Model-Space Reasoning as Search in Feedback Space for Planning-Domain Generation

Presenter: James Oswald (RPI)

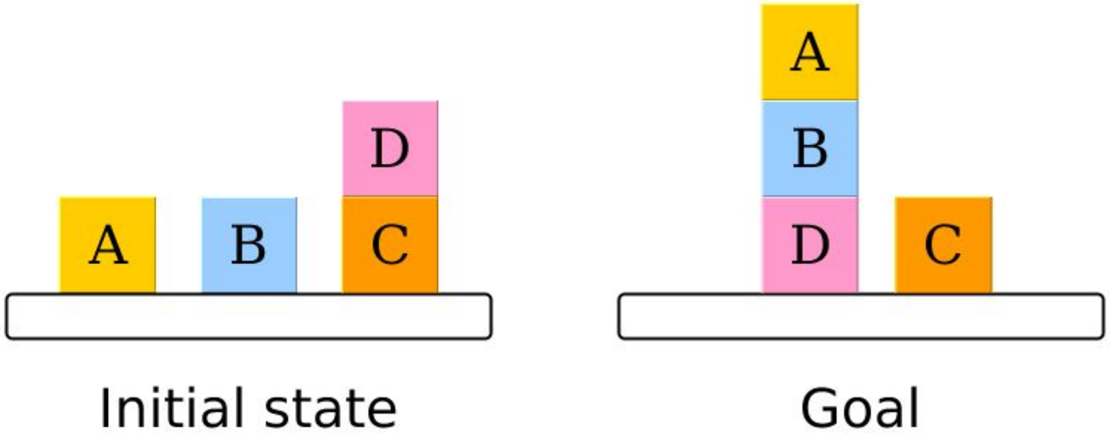
PIs: Kavitha Srinivas (IBM), Harsha Kokel (IBM), Michael Katz (IBM), Shirin Sohrabi (IBM), Selmer Bringsjord (RPI)

Overview

- A Review of Automated Planning.
- How can we approach Planning Domain Construction With LLMs?
- Automatically evaluating Domain Quality.
- Feedback Based Approaches.
- Search Over Feedback Space.

What is (Classical) Planning?

Planning is the process of creating a list of actions to get from some initial state to a goal state.



How Do We Represent Planning Problems?

A lifted planning problem is defined as tuple (F, C, A, s_0, g) we split this into an abstract domain consisting of just the world description and action model:

1) *F*, A set of (lifted) *predicates* that describe the world.
2) *A*, A set of (lifted) *actions*, each of which is composed of:
    a) pre(a), A set of predicates in S required to be true to perform the action.
    b) add(a), A set of predicates in S that are added to the state after the action.
    c) del(a), A set of predicates in S removed from the state after the action.

And a problem, consisting of objects, initial, and goal states:

1) *C*, A set of objects / constants that ground F into a set of propositions F_g.
2) $s_0 \in P(F_g)$, An initial state, composed of grounded predicates from F_g.
3) $g \subseteq P(F_g)$, A set of goal states.

Predicates and State Space

Initial State

(handempty)

(on-table A)

(on-table B)

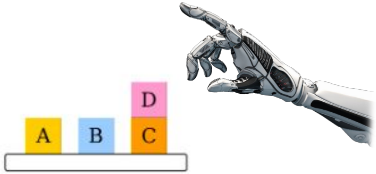
(on-table C)

(clear A)

(clear B)

(clear D)

(on-top-of D C)



Initial state

State after Picking up B

(holding B)

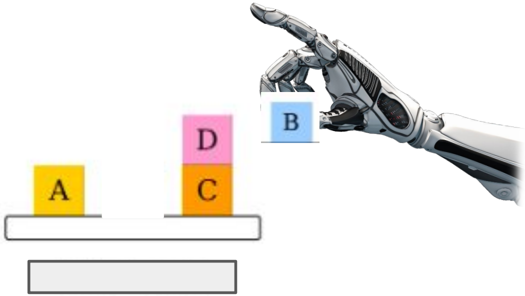
(on-table A)

(on-table C)

(clear A)

(clear D)

(on-top-of D C)



State after Stacking B on A

(handempty)

(on-table A)

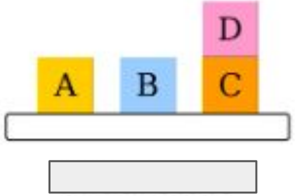
(on-table C)

(clear B)

(clear D)

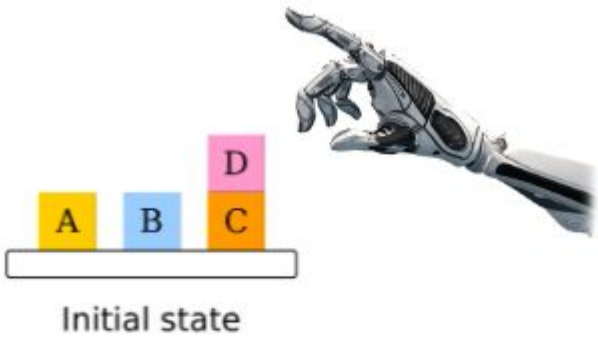
(on-top-of D C)

