Parser PDDL to BDDs

Release 0.1

André Nogueira
Henri Michel
João Guilherme
Matheus Sanches

Jul 25, 2024

CONTENTS:

CHAPTER

ONE

CUSTOM_TYPES MODULE

class src.custom_types.**Action**(*name*, *parameters*, *preconditions*, *effects*)

Represents a PDDL Action.

name

A descriptive name for the action.

Type
str

parameters

The list of parameters of the action.

Type
list[Objects]

preconditions

A list of tuples, with a proposition and its corresponding (boolean) value.

Type
list[(*Proposition*, bool)]

effects

A list of effects; each effect is a list of propositions and their corresponding values.

Type
list[list[(*Proposition*, bool)]]

get_effects()

Gets action effects list.

Return type
list[list[tuple[*Proposition*, bool]]]

get_name()

Gets action name.

Return type
str

get_parameters()

Gets action parameters list.

Return type
list[*Object*]

get_preconditions()

Gets action preconditions list.

Return type
list[tuple[*Proposition*, bool]]

class src.custom_types.**Object**(*name*, *type*)

Represents a PDDL object.

name

A descriptive name for the object.

Type
str

type

The type of the object.

Type
str

Examples

```
>>> rooma = Object("rooma", "room")
>>> ball1 = Object("ball1", "ball")
```

get_name()

Gets object's name.

Return type
str

get_type()

Gets object's type.

Return type
str

class src.custom_types.**Predicate**(*name*, *variable_types=[]*)

Represents a PDDL Predicate.

name

A descriptive name for the predicate.

Type
str

variable_types

The list of variable types of the predicate.

Type
list[str]

Examples

```
>>> at_robby = Predicate("at-robby", [ "room" ])
>>> at_ball = Predicate("at-ball", [ "ball", "room" ])
```

get_name()

Gets the predicate's name.

Return type
str

get_variable_types()

Gets the variable types

Return type
list[str]

class src.custom_types.**Proposition**(*predicate*, *objects*, *index=-1*)

Represents a PDDL Proposition, which is an instantiated predicate.

name

A descriptive name for the proposition.

Type
str

predicate

The predicate corresponding to the proposition.

Type
Predicate

objects

The list of (instantiated) objects corresponding to the proposition.

Type
list[*Object*]

index

An index associated with the proposition.

Type
int

Examples

```
>>> at_ball = Predicate("at-ball", [ "ball", "room" ])
>>> objects = [ Object("ball1", "ball"), Object("rooma", "room") ]
>>> at_ball_ball1_rooma = Proposition(at_ball, objects, 0)
```

__build_proposition_name()

Builds a proposition name by combining the predicate name and object names.

Return type
str

compare_names(*prop_name*)

Compare the name of the proposition with the strings 'prop_name'.

Parameters
prop_name (*str*) – The string to be compared to the proposition name.

Returns
True if the name of the proposition is equal to 'prop_name'; False otherwise.

Return type
bool

get_index()

Gets proposition index.

Return type
int

get_objects()

Gets objects.

Return type
list[*Object*]

get_predicate()

Gets predicate.

Return type
Predicate

CHAPTER

TWO

DOMAIN MODULE

class src.domain.**Domain**(*parsed_domain*)

Represents a PDDL domain.

constants

A map from the names of the constants to the 'Object' objects.

Type
dict[str, list[*Object*]]

predicates

A map from the names of the predicates to the 'Predicate' objects.

Type
dict[str, *Predicate*]

actions

A list of actions.

Type
list[*Action*]

pred_to_actions

A dictionary mapping predicates to lists of actions that have those predicates in their preconditions.

Type
dict[*Predicate*, list[*Action*]]

Examples

```
>>> parsed_domain = parse_domain("tests/examples/gripper3.pddl")
>>> domain = Domain(parsed_domain)
```

__build_action_instance(*parsed_action*, *stored_predicates*)

Sets attributes, and builds an 'Action' object.

Parameters

- **parsed_action** – The parsed action description.
- **stored_predicates** (*dict[str, Predicate]*) – A map from the name of the predicates to the 'Predicate' objects.

