Planning Story Skeletons

Daniele S. Cardullo - 2127806
cardullo.2127806@studenti.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

Applying Planning to NLP

Planning and reasoning approaches can be used to enhance the text comprehension and generation of structured stories and dialogues, their applications spanning a wide range of domains, from robotics, to reasoning LLMs.

Planning allows for generating a structure to a conversation or a story, thus allowing to retain logic flow.

Reasoning allows for understanding implicit features and conversation nuances given a knowledge base.

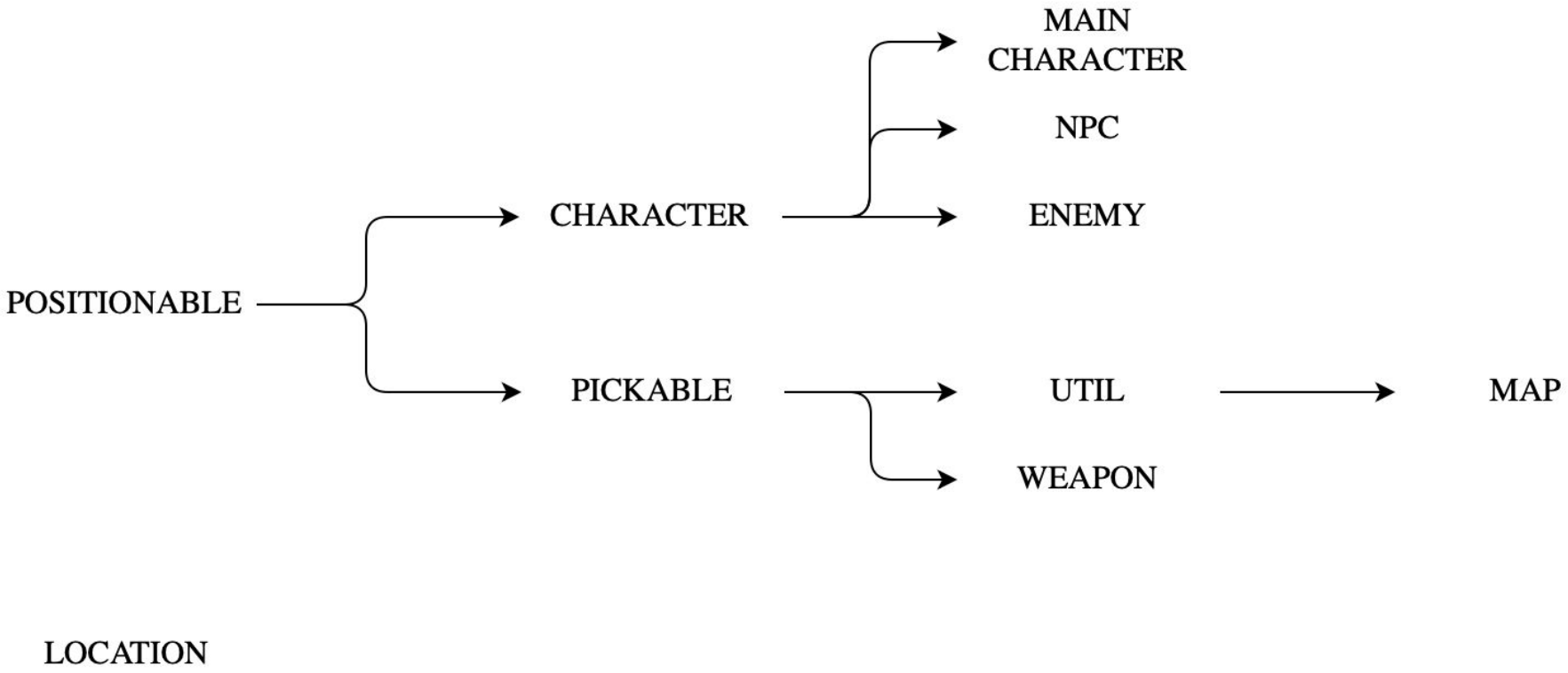
Applying Planning to NLP (This Work)

In this work a prototype of application of P&R to NLP is developed: **generator for fantasy stories' structures** (PDDL Task), **reasoning over story elements and a controller for interactive stories** (Indigolog Task).