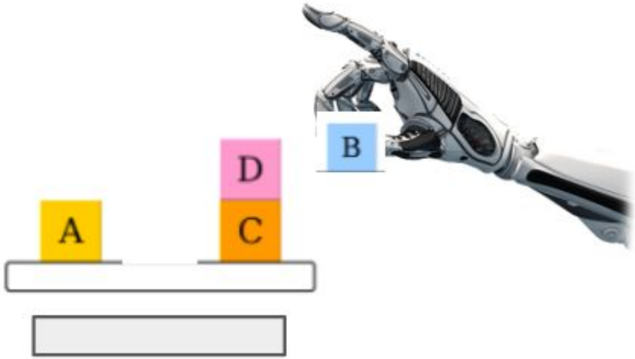
(on-top-of B A)



Actions

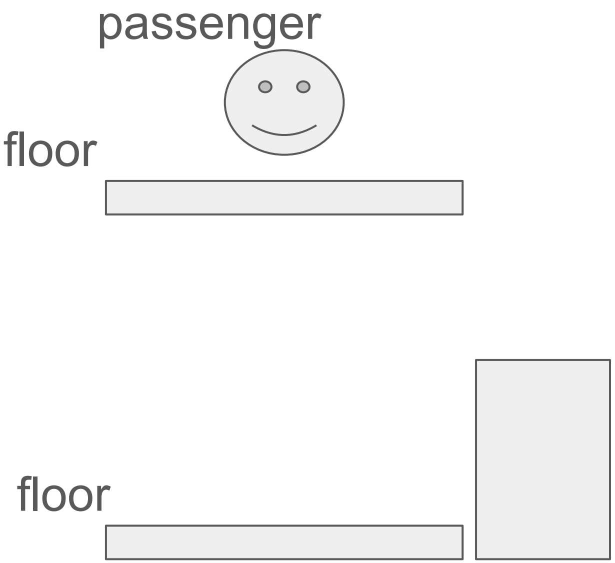
```
(:action pick-up
    :parameters (?x - block)
    :precondition (and
        (clear ?x)
        (ontable ?x)
        (handempty))
    :effect (and
        (not (ontable ?x))
        (not (clear ?x))
        (not (handempty))
        (holding ?x)
    )
)
```





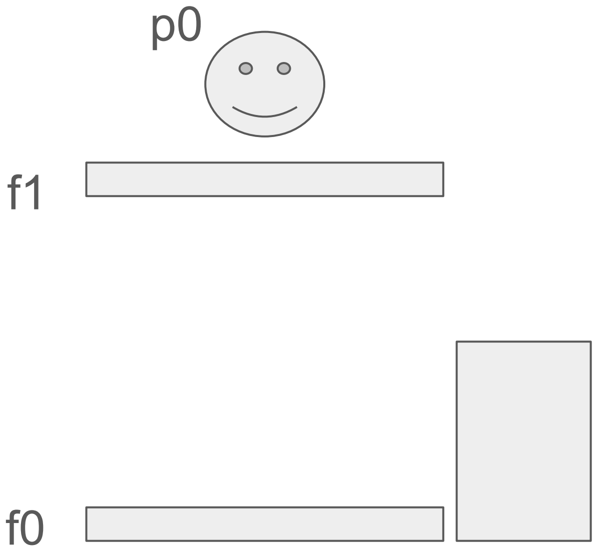
PDDL: Domains, State, Types, and Actions

```
(define (domain miconic)
    (:requirements :typing :strips)
    (:types passenger floor)

    (:predicates
        (origin ?person - passenger ?floor - floor)
        (destin ?person - passenger ?floor - floor)
        (above ?floor1 - floor  ?floor2 - floor)
        (boarded ?person - passenger)
        (not-boarded ?person - passenger)
        (served ?person - passenger)
        (not-served ?person - passenger)
        (lift-at ?floor - floor))

    (:action board
      :parameters (?f - floor ?p - passenger)
      :precondition (and (lift-at ?f) (origin ?p ?f))
      :effect (boarded ?p))

    (:action depart
      :parameters (?f - floor ?p - passenger)
      :precondition (and (lift-at ?f) (destin ?p ?f) (boarded ?p))
      :effect (and (not (boarded ?p))
                   (served ?p)))

    (:action up
      :parameters (?f1 - floor ?f2 - floor)
      :precondition (and (lift-at ?f1) (above ?f1 ?f2))
      :effect (and (lift-at ?f2) (not (lift-at ?f1))))

    (:action down
      :parameters (?f1 - floor ?f2 - floor)
      :precondition (and (lift-at ?f1) (above ?f2 ?f1))
      :effect (and (lift-at ?f2) (not (lift-at ?f1))))
)
```



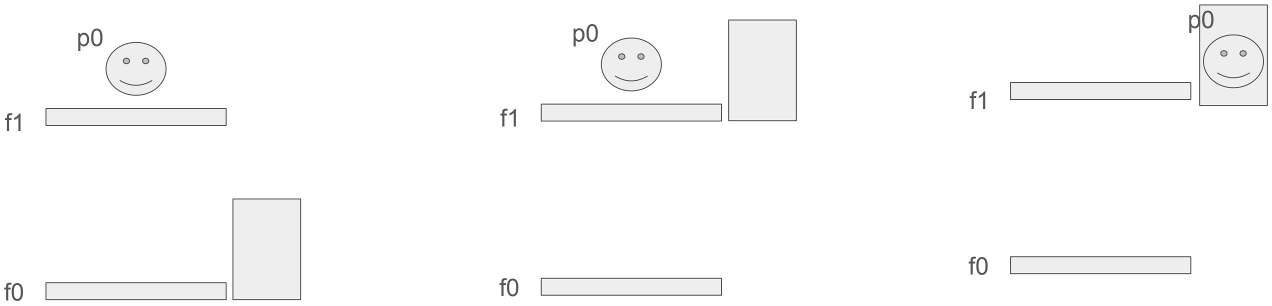
PDDL Problems: Initial State and Goal State

