pyPMT: Planning Modulo Theories

Generated by Doxygen 1.10.0

Chapter 1

pyPMT

A Python library for Planning Modulo Theories using SMT

1.1 Installing pyPMT

Install the package using `pip`:

```
python -m pip install .
```

1.2 Using pyPMT

Getting help

To see the list of input arguments, type

```
pypmtcli -h
```

Running pyPMT

To run pyPMT on a problem from the CLI, type, e.g.,

```
pypmtcli --seq --bound 3 --domain path_to_domain.pddl --problem path_to_problem.pddl
```

To produce an SMT-LIB encoding of the problem (instead of solving it), type, e.g.

```
pypmtcli --seq --bound 3 --domain path_to_domain.pddl --problem path_to_problem.pddl --dump
```

pyPMT can be used as a library too. See `here` for some examples.

1.3 Documentation

Further documentation is available `here`.

1.4 Authors

`Mustafa F Abdelwahed` `Joan Espasa Arxer` `Francesco Leofante`

Do not hesitate to contact us if you have problems using pyPMT, or if you find bugs :)

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 pyPMT.pypmt.config.config Class Reference

Public Member Functions

- **get** (key)
- set (key, value)
- set_verbosity (value)
- set_config (parameters)

Static Public Attributes

- **format**
- dict valid_config_values

4.1.1 Member Function Documentation

4.1.1.1 set()

```
pyPMT.pypmt.config.config.set (
              key,
              value )
```

```
Set a value in the config
```

4.1.1.2 set_config()

```
pyPMT.pypmt.config.config.set_config (
              parameters )
```

```
Set the global config with the parameters given
```

4.1.1.3 set_verbosity()

```
pyPMT.pypmt.config.config.set_verbosity (
              value )
```

```
Configures the logger to output the right amount of messages
Roughly, the idea is to have:
0: nothing. Useful when used as a library I guess
1: basic informative messages and solutions printed
2: add timing information
3: add both z3 and general stats
4: step by step traces on what the code is doing
```

4.1.2 Member Data Documentation

4.1.2.1 config

```
dict pyPMT.pypmt.config.config.config  [static]
```

Initial value:

```
=  {
        "verbose" : 1,
        "ub" : 100,
        "logger" : logging.getLogger(__name__),

        "encoder" : None,
        "search" : None,
        "propagator" : None,
        "extractor" : None, # Not used for now
        "validator" : None # Not used for now
    }
```

4.1.2.2 valid_config_values

```
dict pyPMT.pypmt.config.config.valid_config_values  [static]
```

Initial value:

```
=  {
        "verbose" : "Controls the level of verbosity (0,4)",
        "ub" : "the upper bound on the number of possible steps considered",
        "logger" : "a logging python object that controls where messages go",

        "encoder" : "The encoder class used to encode the problem",
        "search" : "The search algorithm that the class will use",
        "propagator" : "If a propagator class has to be used to help during search",
        "extractor" : "The way of extracting the plan from a model",
        "validator" : "The method to validate the plan"
    }
```

The documentation for this class was generated from the following file:

- pypmt/config.py

4.2 pyPMT.pypmt.encoders.base.Encoder Class Reference

The main role of an Encoder is to receive a Unified Planning task and produce a SMT encoding.

Inheritance diagram for pyPMT.pypmt.encoders.base.Encoder:

Public Member Functions

- __iter__ (self)

  The iterator goes through the raw actions, allowing a clean interface when for example extracting a plan from a model.
- __len__ (self)
- create_variables (self, t)

  Creates the Z3 variables needed for a given timestep.
- encode (self, t)

  Encodes and returns the formula for a single transition step (from t to t+1).
- encode_initial_state (self)

  Encodes the initial state.
- encode_goal_state (self)

  Encodes the goal state.
- encode_actions (self)

  Encodes the transition function.
- encode_frame (self)

  Encodes the frame axioms.
- encode_execution_semantics (self)

  Encodes the possible needed mutexes between actions.

Protected Member Functions

- _ground (self)

  Implements the grounding of the task, if needed.

4.2.1 Detailed Description

The main role of an Encoder is to receive a Unified Planning task and produce a SMT encoding.

This class is an interface for all encodings to implement.

Typical encodings iteratively add layers (or timesteps) until the solver proves its satisfiability. That is, finds a plan of a given number of steps. Therefore, encoders generally keep a record of which timesteps have been encoded. Note that requirements of some encodings will leave some methods unimplemented if they do not need them.

4.2.2 Member Function Documentation

4.2.2.1 __iter__()

```
pyPMT.pypmt.encoders.base.Encoder.__iter__ (
                self )
```

The iterator goes through the raw actions, allowing a clean interface when for example extracting a plan from a model.

Reimplemented in pyPMT.pypmt.encoders.basic.EncoderGrounded, and pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists.

4.2.2.2 __len__()

```
pyPMT.pypmt.encoders.base.Encoder.__len__ (
            self )
```

Returns

the number of timesteps that have been encoded

Reimplemented in pyPMT.pypmt.encoders.basic.EncoderGrounded, pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifte
and pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.

4.2.2.3 _ground()

```
pyPMT.pypmt.encoders.base.Encoder._ground (
            self ) [protected]
```

Implements the grounding of the task, if needed.

Reimplemented in pyPMT.pypmt.encoders.basic.EncoderGrounded, pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifte
and pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.

4.2.2.4 create_variables()

```
pyPMT.pypmt.encoders.base.Encoder.create_variables (
            self,
            t )
```

Creates the Z3 variables needed for a given timestep.