Returns
The instantiated action.

Return type
Action

`__build_object_instance`(*parsed_object*)

Sets attributes, and build an 'Object' instance.

Parameters
parsed_object – The parsed object description.

Returns
The instantiated object.

Return type
Object

`__get_predicate_and_boolean_value`(*proposition*)

Extracts the predicate and truth value from a proposition.

Parameters
pred – The parsed description of a proposition.

Returns
The parsed description of the predicate within 'proposition', and its value.

Return type
(PDDL Predicate, bool)

`__merge_effects`(*all_possible_effects*)

Combines deterministic and non-deterministic effects to a list of possible outcome scenarios.

Parameters
all_possible_effects (*list[list[(Proposition, bool)] or (Proposition, bool)]*) – A list containing both deterministic effects (represented as tuples) and non-deterministic effects (represented as lists of tuples).

Returns
A list of lists, where each inner list represents one possible combination of effects after the action. Deterministic effects are included in every outcome scenario.

Return type
list[list[tuple[*Proposition*, bool]]]

Note: This function assumes that non-deterministic effects have at most one level of alternative outcomes (i.e., no nested "OneOf" effects). This simplifies the merging process and limits the "depth" of potential effect combinations.

`__process_action_parameters`(*parsed_action*)

Builds the list of parameters for the Action

Parameters
parsed_action – The parsed action description.

Returns
The list of parameters for the action.

Return type
list[*Object*]

__store_actions(*parsed_domain*, *stored_predicates*)

Builds a list of actions, along with a map from the predicates to a list of actions
they are associated with, i.e., each action in the list has a precondition containing the predicate.

Parameters

- **parsed_domain** – The parsed domain description.
- **stored_predicates** (*dict[str, Predicate]*) – A map from the name of the predicates to the 'Predicate' objects.

Returns
The list of actions of the corresponding PDDL domain. dict[Predicate, list[Action]]): A map from predicates to the actions they are associated with.

Return type
list[*Action*]

__store_actions_by_preconditions(*action*, *pred_to_actions*)

Updates 'pred_to_actions', which is a map from predicates to the actions containing them in their preconditions.

Parameters
action (Action) – An instantiated action.

pred_to_actions (dict[Predicate, list[Action]]): A dictionary mapping predicates to lists of actions that have those predicates in their preconditions.

Returns
The updated mapping.

Return type
dict[str, *Predicate*]

__store_constants(*parsed_domain*)

Stores constants corresponding to the instantiated domain.

Parameters
parsed_domain – The parsed domain description.

Returns
A map from the names of the constants to the 'Object' objects.

Return type
dict[str, list[*Object*]]

__store_effects_of_action(*action_effects*, *stored_predicates*, *all_possible_effects=[]*)

Recursively builds a list of all possible effects (deterministic and non-deterministic) of an action.

Parameters

- **action_effects** – The parsed description of the action's effects.
- **stored_predicates** (*dict[str, Predicate]*) – A map from predicate names to 'Predicate' objects.
- **all_possible_effects** (*list[list[(Proposition, bool)] or (Proposition, bool)]*) – The list to populate with the possible effects.

Return type
None

Base Case:
If 'action_effects' is a single effect, appends its tuple to 'all_possible_effects'.

Recursive Case:

- 'And' Effect: Recursively processes each sub-effect, appending their results to the SAME 'all_possible_effects' list.
- 'OneOf' Effect: Recursively processes each sub-effect, appending their results to SEPARATE lists within 'all_possible_effects' (representing alternative outcomes).

`__store_one_effect_or_precondition_predicate`(*pred*, *stored_predicates*)

Builds a tuple (Proposition, bool) representing a single effect or precondition.

Parameters

- **pred** – The parsed description of the effect or precondition.
- **stored_predicates** (*dict[str, Predicate]*) – A map from predicate names to 'Predicate' objects.

Returns
The proposition corresponding to the effect. bool: The truth value assgined to the Proposition.

Return type
Proposition

`__store_preconditions_of_action`(*action*, *stored_predicates*)

Builds a list of preconditions for an action.

