

NL2TLf translation for restraining bolts application in BabyAI environment

Kaszuba Sara, 1695639

Postolache Emilian, 1649271

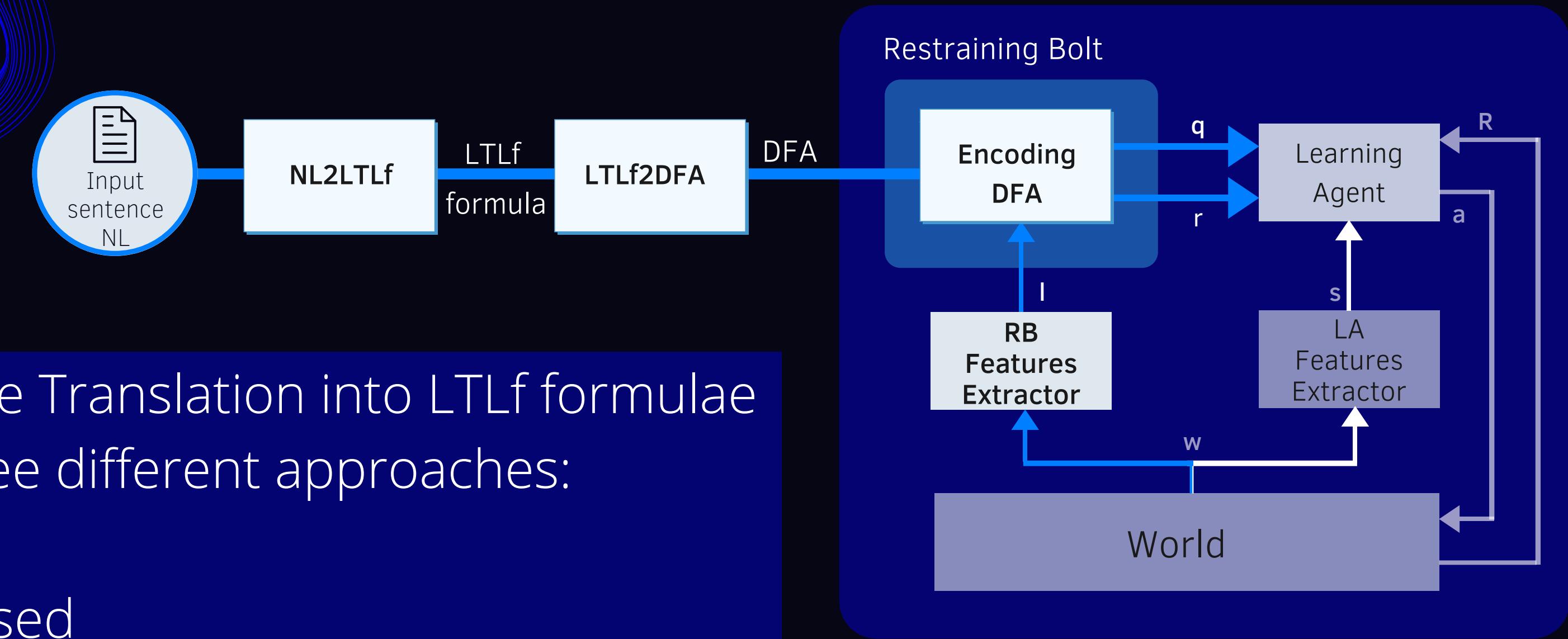
Ratini Riccardo, 1656801

Simionato Giada, 1822614

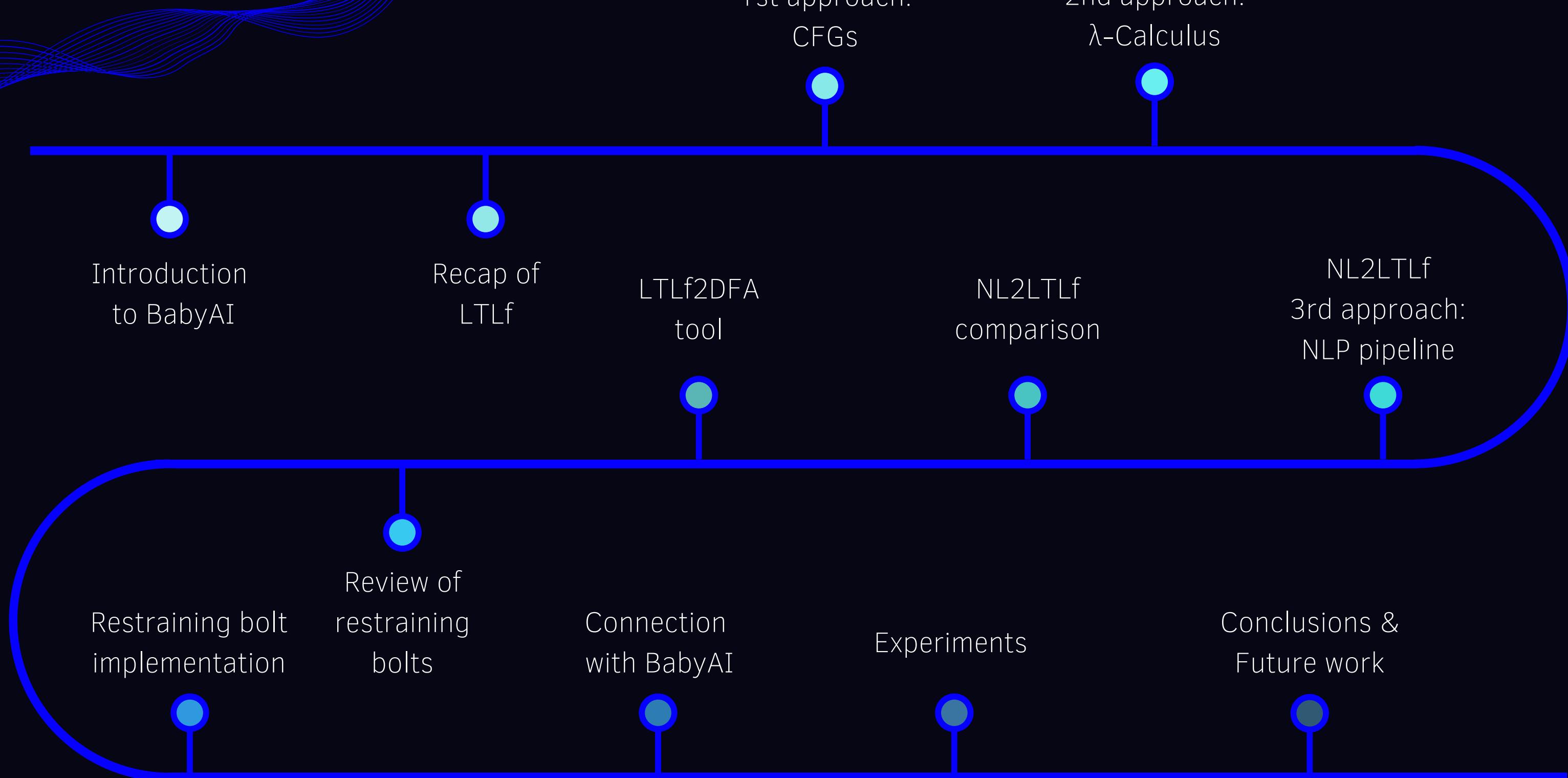
Our work

- Natural Language Translation into LTLf formulae according to three different approaches:
 - CFG Based
 - λ -Calculus Based
 - NLP Based

- Restraining Bolt implementation within BabyAI environment



Overview



BabyAI

AN INTRODUCTION

*Description of bot functioning in Appendix 1

POWERFUL RESEARCH PLATFORM

Facilitates research on grounded language learning as to include human in the loop.

SUPPORTS CURRICULUM LEARNING

Levels of gradually increasing difficulty to boost performances in Human-Machine teaching.

