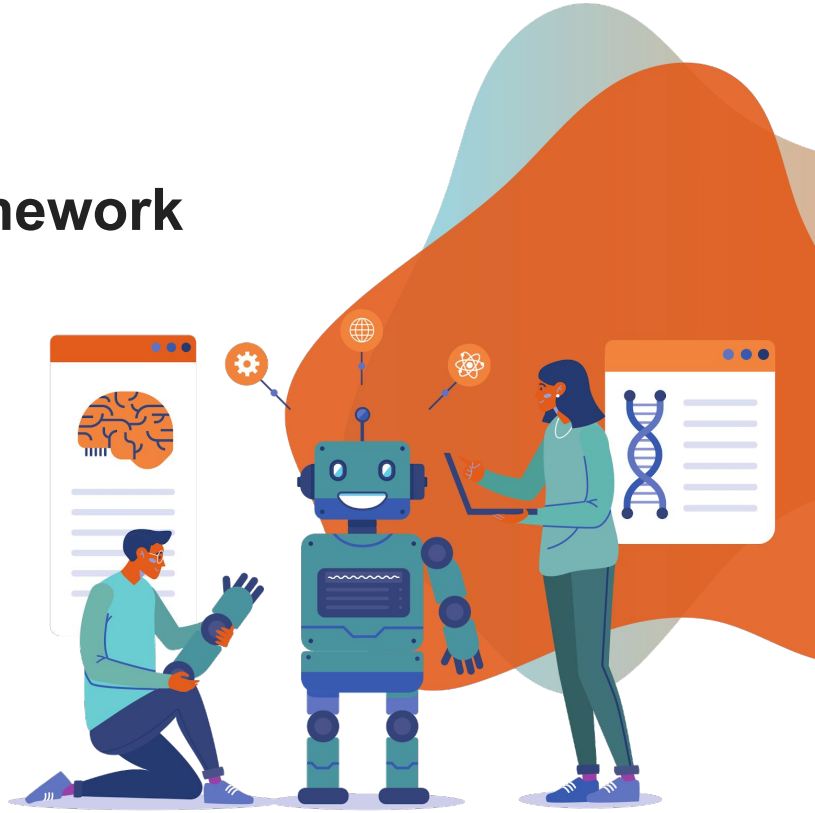
The AIPlan4EU Unified Planning Framework



2022/05/19

Alessandro Trapasso
https://alee08.github.io/

The AIPlan4EU project is funded by the European Commission - H2020 research and innovation programme under grant agreement No 101016442

Summary

- Modeling a problem using the UP library (single agent)
  - Installation and modeling
  - Transformation of the problem into the PDDL formalism
  - Generation of plans

- Modeling a MA-problem using the UP library (Multi agent)
  - Class diagram
  - Multi-Agent planning Structure
  - Modelling MA-Problem
  - MA-PDDL and FMAP

Let's install the Unified Planning Library

Let's start by installing the library! The UP library can be installed directly from PIP with a single command. (The `--pre` option installs the latest pre-release version)

```
!pip install --pre -U unified-planning
```

We install pyperplan from its repository

```
!rm -rf up-pyperplan
!git clone https://github.com/aiplan4eu/up-pyperplan
!pip install up-pyperplan/
```

Fast-Downward is also integrated

```
!rm -rf up-fast-downward
!git clone https://github.com/aiplan4eu/up-fast-downward
!pip install up-fast-downward/
```

ENHSP requires Java 17 and is then installed from its repo

```
!apt-get install openjdk-17-jdk
!rm -rf up-enhsp
!git clone https://github.com/aiplan4eu/up-enhsp.git
!pip install up-enhsp/
```

Tamer can also be automatically installed from its repository

```
!rm -rf up-tamer
!git clone https://github.com/aiplan4eu/up-tamer
!pip install up-tamer/
```