Parameters

t	The timestep for which the variables need to be created

Reimplemented in pyPMT.pypmt.encoders.basic.EncoderGrounded, pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists, and pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted.

4.2.2.5 encode()

```
pyPMT.pypmt.encoders.base.Encoder.encode (
            self,
            t )
```

Encodes and returns the formula for a single transition step (from t to t+1).

Parameters

t	the timestep to consider when encoding the single transition step

Reimplemented in pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted, pyPMT.pypmt.encoders.basic.EncoderGrounde
pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists, and pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.

4.2.2.6 encode_actions()

```
pyPMT.pypmt.encoders.base.Encoder.encode_actions (
              self )
```

Encodes the transition function.

Returns

the encoded formula/s

Reimplemented in pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists, pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentia
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF, and pyPMT.pypmt.encoders.basic.EncoderGrounded.

4.2.2.7 encode_execution_semantics()

```
pyPMT.pypmt.encoders.base.Encoder.encode_execution_semantics (
              self )
```

Encodes the possible needed mutexes between actions.

Returns

the encoded formula/s

Reimplemented in pyPMT.pypmt.encoders.basic.EncoderGrounded, pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifte
and pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.

4.2.2.8 encode_frame()

```
pyPMT.pypmt.encoders.base.Encoder.encode_frame (
              self )
```

Encodes the frame axioms.

Returns

the encoded formula/s

Reimplemented in pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted, pyPMT.pypmt.encoders.SequentialQFUF.Encod
and pyPMT.pypmt.encoders.basic.EncoderGrounded.

4.2.2.9 encode_goal_state()

```
pyPMT.pypmt.encoders.base.Encoder.encode_goal_state (
              self )
```

Encodes the goal state.

Returns

the encoded formula/s

Reimplemented in pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted, pyPMT.pypmt.encoders.SequentialQFUF.Encod
and pyPMT.pypmt.encoders.basic.EncoderGrounded.

4.2.2.10 encode_initial_state()

```
pyPMT.pypmt.encoders.base.Encoder.encode_initial_state (
              self )
```

Encodes the initial state.

Returns

the encoded formula/s

Reimplemented in pyPMT.pypmt.encoders.basic.EncoderGrounded, pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifte
and pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.

The documentation for this class was generated from the following file:

- pypmt/encoders/base.py

4.3 pyPMT.pypmt.encoders.basic.EncoderForall Class Reference

Inheritance diagram for pyPMT.pypmt.encoders.basic.EncoderForall:

Collaboration diagram for pyPMT.pypmt.encoders.basic.EncoderForall:

Public Member Functions

- __init__ (self, task)

Public Member Functions inherited from pyPMT.pypmt.encoders.basic.EncoderGrounded

- __iter__ (self)

  The iterator goes through the raw actions, allowing a clean interface when for example extracting a plan from a model.
- __len__ (self)
- get_action_var (self, name, t)

  Given a str representation of a fluent/action and a timestep, returns the respective Z3 var.
- extract_plan (self, model, horizon)

  Given a model of the encoding generated by this class and its horizon, extract a plan from it.
- encode (self, t)

  Builds and returns the formulas for a single transition step (from t to t+1).
- base_encode (self)

  Builds the encoding.
- encode_execution_semantics (self)

  Encodes execution semantics as specified by the modifier class held.
- create_variables (self, t)

  Creates state variables needed in the encoding for step t.
- encode_initial_state (self)

  Encodes formula defining initial state.
- encode_goal_state (self, t)

  Encodes formula defining goal state.
- encode_actions (self, t)

  Encodes the transition function.
- encode_frame (self, t)

  Encodes the explanatory frame axiom.

Additional Inherited Members

Public Attributes inherited from pyPMT.pypmt.encoders.basic.EncoderGrounded

- **task**
- **name**
- **modifier**
- **ctx**
- **compilation_results**

  linearize partial-order plan
- **grounding_results**
- **ground_problem**
- **z3_actions_to_up**
- **up_actions_to_z3**
- **up_fluent_to_z3**
- **frame_add**
- **frame_del**
- **frame_num**
- **formula**
- **formula_length**

Protected Member Functions inherited from pyPMT.pypmt.encoders.basic.EncoderGrounded

- _ground (self)

  Removes quantifiers from the UP problem via the QUANTIFIERS_REMOVING compiler and then grounds the problem using an available UP grounder.
- _populate_modifiers (self)

  Populates an index on which grounded actions can modify which fluents.
- _expr_to_z3 (self, expr, t, c=None)

  Traverses a UP AST in in-order and converts it to a Z3 expression.

4.3.1 Detailed Description

```
Implementation of a generalisation for numeric planning of the original work
in Kautz & Selman 1996
```

4.3.2 Constructor & Destructor Documentation

4.3.2.1 __init__()

```
pyPMT.pypmt.encoders.basic.EncoderForall.__init__ (
              self,
              task )
```

Reimplemented from pyPMT.pypmt.encoders.basic.EncoderGrounded.

The documentation for this class was generated from the following file:

- pypmt/encoders/basic.py

4.4 pyPMT.pypmt.encoders.basic.EncoderGrounded Class Reference

As its filename implies, it's the most basic encoding you can imagine.