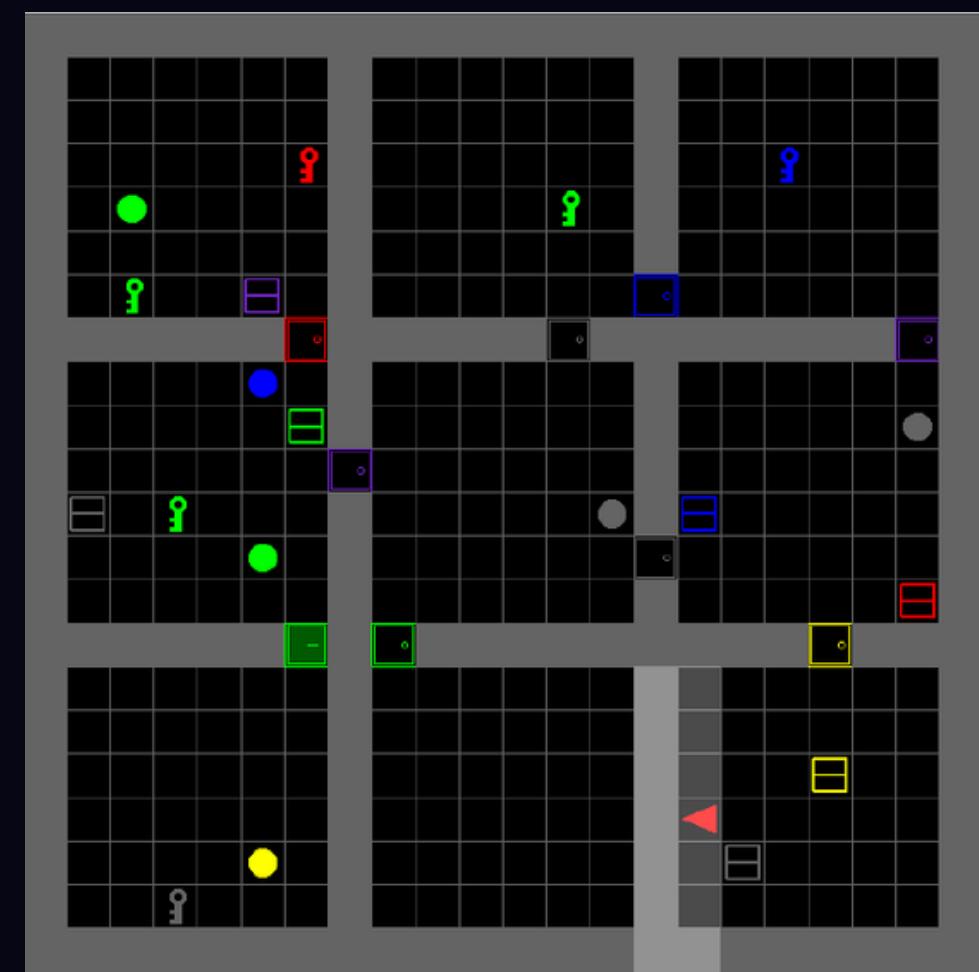
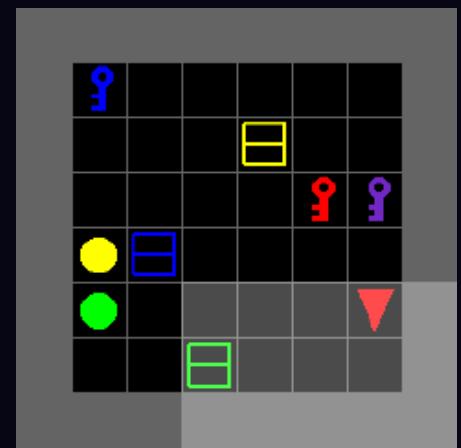
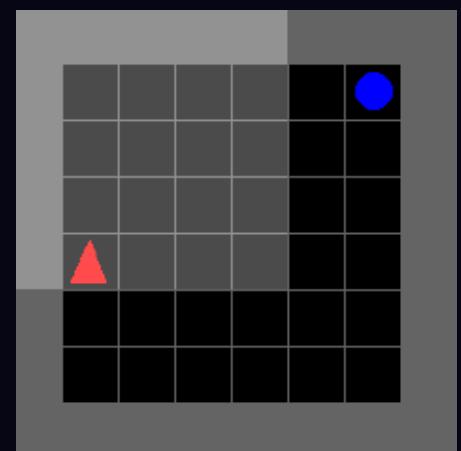
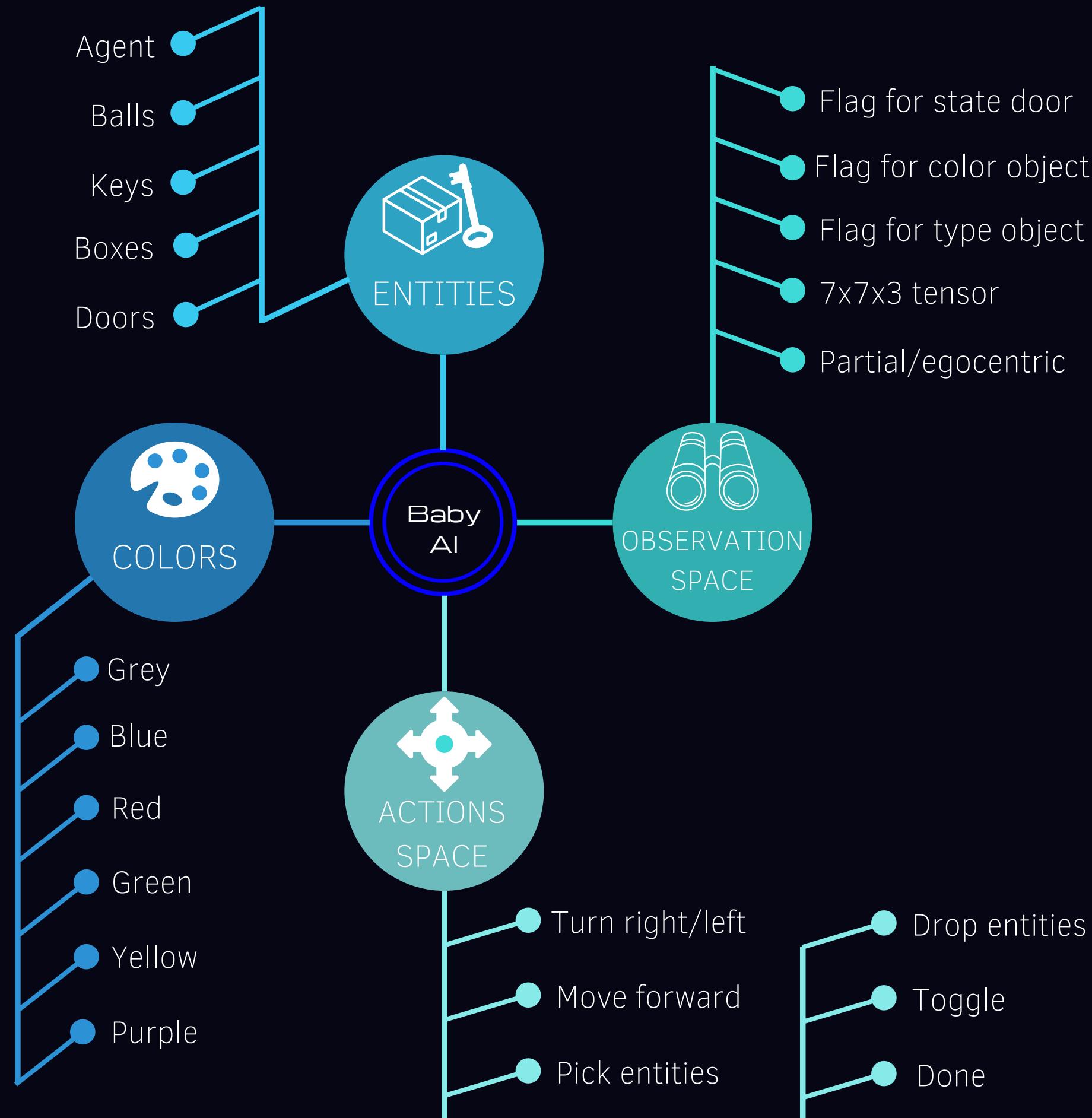
SUPPORTS INTERACTIVE TEACHING

Tailored teaching based on agent's abilities currently achieved.

BOT-BASED HUMAN IMITATION

Generation of demonstrations and action suggestions for each level, to assist the agent in learning.*

BabyAI Structure



BabyAI levels with MiniGrid environment

Baby Language

- Synthetic language with known context-free grammar
- Verifier for the language: check satisfiability of the instructions w.r.t. the environment
- Not all instructions are explicit

BNF of the grammar of the Baby Language

$\langle \text{Sent} \rangle \vdash \langle \text{Sent1} \rangle \mid \langle \text{Sent1} \rangle ; ; \text{ then } \langle \text{Sent1} \rangle \mid \langle \text{Sent1} \rangle \text{ after you } \langle \text{Sent1} \rangle$
$\langle \text{Sent1} \rangle \vdash \langle \text{Clause} \rangle \mid \langle \text{Clause} \rangle \text{ and } \langle \text{Clause} \rangle$
$\langle \text{Clause} \rangle \vdash \text{go to } \langle \text{Descr} \rangle \mid \text{pick up } \langle \text{DescrNotDoor} \rangle \mid \text{open } \langle \text{DescrDoor} \rangle \mid \text{put } \langle \text{DescrNotDoor} \rangle \text{ next to } \langle \text{Descr} \rangle$
$\langle \text{DescrDoor} \rangle \vdash \langle \text{Article} \rangle \langle \text{Color} \rangle \text{ door } \langle \text{LocSpec} \rangle$
$\langle \text{DescrBall} \rangle \vdash \langle \text{Article} \rangle \langle \text{Color} \rangle \text{ ball } \langle \text{LocSpec} \rangle$
$\langle \text{DescrBox} \rangle \vdash \langle \text{Article} \rangle \langle \text{Color} \rangle \text{ box } \langle \text{LocSpec} \rangle$
$\langle \text{DescrKey} \rangle \vdash \langle \text{Article} \rangle \langle \text{Color} \rangle \text{ key } \langle \text{LocSpec} \rangle$
$\langle \text{Descr} \rangle \vdash \langle \text{DescrDoor} \rangle \mid \langle \text{DescrBall} \rangle \mid \langle \text{DescrBox} \rangle \mid \langle \text{DescrKey} \rangle$
$\langle \text{DescrNotDoor} \rangle \vdash \langle \text{DescrBall} \rangle \mid \langle \text{DescrBox} \rangle \mid \langle \text{DescrKey} \rangle$
$\langle \text{LocSpec} \rangle \vdash \epsilon \mid \text{on your left} \mid \text{on your right} \mid \text{in front of you} \mid \text{behind you}$
$\langle \text{Color} \rangle \vdash \epsilon \mid \text{red} \mid \text{green} \mid \text{blue} \mid \text{purple} \mid \text{yellow} \mid \text{grey}$
$\langle \text{Article} \rangle \vdash \text{the} \mid \text{a}$

Examples of instructions in Baby Language

"go to the red ball"

"put a ball next to the blue door"

"put a ball next to a purple door after you put a blue box next to a grey box and pick up the purple key"

BabyAI Levels

- Suite of 19 levels (extendable)
- Varying environment structure and difficulty of instruction
- Each level is a distribution of missions
- Each level requires a set of competencies*

Levels and their competencies

	ROOM	DISTR-BOX	DISTR	MAZE	UNBLOCK	UNLOCK	IMP-UNLOCK	GOTO	OPEN	PICKUP	PUT	LOC	SEQ
GoToObj	x												
GoToRedBallGrey	x	x											
GoToRedBall	x	x	x										
GoToLocal	x	x	x										
PutNextLocal	x	x	x										
PickupLoc	x	x	x										
GoToObjMaze	x			x									
GoTo	x	x	x	x									
Pickup	x	x	x	x									
UnblockPickup	x	x	x	x	x								
Open	x	x	x	x					x				
Unlock	x	x	x	x		x			x				
PutNext	x	x	x	x							x		
Synth	x	x	x	x	x	x		x	x	x	x		
SynthLoc	x	x	x	x	x	x		x	x	x	x	x	
GoToSeq	x	x	x	x				x				x	
SynthSeq	x	x	x	x	x	x		x	x	x	x	x	x
GoToImpUnlock	x	x	x	x			x	x					
BossLevel	x	x	x	x	x	x	x	x	x	x	x	x	x