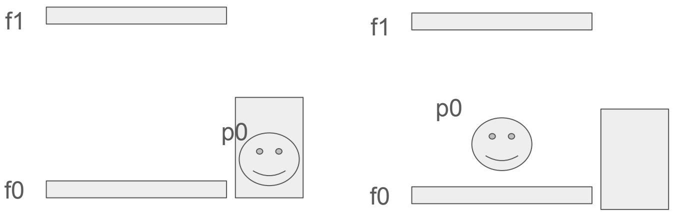
```
(define (problem p1)
    (:domain miconic)
    (:objects p0 - passenger
              f0 f1 - floor)

    (:init
        (above f0 f1)
        (origin p0 f1)
        (destin p0 f0)
        (lift-at f0)
    )


    (:goal (and
        (served p0)
    ))
)
```



Plans

Plans are sequences of grounded actions. A valid plan for a problem is one that will reach the goal while respecting all preconditions.



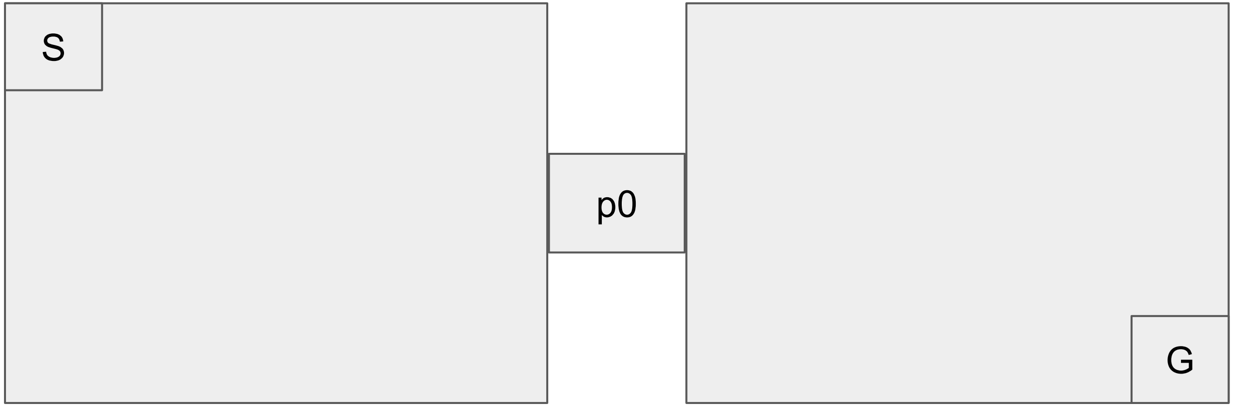


(up f0 f1)
(board f1 p0)
(down f1 f0)
(depart f0 p0)

State Landmarks

State Landmarks are (logical formula over) grounded predicates that must hold at some point along any valid plan for the problem.

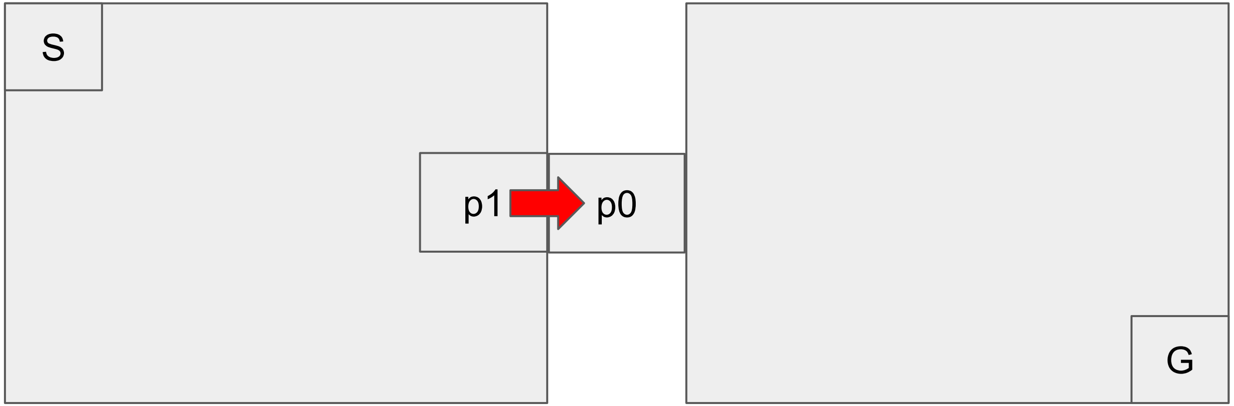
Imagine the following grid navigation problem:



On all valid plans we will at some point have (at p0), thus it is a state landmark.

Action Landmarks

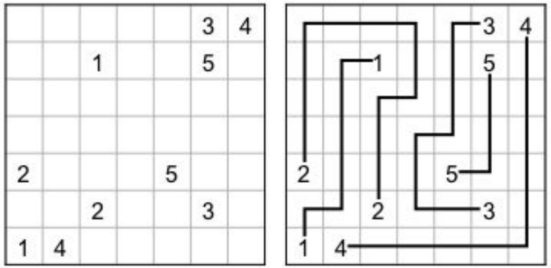
Disjunctive action landmarks are sets of actions at least one of which must appear on any valid plan for the problem.