Inheritance diagram for pyPMT.pypmt.encoders.basic.EncoderGrounded:

Collaboration diagram for pyPMT.pypmt.encoders.basic.EncoderGrounded:

Public Member Functions

- **__init__** (self, name, task, modifier)
- __iter__ (self)

  The iterator goes through the raw actions, allowing a clean interface when for example extracting a plan from a model.
- __len__ (self)
- get_action_var (self, name, t)

  Given a str representation of a fluent/action and a timestep, returns the respective Z3 var.
- extract_plan (self, model, horizon)

  Given a model of the encoding generated by this class and its horizon, extract a plan from it.
- encode (self, t)

  Builds and returns the formulas for a single transition step (from t to t+1).
- base_encode (self)

  Builds the encoding.
- encode_execution_semantics (self)

  Encodes execution semantics as specified by the modifier class held.
- create_variables (self, t)

  Creates state variables needed in the encoding for step t.
- encode_initial_state (self)

  Encodes formula defining initial state.
- encode_goal_state (self, t)

  Encodes formula defining goal state.
- encode_actions (self, t)

  Encodes the transition function.
- encode_frame (self, t)

  Encodes the explanatory frame axiom.

Public Attributes

- **task**
- **name**
- **modifier**
- **ctx**
- **compilation_results**

  linearize partial-order plan
- **grounding_results**
- **ground_problem**
- **z3_actions_to_up**
- **up_actions_to_z3**
- **up_fluent_to_z3**
- **frame_add**
- **frame_del**
- **frame_num**
- **formula**
- **formula_length**

Protected Member Functions

- _ground (self)

  Removes quantifiers from the UP problem via the QUANTIFIERS_REMOVING compiler and then grounds the problem using an available UP grounder.
- _populate_modifiers (self)

  Populates an index on which grounded actions can modify which fluents.
- _expr_to_z3 (self, expr, t, c=None)

  Traverses a UP AST in in-order and converts it to a Z3 expression.

4.4.1 Detailed Description

As its filename implies, it's the most basic encoding you can imagine.

It first uses UP to ground the problem, and then implements a state-based encoding of Planning as SAT. Details of the encoding can be found in the recent Handbooks of Satisfiability in the chapter written by Rintanen: Planning and SAT.

The classical way of improving the performance of encodings is to allow more than one action per step (layer). This class is really a "base class" for two encodings:

- sequential encoding: Kautz & Selman 1992 for the original encoding
- ForAll semantics: this implements a generalisation for numeric planning of the original work in Kautz & Selman 1996

4.4.2 Member Function Documentation

4.4.2.1 __iter__()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.__iter__ (
            self )
```

The iterator goes through the raw actions, allowing a clean interface when for example extracting a plan from a model.

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

Reimplemented in pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists.

4.4.2.2 __len__()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.__len__ (
            self )
```

Returns

the number of timesteps that have been encoded

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.4.2.3 _expr_to_z3()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded._expr_to_z3 (
            self,
            expr,
            t,
            c = None )  [protected]
```

Traverses a UP AST in in-order and converts it to a Z3 expression.

Parameters

expr	The tree expression node. (Can be a value, variable name, or operator)
t	The timestep for the Fluents to be considered
c	The context, which can be used to take into account free params

Returns

: An equivalent Z3 expression

Reimplemented in pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists.

4.4.2.4 _ground()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded._ground (
              self )  [protected]
```

Removes quantifiers from the UP problem via the QUANTIFIERS_REMOVING compiler and then grounds the problem using an available UP grounder.

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.4.2.5 _populate_modifiers()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded._populate_modifiers (
              self )  [protected]
```

Populates an index on which grounded actions can modify which fluents.

These are used afterwards for encoding the frame.

4.4.2.6 base_encode()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.base_encode (
              self )
```

Builds the encoding.

Populates the formula dictionary class attribute, where all the "raw" formulas are stored. Those will later be used by the encode function.

Reimplemented in pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists.

4.4.2.7 create_variables()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.create_variables (
              self,
              t )
```

Creates state variables needed in the encoding for step t.

Parameters

t	the timestep

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

Reimplemented in pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists.

4.4.2.8 encode()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.encode (
              self,
              t )
```

Builds and returns the formulas for a single transition step (from t to t+1).

Parameters

t	the current timestep we want the encoding for

Returns

: A dict with the different parts of the formula encoded

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

Reimplemented in pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists.

4.4.2.9 encode_actions()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.encode_actions (
              self,
              t )
```

Encodes the transition function.

That is, the actions. a -> Pre a -> Eff

Parameters

t	the timestep

Returns

: list of Z3 formulas asserting the actions

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

Reimplemented in pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists.

4.4.2.10 encode_execution_semantics()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.encode_execution_semantics (
              self )
```

Encodes execution semantics as specified by the modifier class held.

Returns

: axioms that specify execution semantics.

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.4.2.11 encode_frame()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.encode_frame (
              self,
              t )
```

Encodes the explanatory frame axiom.

basically for each fluent, to change in value it means that some action that can make it change has been executed f(x,y,z, t) != f(x,y,z, t+1) -> a $\vee$ b $\vee$ c

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.4.2.12 encode_goal_state()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.encode_goal_state (
              self,
              t )
```

Encodes formula defining goal state.

Parameters

t	the timestep

Returns

: Z3 formula asserting propositional and numeric subgoals

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.4.2.13 encode_initial_state()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.encode_initial_state (
              self )
```

Encodes formula defining initial state.

Returns

: Z3 formula asserting initial state

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.4.2.14 extract_plan()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.extract_plan (
              self,
              model,
              horizon )
```

Given a model of the encoding generated by this class and its horizon, extract a plan from it.

Returns

: an instance of a SMTSequentialPlan

Reimplemented in pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists.