*Description of the competencies in Appendix 2

LTLf

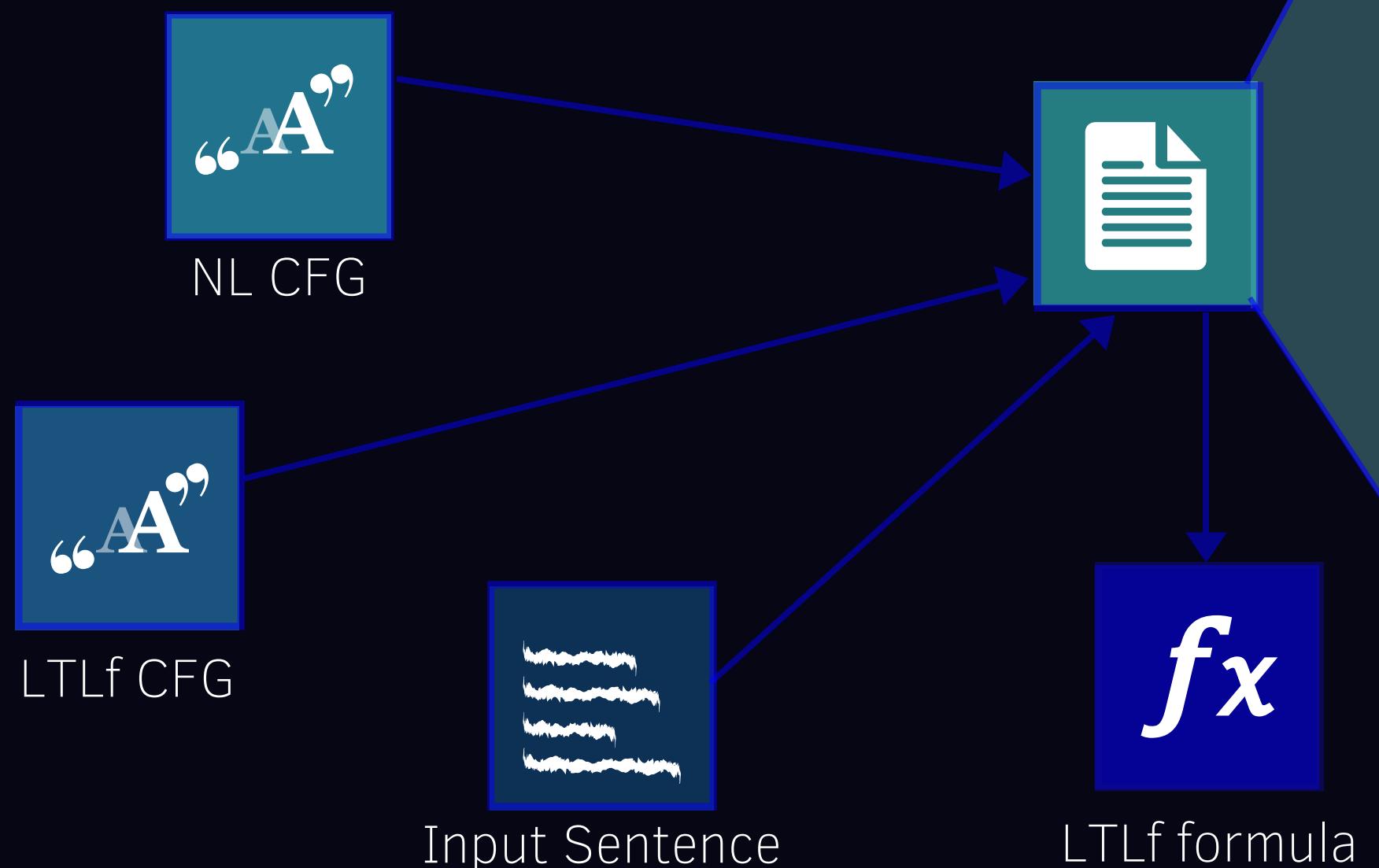
- Linear Temporal Logic over finite traces
- Used to express the goal of the restraining bolt
- Atomic propositions model world features
- Temporal operators and connectives for goal composition
-

$\varphi, \psi \rightarrow$	$p \mid \in \Sigma$	(atomic proposition)
$\top \mid$		(True)
$\perp \mid$		(False)
$\neg\varphi \mid$		(complement)
$\varphi \wedge \psi \mid$		(conjunction)
$\varphi \vee \psi \mid$		(disjunction)
$\bigcirc\varphi \mid$		(next)
$\varphi \mathcal{U} \psi \mid$		(until)
$\neg\lozenge\neg\varphi \mid$	$\lozenge\varphi \mid$	(eventually)
$\neg\bigcirc\neg\varphi \mid$	$\Box\varphi \mid$	(always)
$\neg\bigcirc True \mid$	$\bullet\varphi \mid$	(weak next)
	$Last \mid$	(last)

$$\varphi \models A | \neg\varphi | \varphi_1 \wedge \varphi_2 | \bigcirc \varphi | \varphi_1 \mathcal{U} \varphi_2$$

CFG2LTLf

- Matches structure of parsed tree against LTLf CFG
- CGF of input sentence must be known
- CFG of LTLf must mirror structure NL CFG



CHECK
CONSISTENCY

SENTENCE
PREPROCESSING

PARSING
WITH NL CFG

MATCHING TREE
AGAINST LTLf CFG

POSTPROCESSING
LTLf FORMULA

GROUNDING

CFG2LTLf (cont'd)

Input sentence:

"Pick up the green ball and put a box next to a key, then open the door."

LTLf (not grounded):

$$\Diamond((\text{pick(green_ball)} \wedge \text{put_next_to(box, key)}) \wedge \Diamond\text{open(door)})$$

LTLf (grounded):

$$\Diamond((\text{pick_green_ball} \wedge \text{put_next_to_box_key}) \wedge \Diamond\text{open_door})$$

```

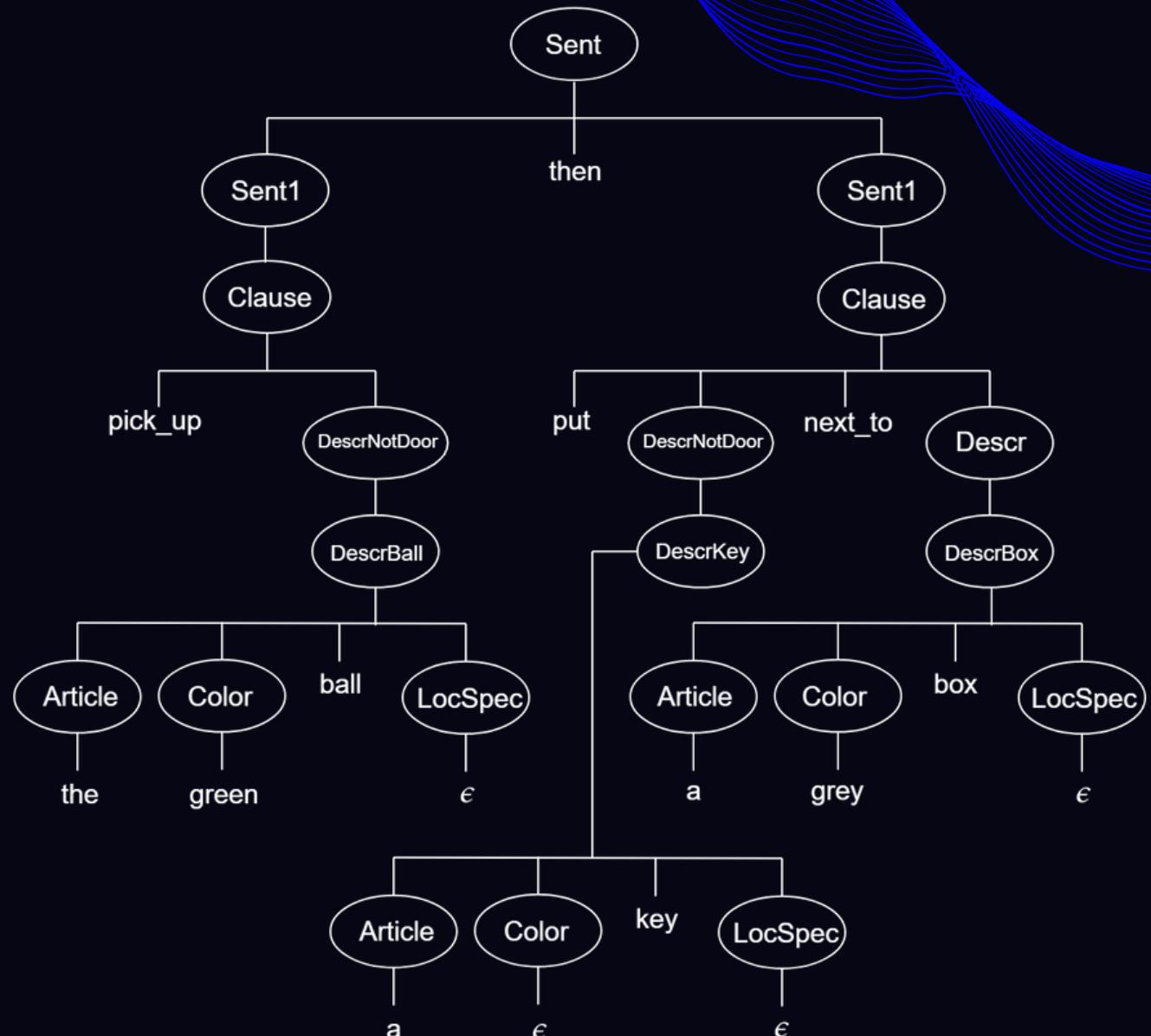
⟨Sent⟩ ⊢ ⟨Sent1⟩ | ⟨Sent1⟩ ; then ⟨Sent1⟩ | ⟨Sent1⟩ after you ⟨Sent1⟩
⟨Sent1⟩ ⊢ ⟨Clause⟩ | ⟨Clause⟩ and ⟨Clause⟩
⟨Clause⟩ ⊢ go to ⟨Descr⟩ | pick up ⟨DescrNotDoor⟩ | open ⟨DescrDoor⟩ | put ⟨DescrNotDoor⟩ next to ⟨Descr⟩
⟨DescrDoor⟩ ⊢ ⟨Article⟩ ⟨Color⟩ door ⟨LocSpec⟩
⟨DescrBall⟩ ⊢ ⟨Article⟩ ⟨Color⟩ ball ⟨LocSpec⟩
⟨DescrBox⟩ ⊢ ⟨Article⟩ ⟨Color⟩ box ⟨LocSpec⟩
⟨DescrKey⟩ ⊢ ⟨Article⟩ ⟨Color⟩ key ⟨LocSpec⟩
⟨Descr⟩ ⊢ ⟨DescrDoor⟩ | ⟨DescrBall⟩ | ⟨DescrBox⟩ | ⟨DescrKey⟩
⟨DescrNotDoor⟩ ⊢ ⟨DescrBall⟩ | ⟨DescrBox⟩ | ⟨DescrKey⟩
⟨LocSpec⟩ ⊢ ε | on your left | on your right | in front of you | behind you
⟨Color⟩ ⊢ ε | red | green | blue | purple | yellow | grey
⟨Article⟩ ⊢ the | a
  
