Planning

Lara J. Martin (she/they)

https://laramartin.net/interactive-fiction-class

Slides adapted from Chris Callison-Burch

Learning Objectives

Identify the components of a planning problem and how they are used in PDDL

Distinguish between search and planning (and MDPs)

Discover limitations and benefits of planning for storytelling

Assess how LLMs and planning can work together for storytelling

Review: Formal Definition of a Search Problem

1. **States:** a set **S**
2. An **initial state** $s_i \in S$
3. **Actions:** a set **A**

   $\forall$ **s Actions(s)** = the set of actions that can be executed in **s**.

4. **Transition Model:** $\forall s \forall a \in$ **Actions(s)**

   Result(s, a) $\rightarrow s_r$

   s_r is called a successor of **s**

   $\{s_i\} \cup$ **Successors**$(s_i)^*$ = state space

5. **Path cost** (Performance Measure):

   Must be additive, e.g. sum of distances, number of actions executed, …

   c(x,a,y) is the step cost, assumed ≥ 0

   - (where action **a** goes from state **x** to state **y**)

6. **Goal test: Goal(s)**

   s is a goal state if **Goal(s)** is true.
   Can be implicit, e.g. **checkmate(s)**

Planning

Planning: The process of **searching** for a plan

- This is why we can use algorithms like BFS to find plans
- "Plain" state-based search is useful when we just want to get to the goal (efficiently); planning is useful when we care about the *path*
- What we think of as "planning" is a combination of search and logic

Plan: The result of planning; a sequence of steps from the initial state to a goal state

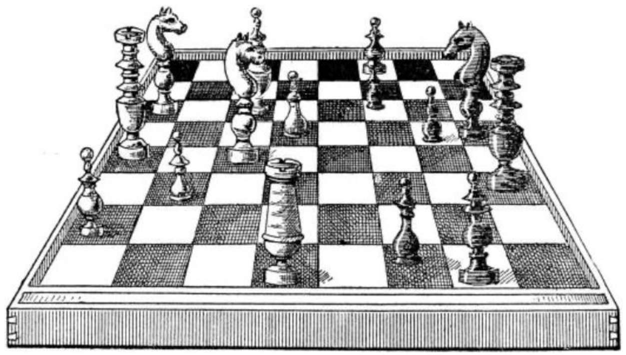
Policy: A collection of transition functions (Result(s, a) $\rightarrow s_r$) that tell the agent what action it should take for a given state

- This will become more relevant when talking about reinforcement learning

Classical Planning

Classical planning: The task of finding a sequence of action to accomplish a goal in an environment that:

- Is deterministic
- Is fully observable
- Contains a single agent
- Has a single initial state
- Is discrete



The solution to any problem in such an environment is a **fixed sequence of actions**.

More complicated planning

In environments that are

- Nondeterministic
- Partially observable
- Etc.

Nondeterministic actions (with assigned probabilities) turn classical planning problems into an MDP!

The solution must recommend different future actions depending on the what percepts it receives. This could be in the form of a branching strategy.

Planning languages

PDDL (Planning Domain Definition Language)

STRIPS (Stanford Research Institute Problem Solver)

ADL (Action Description Language)

…

We’ll focus on PDDL

PDDL breaks the planning problem into a **domain** and a **problem description**

The **domain** is consistent across problems (e.g., the description of the environment)

The **problem** defines what is going to be planned over

Domain

```
(define (domain action-castle)
  (:requirements :strips :typing)
  (:types
            player location direction
            fish crown - item
   )

  (:action go
    :parameters (?dir - direction ?p - player ?l1 - location ?l2 - location)
    :precondition (and (at ?p ?l1)
                       (connected ?l1 ?dir ?l2)
                       (not (blocked ?l1 ?dir ?l2)))
    :effect (and (at ?p ?l2)
            (not (at ?p ?l1)))
  )

  (:action get
    :parameters (?item - item  ?p - player  ?l1 - location )
    :precondition (and (at ?p ?l1)
                       (at ?item ?l1))
    :effect (and (inventory ?p ?item)
             (not (at ?item ?l1)))
))
```

Domain name

Object Types (can be hierarchical)

Parameters (variables)