In the PDDL task the domain is defined by specifying the story characters, places, weapons and treasures; the different problems specify increasing order of difficulty to generate structures for more intriguing and sophisticated stories.

In Indigolog the domain is modeled and a story is defined as a procedure, the reasoning tasks are then performed

PDDL Domain (Types hierarchy)



PDDL Domain (Predicates)

at ?x - positionable ?where - location whether ?x is in location ?where

reachable ?x1 ?x2 - location whether the main character know the path between ?x1 and ?x2

has ?x - character ?y - pickable whether character ?x has the pickable ?y

know ?x - character ?what - positionable ?where - location whether character ?x know the position ?where of the positionable ?what

PDDL Domain (Predicates ctd.)

knowInExchange ?x - character ?y - pickable ?what - positionable ?where - location whether character ?x know the position ?where of the positionable ?what and dispatch it in exchange of the pickable ?y

atHome ?x - location whether the location ?x is considered the house of the main character

PDDL Domain (Actions)

move ?x - main_character ?from ?to - location:

pre: reachable ?from ?to ∧ at ?x ?from

eff: at ?x ?to ∧ ¬at ?x ?from

main character moves from ?from to location ?to if they know the path and start at ?from. After performing the action the main character is at ?to and no more in ?from.

Cost 3 if going home else 5

pick ?x - main_character ?what - pickable ?where - location:

pre: at ?what ?where ∧ at ?x ?where

∧ ¬ ∃ ?e - enemy . at ?e ?where

eff: has ?x ?what ∧ ¬at ?what ?where

main character picks ?what at ?where if both are at ?where and no enemy around. The effect is that the main character now has ?what

PDDL Domain (Actions ctd.)

fight ?x - main_character ?where - location ?weapon - weapon:

pre: at ?x ?where ∧ has ?x ?weapon
∧ ¬∃ ?e - enemy . at ?e ?where

eff: ∀ ?e - enemy (at ?e ?where ▷ ¬at ?e ?where)

if there are enemies in the location where ?x is, and ?x has ?weapon then all enemies at ?e are eliminated

read ?x - main_character ?m - map
?from ?to - location:

pre: has ?x ?m ∧ at ?x ?from
∧ (∃ ?z - positionable . know ?x ?z ?to ∨ atHome ?to)
∧ ¬∃ ?e - enemy . at ?e ?from

eff: has ?x ?what ∧ ¬at ?what ?where

the main character reads the map they have to discover the path between two locations if they know there is something of interest there or that place is their home. Can be performed if no enemy around Cost 1.

PDDL Domain (Actions ctd.)

talkTo ?x - main_character ?y - npc ?where - location
?what - positionable ?place - location:

pre: at ?x ?where ∧ at ?y ?where ∧ know ?y ?what ?place
∧ ¬∃ ?e - enemy . at ?e ?where

eff: know ?x ?what ?place ∧ at ?what ?place

if in the same place with a npc and no enemy around, the main character can talk to the npc to get information about an object in a place

talkToInExchange ?x - main_character ?y - npc
?z - pickable ?where - location
?what - positionable ?place - location:

pre: at ?x ?where ∧ at ?y ?where ∧ has ?y ?z
∧ knowInExchange ?x ?y ?what ?place
∧ ¬∃ ?e - enemy . at ?e ?where

eff: know ?x ?what ?place ∧ at ?what ?place

if in the same place with a npc and no enemy around, the main character can talk to the npc to get information about an object in a place in exchange for an object

PDDL Domain (Actions ctd.)