4.4.2.15 get_action_var()

```
pyPMT.pypmt.encoders.basic.EncoderGrounded.get_action_var (
              self,
              name,
              t )
```

Given a str representation of a fluent/action and a timestep, returns the respective Z3 var.

Parameters

name	str representation of a fluent or action
t	the timestep we are interested in

Returns

: the corresponding Z3 variable

The documentation for this class was generated from the following file:

- pypmt/encoders/basic.py

4.5 pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists Class Reference

Inheritance diagram for pyPMT.pypmt.encoders.R2E.EncoderRelaxed2Exists:

4.6 pyPMT.pypmt.encoders.basic.EncoderSequential Class Reference

Inheritance diagram for pyPMT.pypmt.encoders.basic.EncoderSequential:

Collaboration diagram for pyPMT.pypmt.encoders.basic.EncoderSequential:

Public Member Functions

- __init__ (self, task)

Public Member Functions inherited from pyPMT.pypmt.encoders.basic.EncoderGrounded

- __iter__ (self)

    The iterator goes through the raw actions, allowing a clean interface when for example extracting a plan from a model.
- __len__ (self)
- get_action_var (self, name, t)

    Given a str representation of a fluent/action and a timestep, returns the respective Z3 var.
- extract_plan (self, model, horizon)

    Given a model of the encoding generated by this class and its horizon, extract a plan from it.
- encode (self, t)

    Builds and returns the formulas for a single transition step (from t to t+1).
- base_encode (self)

    Builds the encoding.
- encode_execution_semantics (self)

    Encodes execution semantics as specified by the modifier class held.
- create_variables (self, t)

    Creates state variables needed in the encoding for step t.
- encode_initial_state (self)

    Encodes formula defining initial state.
- encode_goal_state (self, t)

    Encodes formula defining goal state.
- encode_actions (self, t)

    Encodes the transition function.
- encode_frame (self, t)

    Encodes the explanatory frame axiom.

Additional Inherited Members

Public Attributes inherited from pyPMT.pypmt.encoders.basic.EncoderGrounded

- **task**
- **name**
- **modifier**
- **ctx**
- **compilation_results**

    linearize partial-order plan
- **grounding_results**
- **ground_problem**

- **z3_actions_to_up**
- **up_actions_to_z3**
- **up_fluent_to_z3**
- **frame_add**
- **frame_del**
- **frame_num**
- **formula**
- **formula_length**

Protected Member Functions inherited from pyPMT.pypmt.encoders.basic.EncoderGrounded

- _ground (self)

  Removes quantifiers from the UP problem via the QUANTIFIERS_REMOVING compiler and then grounds the problem using an available UP grounder.
- _populate_modifiers (self)

  Populates an index on which grounded actions can modify which fluents.
- _expr_to_z3 (self, expr, t, c=None)

  Traverses a UP AST in in-order and converts it to a Z3 expression.

4.6.1 Detailed Description

```
Implementation of the classical sequential encoding of Kautz & Selman 1992
where each timestep can have exactly one action.
```

4.6.2 Constructor & Destructor Documentation

4.6.2.1 __init__()

```
pyPMT.pypmt.encoders.basic.EncoderSequential.__init__ (
              self,
              task )
```

Reimplemented from pyPMT.pypmt.encoders.basic.EncoderGrounded.

The documentation for this class was generated from the following file:

- pypmt/encoders/basic.py

4.7 pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted Class Reference

Inheritance diagram for pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted:

Collaboration diagram for pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted:

Public Member Functions

- **__init__** (self, task)
- __len__ (self)
- encode_execution_semantics (self)

    Encodes execution semantics as specified by modifier class.
- create_variables (self, t)

    Creates variables needed in the encoding.
- encode_initial_state (self)

    Encodes formula defining initial state.
- encode_goal_state (self)

    Encodes formula defining goal state.
- encode_actions (self)

    Encodes the Actions.
- encode_frame (self)

    Encode explanatory frame axioms: a predicate retains its value unless it is modified by the effects of an action.
- extract_plan (self, model, horizon)

    Extracts plan from model of the formula.
- encode (self)

    Builds and returns the formulas for a single transition step (from t to t+1).

Public Member Functions inherited from pyPMT.pypmt.encoders.base.Encoder

- __iter__ (self)

    The iterator goes through the raw actions, allowing a clean interface when for example extracting a plan from a model.

Public Attributes

- **name**
- **task**
- **ctx**
- **z3_timestep_sort**
- **z3_timestep_var**
- **z3_objects_sort**
- **up_objects_to_z3**
- **z3_objects_to_up**
- **z3_actions_sort**
- **z3_action_variable**
- **z3_actions_mapping**
- **up_actions_mapping**
- **z3_action_parameters**
- **z3_fluents**
- **z3_typing_functions**
- **frame_idx**
- **formula**
- **formula_length**

Protected Member Functions

- _populate_modifiers (self)

    Populates an index on which grounded actions can modify which fluents.
- **_setup_typing** (self)

    Map the objects and types in the UP problem to Z3 clauses.
- _up_type_to_z3_type (self, type)
- _setup_actions (self)

    Create all the action execution infrastructure for Z3.
- **_setup_state** (self)

    Creates a UF representation for each planning fluent in UP.
- _ground (self)

    Implements the grounding of the task, if needed.
- _expr_to_z3 (self, expr, t, ctx=None)