Preconditions

Effects

Logical statements

Actions

Problem

```
(define (problem navigate-to-location)
  (:domain action-castle)

  (:objects
    npc - player
    cottage gardenpath fishingpond gardenpath
    windingpath talltree drawbridge courtyard
    towerstairs tower dungeonstairs dungeon
    greatfeastinghall throneroom - location
    in out north south east west up down - direction
  )

  (:init
    (at npc cottage)
    (connected cottage out gardenpath)
    (connected gardenpath in cottage)
    (connected gardenpath south fishingpond)
    (connected fishingpond north gardenpath)
  )

  (:goal (and (at npc throneroom) (sitting npc throne)))
)
```

Problem name

What domain to use

Objects
(the atoms)

Initial State

Goal

State Representation

In PDDL, a **state** is represented as a **conjunction** of logical sentences that are **ground atomic fluents**.

Ground means they contain no variables

Atomic sentences logical statements that can't be broken down

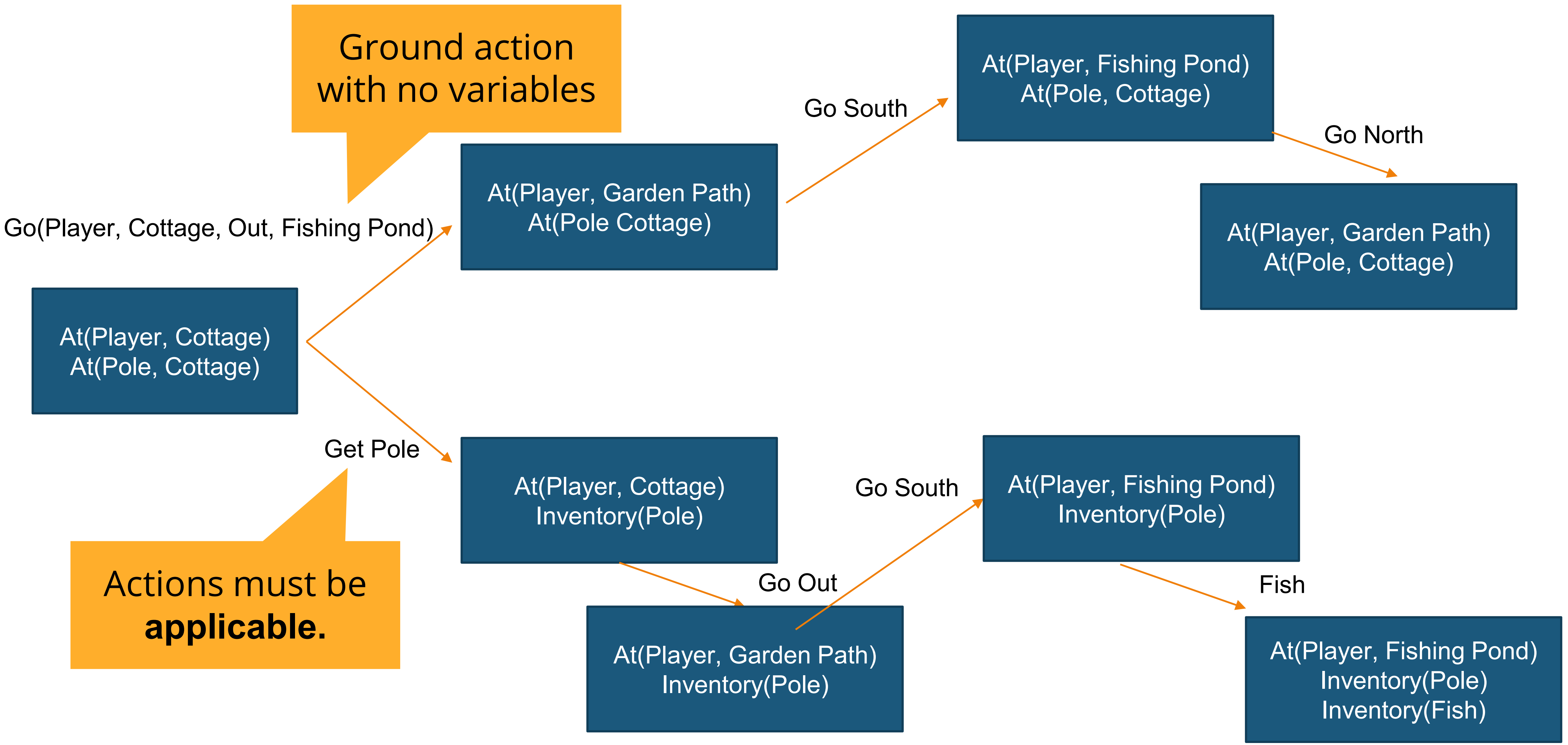
Fluent means an aspect of the world that can change over time