Unified Planning (UP) Library

```python
import unified_planning
from unified_planning.shortcuts import *
from collections import namedtuple
from unified_planning.io.pddl_writer import PDDLWriter

Example = namedtuple('Example', ['problem', 'plan'])

def get_example_problems():
    problems = {}
    # robot
    Location = UserType('Location')
    robot_at = Fluent('robot_at', BoolType(), [Location])
    battery_charge = Fluent('battery_charge', RealType(0, 100))
    move = InstantaneousAction('move', l_from=Location, l_to=Location)
    l_from = move.parameter('l_from')
    l_to = move.parameter('l_to')
    move.add_precondition(GE(battery_charge, 10))
    move.add_precondition(Not(Equals(l_from, l_to)))
    move.add_precondition(robot_at(l_from))
    move.add_precondition(Not(robot_at(l_to)))
    move.add_effect(robot_at(l_from), False)
    move.add_effect(robot_at(l_to), True)
    move.add_effect(battery_charge, Minus(battery_charge, 10))
    l1 = Object('l1', Location)
    l2 = Object('l2', Location)
    problem = Problem('robot')
    problem.add_fluent(robot_at)
    problem.add_fluent(battery_charge)
    problem.add_action(move)
    problem.add_object(l1)
    problem.add_object(l2)
    problem.set_initial_value(robot_at(l1), True)
    problem.set_initial_value(robot_at(l2), False)
    problem.set_initial_value(battery_charge, 100)
    problem.add_goal(robot_at(l2))
    plan = unified_planning.plan.SequentialPlan([unified_planning.plan.ActionInstance(move, (ObjectExp(l1), ObjectExp(l2)))])
    robot = Example(problem=problem, plan=plan)
    problems['robot'] = robot

    w = PDDLWriter(problem)
    print(w.get_domain())
    print(w.get_problem())
```

Fluent

Action

Object

Problem

PDDLWriter

Transformation of the problem into PDDL

PDDL Writer

```
w = PDDLWriter(problem)
print(w.get_domain())
print(w.get_problem())
```

PDDL domain and PDDL problem

```
(unified-planning) alee8@alee8-XPS-15-9510:~/Scrivania/unified-planning/unified_planning/test/examples$ python3 realistic.py
(define (domain robot-domain)
 (:requirements :strips :typing :negative-preconditions :equality :numeric-fluents)
 (:types Location)
 (:predicates (robot_at ?p0 - Location))
 (:functions (battery_charge))
 (:action move
  :parameters ( ?l_from - Location ?l_to - Location)
  :precondition (and (<= 10 (battery_charge)) (not (= ?l_from ?l_to)) (robot_at ?l_from) (not (robot_at ?l_to)))
  :effect (and (not (robot_at ?l_from)) (robot_at ?l_to) (assign (battery_charge) (- (battery_charge) 10))))
)

(define (problem robot-problem)
 (:domain robot-domain)
 (:objects
   l1 l2 - Location
 )
 (:init (robot_at l1) (= (battery_charge) 100))
 (:goal (and (robot_at l2)))
)
```

Planners

```python
from unified_planning.solvers import PlanGenerationResultStatus

for planner_name in ['pyperplan', 'fast_downward']:
    with OneshotPlanner(name=planner_name) as planner:
        result = planner.solve(problem)
        if result.status == PlanGenerationResultStatus.SOLVED_SATISFICING:
            print(f'{planner_name} found a plan.\nThe plan is: {result.plan}')
        else:
            print("No plan found.")
```

```
Pyperplan returned: [move(l1, l2)]
```

We can invoke different instances of a planner in parallel or different planners and return the first plan that is generated effortlessly.

```python
with OneshotPlanner(name='tamer') as planner:
    result = planner.solve(problem)
    if result.status == PlanGenerationResultStatus.SOLVED_SATISFICING:
        print(f'{planner.name} found a plan.\n The plan is: {result.plan}')
    else:
        print("No plan found.")
```