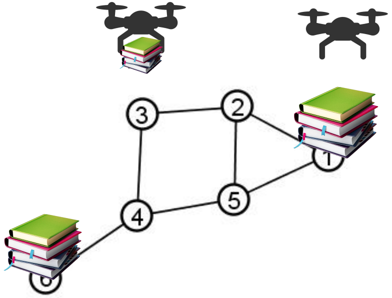
{(move-east p1 p0)} is an action landmark,
because it will hold on all valid plans.

Our Domains

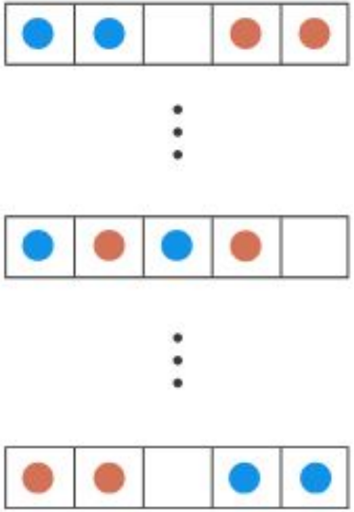
Numberlink / Flow



Book Seller



Checkers Jumping



Bloxorz

LLMs as Domain Generators

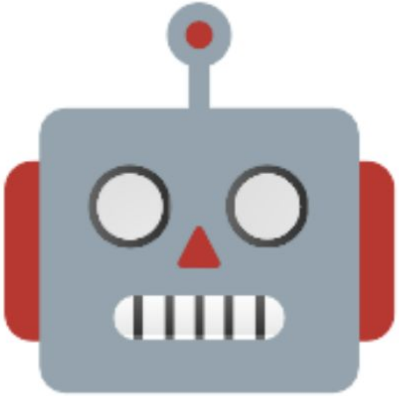
Somewhere in the Wild

"Hello SomeGPT, I'm Daniel, can you provide me with a PDDL domain for the following scenario... ?"

"Sure, Daniel! Here is a PDDL domain for your scenario:

```
(define (domain scenario)
    ...
)
```

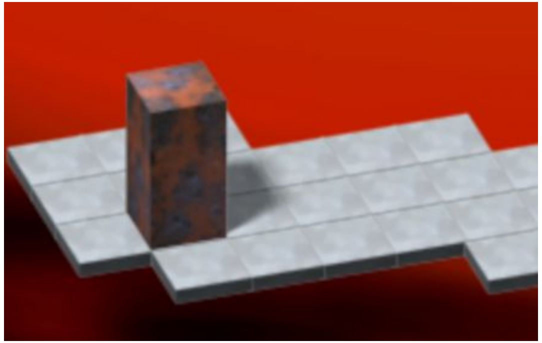
Good luck!"



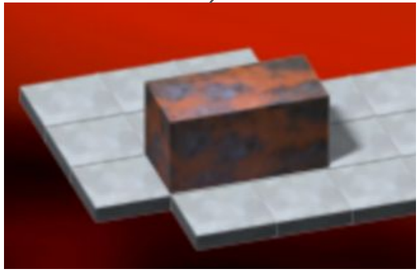
Does This Work?

Can You give me STRIPS style PDDL domain for the game Bloxorz, where the goal is to maneuver a 1x1x2 rectangular prism on a grid of tiles to be standing on a particular tile? Your actions are: stand-up, lay-down, and roll.

```
;; 3. Lay down from standing horizontally
(:action lay-down-h
  :parameters (?b - block ?t - tile ?t2 - tile)
  :precondition (and
    (standing ?b ?t)
    (adjacent ?t ?t2)
  )
  :effect (and
    (not (standing ?b ?t))
    (lying-h ?b ?t ?t2)
  )
)
```