Parameters

- **action** – The parsed action description.
- **stored_predicates** (*dict[str, Predicate]*) – a map from the name of the predicates to the 'Predicate' objects.

Returns
A list of propositions, and their respective truth values.

Return type
list[(*Proposition*, bool)]

`__store_predicates`(*parsed_domain*)

Builds a map from the predicates in the domain description to 'Predicate' objects.

Parameters
parsed_domain – The parsed domain description.

Returns
A map from the names of the predicates to the 'Predicate' objects.

Return type
dict[str, *Predicate*]

`get_actions`()

Gets list of domain actions.

Return type
list[*Action*]

`get_constants`()

Gets name-to-Object mapping for domain constants.

Return type
dict[str, list[*Object*]]

`get_pred_to_actions`()

Gets Predicate-to-actions mapping.

Return type
dict[*Predicate*, list[*Action*]]

`get_predicates`()

Gets name-to-Predicate mapping.

Return type
dict[str, *Predicate*]

CHAPTER

THREE

GROUND MODULE

src.ground.**add_proposition_to_reached**(*reached_list*, *proposition_value*, *proposition_index*, *num_propositions*)

Assigns value 1 to the entry of the list of reached (valued) propositions corresponding to the valued proposition.

Parameters

- **reached_list** (*list[int]*) – The list reached propositions.
- **proposition_value** (*int*) – The truth value of the proposition (1 for true, 0 for false).
- **proposition_index** (*int*) – The index corresponding to the proposition.
- **num_propositions** (*int*) – The total number of propositions.

Return type
None

Note: If a proposition P has an index i, the i-th entry of the returned list correspond to the tuple (P, True), and the (n + i)-th entry to the tuple (P, False).

---

src.ground.**create_reached_list**(*initial_state*)

Creates the list of reached propositions at the initial state.

Parameters
initial_state (*list[int]*) – The bitmask representing the initial truth values of propositions (1 for true, 0 for false).

Returns
The list indicating whether a proposition is reached or not; for those reached, the value is 0; otherwise, value is -1.

Return type
list[int]

Note: Each proposition P has an index i; the i-th entry of the returned list correspond to the tuple (P, True), and the (n + i)-th entry to the tuple (P, False).

---

src.ground.**enqueue_effects**(*frontier_queue*, *action*, *object_combination*, *propositions*, *parameters*, *reached*)

Enqueues propositions and their respective truth values onto a frontier queue based on an action's effects.

Parameters

- **frontier_queue** (*deque[tuple[Proposition, int]]*) – A queue of (proposition, truth value) pairs representing propositions at the frontier.

- **action** (Action) – The action whose effects are being processed.
- **object_combination** (*tuple[Object]*) – The combination of objects for which the action's effects are being evaluated.
- **propositions** (*dict[str, Proposition]*) – A dictionary mapping proposition names to Proposition objects.
- **parameters** (*list[Object]*) – The list of parameters (objects) of the action.
- **reached** (*list[int]*) – A list indicating which propositions have already been reached.

Return type
None

Note: The function updates the 'reached' list to mark new propositions as reached, and appends the corresponding (proposition, truth value) pairs to the 'frontier_queue'.

---

src.ground.**find_proposition**(*generic_proposition*, *object_combination*, *propositions*, *parameters*)

Finds a specific proposition within a dictionary given a generic proposition and an object combination.

Parameters

- **generic_proposition** (Proposition) – The generic proposition (template) to match.
- **object_combination** (*tuple[Object]*) – The objects to substitute into the generic proposition.
- **propositions** (*dict[str, Proposition]*) – A dictionary mapping proposition names to Proposition objects.
- **parameters** (*list[Object]*) – The list of parameters (objects) used in the propositions.

Returns
The matching proposition from the dictionary, or None if not found.

Return type
Proposition

src.ground.**find_reached_predicate_in_preconditions**(*preconditions*, *predicate*, *value*)

Among all the preconditions of an action, finds the one (if one exists) that has a proposition whose predicate is the desired one.

Parameters

- **preconditions** (*list[tuple[Proposition, int]]*) – The list of preconditions of an action.
- **predicate** (Predicate) – The predicate we're looking for in the propositions of the preconditions.
- **value** (*int*) – The truth value of the proposition (1 for true, 0 for false).