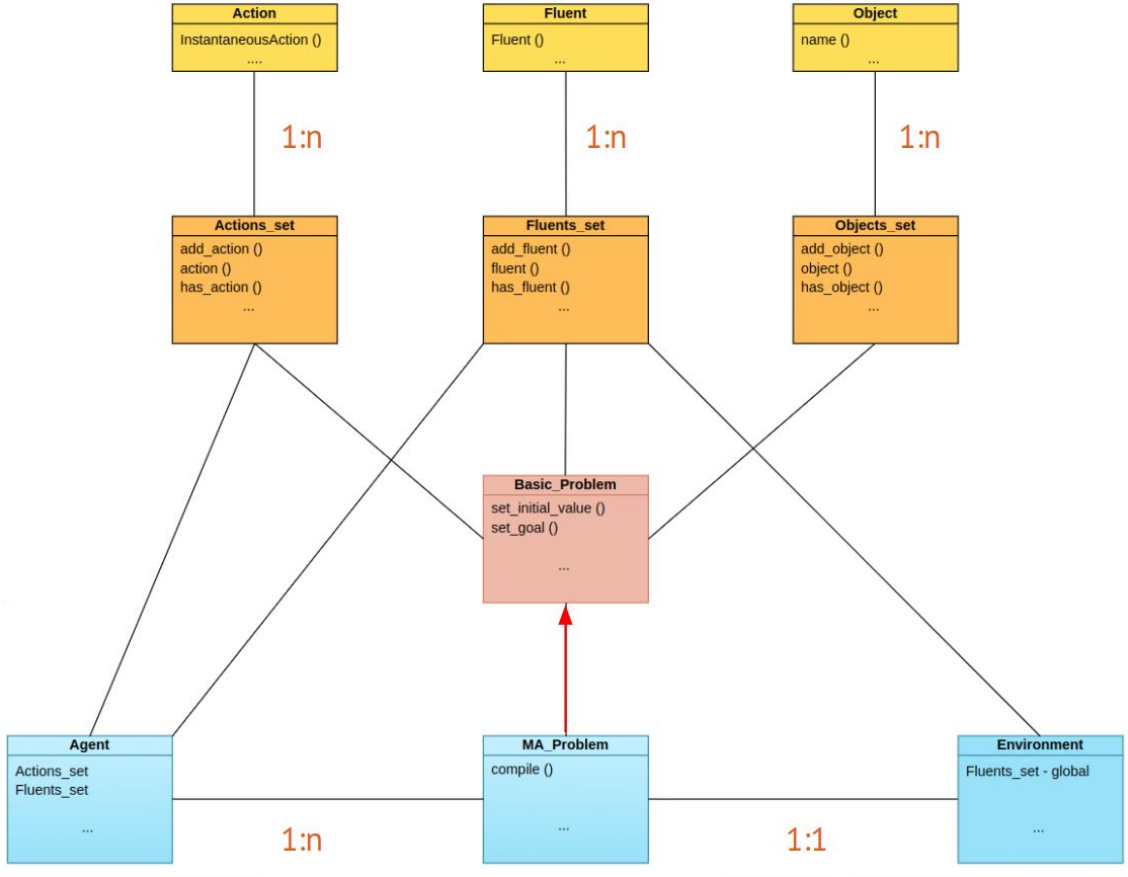
```
Tamer found a plan.
 The plan is: [move(l1, l2)]
```

```python
with OneshotPlanner(names=['tamer', 'tamer', 'enhsp'],
                    params=[{'heuristic': 'hadd'}, {'heuristic': 'hmax'}, {}]) as planner:
    plan = planner.solve(problem).plan
    print(f'{planner.name} returned: {plan}')
```

```
Parallel returned: [move(l1, l2)]
```

Planner	NEGATIVE_CONDITIONS	DECREASE_EFFECTS	NUMERIC_FLUENTS	CONTINUOUS_NUMBERS	EQUALITY	FLAT_TYPING
tamer	True	False	True	True	True	True
pyperplan	False	False	False	False	False	True

Class diagram



Work in progress

Human

Multi-agent planning structure

Agent

Environment

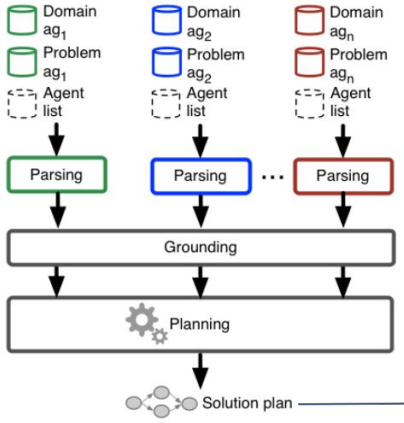
PDDLWriter_MA

```
ma_realistic.py

problem_d = examples_realistic['depot'].problem
problem_t = examples_realistic['depot_truck'].problem
...
robot1 = Agent('depot0')
robot2 = Agent('distributor0')
robot3 = Agent('distributor1')
robot4 = Agent('truck0')
robot5 = Agent('truck1')
environment = Environment_ma()
...
ma_problem = MultiAgentProblem('ma_depot')
ma_problem.add_agent(robot1)
...
ma_problem.add_shared_data(...)
ma_problem.add_fluent_function(...)
ma_depot = ma_problem.compile_ma()
...
ma_problem.write_ma_problem(ma_depot)
plan = ma_problem.FMAP_planner()
```

