LLM-Reasoners Demo

This notebook is accompanied with our tutorial at SIGIR VF: [slides] [video (starting at 37:20)]

Setup

The following code assumes you have cloned our library with `git clone https://github.com/maitrix-org/llm-reasoners.git --recursive`

Set cuda device and initialize an ExllamaModel use our unified LLM interface.

In [1]:

```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0,1'
```

In [2]:

```
from reasoners.lm import ExLlamaModel
import torch
```

```
/home/minzheguo/anaconda3/envs/reasoners/lib/python3.10/site-packages/tqdm/a
uto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywi
dgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [3]:

```
from reasoners.lm import ExLlamaModel
import torch

# https://huggingface.co/TheBloke/Llama-2-70B-GPTQ
# It may take a few minutes to download the model

model = ExLlamaModel(model_dir='TheBloke/Llama-2-70B-GPTQ',
                     lora_dir=None,
                     device = torch.device("cuda:0"),
                     max_batch_size=1,
                     max_new_tokens=200,
                     mem_map=[16,22], # For 2 * 24GB GPUs. If you have > 40G
                     max_seq_length=2048)

# Or use any other model providers:

# HFModel(llama_path, llama_path, device=device, max_batch_size=1, max_new_t
# Llama3Model(llama2_ckpts, llama_size, max_batch_size=1)
# OpenAIModel(openai_mode)
# ClaudeModel('claude-3-opus-20240229')
```

```
/home/minzheguo/pytorch/torch/utils/cpp_extension.py:2068: UserWarning: TORC
H_CUDA_ARCH_LIST is not set, all archs for visible cards are included for co
mpilation.
If this is not desired, please set os.environ['TORCH_CUDA_ARCH_LIST'].
  warnings.warn(
Fetching 13 files: 100%|██████████| 13/13 [00:00<00:00, 69994.80it/s]
```

We gather one example from the Blocksworld dataset, and the proper prompt for in-context learning examples. We will talk more about Evaluators later.

In [ ]:

```
from reasoners.benchmark import BWEvaluator
import json

with open('examples/CoT/blocksworld/prompts/pool_prompt_v1.json') as f:
    prompt = json.load(f)
# print(prompt)
evaluator = BWEvaluator(config_file='examples/CoT/blocksworld/data/bw_config
                        domain_file='examples/CoT/blocksworld/data/generated
                        data_path='examples/CoT/blocksworld/data/split_v1/sp
                        init_prompt=prompt)
prompt = evaluator.sample_prompt(shuffle_prompt=False, num_shot=4)
example = evaluator.full_dataset[1]
print(prompt['icl'])
cot_inputs = (prompt['icl'].replace('<init_state>', example["init"])
                           .replace('<goals>', example["goal"])
                           .replace('<action>', ''))

# import difflib
# diff = difflib.ndiff(prompt['icl'], cot_inputs)
# print(''.join(diff))
# print(cot_inputs)
```

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block

I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.
I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.
I can only unstack a block from on top of another block if the block I am unstacking is clear.
Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding the block being stacked.
I can only stack a block on top of another block if the block onto which I am stacking the block is clear.
Once I put down or stack a block, my hand becomes empty.

[STATEMENT]
As initial conditions I have that, the red block is clear, the orange block is clear, the hand is empty, the orange block is on top of the blue block, the red block is on the table and the blue block is on the table.
My goal is to have that the blue block is on top of the orange block.

My plan is as follows:

[PLAN]
unstack the orange block from on top of the blue block
put down the orange block
pick up the blue block
stack the blue block on top of the orange block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the blue block is clear, the orange block is clear, the hand is empty, the red block is on top of the yellow block, the orange block is on top of the red block, the blue block is on the table and the yellow block is on the table.
My goal is to have that the blue block is on top of the yellow block and the orange block is on top of the blue block.

My plan is as follows:

[PLAN]
unstack the orange block from on top of the red block
put down the orange block
unstack the red block from on top of the yellow block
put down the red block

```
pick up the blue block
stack the blue block on top of the yellow block
pick up the orange block
stack the orange block on top of the blue block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the red block is clear, the yellow block
is clear, the hand is empty, the red block is on top of the blue block, the
blue block is on top of the orange block, the orange block is on the table a
nd the yellow block is on the table.
My goal is to have that the blue block is on top of the orange block and the
yellow block is on top of the red block.