```

NL grammar

```

⟨Sent⟩ ⊢ ⟨Sent1⟩ | ∆(⟨Sent1⟩ ∧ ∆⟨Sent1⟩) | ∆(∆⟨Sent1⟩ ∧ ⟨Sent1⟩)
⟨Sent1⟩ ⊢ ⟨Clause⟩ | (⟨Clause⟩ ∧ ⟨Clause⟩)
⟨Clause⟩ ⊢ at(⟨Descr⟩) | pick(⟨DescrNotDoor⟩) | open(⟨DescrDoor⟩) | put_next_to(⟨DescrNotDoor⟩, ⟨Descr⟩)
⟨DescrDoor⟩ ⊢ ⟨Article⟩⟨Color⟩door⟨LocSpec⟩
⟨DescrBall⟩ ⊢ ⟨Article⟩⟨Color⟩ball⟨LocSpec⟩
⟨DescrBox⟩ ⊢ ⟨Article⟩⟨Color⟩box⟨LocSpec⟩
⟨DescrKey⟩ ⊢ ⟨Article⟩⟨Color⟩key⟨LocSpec⟩
⟨Descr⟩ ⊢ ⟨DescrDoor⟩ | ⟨DescrBall⟩ | ⟨DescrBox⟩ | ⟨DescrKey⟩
⟨DescrNotDoor⟩ ⊢ ⟨DescrBall⟩ | ⟨DescrBox⟩ | ⟨DescrKey⟩
⟨LocSpec⟩ ⊢ _onyourleft_|_onyourright_|_infrontofyou_|_behindyou_
⟨Color⟩ ⊢ red_|green_|blue_|purple_|yellow_|grey_|
⟨Article⟩ ⊢ _|_
  
```

LTLf grammar

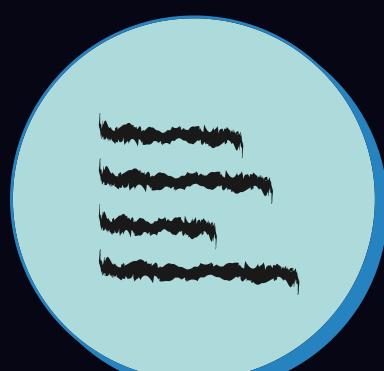


Tree corresponding to the input sentence parsed with NL CFG

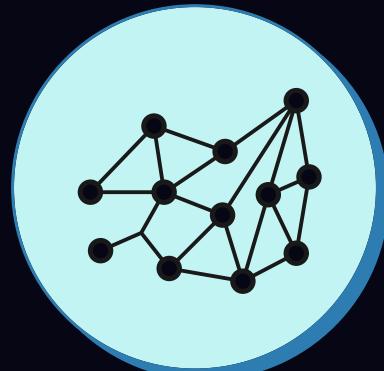
λ Calc2LTLf

- Inspired by [4]
- Based on Lambda Calculus
- Relies on the assumptions:
 - each word in the sentence belongs to the lexicon
 - sentences follow certain structure
- Handles semantic ambiguities
- Supports pronouns handling

REQUIRE:



Input
Sentence
NL



Mapping
words -
abstractions

BUILD DICTIONARY OF ABSTRACTIONS

SENTENCE PREPROCESSING

PARSING & ABSTRACTIONS SELECTION

β -REDUCTION

FORMULA POSTPROCESSING & GROUNDING

λ Calc2LTLf (cont'd)

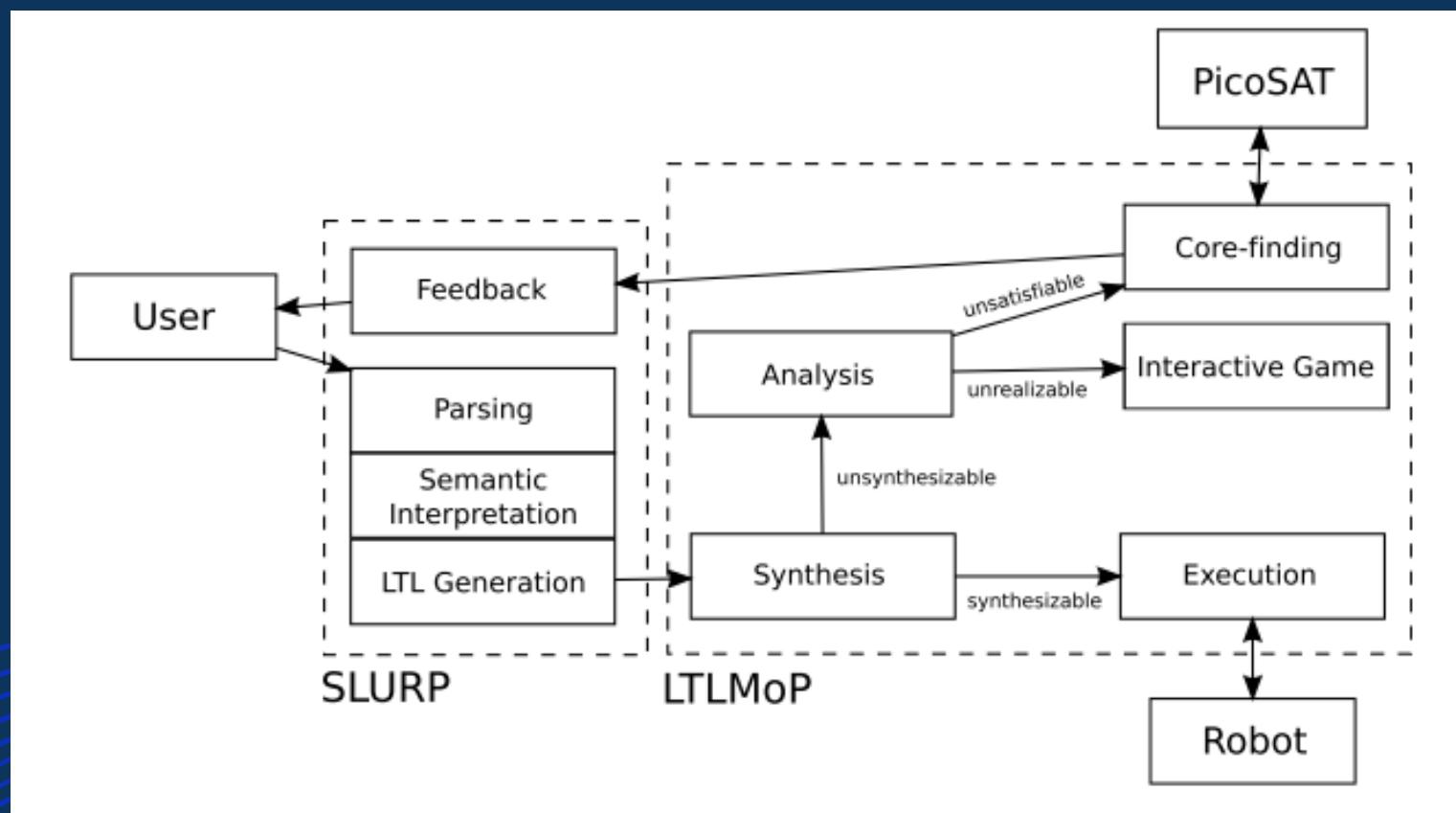
INPUT SENTENCE:	
<hr/>	
<i>"Go to room 1 and pick up the blue box."</i>	
$\lambda x.at(x)$	$\lambda x.x@room_1$
$at(room_1)$	$\lambda x\lambda y.\Diamond(x \wedge \Diamond y)$
	$\lambda x.pick(x)$
	$\lambda x.x$
	$\lambda x.x@blue_box$
<hr/>	<hr/>
	$\lambda x\lambda y.\Diamond(x \wedge \Diamond y)$
	$\lambda x.pick(x)$
	<i>blue_box</i>
<hr/>	<hr/>
not grounded:	$\Diamond(at(room_1) \wedge \Diamond pick(blue_box))$
	
grounded:	$\Diamond(at_room_1 \wedge \Diamond pick_blue_box)$

MAPPING:

Word	LT-abstraction
go to	$\lambda x.at(x)$
the	$\lambda x.x$
and	$\lambda x\lambda y.\Diamond(x \wedge \Diamond y)$
of	$\lambda x.x$
when	$\lambda x\lambda y.\Box(x \rightarrow y)$
open	$\lambda x.open(x)$
do not	$\lambda x.\neg x$
eventually	$\lambda x.\Diamond x$
pick up	$\lambda x.pick(x)$
or	$\lambda x\lambda y.x \vee y$
until	$\lambda x\lambda y.x \mathcal{U} y$
green ball	$\lambda x.x@green_ball$
red key	$\lambda x.x@red_key$
door	$\lambda x.x@door$

NL2LTLf with NLP

Reference paper [5]