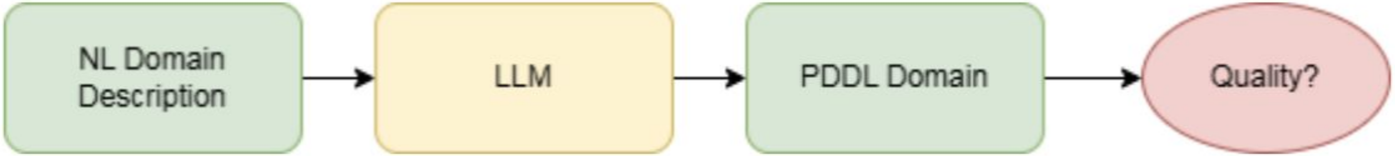


Incorrect

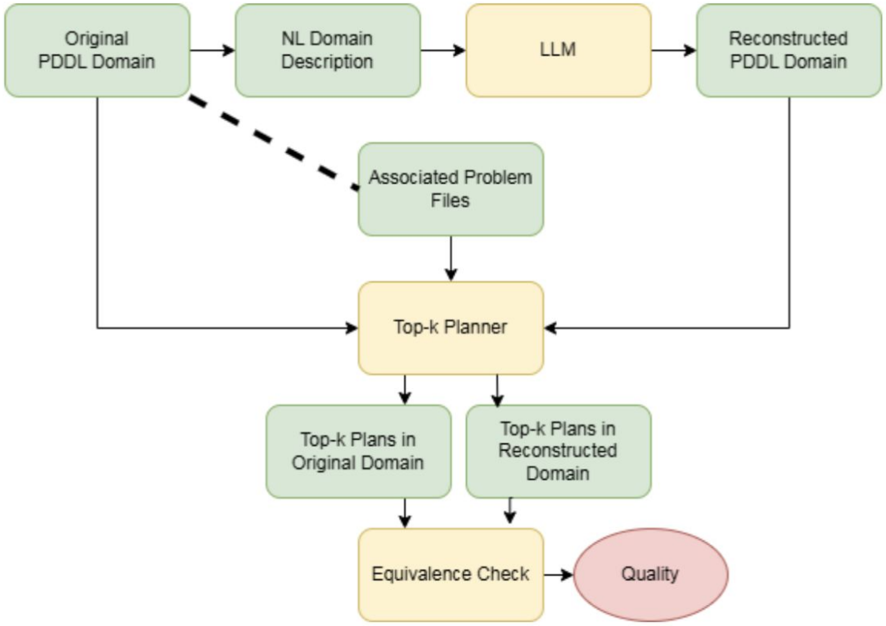


Correct

How can we Evaluate Domain Construction Quality?



Heuristic Domain Equivalence For Domain Quality



Strategy from, *Large Language Models as Planning Domain Generators,* Oswald et al. Published at ICAPS 2024.

Can We Do Better?

Ask on an action by action basis, allow the model to build up what predicates it needs.

“Hi SomeGPT, Im trying to get PDDL for blocksworld… can you give me a PDDL action for picking a block up?”

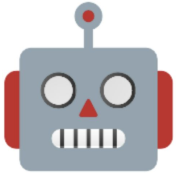
(:action pickup (:params ?x - block) …

“Hi SomeGPT, Im trying to get PDDL for blocksworld… can you give me a PDDL action for putting a block down?”

(:action putdown (:params ?x - block) …

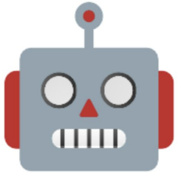
Can we do Better? Simple Syntax & Semantic Feedback

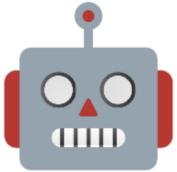
Once we have an action we can check its syntactic correctness trivially. We can do the same once we have our final domain.



"(domain (blocks) (:action (and (or not))))"

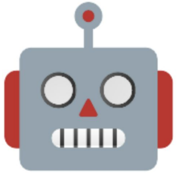
"This is syntactically invalid expected ':parameters', try again"



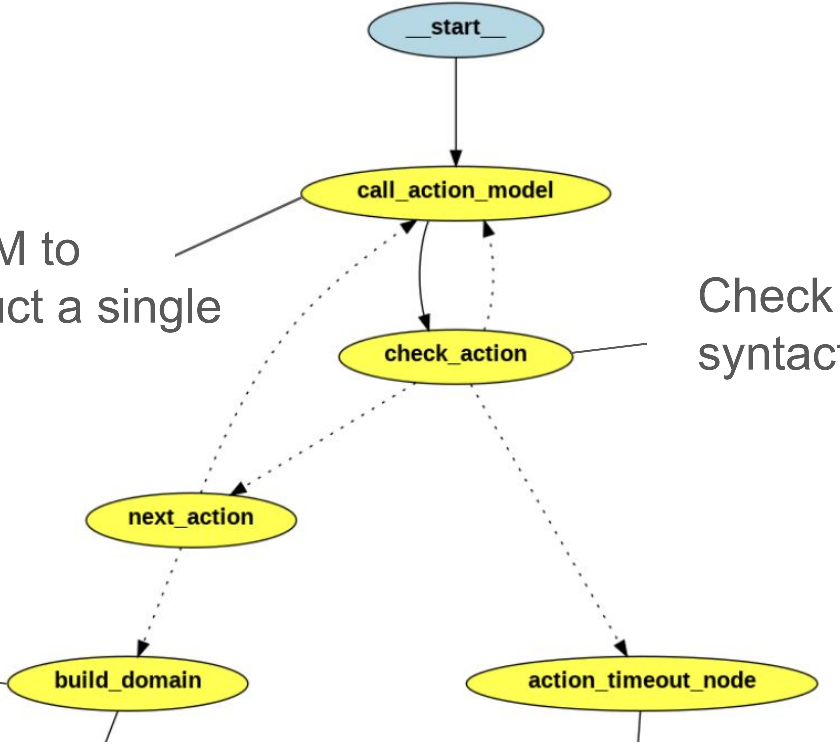


… "(:action stack (:parameters ?x - block) (:precondition (and (not (hand-empty)))"

"This is invalid, expected a pure STRIPS domain but action has a negated precondition"