give ?x - character ?y - character ?where - location
?what - pickable:

pre: at ?x ?where ∧ at ?y ?where ∧ has ?x ?what
∧ ¬∃ (?smth - positionable ?smwh - location)
. knowInExchange ?x ?what ?smth ?smwh

eff: has ?y ?what ∧ ¬has ?x ?what

if a character ?x has an object ?what and another character ?y both at ?where and ?x does not know in exchange for ?what anything, then ?x can give ?what to ?y

PDDL Problem - The Boring Story

```
(:objects
    main_c - main_character
    map_1 - map
    treasure - pickable
    village castle cave - location
    sword - weapon
    E1 E2 E3 E4 - enemy
)

(:goal (and
    (has main_c treasure)
    (at main_c village)
))
```

```
(:init
    (atHome village)
    (at main_c village)
    (has main_c map_1)
    (at treasure cave)
    (at E1 cave)
    (at E2 cave)
    (at E3 cave)
    (at E4 cave)
    (at sword castle)
    (know main_c sword castle)
    (know main_c treasure cave)
)
```

PDDL Problem - The Boring Story (Results)

		Time (s)	Expanded Nodes	Memory (kb)	Solution Length	Solution Cost
FD A*	BLIND	0.067041	125	214’540	9	—-
	h-max	0.071277	25	214’580	9	—-
	h-add	0.072258	11	214’584	10	—-
ENHSP	h-max	0.075	66	—-	11	36
	h-add	0.043	15	—-	11	29

PDDL Problem - The Princess Tale

```
(:objects
      main_c - main_character
      village castle ruin cavern - location
      princess knight - npc
      enemy_1 enemy_2 - enemy
      sword - weapon
      treasure crown - pickable
      map_1 - map
)


(:goal (and
      (at main_c village)
      (has main_c treasure)
))
```

```
(:init
      (atHome village)
      (has main_c map_1)
      (at main_c village)
      (at princess castle)
      (know main_c princess castle)
      (know princess knight ruin)
      (knowInExchange princess crown
            treasure castle)
      (know knight crown cavern)
      (has knight sword)
      (at enemy_1 cavern)
      (at enemy_2 cavern)
)
```

PDDL Problem - The Princess Tale (Results)

		Time (s)	Expanded Nodes	Memory (kb)	Solution Length	Solution Cost
FD A*	BLIND	0.088481	7’996	214’932	18	—-
	h-max	0.077550	324	214’580	18	—-
	h-add	0.083778	1’133	214’584	18	—-
ENHSP	h-max	0.138	637	—-	33	82
	h-add	0.075	81	—-	27	73

PDDL Problem - The Predestined Tale

```
(:objects
    main_c - main_character
    friend mysterious_man
        monk knight princess - npc
    enemy_1 enemy_2 enemy_3
        evil_wizard thief - enemy
    village tavern castle monastery
        ruins thief_house
        evil_wizard_house - location
    map_1 - map
    old_book crown treasure - pickable
    dagger sword - weapon
)


(:goal (and
    (at main_c village)
    (has main_c treasure)
))
```

```
(:init
    (atHome village)
    (at main_c village)
    (at friend tavern)
    (know main_c friend tavern)
    (reachable village tavern)
    (has mysterious_man map_1)
    (know friend mysterious_man tavern)
    (know mysterious_man monk monastery)
    (knowInExchange monk old_book treasure
        evil_wizard_house)
    (has monk dagger)
    (know monk knight castle)
    (knowInExchange knight dagger sword castle)
    (know knight old_book ruins)
    (at enemy_1 ruins)
    (at enemy_2 ruins)
    (at old_book ruins)
    (at evil_wizard evil_wizard_house)
    (at treasure evil_wizard_house)
)
```

PDDL Problem - The Predestined Tale (Results)

		Time (s)	Expanded Nodes	Memory (kb)	Solution Length	Solution Cost
FD A*	BLIND	17.48	3’312’117	500’332	25	—-
	h-max	0.097	1’430	214’872	25	—-
	h-add	0.97	42’175	221’780	25	—-
ENHSP	h-max	1.373	10’190	—-	48	133
	h-add	0.115	68	—-	33	84

Indigolog Implementation

Another important step is applying reasoning to the story generation and understanding, this is done by using an indigolog implementation.