INPUT SENTENCE

LEMMATIZATION

LTL TREE GENERATION

NLP PIPELINE

FORUMULA RECONSTRUCITON

LTL FORMULA

Resources

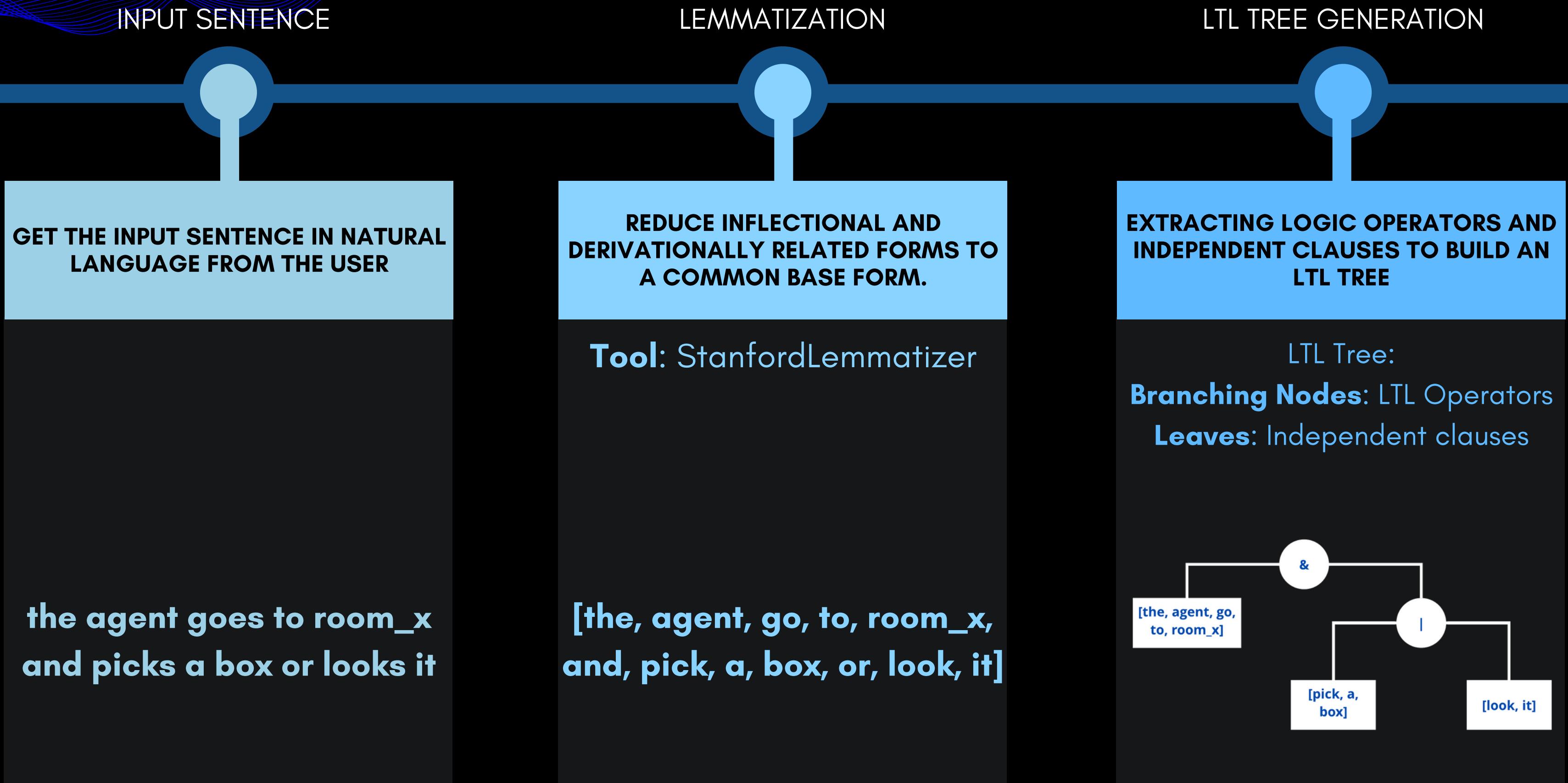
- Stanford CoreNLP
- WordNet
- VerbNet

Frame **Go-02**
Subject: The Agent
Verb: Go
To: the kitchen

Frame **Go-03**
Subject: The Agent
Verb: Go
Through: the hallway

Frame **Go-04**
Subject: The Agent
Verb: Go
From: the hallway
To: the kitchen

NL2LTLf: pt1



NLP Pipeline: pt1

NLP PIPELINE

POS Tagging

ASSIGNING TO EACH WORD CORRESPONDING PART OF SPEECH (POS), ACCORDING TO THE CONTEXT.

Tool: Stanford POS Tagger

- 1) [the/DT, agent/NN, go/VB, to/IN, room_x/NN]
- 2) [pick/VB, a/DT, box/NN]
- 3) [look/VB, it/PRP]

Null-Element Restoration

ADDING THE SUBJECT IN THE SENTENCES IN WHICH IT IS IMPLICIT.

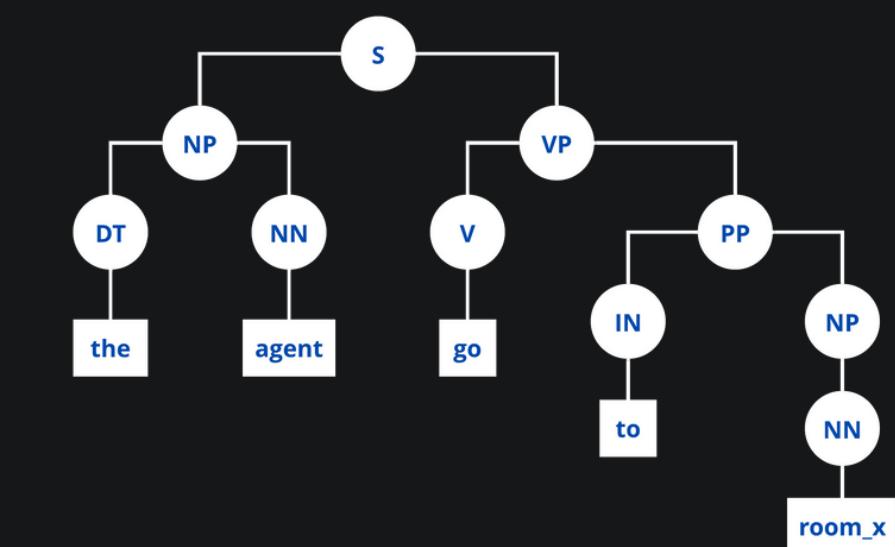
Tool: Stanford Dependency Parser

- 1) already contains a subject
- 2) [the/DT, agent/NN, pick/VB, a/DT, box/NN]
- 3) [the/DT, agent/NN, look/VB, it/PRP]

Syntactic Parsing

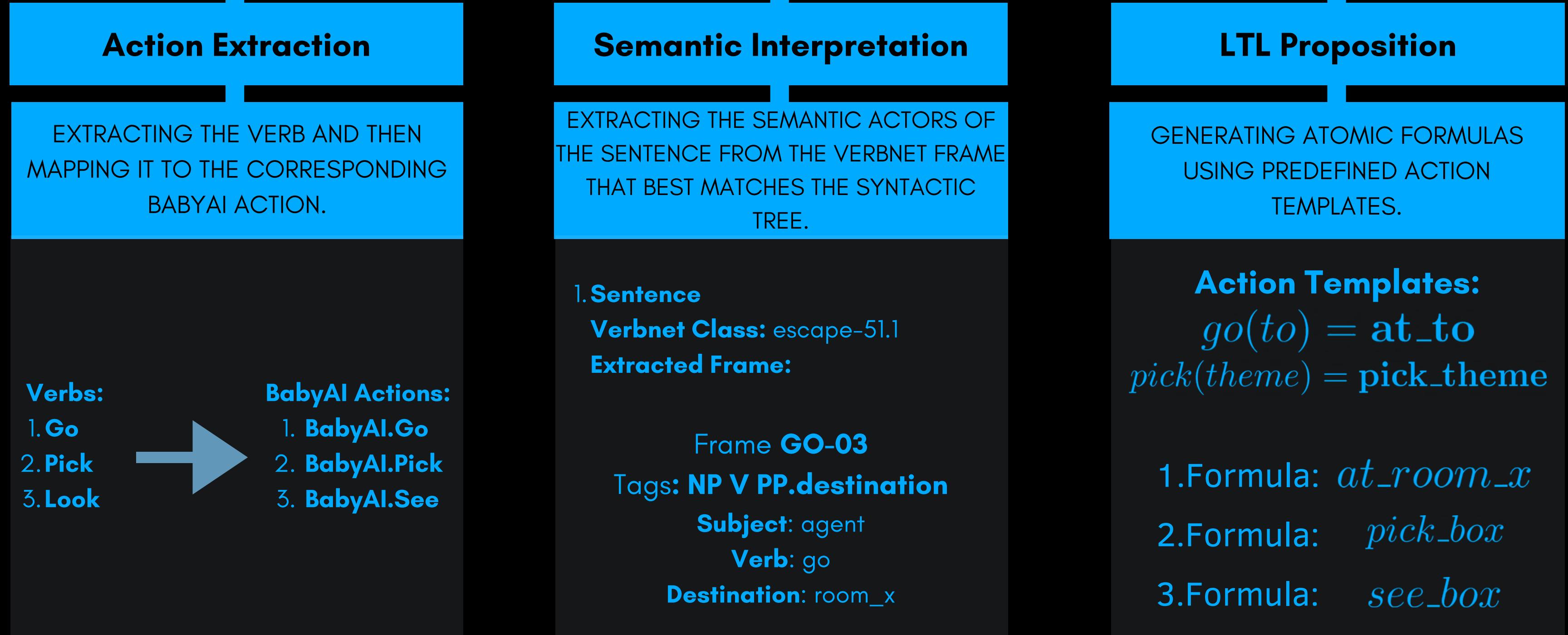
BUILDING A COSTITUENCY-BASED PARSING TREE.