Domain Construction Workflow

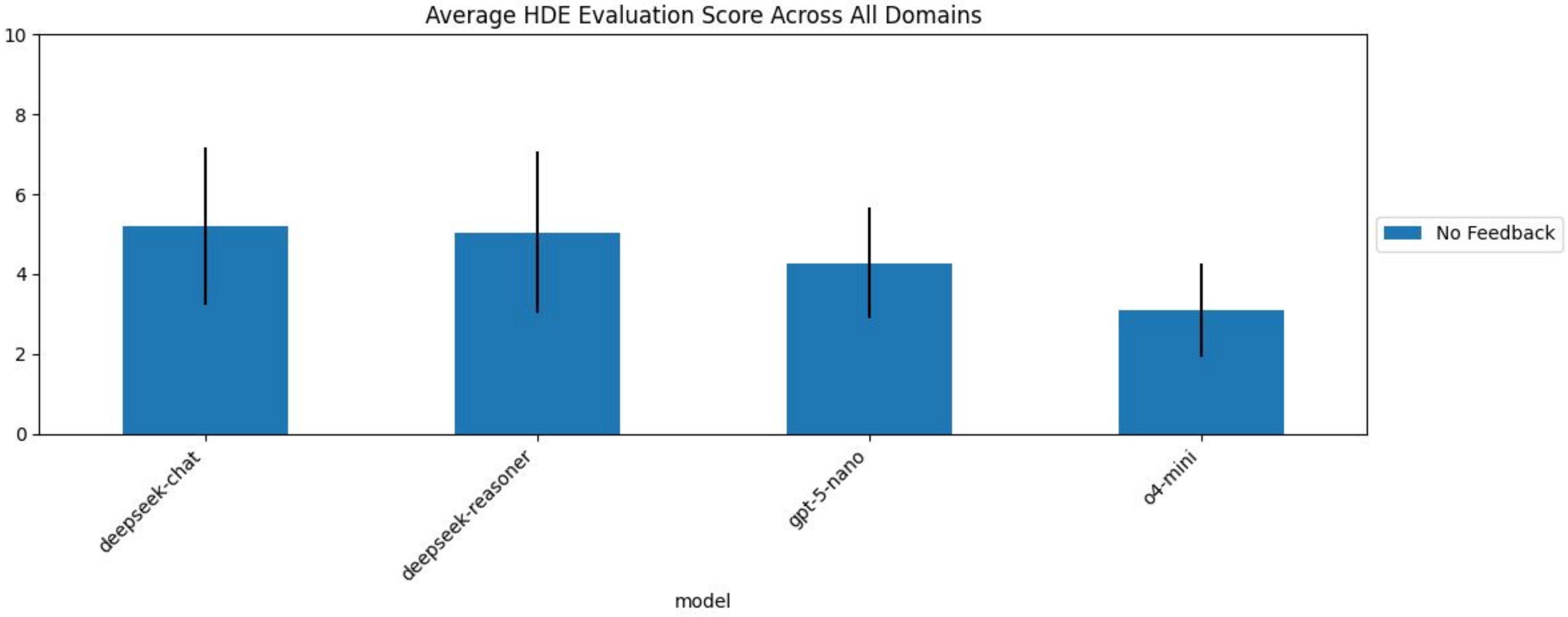


Call LLM to Construct a single action.

Check if actions are syntactically valid

Combine all actions into a final domain.

Few Shot Domain Construction



More feedback? Our Approach

Once we have a syntactically valid domain, how can we improve it further?

More feedback?

Once we have a syntactically valid domain, how can we improve it further?

Unless we have more info about what the user wants we are now essentially stuck.

However, users will often have an idea about what they want, if they specify problems and associated landmarks or plans, we can provide better feedback.

Assumptions In Our Approach

We assume:

- A structured natural language description, describing all actions required, and lists predicates used in initial states.
- A set of sample problems and associated plans or landmarks (For Feedback)
- A set of ground truth domains for evaluation (that are never seen by the model)

Is this Reasonable?

When specifying domains, users often have an idea about not just the action model, but also small problem instances.

Providing toy example(s) for a domain and some valid solution(s) or even JUST landmark(s) that appear on solutions isn't that much extra work, particularly if it helps improve correctness.

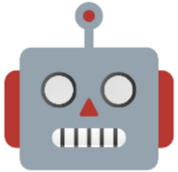
Plan Feedback

“(domain (blocks) (:action stack (?x) …”

The domain was supposed to work with the following problem and plan (...) but on the domain you provided, trying to execute action (pickup …) violated a precondition, check the effects of previous actions or precondition of the current action for issues.

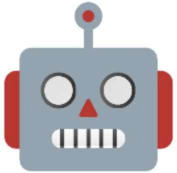
Feedbacks on action execution failure and goal satisfaction failure.