My plan is as follows:

[PLAN]
pick up the yellow block
stack the yellow block on top of the red block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the blue block is clear, the yellow block
is clear, the hand is empty, the red block is on top of the orange block, th
e blue block is on top of the red block, the orange block is on the table an
d the yellow block is on the table.
My goal is to have that the blue block is on top of the red block and the ye
llow block is on top of the blue block.

My plan is as follows:

[PLAN]
pick up the yellow block
stack the yellow block on top of the blue block
[PLAN END]

[STATEMENT]
As initial conditions I have that, <init_state>
My goal is to <goals>

My plan is as follows:

[PLAN]
<action>
I am playing with a set of blocks where I need to arrange the blocks into st
acks. Here are the actions I can do

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block

I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the block is cle
```

ar. A block is clear if the block has no other blocks on top of it and if the block is not picked up.
I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.
I can only unstack a block from on top of another block if the block I am unstacking is clear.
Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding the block being stacked.
I can only stack a block on top of another block if the block onto which I am stacking the block is clear.
Once I put down or stack a block, my hand becomes empty.

[STATEMENT]
As initial conditions I have that, the red block is clear, the orange block is clear, the hand is empty, the orange block is on top of the blue block, the red block is on the table and the blue block is on the table.
My goal is to have that the blue block is on top of the orange block.

My plan is as follows:

[PLAN]
unstack the orange block from on top of the blue block
put down the orange block
pick up the blue block
stack the blue block on top of the orange block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the blue block is clear, the orange block is clear, the hand is empty, the red block is on top of the yellow block, the orange block is on top of the red block, the blue block is on the table and the yellow block is on the table.
My goal is to have that the blue block is on top of the yellow block and the orange block is on top of the blue block.

My plan is as follows:

[PLAN]
unstack the orange block from on top of the red block
put down the orange block
unstack the red block from on top of the yellow block
put down the red block
pick up the blue block
stack the blue block on top of the yellow block
pick up the orange block
stack the orange block on top of the blue block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the red block is clear, the yellow block is clear, the hand is empty, the red block is on top of the blue block, the blue block is on top of the orange block, the orange block is on the table and the yellow block is on the table.
My goal is to have that the blue block is on top of the orange block and the

```
yellow block is on top of the red block.

My plan is as follows:

[PLAN]
pick up the yellow block
stack the yellow block on top of the red block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the blue block is clear, the yellow block
is clear, the hand is empty, the red block is on top of the orange block, th
e blue block is on top of the red block, the orange block is on the table an
d the yellow block is on the table.
My goal is to have that the blue block is on top of the red block and the ye
llow block is on top of the blue block.

My plan is as follows:

[PLAN]
pick up the yellow block
stack the yellow block on top of the blue block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the blue block is clear, the orange block
is clear, the hand is empty, the orange block is on top of the red block, th
e red block is on the table and the blue block is on the table
My goal is to the red block is on top of the blue block

My plan is as follows:

[PLAN]
```

Here is the example.

In [6]:

```
print(example['init'])
```

```
the blue block is clear, the orange block is clear, the hand is empty, the o
range block is on top of the red block, the red block is on the table and th
e blue block is on the table
```

In [7]:

```
print(example['goal'])
```

```
the red block is on top of the blue block
```

Chain-of-Thought

We first experiment with the Chain-of-Thought method. Since we are having the simplest generation algorithm, we directly ask the model to generate all the steps. We look at the 4-shot prompt and the generated answer.

In [15]:

```
print(cot_inputs)
```

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block

I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.
I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.
I can only unstack a block from on top of another block if the block I am unstacking is clear.
Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding the block being stacked.
I can only stack a block on top of another block if the block onto which I am stacking the block is clear.
Once I put down or stack a block, my hand becomes empty.

[STATEMENT]
As initial conditions I have that, the red block is clear, the orange block is clear, the hand is empty, the orange block is on top of the blue block, the red block is on the table and the blue block is on the table.
My goal is to have that the blue block is on top of the orange block.

My plan is as follows:

[PLAN]
unstack the orange block from on top of the blue block
put down the orange block
pick up the blue block
stack the blue block on top of the orange block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the blue block is clear, the orange block is clear, the hand is empty, the red block is on top of the yellow block, the orange block is on top of the red block, the blue block is on the table and the yellow block is on the table.
My goal is to have that the blue block is on top of the yellow block and the orange block is on top of the blue block.

My plan is as follows:

[PLAN]
unstack the orange block from on top of the red block
put down the orange block
unstack the red block from on top of the yellow block
put down the red block

```
pick up the blue block
stack the blue block on top of the yellow block
pick up the orange block
stack the orange block on top of the blue block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the red block is clear, the yellow block
is clear, the hand is empty, the red block is on top of the blue block, the
blue block is on top of the orange block, the orange block is on the table a
nd the yellow block is on the table.
My goal is to have that the blue block is on top of the orange block and the
yellow block is on top of the red block.