MultiAgentProblem

```
realistic.py
place = UserType('place', None)
depot = UserType('depot', place)
truck = UserType('depot', None)
...
clear_h = Fluent('clear_h', BoolType(), [hoist])
located = Fluent('located', BoolType(), [hoist, place])
on_h = Fluent('on_h', BoolType(), [crate, hoist])
...
truck0 = Object('truck0', truck)
truck1 = Object('truck1', truck)
...
drive = InstantaneousAction('Drive', t=truck, x=place, y=place)
t = drive.parameter('t')
x = drive.parameter('x')
y = drive.parameter('y')
drive.add_precondition(at(t, x))
drive.add_effect(at(t, y), True)
...
problem.add_object(crate0)
problem.add_object(crate1)
...
problem.add_action(drive)
problem.add_action(load)
problem.add_action(unload)
...
problem.add_fluent(located, default_initial_value=False)
problem.add_fluent(placed, default_initial_value=False)
...
problem.set_initial_value(pos_p(crate0, distributor0), True)
problem.set_initial_value(clear_s(crate0), True)
...
problem.add_goal(on_s(crate0, pallet2))
problem.add_goal(on_s(crate1, pallet1))
...
depot_truck = Example(problem=problem, plan=plan)
problems['depot_truck'] = depot_truck
```

FMAP



POP

https://bitbucket.org/altorler/fmap/src/master/

A. Torreño, O. Sapena, E. Onaindia
"FMAP: A platform for the development of distributed multi-agent planning systems"
Knowledge-Based Systems, vol. 145, pp.166–168, Elsevier, 2018

Alejandro Torreño, Oscar Sapena e Eva Onaindia
Global Heuristics for Distributed Cooperative Multi-Agent Planning
ICAPS 2015. 25th International Conference on Automated Planning and Scheduling, AAAI Press, pp. 225-233, (2015)

Depot problem

```python
place = UserType('place', None)
hoist = UserType('hoist', None)
surface = UserType('surface', None)
depot = UserType('depot', place)
distributor = UserType('distributor', place)
...
clear_h = Fluent('clear_h', None, [hoist])
clear_s = Fluent('clear_s', None, [surface])
at = Fluent('at', None, [truck, place])
...
truck0 = Object('truck0', truck)
truck1 = Object('truck1', truck)
drive = InstantaneousAction('Drive', t=truck, x=place, y=place)
t = drive.parameter('t')
x = drive.parameter('x')
y = drive.parameter('y')
drive.add_precondition(at(t, x))
drive.add_effect(at(t, y), True)
load = InstantaneousAction('Load', t=truck, p=place, c=crate, h=hoist)
...
unload = InstantaneousAction('Unload', t=truck, p=place, c=crate, h=hoist)
...
problem = Problem('depot_truck')
depot0 = Object('depot0', depot)
distributor0 = Object('distributor0', distributor)
distributor1 = Object('distributor1', distributor)
...
problem.add_object(distributor0)
problem.add_object(distributor1)
...
problem.add_action(drive)
problem.add_action(load)
problem.add_action(unload)
problem.add_fluent(clear_s, default_initial_value=False)
problem.add_fluent(clear_h, default_initial_value=False)
...
problem.set_initial_value(at(truck0, depot0), False)
problem.set_initial_value(at(truck0, distributor0), False)
...
problem.add_goal(on_s(crate0, pallet2))
problem.add_goal(on_s(crate1, pallet1))
depot_truck = Example(problem=problem)
problems['depot_truck'] = depot_truck
```

Type and Subtype

Same fluent associated with two different types

Set Objects

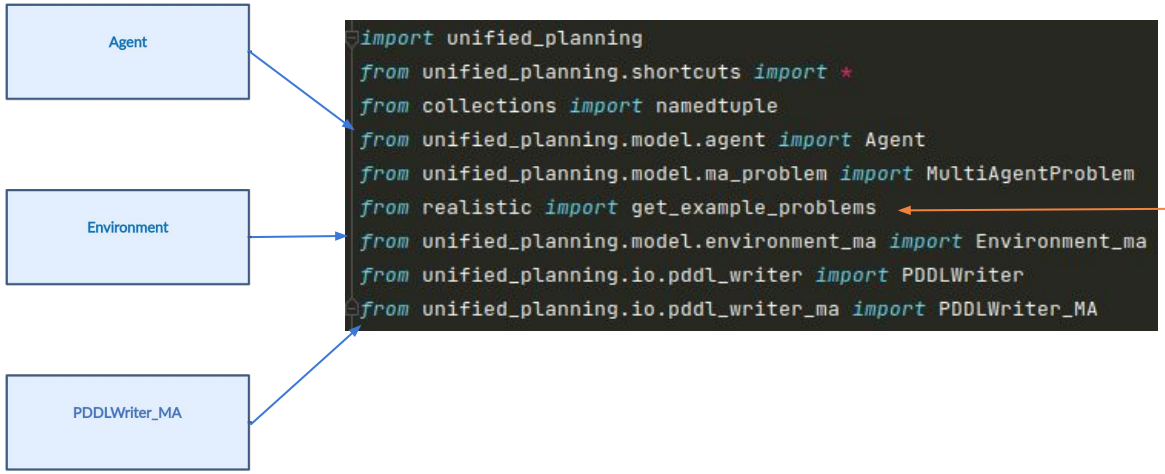
Set Actions:
parameters,
preconditions,
effects