E.g.,

(*connected* gardenpath in cottage)

We make the *closed world assumption*: any fluent not mentioned is false

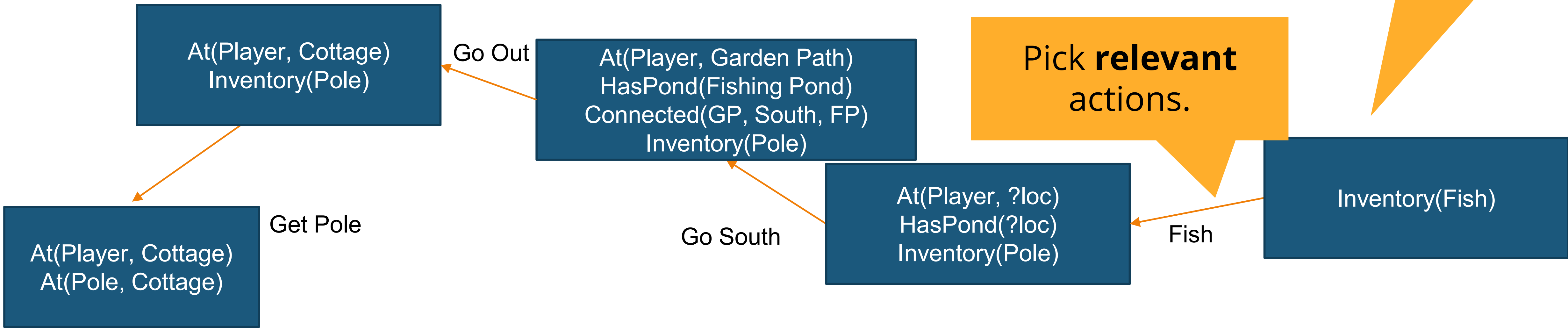
Forward State-Space Search for Planning



Backward State-Space Search for Planning (aka Regression Search)

Start with the **goal**, work **backwards** to initial state

Pick **relevant** actions.



Given a goal **g** and action **a,** the **regression** from g to a gives a state **g'** description whose literals are given by:

POS(g') = (POS(g)-ADD(a)) U POS(Preconditions(a))

NEG(g') = (NEG(g)-DEL(a)) U NEG(Preconditions(a))

Negative literals in the effects are kept in a **delete list** DEL

Positive literals in the effects are kept in an ADD list

Backward State-Space Search for Planning (aka Regression Search)

Given a goal **g** and action **a,** the **regression** from g to a gives a state **g'** description whose literals are given by:

POS(g') = (POS(g)-ADD(a)) U POS(Preconditions(a))

NEG(g') = (NEG(g)-DEL(a)) U NEG(Preconditions(a))

Or simply:

g' = (g - effects(a)) U Preconditions(a)

Partial-Order Planning

Keep a **partial order of steps** and only commit to an ordering when forced to

For example:

- Go north
- Pick up sword; Pick up lantern
- Go west

Review: Partial Order Causal Link (POCL) planning

Figure 1: Example CPOCL Problem and Domain

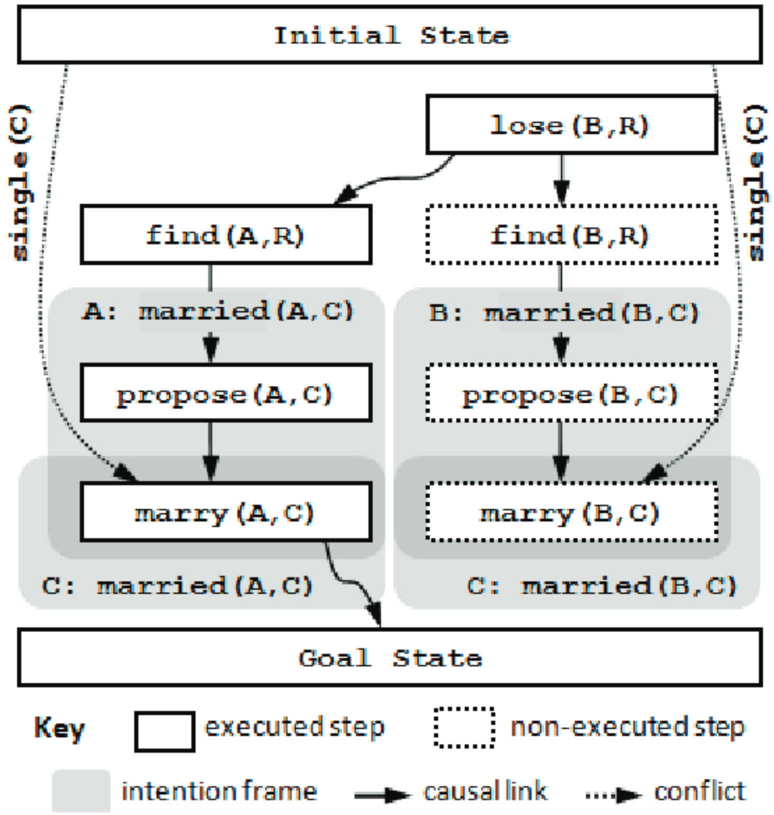
```
Initial:    single(A)∧single(B)∧single(C)∧
            loves(A,C)∧intends(A,married(A,C))∧
            loves(B,C)∧intends(B,married(B,C))∧has(B,R)
Goal:       married(A,C)

lose(?p,?i)                    find(?p,?i)
A: ∅                           A: ∅
P: has(?p,?i)                  P: lost(?i)
E: lost(?i)∧¬has(?p,?i)        E: has(?p,?i)∧¬lost(?i)

give(?p1,?p2,?i)               marry(?b,?g)
A: ?p1 ?p2                     A: ?b ?g
P: has(?p1,?i)                 P: loves(?b,?g)∧loves(?g,?b)
E: has(?p2,?i)∧¬has(?p1,?i)    ∧single(?b)∧single(?g)
                               E: married(?b,?g)∧
                               ¬single(?b)∧¬single(?g)

        propose(?b,?g)
        A: ?b
        P: loves(?b,?g)∧has(?b,R)
        E: loves(?g,?b)∧intends(?g,married(?b,?g))
```

Figure 2: Example CPOCL Plan



S. G. Ware and R. M. Young, "CPOCL: A Narrative Planner Supporting Conflict," *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 7, no. 1, pp. 97–102, 2011, doi: 10.1609/aiide.v7i1.12428.

Think-Pair-Share

What are some limitations of planning for storytelling?

What are some benefits?