Tool: Stanford Syntactic Parser

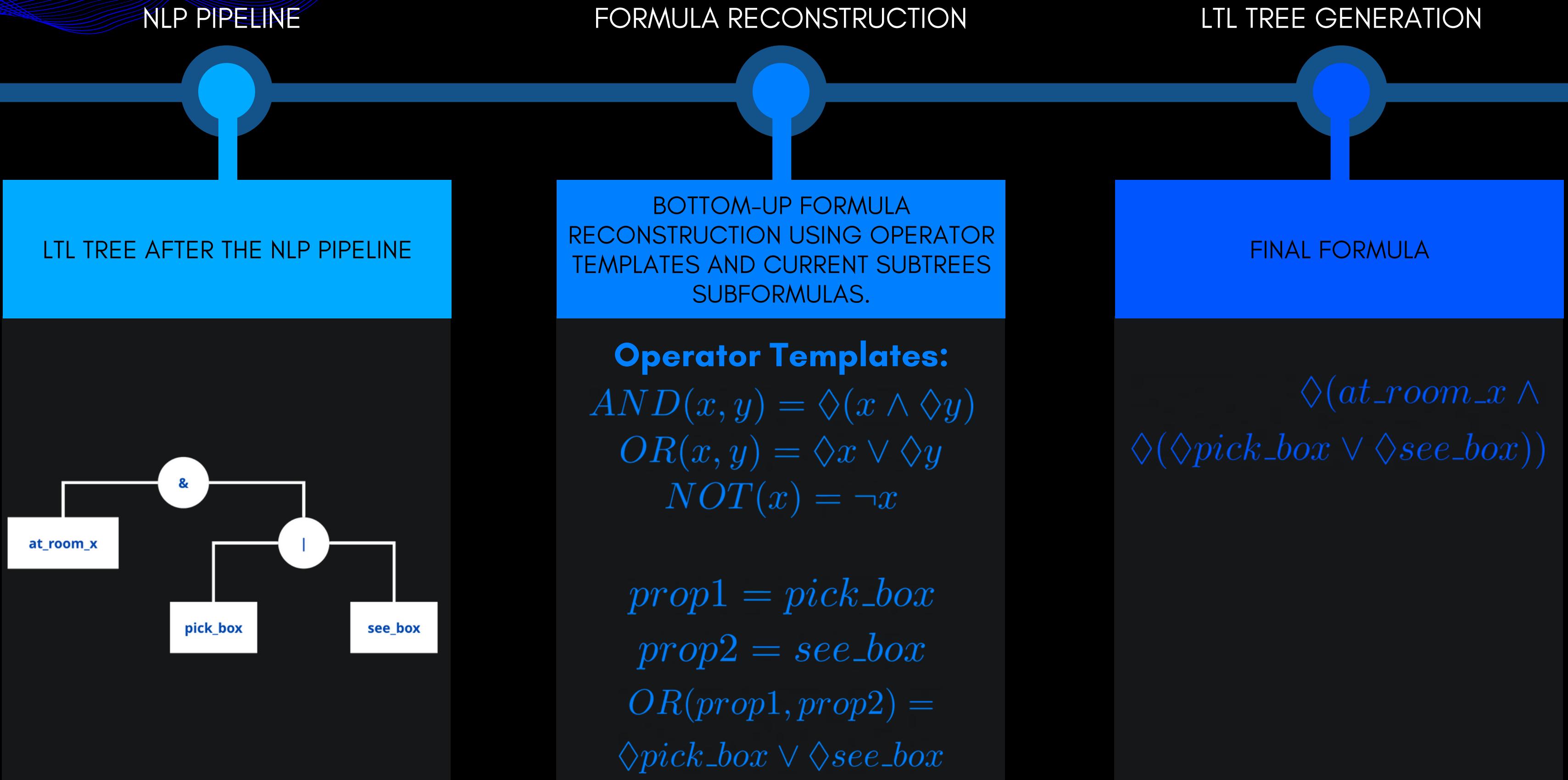


NLP Pipeline: pt2

NLP PIPELINE (Cont'd)



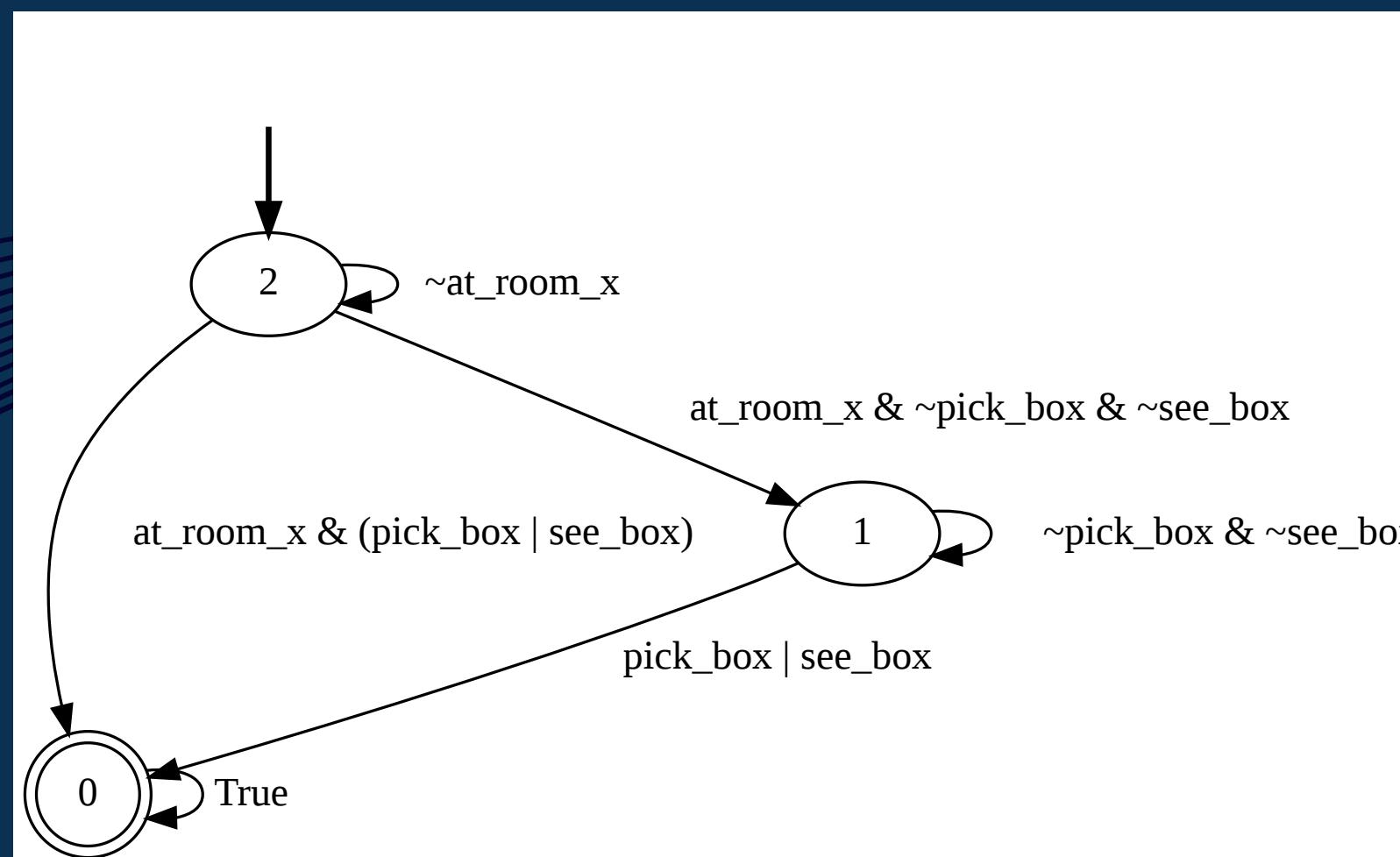
NL2LTLf: pt2



Comparison

	Independent from structure sentence	Accepts non lemmatized words	Pronouns handling	No knowledge CFG required	Autonomy	Upgradeability	Portability	Time	Overall performance
CFG Based									
λ -Calculus Based									
NLP Based									

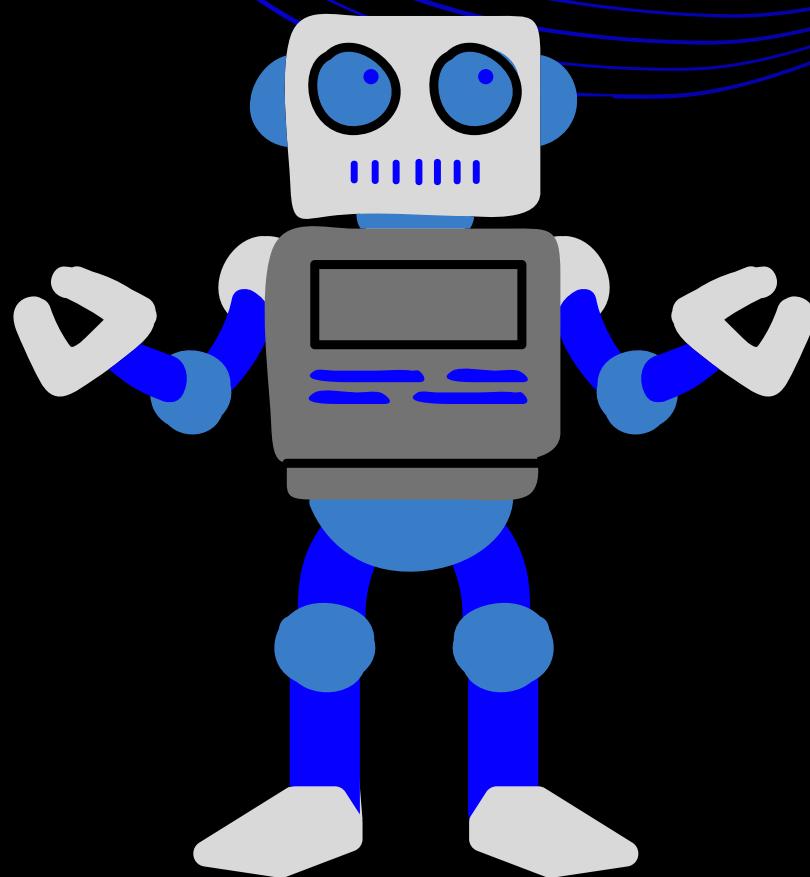
Online tool for Automata generation



LTLf Syntax

Syntax	Semantics	Example
$[A-Za-z0-9]^+$	Atomic	$A, b01$
true false	True	true
! f	Not	$\neg A$
$f_1 \wedge \dots \wedge f_n$	And, Or	$A \wedge (B \vee \neg C)$
$f_1 \rightarrow f_2$	Implication	$G(A \rightarrow B)$
$f_1 \leftrightarrow f_2$	Equivalence	$F(A \leftrightarrow B)$
$X f_1$	Next	$X A$
$WX f_1$	Weak Next	$WX A$
$f_1 U f_2$	Until	$A U B$
$f_1 R f_2$	Release	$A R B$
$F f_1$	Eventually	$F A$
$G f_1$	Always	$G A$