Set Objects

I add to the problem:
objects,
actions,
fluents,
initial values,
goals

```
(define (problem pddl-problem)
(:domain pddl-domain)
(:objects
 crate0 crate1 - crate
 truck0 truck1 - truck
 depot0 - depot
 distributor0 distributor1 - distributor
 pallet0 pallet1 pallet2 - pallet
 hoist0 hoist1 hoist2 - hoist
)
(:shared-data
  (clear ?x - (either hoist surface))
  ((at ?t - truck) - place)
  ((pos ?c - crate) - (either place truck))
  ((on ?c - crate) - (either hoist truck surface)) -
(either depot0 distributor0 distributor1 truck1)
)
(:init
 (myAgent truck0)
 (= (pos crate0) distributor0)
 (clear crate0)
 (= (on crate0) pallet1)
 (= (pos crate1) depot0)
 (clear crate1)
 (= (on crate1) pallet0)
 (= (at truck0) distributor1)
 (= (at truck1) depot0)
 (= (located hoist0) depot0)
 (clear hoist0)
 (= (located hoist1) distributor0)
 (clear hoist1)
 (= (located hoist2) distributor1)
 (clear hoist2)
 (= (placed pallet0) depot0)
 (not (clear pallet0))
 (= (placed pallet1) distributor0)
 (not (clear pallet1))
 (= (placed pallet2) distributor1)
 (clear pallet2)
)
(:global-goal (and
 (= (on crate0) pallet2)
 (= (on crate1) pallet1)
))
)
```

Import for Ma-Problem



The problem depot in this specific case was modeled in *realistic.py*

```python
place = UserType('place', None)
hoist = UserType('hoist', None)
surface = UserType('surface', None)
depot = UserType('depot', place)
distributor = UserType('distributor', place)
...
clear_h = Fluent('clear_h', None, [hoist])
clear_s = Fluent('clear_s', None, [surface])
at = Fluent('at', None, [truck, place])
...
truck0 = Object('truck0', truck)
truck1 = Object('truck1', truck)
drive = InstantaneousAction('Drive', t=truck, x=place, y=place)
t = drive.parameter('t')
x = drive.parameter('x')
y = drive.parameter('y')
drive.add_precondition(at(t, x))
drive.add_effect(at(t, y), True)
load = InstantaneousAction('Load', t=truck, p=place, c=crate, h=hoist)
...
unload = InstantaneousAction('Unload', t=truck, p=place, c=crate, h=hoist)
...
problem = Problem('depot_truck')
depot0 = Object('depot0', depot)
distributor0 = Object('distributor0', distributor)
distributor1 = Object('distributor1', distributor)
...
problem.add_object(distributor0)
problem.add_object(distributor1)
...
problem.add_action(drive)
problem.add_action(load)
problem.add_action(unload)
problem.add_fluent(clear_s, default_initial_value=False)
problem.add_fluent(clear_h, default_initial_value=False)
...
problem.set_initial_value(at(truck0, depot0), False)
problem.set_initial_value(at(truck0, distributor0), False)
...
problem.add_goal(on_s(crate0, pallet2))
problem.add_goal(on_s(crate1, pallet1))
depot_truck = Example(problem=problem)
problems['depot_truck'] = depot_truck
```