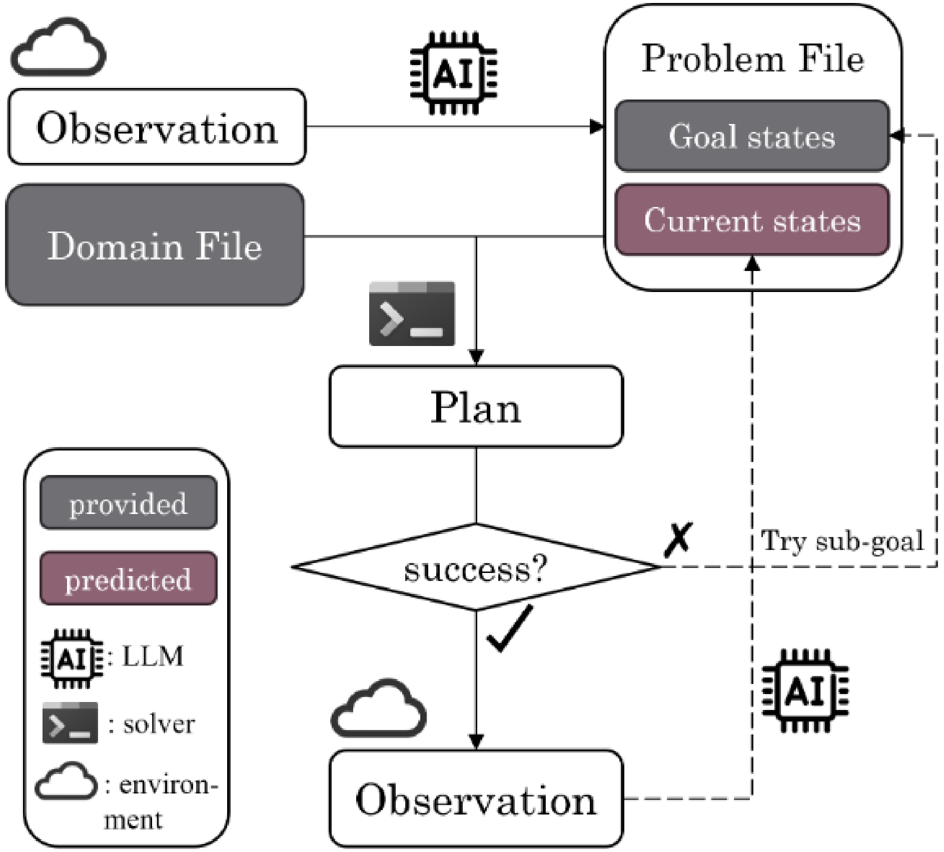
Neural Planning

Types of Neural Planning

Generation of planning language code

- To be run through planner

PDDLGO



Iterative creation of PDDL problem

Treat as partially-observed

Keep regenerating until plan succeeds

Li Zhang, Peter Jansen, Tianyi Zhang, Peter Clark, Chris Callison-Burch, and Niket Tandon. 2024. PDDLEGO: Iterative Planning in Textual Environments. In *Proceedings of the 13th Joint Conference on Lexical and Computational Semantics (*SEM 2024)*, pages 212–221, Mexico City, Mexico. Association for Computational Linguistics.

Types of Neural Planning

Generation of planning language code

- To be run through planner

Generation of planning-esque components

Neural Story Planning

Algorithm 1: Neural Plot Planner

Input: ending event sentence g; Initial conditions I.
Initialize a plan $P \leftarrow \emptyset$; Initialize $queue \leftarrow \{g\}$.
while $queue \neq \emptyset$ **do**
  Let $event \leftarrow$ **pop**$(queue)$
  Let $context \leftarrow$ sequence of events collected by running a breadth-first search from $event$ to g.
  Let $\Lambda \leftarrow$ all satisfied preconditions
  Let $\Gamma \leftarrow$ $generate\ preconds(event)$ for each character in $event$
  if adding any precondition in Γ creates a cycle **then**
    remove $event$ from P.
    $\Gamma \leftarrow$unsatisfied preconditions due to removing $event$.
  for each $c \in \Gamma$ **do**
    if $c \in I$ or c meets conditions for not being expanded **then**
      $P \leftarrow P \cup \{nil \xrightarrow{c} event\}$ ▷ Dangling precondition
    else if there exists a precondition $c' \in \Lambda$ that is similar to c **then**
      $event' \leftarrow$ event that satisfies c'
      $P \leftarrow P \cup \{event' \xrightarrow{c'} event\}$ ▷ Reuse precondition
    else
      $event' \leftarrow generate\ event(c, context)$
      $P \leftarrow P \cup \{event' \xrightarrow{c} event\}$ ▷ Satisfy with new event
      $queue \leftarrow queue \cup \{event'\}$

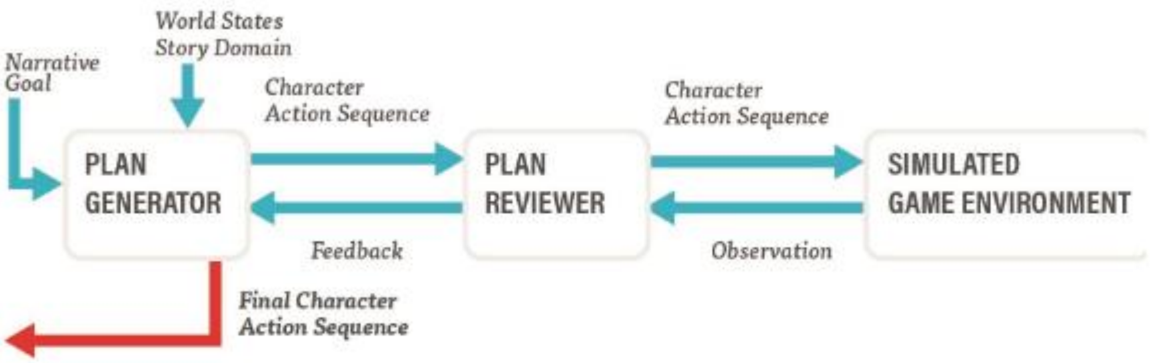
Generates preconditions for characters

- Items needed
- Locations
- Item states
- Context (How)
- Interactions with others
- Reasons

