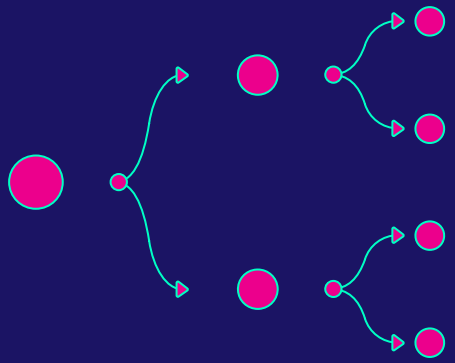
Assim, terminamos de definir nosso **domínio**, e podemos olhar para o **Arquivo de Problema**. Vamos começar deixando ele saber qual domínio está associado ao arquivo de problema e definir os objetos que existem no mundo.

Note que o domínio não restringe coisas como a quantidade de cupcakes na mesa. Deste modo, podemos dar quantos cupcakes quisermos à nossa pessoa (neste caso, 2).

```
(define (problem letseat)
    (:domain letseat)
    (:objects
        arm - bot
        cupcake second-cupcake - cupcake
        table - location
        plate - location
        person - person
    )
```

Exemplos - Let's eat

Agora basta definir o **estado inicial** e o **estado em que queremos chegar (estado objetivo)**.
Começamos com o braço vazio e os dois cupcakes na mesa. Também definimos que o braço pode se mover livremente entre a mesa e o prato, e que a pessoa só pode comer coisas que estão no prato.



```
(:init
    (on arm table)
    (on cupcake table)
    (on second-cupcake table)
    (arm-empty)
    (path table plate)
    (path plate table)
    (used-to-eat plate)
)

(:goal
    (and
        (eaten person cupcake)
        (eaten person second-cupcake)
    )
)
```

Exemplos – Let's eat

Colocando tudo em um arquivo teremos um **Arquivo de Problema**. Podemos rodar o problema, junto com o domínio, usando um planner, como http://editor.planning.domains.

Obtemos um relatório contendo o plano encontrado. Podemos navegar pelo plano, explorando quais ações devem ser tomadas para chegar no objetivo.

Found Plan (output)

(pick-up arm cupcake table)
(move arm table plate)
(drop arm cupcake plate)
(eat person cupcake plate)
(move arm plate table)
(pick-up arm second-cupcake table)
(move arm table plate)
(drop arm second-cupcake plate)
(eat person second-cupcake plate)

```
(:action pick-up
  :parameters (arm cupcake table)
  :precondition
    (and
      (on arm table)
      (on cupcake table)
      (arm-empty)
    )
  :effect
    (and
      (not
        (on cupcake table)
      )
      (holding arm cupcake)
      (not
        (arm-empty)
      )
    )
)
```

Exemplos – Gripper

- **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.

Exemplos – Gripper

Objects:

Rooms: `rooma`, `roomb`

Balls: `ball1`, `ball2`, `ball3`, `ball4`

Robot arms: `left`, `right`

In PDDL:

```
(:objects rooma roomb
          ball1 ball2 ball3 ball4
          left right)
```

Exemplos – Gripper

Predicates:

ROOM(x)	– true iff x is a room
BALL(x)	– true iff x is a ball
GRIPPER(x)	– true iff x is a gripper (robot arm)
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 (ROOM ?x) (BALL ?x) (GRIPPER ?x)
             (at-robby ?x) (at-ball ?x ?y)
             (free ?x) (carry ?x ?y))
```

Exemplos – Gripper

Initial state:

`ROOM(rooma)` and `ROOM(roomb)` are true.
`BALL(ball1)`, ..., `BALL(ball4)` are true.
`GRIPPER(left)`, `GRIPPER(right)`, `free(left)` and `free(right)` are true.
`at-robby(rooma)`, `at-ball(ball1, rooma)`, ..., `at-ball(ball4, rooma)` are true.
Everything else is false.

In PDDL:

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

Exemplos – Gripper

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)))
```

Exemplos – Gripper

Action/Operator:

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

In PDDL:

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

Exemplos – Gripper

Action/Operator:

Description:	The robot can pick up x in y with z.
Precondition:	`BALL(`x`)`, `ROOM(`y`)`, `GRIPPER(`z`)`, `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 ?y ?z)
   :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
                      (at-ball ?x ?y) (at-robby ?y) (free ?z))
   :effect       (and (carry ?z ?x)
                      (not (at-ball ?x ?y)) (not (free ?z))))
```

Exemplos – Gripper

Action/Operator:

Description: The robot can drop x in y from z.

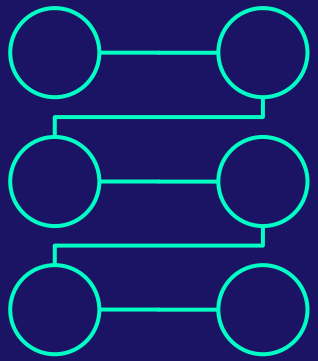
(Preconditions and effects similar to the pick-up operator.)

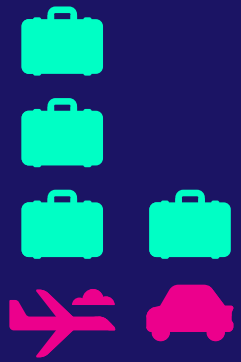
In PDDL:

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

Exemplos – Delivery Drivers

Neste segundo exemplo, modelamos a entrega de cargas. Podem existir diferentes veículos, com diferentes capacidades de carga. Cada carga tem um tamanho, que deve ser menor ou igual à capacidade do veículo. Para tornar as viagens mais eficientes, cada veículo tem uma distância mínima que deve percorrer para fazer com que a viagem valha a pena. Caso a carga não esteja distante o suficiente para um veículo a levar, outro veículo deve ser usado. Os objetos, predicados e ações são parecidas com o exemplo anterior, mas este domínio requer o uso de funções.





Exemplos - Delivery Drivers

O arquivo de domínio só define quais funções existem, e quais seus parâmetros. É responsabilidade dos arquivos de problema definirem seus valores.