4.7.1 Detailed Description

```
Roughly uses the encoding idea of max-parameters from:
Espasa Arxer, J., Miguel, I. J., Villaret, M., & Coll, J. (2019). Towards
Lifted Encodings for Numeric Planning in Essence Prime. Paper presented at
The 18th workshop on Constraint Modelling and Reformulation, Stamford,
Connecticut, United States.

It uses quantifiers to allow a very compact representation of the encoding
but that makes in general non-decidable.  The encoding seems to be correct
although has not been thoroughly tested as it is awfully slow for now.
```

4.7.2 Member Function Documentation

4.7.2.1 __len__()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted.__len__ (
              self )
```

Returns

the number of timesteps that have been encoded

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.7.2.2 _expr_to_z3()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted._expr_to_z3 (
              self,
              expr,
              t,
              ctx = None )  [protected]
```

```
Traverses a tree expression in-order and converts it to a Z3 expression.
expr: The tree expression node. (Can be a value, variable name, or operator)
t: The timestep for the Fluents to be considered
ctx: A context manager, as we need to take into account parameters from actions, fluents, etc ...
Returns A Z3 expression or Z3 value.
```

4.7.2.3 _ground()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted._ground (
              self )  [protected]
```

Implements the grounding of the task, if needed.

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.7.2.4 _populate_modifiers()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted._populate_modifiers (
              self )  [protected]
```

Populates an index on which grounded actions can modify which fluents.

These are used afterwards for encoding the frame.

4.7.2.5 _setup_actions()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted._setup_actions (
              self )  [protected]
```

Create all the action execution infrastructure for Z3.

We will have a UF named Exec, that gets a timestep and returns which action is being executed at that timestep.

To store the parameters for the actions in each timestep, we will define too a set of param_x uninterpreted functions. We will have a number of those variables equal to the maximum number of parameters between all actions. These, similarly to the Exec function will given a timestep, is going to tell us to which object that parameter is being mapped to.

4.7.2.6 _up_type_to_z3_type()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted._up_type_to_z3_type (
              self,
              type )  [protected]
```

```
 Given a UP type, return the Z3 sort
```

4.7.2.7 create_variables()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted.create_variables (
              self,
              t )
```

Creates variables needed in the encoding.

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.7.2.8 encode()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted.encode (
              self )
```

Builds and returns the formulas for a single transition step (from t to t+1).

Parameters

t	timestep where the goal is true

Returns

: A dict with the different parts of the formula encoded

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.7.2.9 encode_actions()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted.encode_actions (
            self )
```

Encodes the Actions.

Returns

actions: list of Z3 formulas asserting the actions

The main idea is to do: forall t - int (t >= 0 ∧ t < t_goal ∧ is_plane(param_1(t)) ∧ is_city(param_2(t)) ∧ is_↩ city(param_3(t)) ∧ # typing exec(t) = fly) -> (... # precondition ... # effect)

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.7.2.10 encode_execution_semantics()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted.encode_execution_semantics (
            self )
```

Encodes execution semantics as specified by modifier class.

Returns

axioms that specify execution semantics.

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.7.2.11 encode_frame()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted.encode_frame (
            self )
```

Encode explanatory frame axioms: a predicate retains its value unless it is modified by the effects of an action.

f(x,y,z, t) != f(x,y,z, t+1) -> (exec(t) = action1 ∧ param2(t) = z) ∨ (exec(t) = action3 ∧ param1(t) = x ∧ param3(t) = y) ∨ ...

The empty disjunction on the RHS will evaluate to false if there are no actions that can change the value of f

Returns

: list of frame axioms

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.7.2.12 encode_goal_state()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted.encode_goal_state (
              self )
```

Encodes formula defining goal state.

Returns

goal: Z3 formula asserting propositional and numeric subgoals

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.7.2.13 encode_initial_state()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted.encode_initial_state (
              self )
```

Encodes formula defining initial state.

Returns

initial: Z3 formula asserting initial state

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.7.2.14 extract_plan()

```
pyPMT.pypmt.encoders.SequentialLifted.EncoderSequentialLifted.extract_plan (
              self,
              model,
              horizon )
```

Extracts plan from model of the formula.

Plan returned is linearized.

Parameters

model	Z3 model of the planning formula.
encoder	encoder object, contains maps variable/variable names.

Returns

: dictionary containing plan. Keys are steps, values are actions.

The documentation for this class was generated from the following file:

- pypmt/encoders/SequentialLifted.py

4.8 pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF Class Reference

Inheritance diagram for pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF:

Collaboration diagram for pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF:

Public Member Functions

- **__init__** (self, task)
- __len__ (self)
- encode_execution_semantics (self)

  Encodes execution semantics as specified by modifier class.
- encode_initial_state (self)

  Encodes formula defining initial state.
- encode_goal_state (self)

  Encodes formula defining goal state.
- encode_actions (self)

  Encodes the Actions.
- encode_frame (self)

  f(x,y,z, t) != f(x,y,z, t+1) -> (exec(t) = action1 /\ param2(t) = z) \/ (exec(t) = action3 /\ param1(t) = x /\ param3(t) = y) \/ ... for each possible grounding of f()
- extract_plan (self, model, horizon)

  Extracts plan from model of the formula.
- encode (self, t)

  Builds and returns the formulas for a single transition step (from t to t+1).
- base_encode (self)

  Builds the encoding.

Public Member Functions inherited from pyPMT.pypmt.encoders.base.Encoder

- __iter__ (self)

  The iterator goes through the raw actions, allowing a clean interface when for example extracting a plan from a model.
- create_variables (self, t)

  Creates the Z3 variables needed for a given timestep.

Public Attributes

- **name**
- **task**
- **ctx**
- **grounding_results**
- **ground_problem**
- **z3_timestep_sort**
- **z3_timestep_var**
- **z3_objects_sort**
- **up_objects_to_z3**
- **z3_objects_to_up**
- **z3_actions_sort**
- **z3_action_variable**

- **z3_actions_mapping**
- **up_actions_mapping**
- **z3_action_parameters**
- **z3_fluents**
- **z3_typing_functions**
- **frame_idx**
- **formula**
- **formula_length**

Protected Member Functions

- _populate_modifiers (self)

  Populates an index on which grounded actions can modify which fluents.
- **_setup_typing** (self)

  map the objects and types in the UP problem to Z3 clauses
- **_up_type_to_z3_type** (self, type)

  Given a UP type, return the corresponding Z3 sort.
- _setup_actions (self)

  Create all the action execution infrastructure for Z3.
- **_setup_state** (self)

  Creates a UF representation for each planning fluent in UP.
- _ground (self)

  Implements the grounding of the task, if needed.
- _expr_to_z3 (self, expr, t, ctx=None)

  Traverses a tree expression in-order and converts it to a Z3 expression.

4.8.1 Detailed Description