The proposed reasoning tasks are:

- projection task
- legality task
- a controller that reacts to exogenous actions

The domain has been simplified by removing the map reading mechanic.

Indigolog Domain (Predicates and Fluents)

The domain is analogous to the one specified in PDDL:

PREDICATES:

character(c)
location(l)
object(o)
weapon(w)
treasure_item(t)
enemy(e)

FLUENTS:

at(c, s)
has(c, o, s)
know(c, o, l, s)
reachable(c, l, s)
knowInExchange(c, o, e, s)
alive(c, s)
atHome(c, l, s)
enemyPresent(l, s)
someChanges(s)

Indigolog Domain (Actions)

move(char, from, to)

```
poss(move(Char, From, To), (
      at(Char, From),
      reachable(Char, To),
      \+ (=(From, To)),
      \+ enemyPresent(From)
))
```

pick(char, object)

```
poss(pick(Char, Object), (
  at(Char, Location),
  at(Object, Location),
  \+ has(_, Object),
  \+ enemyPresent(Location)
))
```

fight(char, enemy)

```
poss(fight(Char, Enemy), (
  at(Char, Location),
  at(Enemy, Location),
  alive(Enemy),
  has(Char, Weapon),
  weapon(Weapon)
))
```

Indigolog Domain (Actions ctd.)

talkTo(C1, C2)

```
poss(talkTo(C1, C2), (
  at(C1, Location),
  at(C2, Location),
  \+ (=(C1, C2)),
  \+ enemyPresent(Location)
))
```

give(giver, rec, obj)

```
poss(give(Giver, Receiver,
Object), (
  at(Giver, Location),
  at(Receiver, Location),
  has(Giver, Object)
))
```

talkToInExchange(C1, C2, Object)

```
poss(talkToInExchange(C1, C2, Object), (
  at(C1, Location),
  at(C2, Location),
  has(C1, Object),
  knowInExchange(C2, _, Object),
  \+ enemyPresent(Location)
))
```

Indigolog Domain (Exogenous Actions)

```
exog_action(enemy_spawns(Enemy, Location)) :-
enemy(Enemy), location(Location).

exog_action(treasure_appears(Treasure, Location)) :-
treasure_item(Treasure), location(Location).
```

Indigolog Domain (SSA)

```
% causal laws for move action
causes_true(move(Char, From, To), at(Char, To), true).
causes_false(move(Char, From, To), at(Char, From), at(Char,
From)).

% causal laws for pick action
causes_false(pick(Char, Object), at(Object, Location), at(Object,
Location)).
causes_true(pick(Char, Object), has(Char, Object), true).

% causal laws for fight action
causes_false(fight(Char, Enemy), alive(Enemy), alive(Enemy)).
causes_false(fight(Char, Enemy), at(Enemy, Location), at(Enemy,
Location)).
causes_true(fight(Char, Enemy), has(Char, Object),
 (has(Enemy, Object), alive(Enemy))).
```

```
% causal laws for give action
causes_false(give(Giver, Receiver, Object), has(Giver, Object),
has(Giver, Object)).
causes_true(give(Giver, Receiver, Object), has(Receiver, Object), true).

% causal laws for exogenous actions
causes_true(enemy_spawns(Enemy, Location), at(Enemy, Location), true).
causes_true(enemy_spawns(Enemy, Location), alive(Enemy), true).
causes_true(treasure_appears(Treasure, Location), at(Treasure,
Location), true).
```

```
causes_true(talkTo(C1, C2), know(C1, Object, Location), know(C2,
Object, Location)).
causes_true(talkToInExchange(C1, C2, ExchObj), know(C1, Object,
Location),
 (knowInExchange(C2, Object, ExchObj), has(C1, ExchObj))).
```

```
causes_true(enemy_spawns(Enemy, Location),
enemyPresent(Location), true).
causes_false(fight(Char, Enemy), enemyPresent(Location),
 at(Enemy, Location)).

causes_true(enemy_spawns(_, _), some_changes, true).
causes_true(treasure_appears(_, _), some_changes, true).
```