My plan is as follows:

[PLAN]
pick up the yellow block
stack the yellow block on top of the red block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the blue block is clear, the yellow block
is clear, the hand is empty, the red block is on top of the orange block, th
e blue block is on top of the red block, the orange block is on the table an
d the yellow block is on the table.
My goal is to have that the blue block is on top of the red block and the ye
llow block is on top of the blue block.

My plan is as follows:

[PLAN]
pick up the yellow block
stack the yellow block on top of the blue block
[PLAN END]

[STATEMENT]
As initial conditions I have that, the blue block is clear, the orange block
is clear, the hand is empty, the orange block is on top of the red block, th
e red block is on the table and the blue block is on the table
My goal is to the red block is on top of the blue block

My plan is as follows:

[PLAN]
```

In [16]:

```
output = model.generate([cot_inputs],
                        hide_input=True,
                        eos_token_id='\n[').text[0][:-1].strip()
```

```
/home/minzheguo/llm-reasoners/reasoners/lm/exllama_model.py:124: UserWarnin
g: max_new_tokens is not set, we will use the default value: 200
  warnings.warn(f"max_new_tokens is not set, we will use the default value:
{self.max_new_tokens}")
/home/minzheguo/llm-reasoners/reasoners/lm/exllama_model.py:127: UserWarnin
g: do_sample is False while the temperature is non-positive. We will use gre
edy decoding for Exllama
  warnings.warn(
/home/minzheguo/llm-reasoners/reasoners/lm/exllama_model.py:149: UserWarnin
g: the eos_token '\n[' is encoded into tensor([29871,    13, 29961]) with le
ngth != 1, using 29961 as the eos_token_id
  warnings.warn(f'the eos_token {repr(token)} is encoded into {tokenized} wi
th length != 1, '
```

In [17]:

```
print(output)
```

```
pick up the red block
stack the red block on top of the blue block
```

Clearly that's not a valid solution :(The orange block is on the red block, so we cannot pick up the red block as the first step.

Tree-of-Thought

Then let's turn to a tree search algorithm, Tree-of-Thought). We will need to define a simple world model, and a search algorithm, for the Blocksworld task.

In [19]:

```
from reasoners import WorldModel, LanguageModel, SearchConfig, State, Reasor
from reasoners.algorithm import BeamSearch, MCTS
import reasoners.benchmark.bw_utils as utils
from typing import NamedTuple
import copy
import numpy as np


# We use NamedTuple for clearer presentation, you may just use normal tuple
class BWStateToT(NamedTuple):
    step_idx: int
    action_history: list[str]
    end: bool


# We just use the description str as the action, we use a type alias for bet
# You may directly use str of you want a quick experiment.
BWAction = str


class BlocksWorldModelToT(WorldModel):
    def __init__(self,
                 base_model: LanguageModel,
                 prompt: dict,
                 max_steps: int = 4,
                 batch_size: int = 1) -> None:
```

```
        super().__init__()
        self.max_steps = max_steps
        self.base_model = base_model
        self.prompt = prompt
        self.batch_size = batch_size

    def init_state(self) -> BWStateToT:
        return BWStateToT(step_idx=0, action_history=[], end=False)

    def step(self, state: BWStateToT, action: BWAction) -> tuple[BWStateToT,
        state = copy.deepcopy(state)
        if action != "[PLAN END]":
            state = BWStateToT(step_idx=state.step_idx + 1, action_history=s
        else:
            state = BWStateToT(step_idx=state.step_idx + 1, action_history=s
        return state, {}  # the dict is auxiliary information for SearchConf

    def is_terminal(self, state: State) -> bool:
        return state.end or state.step_idx >= self.max_steps


class BWConfigToT(SearchConfig):
    def __init__(self,
                 base_model: LanguageModel,
                 prompt: dict,
                 temperature: float = 0.8,
                 n_candidate: int = 4) -> None:
        super().__init__()
        self.base_model = base_model
        self.example = None
        self.prompt = prompt
        self.n_candidate = n_candidate
        self.temperature = temperature