```
Roughly uses the encoding idea of max-parameters from:
Espasa Arxer, J., Miguel, I. J., Villaret, M., & Coll, J. (2019). Towards
Lifted Encodings for Numeric Planning in Essence Prime. Paper presented at
The 18th workshop on Constraint Modelling and Reformulation, Stamford,
Connecticut, United States.

It derives from the SequentialLifted but removes all quantifiers
```

4.8.2 Member Function Documentation

4.8.2.1 __len__()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.__len__ (
              self )
```

Returns

the number of timesteps that have been encoded

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.8.2.2 _expr_to_z3()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF._expr_to_z3 (
              self,
              expr,
              t,
              ctx = None )  [protected]
```

Traverses a tree expression in-order and converts it to a Z3 expression.

expr: The tree expression node. (Can be a value, variable name, or operator) t: The timestep for the Fluents to be considered ctx: A context manager, as we need to take into account parameters from actions, fluents, etc ... Returns A Z3 expression or Z3 value.

4.8.2.3 _ground()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF._ground (
              self )  [protected]
```

Implements the grounding of the task, if needed.

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.8.2.4 _populate_modifiers()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF._populate_modifiers (
              self )  [protected]
```

Populates an index on which grounded actions can modify which fluents.

These are used afterwards for encoding the frame.

4.8.2.5 _setup_actions()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF._setup_actions (
              self )  [protected]
```

Create all the action execution infrastructure for Z3.

We will have a UF named Exec, that gets a timestep and returns which action is being executed at that timestep.

To store the parameters for the actions in each timestep, we will define too a set of param_x uninterpreted functions. We will have a number of those variables equal to the maximum number of parameters between all actions. These, similarly to the Exec function will given a timestep, is going to tell us to which object that parameter is being mapped to.

4.8.2.6 base_encode()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.base_encode (
              self )
```

Builds the encoding.

Populates the formula dictionary, where all the "raw" formulas are stored

Returns

None

4.8.2.7 encode()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.encode (
              self,
              t )
```

Builds and returns the formulas for a single transition step (from t to t+1).

Parameters

t	timestep where the goal is true

Returns

encoded_formula: A dict with the different parts of the formula encoded

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.8.2.8 encode_actions()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.encode_actions (
              self )
```

Encodes the Actions.

Returns

actions: list of Z3 formulas asserting the actions

The main idea is to do: forall t - int (t >= 0 $\wedge$ t < t_goal $\wedge$ is_plane(param_1(t)) $\wedge$ is_city(param_2(t)) $\wedge$ is_↩
city(param_3(t)) $\wedge$ # typing exec(t) = fly) -> (... # precondition ... # effect)

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.8.2.9 encode_execution_semantics()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.encode_execution_semantics (
              self )
```

Encodes execution semantics as specified by modifier class.

Returns

axioms that specify execution semantics.

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.8.2.10 encode_frame()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.encode_frame (
              self )
```

f(x,y,z, t) != f(x,y,z, t+1) -> (exec(t) = action1 $\wedge$ param2(t) = z) $\vee$ (exec(t) = action3 $\wedge$ param1(t) = x $\wedge$ param3(t) = y) $\vee$ ... for each possible grounding of f()

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.8.2.11 encode_goal_state()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.encode_goal_state (
              self )
```

Encodes formula defining goal state.

Returns

goal: Z3 formula asserting propositional and numeric subgoals

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.8.2.12 encode_initial_state()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.encode_initial_state (
              self )
```

Encodes formula defining initial state.

Returns

initial: Z3 formula asserting initial state

Reimplemented from pyPMT.pypmt.encoders.base.Encoder.

4.8.2.13 extract_plan()

```
pyPMT.pypmt.encoders.SequentialQFUF.EncoderSequentialQFUF.extract_plan (
              self,
              model,
              horizon )
```

Extracts plan from model of the formula.

Plan returned is linearized.

Parameters

model	Z3 model of the planning formula.
encoder	encoder object, contains maps variable/variable names.

Returns

plan: dictionary containing plan. Keys are steps, values are actions.

The documentation for this class was generated from the following file:

- pypmt/encoders/SequentialQFUF.py

4.9 pyPMT.pypmt.planner.lifted.LiftedSearch Class Reference

Inheritance diagram for pyPMT.pypmt.planner.lifted.LiftedSearch:

Collaboration diagram for pyPMT.pypmt.planner.lifted.LiftedSearch:

Public Member Functions

- search (self)
- dump_smtlib_to_file (self, t, path)

Public Member Functions inherited from pyPMT.pypmt.planner.base.Search

- **__init__** (self, encoder, scheduler, ∗∗kwargs)

Public Attributes

- **solution**
- **solver**
- **horizon**

Public Attributes inherited from pyPMT.pypmt.planner.base.Search

- **encoder**
- **found**
- **solution**
- **solver**
- **horizon**
- **scheduler**
- **propagator**

4.9.1 Member Function Documentation

4.9.1.1 dump_smtlib_to_file()