Returns
returns the matching precondition, or None if not found.

Return type
Union[tuple[*Proposition*, int], None]

src.ground.**get_action_parameters_and_preconditions**(*action*)

Retrieves the preconditions and parameters of an action.

Parameters
action (Action) – The action object.

Returns

A tuple containing:

- A list of preconditions, where each precondition is a tuple of a Proposition and its truth value (True or False).
- A list of objects representing the parameters required for the action.

Return type
tuple[list[tuple[*Proposition*, bool]], list[*Object*]]

src.ground.**get_element_from_frontier**(*frontier_queue*)

Pops and returns the front element from the frontier queue along with its proposition's index.

Parameters
frontier_queue (*deque[tuple[Proposition, int]]*) – A queue of (proposition, truth value) pairs representing propositions at the frontier. A proposition reaches the frontier if it lies in the effects list of a reachable action (see definition elsewhere).

Returns

A tuple containing:

- The popped proposition
- Its corresponding truth value (1 for true, 0 for false)
- The index associated with the proposition

Return type
tuple[*Proposition*, int, int]

src.ground.**get_parameters_combinations**(*parameters*, *fixed_object*, *dict_objects*)

Generates all unique combinations of objects that can be assigned to a set of parameters.

Parameters

- **parameters** (*list[Object]*) – A list of parameter objects for which combinations need to be generated.
- **fixed_object** (*dict[Object, list[Object]]*) – A dictionary mapping parameter objects to their fixed values. If a parameter is not in this dictionary, it is considered variable.
- **dict_objects** (*dict[str, list[Object]]*) – A dictionary mapping object types (as strings) to lists of objects of that type.

Returns
A list of tuples, where each tuple represents a unique combination of objects that can be assigned to the parameters.

Return type
list[tuple[*Object*]]

src.ground.**run_ground**(*initial_state*, *list_propositions*, *dict_propositions*, *pred_to_actions*, *dict_objects*)

Given an initial state, computes the list of reachable actions, along with the list of reachable propositions.

Parameters

- **initial_state** (*list[int]*) – The initial state represented as a bitmask (1 for true, 0 for false) for each proposition.

- **list_propositions** (*list[Proposition]*) – The list of all propositions in the domain.
- **dict_propositions** (*dict[str, Proposition]*) – A dictionary mapping proposition names to Proposition objects.
- **pred_to_actions** (*dict[Predicate, list[Action]]*) – A dictionary mapping predicates to lists of actions that have those predicates in their preconditions.
- **dict_objects** (*dict[str, list[Object]]*) – A dictionary mapping object types (as strings) to lists of objects of that type.

Returns

A tuple containing:

- A list of tuples where each tuple represents a reachable action and its corresponding object combination.
- A list indicating whether each proposition (and its negation) is reachable (1) or not (-1).

Return type
tuple[list[tuple[*Action*, tuple[*Object*]]], list[int]]

Note: The algorithm iteratively explores the state space by adding reached propositions to a frontier queue. It checks if actions' preconditions are satisfied by the reached propositions and their combinations. If so, the action's effects are enqueued, expanding the frontier. The process continues until all reachable propositions and actions are found.

---

src.ground.**store_initial_queue**(*initial_state*, *propositions*)

Enqueue the pairs composed by the propositions and their respective truth values at the initial state.

Parameters

- **initial_state** (*list[int]*) – The bitmask representing the initial truth values of propositions (1 for true, 0 for false).
- **propositions** (*list[Proposition]*) – The list of all possible propositions in the domain.

Returns
A queue with the tuples corresponding to the initial state.

Return type
deque[tuple[*Proposition*, int]]

CHAPTER

FOUR

PARSER_PDDL MODULE

class src.parser_pddl.**Parser**(*domain_path*, *problem_path*)

Represents the Parser, the central unit for domain and problem analysis.

domain

The parsed representation of the planning domain.

Type
Domain

problem

The parsed representation of the specific planning problem.

Type
Problem

actions

The list of actions defined in the domain.