    def get_actions(self, state: BWStateToT) -> list[BWAction]:
        prompts = (self.prompt["icl"]
                       .replace("<action>", "\n".join(state.action_history +
                       .replace("<init_state>", utils.extract_init_state(sel
                       .replace("<goals>", utils.extract_goals(self.example,
        outputs = self.base_model.generate([prompts],
                                           num_return_sequences=self.n_candi
                                           max_length=20,
                                           eos_token_id="\n",
                                           temperature=self.temperature,
                                           do_sample=True,
                                           hide_input=True).text
        outputs = [output.split("\n")[0] for output in outputs]
        outputs = list(dict.fromkeys(outputs))  # deduplicate
        return outputs

    # Some reward functions are fast to calculate.
    # We calculate the reward before executing the action, which can be used
    def fast_reward(self, state: BWStateToT, action: BWAction) -> tuple[floa
        # We use two rewards here:
        # 1. Intuition: The loglikelihood of the action given the prompt.
        # 2. Self-eval: Ask the language model whether this step is "Good".
        inputs = self.prompt["icl"].replace("<action>", "\n".join(state.acti
            .replace("<init_state>", utils.extract_init_state(self.example))
            .replace("<goals>", utils.extract_goals(self.example, return_raw

        intuition = self.base_model.get_loglikelihood(inputs, [inputs + "\n"

        self_eval_prompt = (self.prompt["self-eval"].replace("<init_state>",
                                                     .replace("<goals>", util
                                                     .replace("<action>", act
        self_eval = self.base_model.get_loglikelihood(self_eval_prompt, [sel

        return intuition + self_eval, {'intuition': intuition, "self_eval":

    # kwargs is the auxiliary information returned by SearchConfig.fast_rewa
    # so that we do not need duplicated calculations.
    # In this case, we just use the fast_reward result as the reward.
    # Generally, if a reward function depends on the new state, or is slow t
    # we will calculate it here.
    def reward(self, state, action, **kwargs) -> tuple[float, dict]:
        return kwargs['intuition'] + kwargs['self_eval'], kwargs
```

Note: The following command may take to 2 minutes to run

In [12]:

```
world_model = BlocksWorldModelToT(base_model=model, prompt=prompt)
config = BWConfigToT(base_model=model, prompt=prompt)
algorithm = BeamSearch(beam_size=4, max_depth=7)
reasoner_tot = Reasoner(world_model=world_model, search_config=config, searc
result_tot = reasoner_tot(example)
print(result_tot)
```

```
/home/minzheguo/llm-reasoners/reasoners/lm/exllama_model.py:122: UserWarnin
g: max_length is not supported by ExLlamaModel for generation. Use max_new_t
okens instead.
  warnings.warn("max_length is not supported by ExLlamaModel for generation.
Use max_new_tokens instead.")
/home/minzheguo/llm-reasoners/reasoners/lm/exllama_model.py:149: UserWarnin
g: the eos_token '\n' is encoded into tensor([29871,    13]) with length !=
1, using 13 as the eos_token_id
  warnings.warn(f'the eos_token {repr(token)} is encoded into {tokenized} wi
th length != 1, '
```

```
BeamSearchResult(terminal_node=<reasoners.algorithm.beam_search.BeamSearchNo
de object at 0x7f394d1d1ea0>, terminal_state=BWStateToT(step_idx=3, action_h
istory=['pick up the red block', 'stack the red block on top of the blue blo
ck'], end=True), cum_reward=np.float32(-0.6805274), tree=<reasoners.algorith
m.beam_search.BeamSearchNode object at 0x7f394cfd75b0>, trace=[(None, BWStat
eToT(step_idx=0, action_history=[], end=False), 0.0), ('pick up the red bloc
k', BWStateToT(step_idx=1, action_history=['pick up the red block'], end=Fal
se), np.float32(-1.0768182)), ('stack the red block on top of the blue bloc
k', BWStateToT(step_idx=2, action_history=['pick up the red block', 'stack t
he red block on top of the blue block'], end=False), np.float32(-0.7901649
5)), ('[PLAN END]', BWStateToT(step_idx=3, action_history=['pick up the red
block', 'stack the red block on top of the blue block'], end=True), np.float
32(-0.6805274))])
```

In [13]:

```
print('Action, Reward')
for action, _, reward in result_tot.trace:
    print(action, reward)
```