Partial-order-planning–inspired generation

Anbang Ye, Christopher Zhang Cui, Taiwei Shi and Mark Riedl. 2023. Neural Story Planning. In Proceedings of the AAAI-23 Workshop on Creative AI Across Modalities.
https://openreview.net/forum?id=cLBEKlu5WZK

StoryVerse



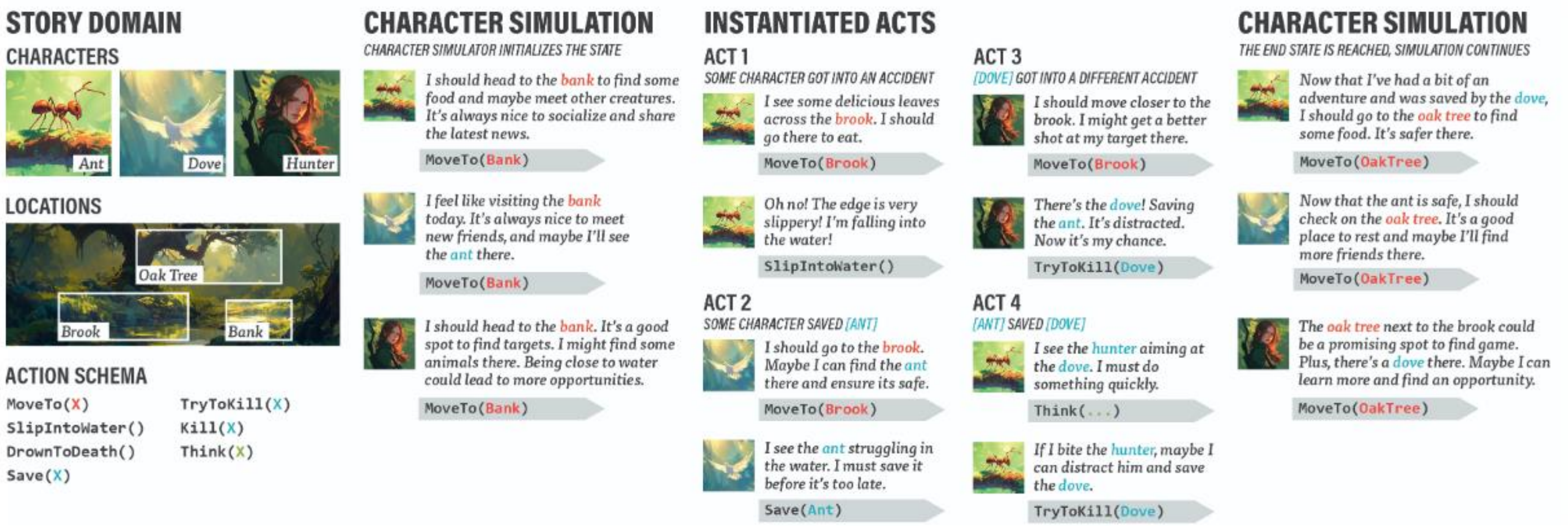


Figure 2: Two example story domains - The Ville (top) and Ant & Dove (bottom) - together with instantiated versions of the four abstract acts from Figure 1. Note that the text for narrations, dialogs, and monologues is all generated by LLMs.

Yi Wang, Qian Zhou, and David Ledo. 2024. StoryVerse: Towards Co-authoring Dynamic Plot with LLM-based Character Simulation via Narrative Planning. In Proceedings of the 19th International Conference on the Foundations of Digital Games (FDG '24). Association for Computing Machinery, New York, NY, USA, Article 55, 1–4. https://dl.acm.org/doi/10.1145/3649921.3656987