Indigolog Domain (Initial Situation)

Same as the “Boring Story” initial state

```
initially(at(hero, home), true).
initially(at(treasure, cave), true).
initially(at(sword, armory), true).
initially(at(enemy1, cave), true).
initially(atHome(hero, home), true).
initially(know(hero, treasure, cave), true).
initially(know(hero, sword, armory), true).
initially(alive(hero), true).
initially(alive(enemy1), true).
initially(enemyPresent(cave), true).
```

Indigolog Domain (Initial Situation ctd.)

Extension to the “princess tale”

```
initially(at(princess, castle), true).
initially(at(magic_gem, mountain), true).
initially(know(princess, magic_gem, mountain), true).
initially(knowInExchange(princess, magic_gem, golden_ring), true).
initially(alive(princess), true).
```

plus default initializations…

First Reasoning Task - Projection Task

Will there still be enemy in the cave after the hero takes the treasure?

$$s_\rho = [\texttt{goto\_loc(armory)}, \texttt{pick(hero, sword)}, \texttt{goto\_loc(cave)}, \texttt{fight(hero, enemy1)}, \texttt{pick(hero,treasure)}]$$

$$\mathcal{D} \models \texttt{at}(\texttt{enemy1}, \texttt{cave}, s_\rho)$$

and, as expected:

PROGRAM: Program fails:
[[], ?(at(enemy1,cave))]
…at history:
[pick(hero,treasure),
fight(hero,enemy1),move(hero,armory,cave),pick(hero,sword),
move(hero,home,armory)]

Second Reasoning Task - Legality Task

Now we can check the legality of performing a certain defined story, in this case, the hero goes from home to the armory to get a weapon and then moves to the cave.

$$\mathcal{D} \models [\texttt{move}(\texttt{hero}, \texttt{home}, \texttt{armory}), \texttt{pick}(\texttt{hero}, \texttt{sword}), \texttt{move}(\texttt{hero}, \texttt{armory}, \texttt{cave})]$$

and result is:

PROGRAM: Program has executed to completion!!
History done:
[move(hero, armory, cave), pick(hero, sword), move(hero, home, armory)]

Third Reasoning Task - Interactive Controller

To make the story more interactive a controller has been implemented, which leads the main character to go around searching for enemies to fight.

```
proc(control(patroling_controller), patroling).
proc(patroling, [
 if((\+ has(hero, sword)),
   [goto_loc(armory), pick(hero,sword)],
   []),

 while((true), [
   if(some([en], (enemy(en), alive(en))),
     [search(pi([en, loc], [?(enemy(en)), ?(alive(en)), ?(at(en, loc)), goto_loc(loc), fight(hero, en)
     ]))],
     [
       if(at(hero, home), [], [goto_loc(home)])
     ]
   )
 ])
]).
```

Third Reasoning Task - Interactive Controller (execution)

No Exogenous Events:

PROGRAM: Program has executed to completion!!
History done:
[move(hero,cave,home),fight(hero,enemy1),move(hero,armory,cave),pick(hero,sword),move(hero,home,armory)]

Exogenous Events:

PROGRAM: Program has executed to completion!!
History done:
[move(hero,cave,home),fight(hero,orc),move(hero,castle,cave),enemy_spawns(orc,cave),fight(hero,goblin),move(hero,cave,castle),fight(hero,enemy1),move(hero,armory,cave),enemy_spawns(goblin,castle),pick(hero,sword),move(hero,home,armory)]

Thank you for the attention!