```
Action, Reward
None 0.0
pick up the red block -1.0768182
stack the red block on top of the blue block -0.79016495
[PLAN END] -0.6805274
```

Still the same error :(

RAP

With RAP, we are truly using the latest block configuration as the state, instead of a history of actions. Thus, we define a new world model to transit between states, which is just a little complex than the previous one.

In [20]:

```
BWAction = str


class BWStateRAP(NamedTuple):
    step_idx: int
    last_blocks_state: str
    blocks_state: str
    buffered_action: BWAction


class BlocksWorldModelRAP(WorldModel):
    def __init__(self,
                 base_model: LanguageModel,
                 prompt: dict,
                 max_steps: int = 4,
                 batch_size: int = 1) -> None:
        super().__init__()
        self.max_steps = max_steps
        self.base_model = base_model
        self.prompt = prompt
        self.batch_size = batch_size

    def init_state(self) -> BWStateRAP:
        return BWStateRAP(step_idx=0, last_blocks_state="", blocks_state=uti
                          extract_init_state(self.example), buffered_action="")

    def step(self, state: BWStateRAP, action: BWAction) -> tuple[BWStateRAP,
        state = copy.deepcopy(state)
        blocks_state = state.blocks_state
        step_idx = state.step_idx
        blocks_state = self.update_blocks(blocks_state, action)
        new_buffered_action = action if state.buffered_action == "" else ""

        state = BWStateRAP(step_idx=step_idx + 1,
                           last_blocks_state=state.blocks_state,
                        blocks_state=blocks_state,
                        buffered_action=new_buffered_action)
        return state, {"goal_reached": utils.goal_check(utils.extract_goals(

    def update_blocks(self, block_states: str, action: BWAction) -> str:
        if "pick" in action:
            key = "world_update_pickup"
        elif "unstack" in action:
            key = "world_update_unstack"
        elif "put" in action:
            key = "world_update_putdown"
        elif "stack" in action:
            key = "world_update_stack"
        else:
            raise ValueError("Invalid action")
        world_update_prompt = self.prompt[key].format(block_states, action.c
        world_output = self.base_model.generate([world_update_prompt],
                                                eos_token_id="\n",
                                                hide_input=True,
                                                temperature=0).text[0].strip
        new_state = utils.apply_change(world_output, block_states)
        return new_state

    def is_terminal(self, state: BWStateRAP) -> bool:
        if utils.goal_check(utils.extract_goals(self.example), state.blocks_
            return True
        elif state.step_idx == self.max_steps:
            return True
        return False
```

In [21]:
```
class BWConfigRAP(SearchConfig):
    def __init__(self,
                 base_model: LanguageModel,
                 prompt: dict,
                 batch_size: int = 1,
                 reward_alpha: float = 0.5,
                 goal_reward_default: float = 0.,
                 goal_reached_reward: float = 100.) -> None:
        super().__init__()
        self.base_model = base_model
        self.example = None
        self.prompt = prompt
        self.batch_size = batch_size
        self.reward_alpha = reward_alpha
        self.goal_reward_default = goal_reward_default
        self.goal_reached_reward = goal_reached_reward

    def get_actions(self, state: BWStateRAP) -> list[BWAction]:
        blocks_state = state.blocks_state
        return utils.generate_all_actions(blocks_state)

    def fast_reward(self, state: BWStateRAP, action: BWAction) -> tuple[floa
        if state.buffered_action == "":
            current_blocks_state = state.blocks_state
        else:
            current_blocks_state = state.last_blocks_state
        previous_action = state.buffered_action + "\n" if state.buffered_act

        # every two steps, we will also reduce the icl examples by 2 steps
        # so that the distribution of step length in examples is more reason
        icl_template = self.prompt["icl_list"][state.step_idx // 2]

        inputs = (icl_template.replace("<init_state>", current_blocks_state)
                              .replace("<goals>", utils.extract_goals(self.e
                              .replace("<action>", previous_action))
        intuition = self.base_model.get_loglikelihood(inputs, [inputs + acti

        self_eval_prompt = (self.prompt["self-eval"]
                                .replace("<init_state>", current_blocks_stat
                                .replace("<goals>", utils.extract_goals(self
                                .replace("<action>", action))
        self_eval = self.base_model.get_loglikelihood(self_eval_prompt, [sel