```
pyPMT.pypmt.planner.lifted.LiftedSearch.dump_smtlib_to_file (
              self,
              t,
              path )
```

Reimplemented from pyPMT.pypmt.planner.base.Search.

4.9.1.2 search()

```
pyPMT.pypmt.planner.lifted.LiftedSearch.search (
              self )
```

Reimplemented from pyPMT.pypmt.planner.base.Search.

The documentation for this class was generated from the following file:

- pypmt/planner/lifted.py

4.10 pyPMT.pypmt.modifiers.modifierLinear.LinearModifier Class Reference

Inheritance diagram for pyPMT.pypmt.modifiers.modifierLinear.LinearModifier:

Collaboration diagram for pyPMT.pypmt.modifiers.modifierLinear.LinearModifier:

Public Member Functions

- __init__ (self)
- encode (self, encoder, variables)

Additional Inherited Members

Public Attributes inherited from pyPMT.pypmt.modifiers.base.Modifier

- **name**
- **mutexes**

4.10.1 Detailed Description

```
Linear modifier, contains method to implement sequential execution semantics.
```

4.10.2 Constructor & Destructor Documentation

4.10.2.1 __init__()

```
pyPMT.pypmt.modifiers.modifierLinear.LinearModifier.__init__ (
              self )
```

Reimplemented from pyPMT.pypmt.modifiers.base.Modifier.

4.10.3 Member Function Documentation

4.10.3.1 encode()

```
pyPMT.pypmt.modifiers.modifierLinear.LinearModifier.encode (
              self,
              encoder,
              variables )
```

Reimplemented from pyPMT.pypmt.modifiers.base.Modifier.

The documentation for this class was generated from the following file:

- pypmt/modifiers/modifierLinear.py

4.11 pyPMT.pypmt.modifiers.base.Modifier Class Reference

Inheritance diagram for pyPMT.pypmt.modifiers.base.Modifier:

Public Member Functions

- **__init__** (self, name)
- encode (self, encoder, variables)

Public Attributes

- **name**
- **mutexes**

4.11.1 Detailed Description

```
Modifier class.
```

4.11.2 Member Function Documentation

4.11.2.1 encode()

```
pyPMT.pypmt.modifiers.base.Modifier.encode (
              self,
              encoder,
              variables )
```

Reimplemented in pyPMT.pypmt.modifiers.modifierParallel.ParallelModifier.

The documentation for this class was generated from the following file:

- pypmt/modifiers/base.py

4.12 pyPMT.pypmt.modifiers.modifierParallel.ParallelModifier Class Reference

Inheritance diagram for pyPMT.pypmt.modifiers.modifierParallel.ParallelModifier:

Collaboration diagram for pyPMT.pypmt.modifiers.modifierParallel.ParallelModifier:

Public Member Functions

- __init__ (self)
- encode (self, encoder, actions)

    Computes mutually exclusive actions: Two actions (a1, a2) are mutex if:

Additional Inherited Members

Public Attributes inherited from pyPMT.pypmt.modifiers.base.Modifier

- **name**
- **mutexes**

4.12.1 Detailed Description

```
Parallel modifier, contains method to implement parallel execution semantics.
```

4.12.2 Constructor & Destructor Documentation

4.12.2.1 __init__()

```
pyPMT.pypmt.modifiers.modifierParallel.ParallelModifier.__init__ (
              self )
```

Reimplemented from pyPMT.pypmt.modifiers.base.Modifier.

4.12.3 Member Function Documentation

4.12.3.1 encode()

```
pyPMT.pypmt.modifiers.modifierParallel.ParallelModifier.encode (
              self,
              encoder,
              actions )
```

Computes mutually exclusive actions: Two actions (a1, a2) are mutex if:

```
1- The effects of a1 can prevent the execution of a2
- intersection pre_a1 and eff_a2 (or viceversa) is non-empty

2- The effects of a1 and a2 interfere
- intersection between eff_a1+ and eff_a2- (or viceversa) is non-empty
- intersection between numeric effects is non-empty
```

Note that condition 1 is non-symmetric, while condition 2 is.

See,'Efficient SMT Encodings for the Petrobras Domain' Espasa et al. Sec 3.3 - Parallel Plans

Returns

mutex: list of tuples defining action mutexes

Reimplemented from pyPMT.pypmt.modifiers.base.Modifier.

The documentation for this class was generated from the following file:

- pypmt/modifiers/modifierParallel.py

4.13 pyPMT.pypmt.planner.QFUF.QFUFSearch Class Reference

Inheritance diagram for pyPMT.pypmt.planner.QFUF.QFUFSearch:

Collaboration diagram for pyPMT.pypmt.planner.QFUF.QFUFSearch:

Public Member Functions

- search (self)
- dump_smtlib_to_file (self, t, path)

Public Member Functions inherited from pyPMT.pypmt.planner.base.Search

- **__init__** (self, encoder, scheduler, **kwargs)

Public Attributes

- **horizon**
- **solution**
- **solver**

Public Attributes inherited from pyPMT.pypmt.planner.base.Search

- **encoder**
- **found**
- **solution**
- **solver**
- **horizon**
- **scheduler**
- **propagator**

4.13.1 Detailed Description