Restraining Bolt



"It is a device that restricts a droid's [agent's] actions when connected to its systems. Droid owners install restraining bolts to limit actions to a set of desired behaviors." [3]

Representations of the world

A
G
E
N
T

Features

Actions

Reward



WORLD

Fluents

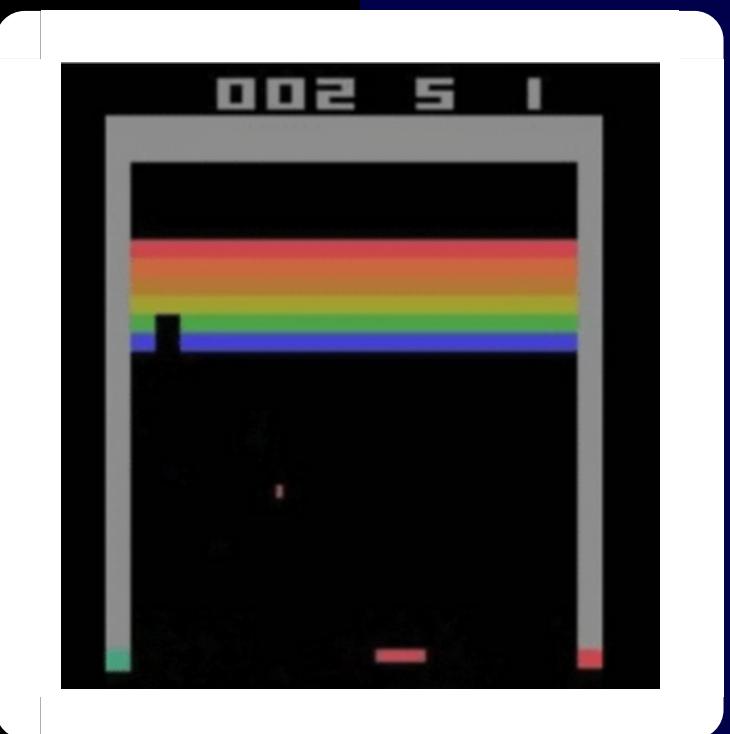
Restraining
specification

B
O
L
T

Representations of the world

Learning Agent

- **Features:** paddle and ball positions, ball speed
- **Actions:** move the paddle (left or right)
- **Rewards:** after hitting a brick

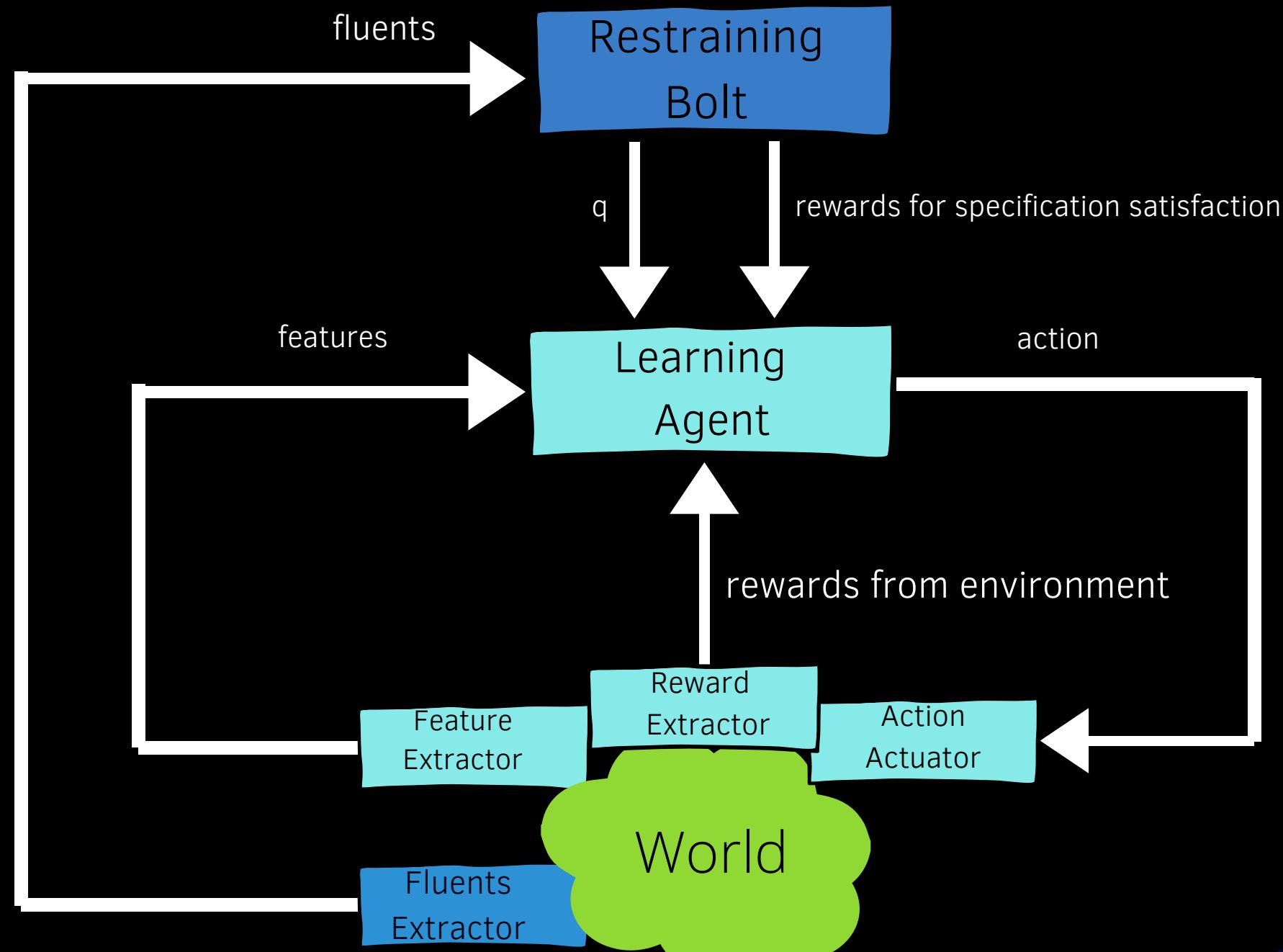


Example Breakout: Representations of the world

Restraining Bolt

- **Fluents:** bricks position and status of the columns (broken or not)
- **Restraining specification:** all bricks in column i must be removed before completing the column j , with $i < j$

RL with LTLf/LDLf restraining specifications



Learning Agent: it is defined as
 $M = (S, A, Tr, R)$

Restraining Bolt: it is formed by m LTLf/LDLf formulas φ_i over the fluents L with associated rewards r_i , it is defined as $RB = (L, \{(\varphi_i, r_i)\}_{i=1}^m)$

Goal: The learning agent tries to learn a non-Markovian policy that maximizes the expected cumulative reward, so $p : S^* \rightarrow A$

Main differences

Classical Reinforcement Learning

The Learning Agent is an MDP (Markov Decision Process) with unknown transitions and rewards, which takes actions in order to maximize the reward. The goal is to find a policy of the form: $p : S \rightarrow A$, so given a set of features, it tells us what action to perform.

RL for non-Markovian reward decision process with LTLf/LDLf rewards

In this case, the rewards are given over sequences of configuration of the feature in MDPs, working with an MDP with non-Markovian rewards. We can overcome the problem of non-Markovian reward by:

1. Taking the formula
2. Finding the corresponding DFA
3. Combining the DFAs with MDP, so to obtain a new system in which the policy to compute is in the form $p : S^* \rightarrow A$, that can be easily solved by working with an MDP with a larger set of states.

Solution for RL with LTLf/LDLf restraining specifications

STEP 1

Transform each formula φ_i into a DFA

$$A_{\varphi_i} = (2^L, Q_i, q_{i0}, \delta_i, F_i)$$

STEP 2

Apply classical Reinforcement Learning over a new MDP

$$M' = (Q_1 \times \dots \times Q_m \times S, A, Tr', R')$$

STEP 3

The optimal policy p' learned for M' is an optimal policy of the original problem

$$R'(q_1, \dots, q_m, s, a, q'_1, \dots, q'_m, s') = \sum_{i: q'_i \in F_i} r_i + R(s, a, s')$$

Restraining Bolt Implementation: File rb.py

Class RestrainingBolt:
superclass of each Restraining Bolt

```
class RestrainingBolt(ABC):
    def __init__(self, num_states, final_states,
                 initial_state=0, reward=1):
        self.num_states = num_states
        self.reward = reward
        self.initial_state = initial_state
        self.final_states = final_states
        self.reset()

    def reset(self):
        self.current_state = self.initial_state

    @abstractmethod
    def transition(self, world_state):
        pass

    def get_reward(self):
        if self.current_state in self.final_states:
            return self.reward
        else:
            return 0
```