Type
list[*Action*]

objects

A dictionary mapping object types (str) to lists of corresponding objects.

Type
dict[str, list[*Object*]]

propositions

The list of all possible propositions in the domain.

Type
list[*Proposition*]

dict_propositions

A dictionary mapping proposition names (str) to Proposition objects.

Type
dict[str, *Proposition*]

initial_state

The bitmask representing the initial truth values of propositions (1 for true, 0 for false).

Type
list[int]

goal_state

The bitmask representing the goal truth values of propositions (1 for true, 0 for false, -1 for don't care).

Type
list[int]

Examples

```
>>> parser1 = Parser("tests/examples/gripper3.pddl", "tests/examples/gripper3_3_
↪balls.pddl")
>>> parser2 = Parser("tests/examples/triangle-tire.pddl", "tests/examples/triangle-
↪tire-1.pddl")
```

__build_instantiated_action_name(*action*, *parameters*)

Builds an instantiated action name by combining the action name and the parameters names.

Return type
str

__build_proposition_names(*parsed_prop*, *is_negated*)

Constructs a standardized proposition name string from a parsed proposition.

Parameters

- **parsed_prop** – The parsed proposition description.
- **is_negated** (*bool*) – Indicates whether the proposition is negated.

Returns
The standardized proposition name

Return type
str

__get_object_combinations(*predicate*)

Generates unique combinations of objects that satisfy a given predicate's variable types.

This method takes a predicate and identifies the types of variables it expects. It then retrieves all objects of those types and creates unique combinations where each object in a combination is of the required type.

Parameters
predicate (Predicate) – The predicate for which object combinations are to be generated.

Returns
The list of tuples, where each tuple represents a unique combination of objects that can satisfy the predicate's variable types.

Return type
list[tuple[*Object*, ...]]

__instantiate_reachable_actions()

Calls the function run_ground and returns the tuple returned by the call.

Return type
tuple[list[tuple[*Action*, tuple[*Object*]]], list[int]]

__is_proposition_negated(*parsed_prop*)

Determines whether a parsed proposition is negated.

Parameters
parsed_prop – The parsed proposition description.

Returns
True if the proposition is negated; False otherwise.

Return type
bool

__merge_obj_const()

Combines domain constants and problem objects into a unified object dictionary.

Returns
A map from types to a list of 'Object' objects.

Return type
dict[str, list[*Object*]]

Note: This method assumes that object types are consistent between the domain and problem definitions.

__print_effects_reachable_action(*action*, *parameters*, *output_file*)

Writes the effects of a reachable action, enclosed in 'begin_nd_effects' and 'end_nd_effects' tags, to the specified output stream.

Parameters
output_file (*TextIO*) – The text stream where the formatted effects of the reachable action should be written.

Return type
None

__print_goal_state(*output_file*)

Writes the goal state, enclosed in 'begin_goal_state' and 'end_goal_state' tags, to the specified output stream.

Parameters
output_file (*TextIO*) – The text stream (file or similar) where the formatted goal state should be written.

Return type
None

Note: Propositions with an indeterminate goal value (-1) are omitted from the output.

__print_initial_state(*output_file*)

Writes the initial state, enclosed in 'begin_initial_state' and 'end_initial_state' tags, to the specified output stream.

Parameters
output_file (*TextIO*) – The text stream where the formatted initial state should be written.

Return type
None

__print_preconditions_reachable_action(*action*, *parameters*, *output_file*)

Writes the preconditions of a reachable action to the specified output stream.

Parameters

- **action** (Action) – The instantiated action.
- **parameters** (*tuple[Object]*) – The parameters of the instantiated action.
- **output_file** (*TextIO*) – The text stream where the formatted effects of the reachable action should be written.

Return type
None

__print_problem_name(*output_file*)

Writes the problem name, enclosed in 'begin_problem_name' and 'end_problem_name' tags, to the specified output stream.

Parameters
output_file (*TextIO*) – The text stream where the formatted problem name should be written.

Return type
None

__print_propositions(*output_file*)

Writes propositions and their indices, enclosed in 'begin_propositions' and 'end_propositions' tags, to the specified output stream.