Types of Neural Planning

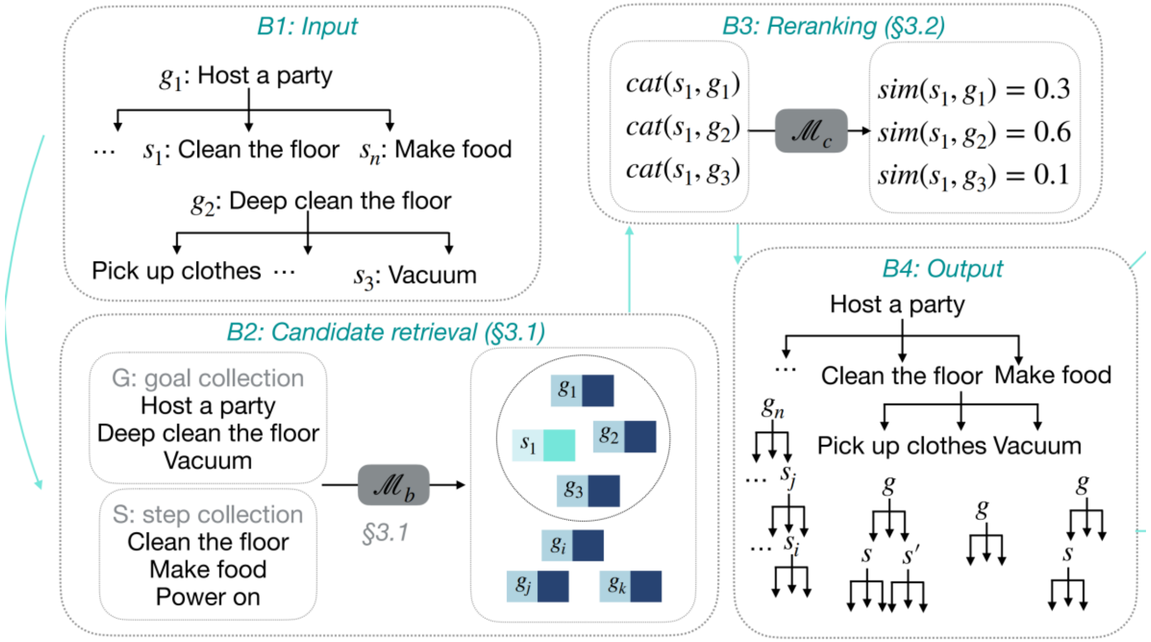
Generation of planning language code

- To be run through planner

Generation of planning-esque components

Use of LLM as *informal* planner

- Like the procedures work we looked at



Types of Neural Planning

Generation of planning language code

- To be run through planner

Generation of planning-esque components

Use of LLM as *informal* planner

- Like the procedures work we looked at
- Or using techniques like CoT

```
{"role": "system", "content": "# You are a helpful fiction writer assistant."},
{"role": "user", "content": f"{plot}\n
  Summarize the plot above into a plot tree of
  {'at most 6' if num_nodes == '' else num_nodes}
  nodes with each node containing the state and goal of {char_name}, and the key
  decision that propels the story forward. Each edge should contain a list of
  events that lead {char_name} to the state of next node. Also, Given the same
  state and goal of {char_name}, imagine an alternate decision that would have led
  {char_name} to a different storyline. Output in JSON format with schema:
  {JSON_SCHEMA}. Make sure that all important plot points are included in
  'edgeEvents' but not in 'state'"
}
```

Table 5: Prompt for generating a tree from the plot (plot-to-tree).

Huang, R., Martin, L.J., & Callison-Burch C. (2025). WHAT-IF: Exploring Branching Narratives by Meta-Prompting Large Language Models. Wordplay: Where Language Meets Games Workshop at EMNLP 2025. https://arxiv.org/abs/2412.10582

Types of Neural Planning

Generation of planning language code

- To be run through planner

Generation of planning-esque components

Use of LLM as *informal* planner

- Like the procedures work we looked at
- Or using techniques like CoT
- Or guided/hierarchical generation like Plan & Write

Dynamic	Storyline	needed → money → computer → bought → happy
	Story	John needed a computer for his birthday. He worked hard to earn money. John was able to buy his computer. He went to the store and bought a computer. John was happy with his new computer.
Static	Storyline	computer → slow → work → day → buy
	Story	I have an old computer. It was very slow. I tried to work on it but it wouldn't work. One day, I decided to buy a new one. I bought a new computer .

Yao, L., Peng, N., Weischedel, R., Knight, K., Zhao, D., & Yan, R. (2019). Plan-And-Write: Towards Better Automatic Storytelling. *AAAI Conference on Artificial Intelligence (AAAI)*, *33*(1), 7378–7385. https://aaai.org/ojs/index.php/AAAI/article/view/4726