Multi-agent problem

```python
problem = examples_realistic['depot_truck'].problem
fluents_problem = problem.fluents()
actions_problem = problem.actions()
init_values_problem = problem.initial_values()
goals_problem = problem.goals()
objects_problem = problem.all_objects()
robot1 = Agent('depot0')
robot2 = Agent('distributor0')
robot3 = Agent('distributor1')
robot4 = Agent('truck0')
robot5 = Agent('truck1')
environment = Environment_ma()
robot1.add_fluents(fluents_problem)
robot2.add_fluents(fluents_problem)
...
robot1.add_actions(actions_problem)
robot2.add_actions(actions_problem)
...
robot1.set_initial_values(init_values_problem)
robot2.set_initial_values(init_values_problem)
...
robot1.add_goals(goals_problem)
robot2.add_goals(goals_problem)
...
ma_problem = MultiAgentProblem('depot_trucks')
ma_problem.add_agent(robot1)
ma_problem.add_agent(robot2)
...
ma_problem.add_environment_(environment)
ma_problem.add_objects(objects_problem)
#Add shared data
problem = ma_problem.compile_ma()
ma_problem.add_shared_data(ma_problem.fluent('clear_h'))
ma_problem.add_shared_data(ma_problem.fluent('clear_s'))
...
#Add fucntions
ma_problem.add_flu_function(ma_problem.fluent('located'))
ma_problem.add_flu_function(ma_problem.fluent('at'))
...
robots = Example(problem=problem)
problems['depot_trucks'] = robots
```

I import the depot_truck problem from realitic.py

Agents

For each agent add:
actions,
initial values,
goals,

I add to the problem:
agent,
environment,
objects

Shared data

Functions

```
(define (domain pddl-domain)
(:requirements :strips :typing :negative-preconditions)
(:types surface agent place hoist - object
        depot distributor - (either place agent)
        truck - agent
        crate pallet - surface
)
(:predicates
  (myAgent ?a - truck)
  (clear ?x - (either surface hoist)))
(:functions
  (located ?h - hoist) - place
  (at ?t - truck) - place
  (placed ?p - pallet) - place
  (pos ?c - crate) - (either place truck)
  (on ?c - crate) - (either hoist truck surface)
)
(:action Drive
 :parameters ( ?t - truck ?x - place ?y - place)
 :precondition (and (myAgent ?t) (= (at ?t) ?x))
 :effect (and (assign (at ?t) ?y)))
(:action Load
 :parameters ( ?t - truck ?p - place ?c - crate ?h
 :precondition (and (myAgent ?t) (= (at ?t) ?p) (=
(not (clear ?c)) (not (clear ?h)) (= (on ?c) ?h) (=
h) ?p))
 :effect (and (clear ?h) (clear ?c) (assign (pos ?c
(on ?c) ?t)))
(:action Unload
 :parameters ( ?t - truck ?p - place ?c - crate ?h
 :precondition (and (myAgent ?t) (= (located ?h) ?p
p) (= (pos ?c) ?t) (= (on ?c) ?t) (clear ?h) (clear
 :effect (and (assign (pos ?c) ?p) (assign (on ?c)
(clear ?c)) (not (clear ?h))))
)
```

```
(define (problem pddl-problem)
(:domain pddl-domain)
(:objects
 crate0 crate1 - crate
 truck0 truck1 - truck
 depot0 - depot
 distributor0 distributor1 - distributor
 pallet0 pallet1 pallet2 - pallet
 hoist0 hoist1 hoist2 - hoist
)
(:shared-data
  (clear ?x - (either hoist surface))
  ((at ?t - truck) - place)
  ((pos ?c - crate) - (either place truck))
  ((on ?c - crate) - (either hoist truck surface)) -
(either depot0 distributor0 distributor1 truck1)
)
(:init
 (myAgent truck0)
 (= (pos crate0) distributor0)
 (clear crate0)
 (= (on crate0) pallet1)
 (= (pos crate1) depot0)
 (clear crate1)
 (= (on crate1) pallet0)
 (= (at truck0) distributor1)
 (= (at truck1) depot0)
 (= (located hoist0) depot0)
 (clear hoist0)
 (= (located hoist1) distributor0)
 (clear hoist1)
 (= (located hoist2) distributor1)
 (clear hoist2)
 (= (placed pallet0) depot0)
 (not (clear pallet0))
 (= (placed pallet1) distributor0)
 (not (clear pallet1))
 (= (placed pallet2) distributor1)
 (clear pallet2)
)
(:global-goal (and
 (= (on crate0) pallet2)
 (= (on crate1) pallet1)
))
```

Competition of Distributed and Multiagent Planners (CoDMAP)
http://agents.fel.cvut.cz/codmap/