Parameters
output_file (*TextIO*) – The text stream where the formatted propositions should be written.

Return type
None

__print_reachable_actions(*output_file*)

Writes the reachable actions, enclosed in 'begin_actions' and 'end_actions' tags, to the specified output stream.

Parameters
output_file (*TextIO*) – The text stream where the formatted reachable actions should be written.

Return type
None

__print_reachable_propositions(*output_file*)

Writes the reachable propositions, enclosed in 'begin_reachable_propositions' and 'end_reachable_propositions' tags, to the specified output stream.

Parameters
output_file (*TextIO*) – The text stream where the formatted reachable proposition should be written.

Return type
None

__process_state(*parsed_state*, *default_value*)

Converts a parsed PDDL state into the list of truth values for propositions.

Parameters

- **parsed_state** – The parsed state object from the PDDL problem.

- **default_value** (*int, optional*) – The default value to use for propositions not explicitly mentioned in the state. Defaults to 'default_value'.

Returns
The list of truth values (1 for true, 0 for false, 'default_value' for don't care) corresponding to the propositions defined in the domain. The list's length matches the number of propositions.

Return type
list[int]

__store_basic_elements(*parsed_problem*)

Pre-proccess and store some complementary attributes.

Parameters
parsed_problem – The parsed problem description.

Return type
None

__store_propositions()

Builds a list of propositions, along with a map from the names to the 'Proposition' objects.

Returns
The list of propositions of the corresponding PDDL domain. dict[str, Proposition]): A map from the proposition names to the 'Proposition' objects.

Return type
list[*Proposition*]

get_actions()

Gets domain action list.

Return type
list[*Action*]

get_dict_propositions()

Gets name-to-Proposition mapping.

Return type
dict[str, *Proposition*]

get_goal_state()

Gets problem goal state mapping.

Return type
list[int]

get_initial_state()

Gets problem initial state bitmap.

Return type
list[int]

get_propositions()

Gets domain propositions list.

Return type
list[*Proposition*]

get_reachable_actions()

Gets the list of reachable actions, which is a list of pairs composed by the action and its respective parameters.

print_bdds(*output_file*)

Writes the problem definition, propositions, initial state, and goal state in a structured format to a file.

This method generates a file containing a structured representation of the planning problem, including:

- Problem Name: Enclosed in 'begin_problem_name' and 'end_problem_name' tags.
- Propositions: Enclosed in 'begin_propositions' and 'end_propositions' tags, along with their indices.
- Initial State: Enclosed in 'begin_initial_state' and 'end_initial_state' tags, with truth values for each proposition.
- Goal State: Enclosed in 'begin_goal_state' and 'end_goal_state' tags, with truth values for defined goal propositions.
- Reachable Actions: Enclosed in 'begin_actions' and 'end_actions' tags, with their respective preconditions and effects.
- Reachable Propositions: Enclosed in 'begin_reachable_propositions' and 'end_reachable_propositions' tags, with the indices of the reachable propositions.

Parameters

output_file (*TextIO*) – The text stream where the output should be written.

Return type

None

CHAPTER

FIVE

PROBLEM MODULE

class src.problem.**Problem**(*parsed_problem*)

Represents a PDDL problem.

name

A descriptive name for the planning problem.

Type
str

objects

A map from object types (strings) to lists of instances of the 'Object' class.

Type
dict[str, list[*Object*]]

Examples

```
>>> parsed_problem = pddl.parse_problem("tests/examples/gripper3_3_balls.pddl")
>>> problem = Problem(parsed_problem)
```

__store_objects(*parsed_problem*)

Stores objects corresponding to the instantiated problem.

Parameters
parsed_problem – The parsed problem description.

Returns
A map from object types (strings) to lists of instances of the 'Object' class.

Return type
dict[str, list[*Object*]]

get_name()

Gets problem name.

Return type
str

get_objects()

Gets type-to-Object mapping for problem objects.

Return type
dict[str, list[*Object*]]

PYTHON MODULE INDEX

S

INDEX

Symbols

A

C

D

E

F

G

I

M

N

O

P

R

S

T

V