Landmark Feedback



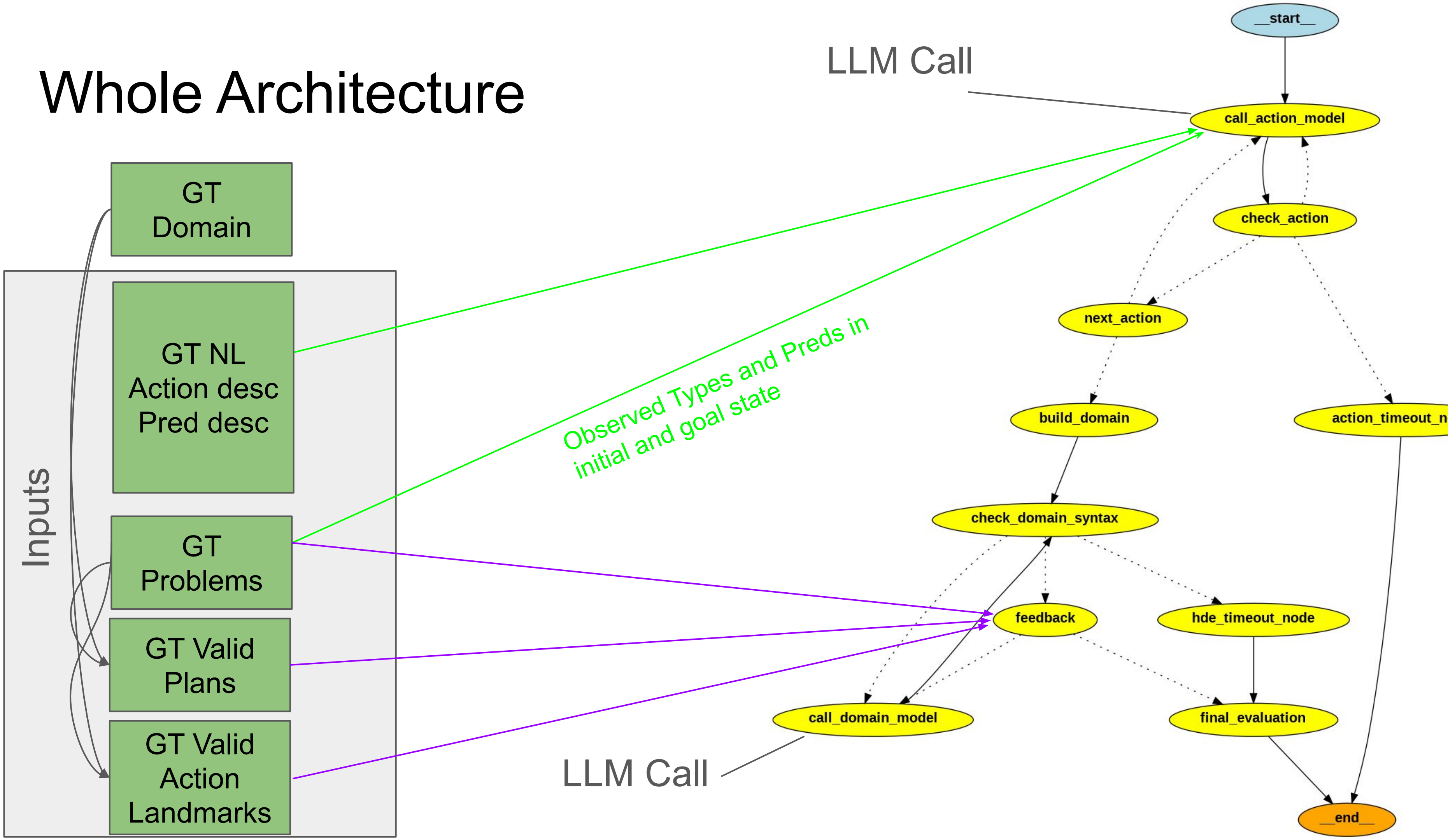
“(domain (blocks) (:action stack (?x) …”

Given the following problem (...) we expected that the following action (...) would appear on all plans, but it does not, please check its preconditions and other action’s effects.

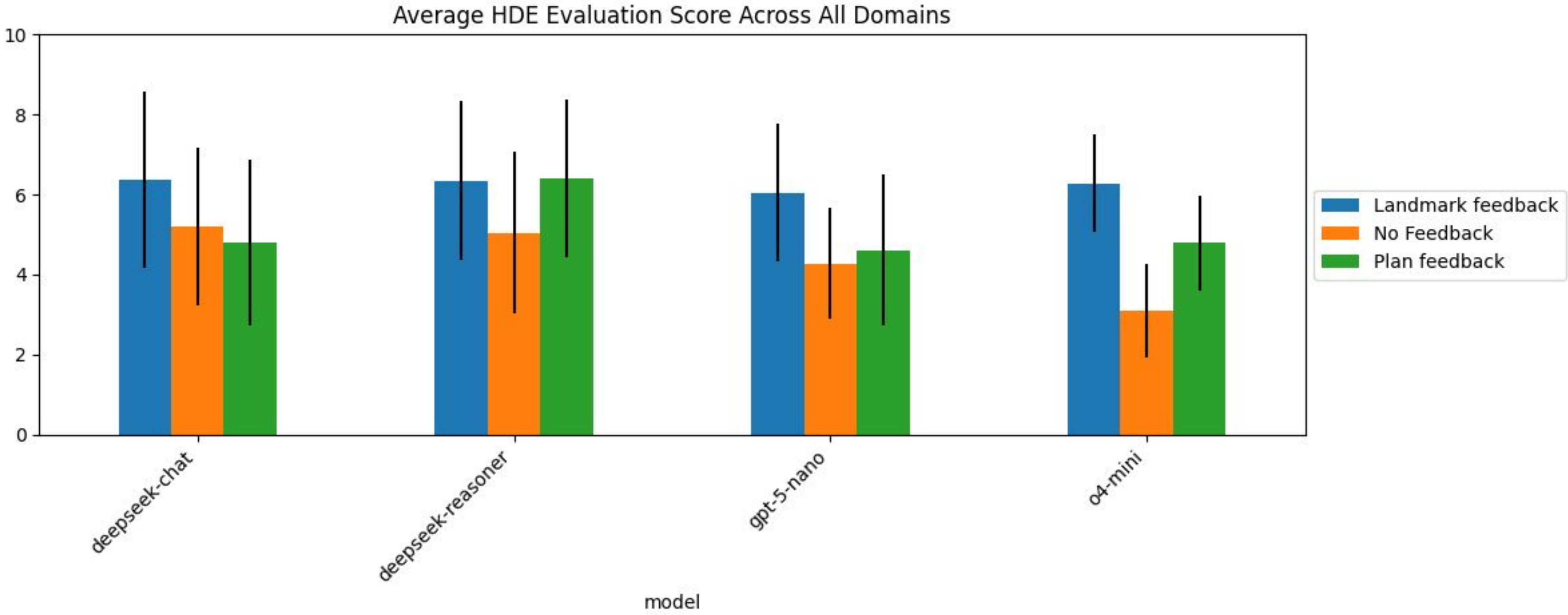


Feedbacks on landmark satisfaction failure.