MA-PDDL and FMAP

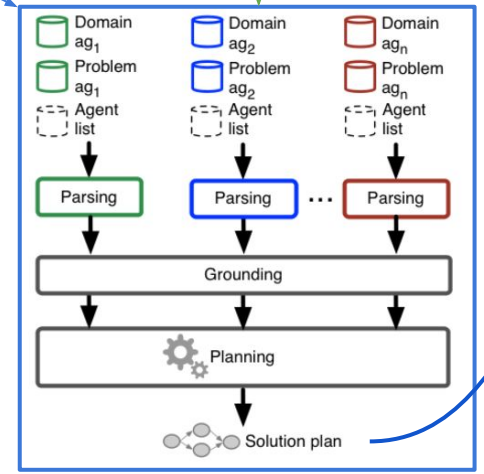
```
problem_depots = problems['depots'].problem
problem.add_agent_list(problem_depots, 'depot0')
problem.add_agent_list(problem_depots, 'distributor0')
problem.add_agent_list(problem_depots, 'distributor1')

problem_trucks = problems['depot_trucks'].problem
problem.add_agent_list(problem_trucks, 'truck0')
problem.add_agent_list(problem_trucks, 'truck1')

problems_ma = [problem_depots, problem_trucks]
problem.write_ma_problem(problem)
plan = problem.FMAP_planner()
```

Problems and domains in ma_pddl

- ProblemDepotsDepot0.pddl
- ProblemDepotsDistributor0.pddl
- ProblemDepotsDistributor1.pddl
- ProblemDepotsTruck0.pddl
- ProblemDepotsTruck1.pddl
- DomainDepot_trucks.pddl
- DomainDepots.pddl

Input FMAP

FMAP



FMAP solution plan

```
; Hdtg = 0, Hlan = 0
; Hdtg = 12, Hlan = 12
; Hdtg = 12, Hlan = 12
; Hdtg = 31, Hlan = 8
; Hdtg = 10, Hlan = 8
; Hdtg = 20, Hlan = 5
; Hdtg = 6, Hlan = 5
; Hdtg = 7, Hlan = 4
; Hdtg = 6, Hlan = 5
; Hdtg = 18, Hlan = 5
; Hdtg = 5, Hlan = 5
; Hdtg = 4, Hlan = 4
; Hdtg = 3, Hlan = 4
; Hdtg = 2, Hlan = 2
; Hdtg = 1, Hlan = 1
; Hdtg = 0, Hlan = 0
```

```
; Solution plan - CoDMAP Distributed format
; ----------------------------------------
0: (LiftP hoist0 crate1 pallet0 depot0)
0: (LiftP hoist1 crate0 pallet1 distributor0)
6: (DropP hoist1 crate1 pallet1 distributor0)
8: (DropP hoist2 crate0 pallet2 distributor1)
1: (Load truck1 depot0 crate1 hoist0)
2: (Drive truck1 depot0 distributor1)
3: (Drive truck1 distributor1 distributor0)
4: (Load truck1 distributor0 crate0 hoist1)
5: (Unload truck1 distributor0 crate1 hoist1)
6: (Drive truck1 distributor0 distributor1)
7: (Unload truck1 distributor1 crate0 hoist2)
```

Multi-agent plan generation

ProblemDepotsTruck0.pddl

```
(define (problem pddl-problem)
(:domain pddl-domain)
(:objects
 crate0 crate1 - crate
 truck0 truck1 - truck
 depot0 - depot
 distributor0 distributor1 - distributor
 pallet0 pallet1 pallet2 - pallet
 hoist0 hoist1 hoist2 - hoist
)
(:shared-data
  (clear ?x - (either hoist surface))
  ((at ?t - truck) - place)
  ((pos ?c - crate) - (either place truck))
  ((on ?c - crate) - (either hoist truck surface)) -
(either depot0 distributor0 distributor1 truck1)
)
(:init
 (myAgent truck0)
 (= (pos crate0) distributor0)
 (clear crate0)
 (= (on crate0) pallet1)
 (= (pos crate1) depot0)
 (clear crate1)
 (= (on crate1) pallet0)
 (= (at truck0) distributor1)
 (= (at truck1) depot0)
 (= (located hoist0) depot0)
 (clear hoist0)
 (= (located hoist1) distributor0)
 (clear hoist1)
 (= (located hoist2) distributor1)
 (clear hoist2)
 (= (placed pallet0) depot0)
 (not (clear pallet0))
 (= (placed pallet1) distributor0)
 (not (clear pallet1))
 (= (placed pallet2) distributor1)
 (clear pallet2)
)
(:global-goal (and
 (= (on crate0) pallet2)
 (= (on crate1) pallet1)
))
)
```

- ProblemDepotsDepot0.pddl
- ProblemDepotsDistributor0.pddl
- ProblemDepotsDistributor1.pddl
- ProblemDepotsTruck0.pddl
- ProblemDepotsTruck1.pddl