        return (self.calculate_reward(intuition, self_eval),
                {'intuition': intuition, "self_eval": self_eval})

    def calculate_reward(self, intuition, self_eval, goal_reached=None) -> f
        # to provide a unified interface for reward and fast_reward
        if goal_reached is None:
            goal_reward = self.goal_reward_default
        elif goal_reached[0]:
            goal_reward = self.goal_reached_reward
        else:
            goal_reward = goal_reached[1]
        return (intuition + self_eval) * self.reward_alpha + goal_reward * (

    def reward(self, state: BWStateRAP, action: BWAction,
               intuition: float = None,
               self_eval: float = None,
               goal_reached: tuple[bool, float] = None) -> tuple[float, dict
        return (self.calculate_reward(intuition, self_eval, goal_reached),
                {'intuition': intuition, 'goal_reached': goal_reached})
```

We just use the MCTS algorithm embedded in Reasoners, and build up the pipeline again. Note: the following command may take 2 minutes to run

In [ ]:

```
print(prompt['world_update_pickup'])
world_model = BlocksWorldModelRAP(base_model=model, prompt=prompt, max_steps
config = BWConfigRAP(base_model=model, prompt=prompt)
algorithm = MCTS(depth_limit=4, disable_tqdm=False, output_trace_in_each_ite
reasoner_rap = Reasoner(world_model=world_model, search_config=config, searc
result_rap = reasoner_rap(example)
# print(result_rap)
```

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block

I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.
I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.
I can only unstack a block from on top of another block if the block I am unstacking is clear. Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding the block being stacked.
I can only stack a block on top of another block if the block onto which I am stacking the block is clear. Once I put down or stack a block, my hand becomes empty.

After being given an initial state and an action, give the new state after performing the action.

[SCENARIO 1]
[STATE 0] I have that, the white block is clear, the cyan block is clear, the brown block is clear, the hand is empty, the white block is on top of the purple block, the purple block is on the table, the cyan block is on the table and the brown block is on the table.
[ACTION] Pick up the brown block.
[CHANGE] The hand was empty and is now holding the brown block, the brown block was on the table and is now in the hand, and the brown block is no longer clear.
[STATE 1] I have that, the white block is clear, the cyan block is clear, the brown block is in the hand, the hand is holding the brown block, the white block is on top of the purple block, the purple block is on the table and the cyan block is on the table.

[SCENARIO 2]
[STATE 0] I have that, the purple block is clear, the cyan block is clear, the white block is clear, the hand is empty, the white block is on top of the brown block, the purple block is on the table, the cyan block is on the table and the brown block is on the table.
[ACTION] Pick up the cyan block.
[CHANGE] The hand was empty and is now holding the cyan block, the cyan block was on the table and is now in the hand, and the cyan block is no longer clear.
[STATE 1] I have that, the cyan block is in the hand, the white block is clear, the purple block is clear, the hand is holding the cyan block, the white block is on top of the brown block, the purple block is on the table and the brown block is on the table.

```
[SCENARIO 3]
[STATE 0] I have that, {}
[ACTION] {}
[CHANGE]
```

In [17]:

```
result_rap.trace
```

Out[17]:

```
([BWStateRAP(step_idx=0, last_blocks_state='', blocks_state='the blue block
is clear, the orange block is clear, the hand is empty, the orange block is
on top of the red block, the red block is on the table and the blue block i
s on the table.', buffered_action=''),
  BWStateRAP(step_idx=1, last_blocks_state='the blue block is clear, the or
ange block is clear, the hand is empty, the orange block is on top of the r
ed block, the red block is on the table and the blue block is on the tabl
e.', blocks_state='the blue block is clear, the orange block is in the han
d, the red block is clear, the hand is holding the orange block, the blue b
lock is on the table, and the red block is on the table.', buffered_action
='unstack the orange block from on top of the red block'),
  BWStateRAP(step_idx=2, last_blocks_state='the blue block is clear, the or
ange block is in the hand, the red block is clear, the hand is holding the
orange block, the blue block is on the table, and the red block is on the t
able.', blocks_state='the blue block is clear, the orange block is clear, t
he red block is clear, the hand is empty, the blue block is on the table, t
he orange block is on the table, and the red block is on the table.', buffe
red_action=''),
  BWStateRAP(step_idx=3, last_blocks_state='the blue block is clear, the or
ange block is clear, the red block is clear, the hand is empty, the blue bl
ock is on the table, the orange block is on the table, and the red block is
on the table.', blocks_state='the blue block is clear, the orange block is
clear, the red block is in the hand, the hand is holding the red block, the
blue block is on the table, and the orange block is on the table.', buffere
d_action='pick up the red block'),
  BWStateRAP(step_idx=4, last_blocks_state='the blue block is clear, the or
ange block is clear, the red block is in the hand, the hand is holding the
red block, the blue block is on the table, and the orange block is on the t
able.', blocks_state='the orange block is clear, the red block is clear, th
e hand is empty, the red block is on top of the blue block, the blue block
is on the table, and the orange block is on the table.', buffered_action
='')],
 ['unstack the orange block from on top of the red block',
  'put down the orange block',
  'pick up the red block',
  'stack the red block on top of the blue block'])
```

Finally, we get a valid solution!

Visualization

Visualization is as simple as calling `visualize(log)`

In [18]:

```
from reasoners.visualization import visualize
from reasoners.visualization.tree_snapshot import NodeData, EdgeData
from reasoners.algorithm.mcts import MCTSNode
```

```
# (Optional) You can write node_data_factory and edge_data_factory to show c
def blocksworld_node_data_factory(n: MCTSNode) -> NodeData:
    return NodeData({"block state": n.state.blocks_state if n.state else "No
                     "# goals satisfied": n.reward_details["goal_reached"][1
                     "# visited": len(n.cum_rewards)})

def blocksworld_edge_data_factory(n: MCTSNode) -> EdgeData:
    return EdgeData({"Q": n.Q,
                     "intuition": n.fast_reward_details["intuition"],
                     "self_eval": n.fast_reward_details["self_eval"],
                     "action": n.action})

visualize(result_rap,
          node_data_factory=blocksworld_node_data_factory,
          edge_data_factory=blocksworld_edge_data_factory)
```

```
Visualizer URL: https://main.d1puk3wdon4rk8.amplifyapp.com/visualizer/80cda3
44-cb51-458a-84db-1dc538ebf791?accessKey=dd9102f5
```

This evaluator module provides standard APIs and easy implementation of multiple popular reasoning datasets.

In [19]:

```
# a helper function to extract the action history from the output of the alg

def bfs_bw_extractor(algo_output):
    if torch.distributed.is_initialized():
        torch.distributed.barrier()
    # to make sure the plan is saved before evaluation in multi-process sett
    try:
        return "\n".join(algo_output.terminal_node.state.action_history)
    except Exception as e:
        print("Error in output extraction,", e)
        return ""
```

In [20]:

```
with open('examples/CoT/blocksworld/prompts/pool_prompt_v1.json') as f:
    prompt = json.load(f)

evaluator = BWEvaluator(config_file='examples/CoT/blocksworld/data/bw_config
                        domain_file='examples/CoT/blocksworld/data/generated
                        data_path='examples/CoT/blocksworld/data/split_v1/sp
                        init_prompt=prompt,
                        output_extractor=bfs_bw_extractor)

evaluator.evaluate(reasoner_tot, shuffle_prompt=True, num_shot=4, resume=0)
```

```
blocksworld:   0%|          | 0/84 [00:00<?, ?it/s]/bin/sh: 1: None/validat
e: not found
blocksworld:   1%|          | 1/84 [03:29<4:49:15, 209.11s/it]
```

```
[+]: Saving plan in tmp_plan.txt
RESPONSE:::
Case #1: correct=False, output='unstack the orange block from on top of the
red block\nstack the orange block on top of the blue block', answer={'init':
'the blue block is clear, the hand is empty, the red block is on top of the
yellow block, the blue block is on top of the orange block, the orange block
is on top of the red block and the yellow block is on the table', 'goal': 't
he orange block is on top of the blue block', 'plan': '\nunstack the blue bl
ock from on top of the orange block\nput down the blue block\nunstack the or
ange block from on top of the red block\nstack the orange block on top of th
e blue block\n[PLAN END]\n', 'question': '\n[STATEMENT]\nAs initial conditio
ns I have that, the blue block is clear, the hand is empty, the red block is
on top of the yellow block, the blue block is on top of the orange block, th
e orange block is on top of the red block and the yellow block is on the tab
le.\nMy goal is to have that the orange block is on top of the blue block.\n
\nMy plan is as follows:\n\n[PLAN]\n', 'instance_file': 'LLMs-Planning/llm_p
lanning_analysis/instances/blocksworld/generated_basic/instance-176.pddl'};a
ccuracy=0.000 (0/1)
```