Whole Architecture



With Feedback



Can we do better?

Not all feedback is equally useful:

“(domain (blocks) (:action stack (?x…”

HDE score 5/10

Feedback about
Landmark #2

Feedback about
Landmark #1

“(domain (blocks) (:action stack (?x) …”

HDE score 4/10

“(domain (blocks) (:action stack (?x) …”

HDE score 10/10

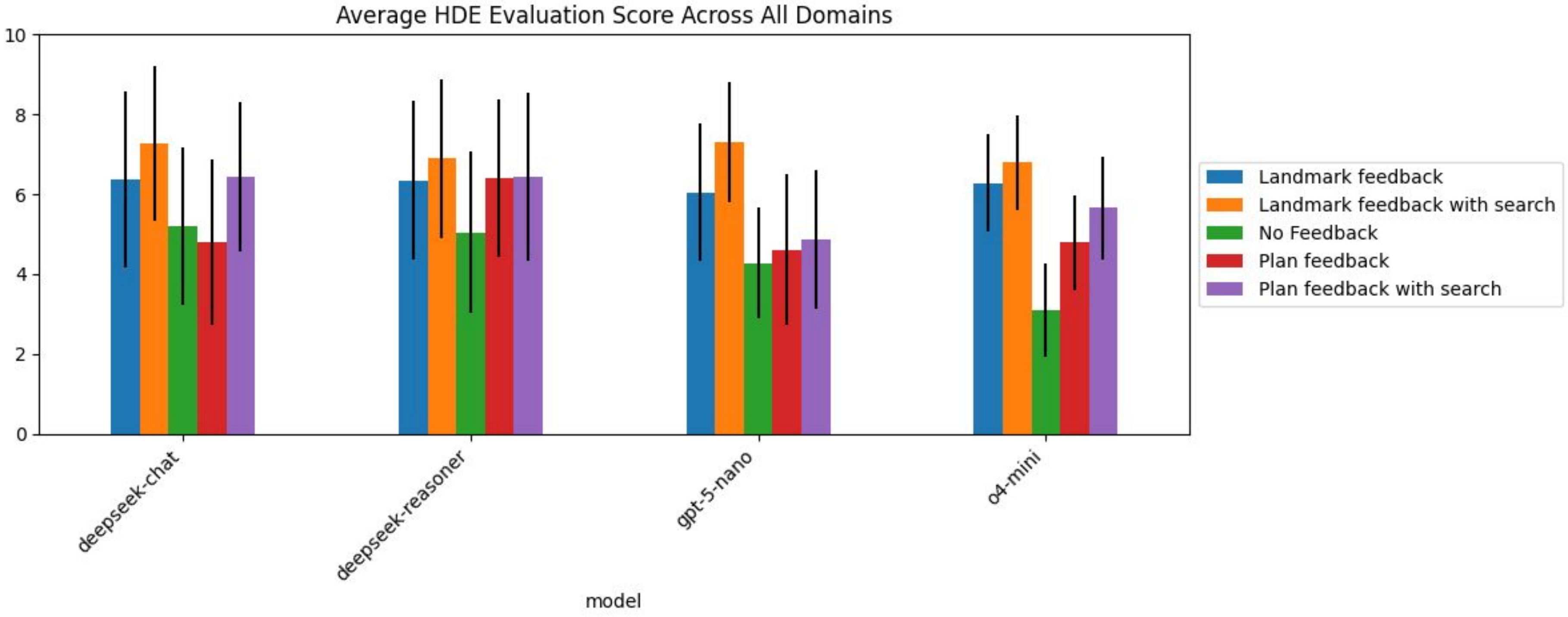
Feedback Selection as A Search Problem

By treating PDDL domains as a search space and feedback as actions over this search space, we can treat LLM PDDL domain construction as a search problem and apply heuristic search techniques to ensure the model doesn't backtrack.

We use best first search.

The downside of this is that we must call the LLM for every single piece of feedback.

With Search


Average HDE Evaluation Score Across All Domains
10
8
6
4
2
0
deepseek-chat
deepseek-reasoner
gpt-5-nano
o4-mini
model
Landmark feedback
Landmark feedback with search
No Feedback
Plan feedback
Plan feedback with search

Conclusions

- Search improves domain quality by ensuring we only investigate the most promising candidates.

No Feedback < Random Feedback < Search Over Feedback

- Landmark feedback compares favorably to plan feedback, despite generally containing less information.

Future Work

- Many other types of feedback:
  - Invariant Feedback
  - LTL Based Feedback
- Evaluation across multiple description classes.
- Exploration of more search heuristics to prevent getting stuck in local optima.

Questions?