```
Base class defining search schemes.
```

4.13.2 Member Function Documentation

4.13.2.1 dump_smtlib_to_file()

```
pyPMT.pypmt.planner.QFUF.QFUFSearch.dump_smtlib_to_file (
              self,
              t,
              path )
```

Reimplemented from pyPMT.pypmt.planner.base.Search.

4.13.2.2 search()

```
pyPMT.pypmt.planner.QFUF.QFUFSearch.search (
              self )
```

Reimplemented from pyPMT.pypmt.planner.base.Search.

The documentation for this class was generated from the following file:

- pypmt/planner/QFUF.py

4.14 pyPMT.pypmt.planner.base.Search Class Reference

Inheritance diagram for pyPMT.pypmt.planner.base.Search:

Public Member Functions

- **__init__** (self, encoder, scheduler, ∗∗kwargs)
- **search** (self)
- **dump_smtlib_to_file** (self, t, path)

Public Attributes

- **encoder**
- **found**
- **solution**
- **solver**
- **horizon**
- **scheduler**
- **propagator**

4.14.1 Detailed Description

```
Base class defining search schemes.
```

The documentation for this class was generated from the following file:

- pypmt/planner/base.py

4.15 pyPMT.pypmt.up.SMTPlanner.SMTPlanner Class Reference

Inheritance diagram for pyPMT.pypmt.up.SMTPlanner.SMTPlanner:

Collaboration diagram for pyPMT.pypmt.up.SMTPlanner.SMTPlanner:

Public Member Functions

- **__init__** (self, ∗∗options)
- str **name** (self)
- **destroy** (self)

Static Public Member Functions

- **supported_kind** ()
- **supports** (problem_kind)

Public Attributes

- **configuration**
- **encoder**
- **search_strategy**
- **upper_bound**
- **schedule**
- **run_validation**
- **name**

Protected Member Functions

- 'up.engines.PlanGenerationResult' **_solve** (self, 'up.model.Problem' problem, Optional[Callable[['up.↩ engines.PlanGenerationResult'], None]] callback=None, Optional[float] timeout=None, Optional[IO[str]] output_stream=None)

The documentation for this class was generated from the following file:

- pypmt/up/SMTPlanner.py

4.16 pyPMT.pypmt.planner.SMT.SMTSearch Class Reference

Inheritance diagram for pyPMT.pypmt.planner.SMT.SMTSearch:

Collaboration diagram for pyPMT.pypmt.planner.SMT.SMTSearch:

Public Member Functions

- search (self)
- dump_smtlib_to_file (self, t, path)

Public Member Functions inherited from pyPMT.pypmt.planner.base.Search

- **__init__** (self, encoder, scheduler, ∗∗kwargs)

Public Attributes

- **horizon**
- **solution**
- **solver**

Public Attributes inherited from pyPMT.pypmt.planner.base.Search

- **encoder**
- **found**
- **solution**
- **solver**
- **horizon**
- **scheduler**
- **propagator**

4.16.1 Detailed Description

```
Basic grounded incremental search
```

4.16.2 Member Function Documentation

4.16.2.1 dump_smtlib_to_file()

```
pyPMT.pypmt.planner.SMT.SMTSearch.dump_smtlib_to_file (
              self,
              t,
              path )
```

Reimplemented from pyPMT.pypmt.planner.base.Search.

4.16.2.2 search()

```
pyPMT.pypmt.planner.SMT.SMTSearch.search (
              self )
```

Reimplemented from pyPMT.pypmt.planner.base.Search.

The documentation for this class was generated from the following file:

- pypmt/planner/SMT.py

4.17 pyPMT.pypmt.planner.plan.smt_sequential_plan.SMTSequential↩ Plan Class Reference

Public Member Functions

- **__init__** (self, plan, task)
- __len__ (self)
  Returns the length of the plan.
- __str__ (self)
  Returns the plan as a stringin PDDL format.
- cost (self)
  Computes the cost of the plan.
- validate (self)
  Validates plan (when one is found).

Public Attributes

- **isvalid**
- **cost_value**
- **validation_fail_reason**
- **plan**
- **task**

4.17.1 Member Function Documentation

4.17.1.1 __len__()

```
pyPMT.pypmt.planner.plan.smt_sequential_plan.SMTSequentialPlan.__len__ (
             self )
```

Returns the length of the plan.

Returns

the length of the actions in the plan.

4.17.1.2 __str__()

```
pyPMT.pypmt.planner.plan.smt_sequential_plan.SMTSequentialPlan.__str__ (
             self )
```

Returns the plan as a stringin PDDL format.

Returns

the plan as a string.

4.17.1.3 cost()

```
pyPMT.pypmt.planner.plan.smt_sequential_plan.SMTSequentialPlan.cost (
             self )
```

Computes the cost of the plan.

Returns

cost: dictionary containing the cost of the plan.

4.17.1.4 validate()

```
pyPMT.pypmt.planner.plan.smt_sequential_plan.SMTSequentialPlan.validate (
             self )
```

Validates plan (when one is found).

Parameters

domain	path to PDDL domain file.
problem	path to PDDL problem file.

Returns

plan: string containing plan if plan found is valid, None otherwise.

The documentation for this class was generated from the following file:

- pypmt/planner/plan/smt_sequential_plan.py

Index