```
/bin/sh: 1: None/validate: not found
blocksworld:   2%|▏         | 2/84 [06:01<4:00:08, 175.72s/it]
```

```
[+]: Saving plan in tmp_plan.txt
RESPONSE:::
Case #2: correct=False, output='pick up the blue block\nstack the blue block
on top of the red block', answer={'init': 'the blue block is clear, the oran
ge block is clear, the hand is empty, the orange block is on top of the red
block, the red block is on the table and the blue block is on the table', 'g
oal': 'the red block is on top of the blue block', 'plan': '\nunstack the or
ange block from on top of the red block\nput down the orange block\npick up
the red block\nstack the red block on top of the blue block\n[PLAN END]\n',
'question': '\n[STATEMENT]\nAs initial conditions I have that, the blue bloc
k is clear, the orange block is clear, the hand is empty, the orange block i
s on top of the red block, the red block is on the table and the blue block
is on the table.\nMy goal is to have that the red block is on top of the blu
e block.\n\nMy plan is as follows:\n\n[PLAN]\n', 'instance_file': 'LLMs-Plan
ning/llm_planning_analysis/instances/blocksworld/generated_basic_3/instance-
52.pddl'};accuracy=0.000 (0/2)
```

```
/bin/sh: 1: None/validate: not found
blocksworld:   4%|▍         | 3/84 [09:21<4:12:13, 186.84s/it]
```

```
[+]: Saving plan in tmp_plan.txt
RESPONSE:::
Case #3: correct=False, output='pick up the orange block\nstack the orange b
lock on top of the blue block\npick up the yellow block\nstack the yellow bl
ock on top of the orange block', answer={'init': 'the blue block is clear, t
he orange block is clear, the yellow block is clear, the hand is empty, the
orange block is on top of the red block, the red block is on the table, the
blue block is on the table and the yellow block is on the table', 'goal': 't
he blue block is on top of the orange block and the orange block is on top o
f the yellow block', 'plan': '\nunstack the orange block from on top of the
red block\nstack the orange block on top of the yellow block\npick up the bl
ue block\nstack the blue block on top of the orange block\n[PLAN END]\n', 'q
uestion': '\n[STATEMENT]\nAs initial conditions I have that, the blue block
is clear, the orange block is clear, the yellow block is clear, the hand is
empty, the orange block is on top of the red block, the red block is on the
table, the blue block is on the table and the yellow block is on the tabl
e.\nMy goal is to have that the blue block is on top of the orange block and
the orange block is on top of the yellow block.\n\nMy plan is as follows:\n
\n[PLAN]\n', 'instance_file': 'LLMs-Planning/llm_planning_analysis/instance
s/blocksworld/generated_basic/instance-301.pddl'};accuracy=0.000 (0/3)
```

```
/bin/sh: 1: None/validate: not found
blocksworld:   5%|▌         | 4/84 [11:27<3:37:12, 162.91s/it]
```

```
[+]: Saving plan in tmp_plan.txt
RESPONSE:::
Case #4: correct=False, output='unstack the blue block from on top of the ye
llow block\nunstack the yellow block from on top of the red block\nstack the
yellow block on top of the orange block', answer={'init': 'the blue block is
clear, the orange block is clear, the hand is empty, the blue block is on to
p of the yellow block, the yellow block is on top of the red block, the red
block is on the table and the orange block is on the table', 'goal': 'the ye
llow block is on top of the orange block', 'plan': '\nunstack the blue block
from on top of the yellow block\nput down the blue block\nunstack the yellow
block from on top of the red block\nstack the yellow block on top of the ora
nge block\n[PLAN END]\n', 'question': '\n[STATEMENT]\nAs initial conditions
I have that, the blue block is clear, the orange block is clear, the hand is
empty, the blue block is on top of the yellow block, the yellow block is on
top of the red block, the red block is on the table and the orange block is
on the table.\nMy goal is to have that the yellow block is on top of the ora
nge block.\n\nMy plan is as follows:\n\n[PLAN]\n', 'instance_file': 'LLMs-Pl
anning/llm_planning_analysis/instances/blocksworld/generated_basic/instance-
388.pddl'};accuracy=0.000 (0/4)
```

```
/bin/sh: 1: None/validate: not found
blocksworld:   6%|▌         | 5/84 [14:55<3:55:47, 179.09s/it]
```