```
(:functions
    (size ?cargo - cargo)
    (capacity ?vehicle - vehicle)
    (minimum-distance ?vehicle - vehicle)
    (distance-from-destination ?cargo - cargo)
)
```

```
(:action pick-up
    :parameters (?vehicle - vehicle ?cargo - cargo ?loc - location
    :precondition (and
        (on ?vehicle ?loc)
        (on ?cargo ?loc)
        (empty ?vehicle)
        (<= (size ?cargo) (capacity ?vehicle))
        (<= (minimum-distance ?vehicle) (distance-from-destination ?cargo))
    )
    :effect (and
        (not (on ?cargo ?loc))
        (holding ?vehicle ?cargo)
        (not (empty ?vehicle))
    )
)
```

Exemplos - Delivery Drivers

O primeiro problema lida com diferentes veículos, com diferentes capacidades, entregando diferentes cargas, com diferentes tamanhos.

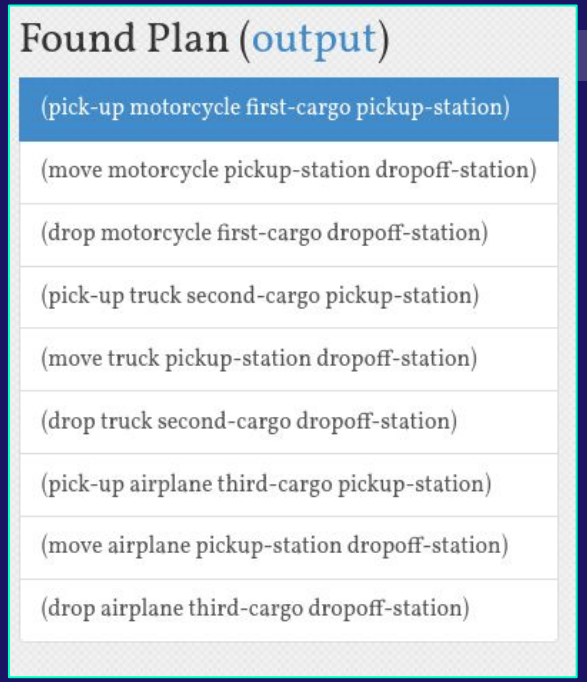
```
(:init
    (on motorcycle pickup)
    (on truck pickup)
    (empty motorcycle)
    (empty truck)
    (on light pickup)
    (on medium pickup)
    (path pickup dropoff)
    (path dropoff pickup)
    (= (capacity motorcycle) 1)
    (= (minimum-distance motorcycle) 0)
    (= (capacity truck) 2)
    (= (minimum-distance truck) 500)
    (= (size light) 1)
    (= (size medium) 2)
    (= (distance-from-destination light) 250)
    (= (distance-from-destination medium) 1000)
)
```

```
(:objects
    motorcycle truck airplane - vehicle
    light medium heavy - cargo
    pickup dropoff - location
)
```

```
(:goal
    (and
        (on light dropoff)
        (on medium dropoff)
    )
)
```

Exemplos – Delivery Drivers

O caminhão não pode levar a carga leve, pois a distância não valeria a pena, enquanto a moto não pode levar a carga média, pois não tem capacidade o suficiente



Exemplos - Delivery Drivers

O segundo problema lida com diferentes rotas. Deste modo, uma carga pode percorrer diferentes caminhos para chegar no destino.

```
(:init
    (on airplane florianopolis)
    (empty airplane)
    (on first florianopolis)
    (on second new-york)
    (on third london)
    (path florianopolis sao-paulo)
    (path sao-paulo new-york)
    (path sao-paulo amsterdam)
    (path amsterdam london)
    (path london new-york)
    (path sao-paulo florianopolis)
    (path new-york sao-paulo)
    (path amsterdam sao-paulo)
    (path london amsterdam)
    (path new-york london)
)
```

```
(:objects
    airplane - vehicle
    first second third - cargo
    florianopolis sao-paulo new-york london
amsterdam - location
)
```

```
(:goal
    (and
        (on first london)
        (on second florianopolis)
        (on third sao-paulo)
    )
)
```

Delivery Drivers

Note que a ordem dos objetivos não importa. O avião primeiro entrega a primeira carga seguindo o caminho Florianópolis > São Paulo > New York > London, e em seguida já pega a terceira carga, que se encontra em London.

Referências e Conteúdos

[1] OPTIC Software
https://nms.kcl.ac.uk/planning/software/optic.html

[2] Tutorial de PDDL feito por fareskalaboud
https://fareskalaboud.github.io/LearnPDDL/

[3] Página da Wikipedia sobre PDDL
https://en.wikipedia.org/wiki/Planning_Domain_Definition_Language

[4] Exemplos de PDDL feitos por jan-dolejsi
https://github.com/jan-dolejsi/vscode-pddl-samples/

[5] Slidesgo
https://slidesgo.com

[6] EH 2750 Modelling planning problems using PDDL Arshad
https://slidetodoc.com/eh-2750-modelling-planning-problems-using-pddl-arshad/