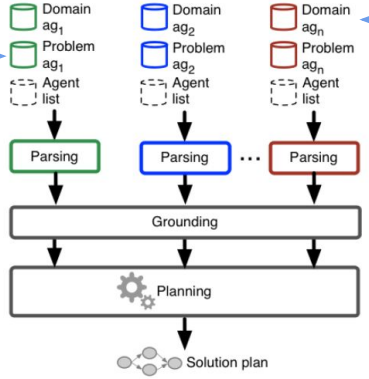
PDDLWriter_MA

- DomainDepot_trucks.pddl
- DomainDepots.pddl

DomainDepots.pddl

```
(define (domain pddl-domain)
(:requirements :strips :typing :negative-preconditions)
(:types surface place hoist agent - object
        truck - agent
        depot distributor - (either place agent)
        crate pallet - surface
)
(:predicates
  (myAgent ?a - place)
  (clear ?x - (either hoist surface)))
(:functions
  (located ?h - hoist) - place
  (at ?t - truck) - place
  (placed ?p - pallet) - place
  (pos ?c - crate) - (either place truck)
  (on ?c - crate) - (either hoist truck surface)
)
(:action LiftP
 :parameters ( ?h - hoist ?c - crate ?z - pallet ?p - place)
 :precondition (and (myAgent ?p) (= (located ?h) ?p) (= (placed ?
  z) ?p) (clear ?h) (= (pos ?c) ?p) (= (on ?c) ?z) (clear ?c))
 :effect (and (assign (on ?c) ?h) (not (clear ?c)) (not (clear ?
  h)) (clear ?z)))
(:action LiftC
 :parameters ( ?h - hoist ?c - crate ?z - crate ?p - place)
 :precondition (and (myAgent ?p) (= (located ?h) ?p) (= (pos ?
  z) ?p) (clear ?h) (= (pos ?c) ?p) (= (on ?c) ?z) (clear ?c))
 :effect (and (assign (on ?c) ?h) (not (clear ?c)) (not (clear ?
  h)) (clear ?z)))
(:action DropP
 :parameters ( ?h - hoist ?c - crate ?z - pallet ?p - place)
 :precondition (and (myAgent ?p) (= (located ?h) ?p) (= (placed ?
  z) ?p) (clear ?z) (= (on ?c) ?h) (not (clear ?c)) (not (clear ?
  h)))
 :effect (and (clear ?h) (clear ?c) (not (clear ?z)) (assign
  (on ?c) ?z)))
(:action DropC
 :parameters ( ?h - hoist ?c - crate ?z - crate ?p - place)
 :precondition (and (myAgent ?p) (= (located ?h) ?p) (= (pos ?
  z) ?p) (clear ?z) (= (on ?c) ?h) (not (clear ?c)) (not (clear ?
  h)))
 :effect (and (clear ?h) (clear ?c) (not (clear ?z)) (assign
  (on ?c) ?z)))
)
```

FMAP




```
; Hdtg = 0, Hlan = 0
; Hdtg = 12, Hlan = 12
; Hdtg = 12, Hlan = 12
; Hdtg = 31, Hlan = 8
; Hdtg = 10, Hlan = 8
; Hdtg = 20, Hlan = 5
; Hdtg = 6, Hlan = 5
; Hdtg = 7, Hlan = 4
; Hdtg = 6, Hlan = 5
; Hdtg = 18, Hlan = 5
; Hdtg = 5, Hlan = 5
; Hdtg = 4, Hlan = 4
; Hdtg = 3, Hlan = 4
; Hdtg = 2, Hlan = 2
; Hdtg = 1, Hlan = 1
; Hdtg = 0, Hlan = 0
```

```
; Solution plan - CoDMAP Distributed format
; ----------------------------------------
0: (LiftP hoist0 crate1 pallet0 depot0)
0: (LiftP hoist1 crate0 pallet1 distributor0)
6: (DropP hoist1 crate1 pallet1 distributor0)
8: (DropP hoist2 crate0 pallet2 distributor1)
1: (Load truck1 depot0 crate1 hoist0)
2: (Drive truck1 depot0 distributor1)
3: (Drive truck1 distributor1 distributor0)
4: (Load truck1 distributor0 crate0 hoist1)
5: (Unload truck1 distributor0 crate1 hoist1)
6: (Drive truck1 distributor0 distributor1)
7: (Unload truck1 distributor1 crate0 hoist2)
```

Competition of Distributed and Multiagent Planners (CoDMAP)
http://agents.fel.cvut.cz/codmap/

Thank you for your attention