Class FluentRestrainingBolt:
allows to manage fluents of the Restraining Bolt

```
class FluentRestrainingBolt(RestrainingBolt):
    # permette di avere tutto più automatizzato, ma è più inefficiente

    def __init__(self, num_states, final_states,
                 initial_state=0, reward=1):
        super().__init__(num_states, final_states, initial_state, reward)

    def transition(self, world_state):
        self.compute_fluents(world_state)
        self.compute_transition()

    @abstractmethod
    def compute_transition(self):
        pass

    @abstractmethod
    def compute_fluents(self, world_state):
        pass

    def reset(self):
        super().reset()
        self.current_fluents = set()
```

Class MultiBolt:
container of Restraining Bolts

```
class MultiBolt():
    def __init__(self, bolts, same_rewards=True):
        self.bolts = bolts
        self.same_rewards = same_rewards
        self.reset()

    def reset(self):
        for bolt in self.bolts:
            bolt.reset()

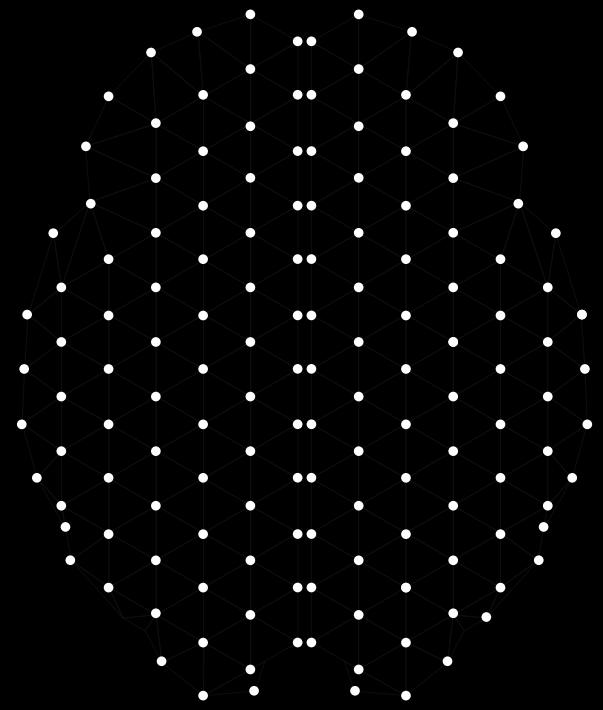
    def transition(self, world_state):
        for bolt in self.bolts:
            bolt.transition(world_state)

    def get_reward(self):
        reward = 0
        for bolt in self.bolts:
            r = bolt.get_reward()
            if self.same_rewards:
                reward += r * 1/len(self.bolts)
            else:
                reward += r
        return reward

    @property
    def current_state(self):
        state = self.bolts[0].current_state
        for bolt in self.bolts[1:]:
            state += bolt.current_state * bolt.num_states
        return state
```

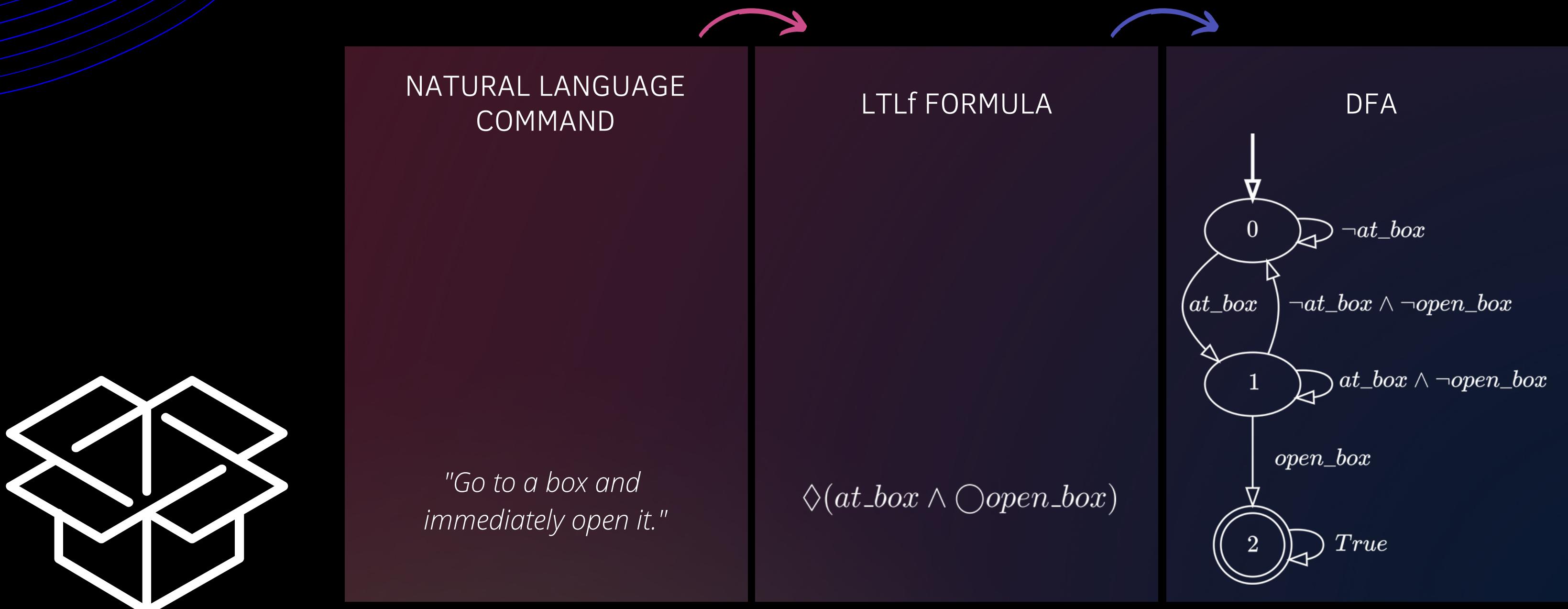
BabyAI integration

- Agent observations:
 - 7x7x3 tensors: the (partial) view of the agent inside the grid world
 - textual instruction
- During each episode parallel frames (generated by multiple Gym environments) are iterated by the PPO algorithm [10], employing an Actor Critic model.
- The Actor Critic maps the view to an embedding vector through a conditional CNN (using the textual instruction), that is mapped to a value scalar (critic) and to a distribution over actions (actor).
- We pass the observation to the bolt, perform a transition, then augment the embedding with the bolt state.
- BabyAI resets each environment when the goal of the textual instruction is reached or when a step limit is exceeded. Reward is proportional to the number of steps (0 if limit is reached, max 1). Bolts assign a reward of 1 if a final state is reached when the environment is reset.



Experiment 1

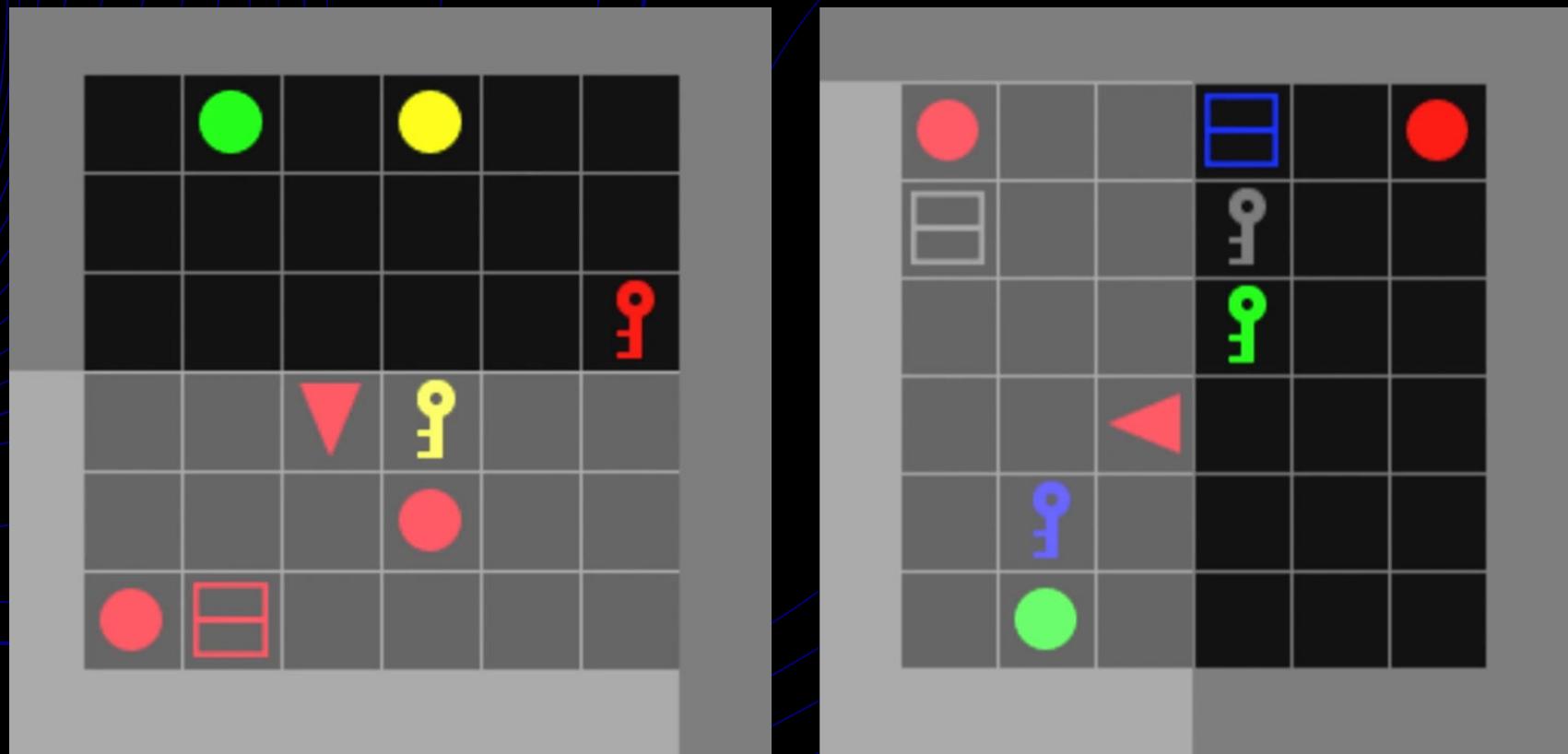
Open Box



Experiment 1

Open box

Demo



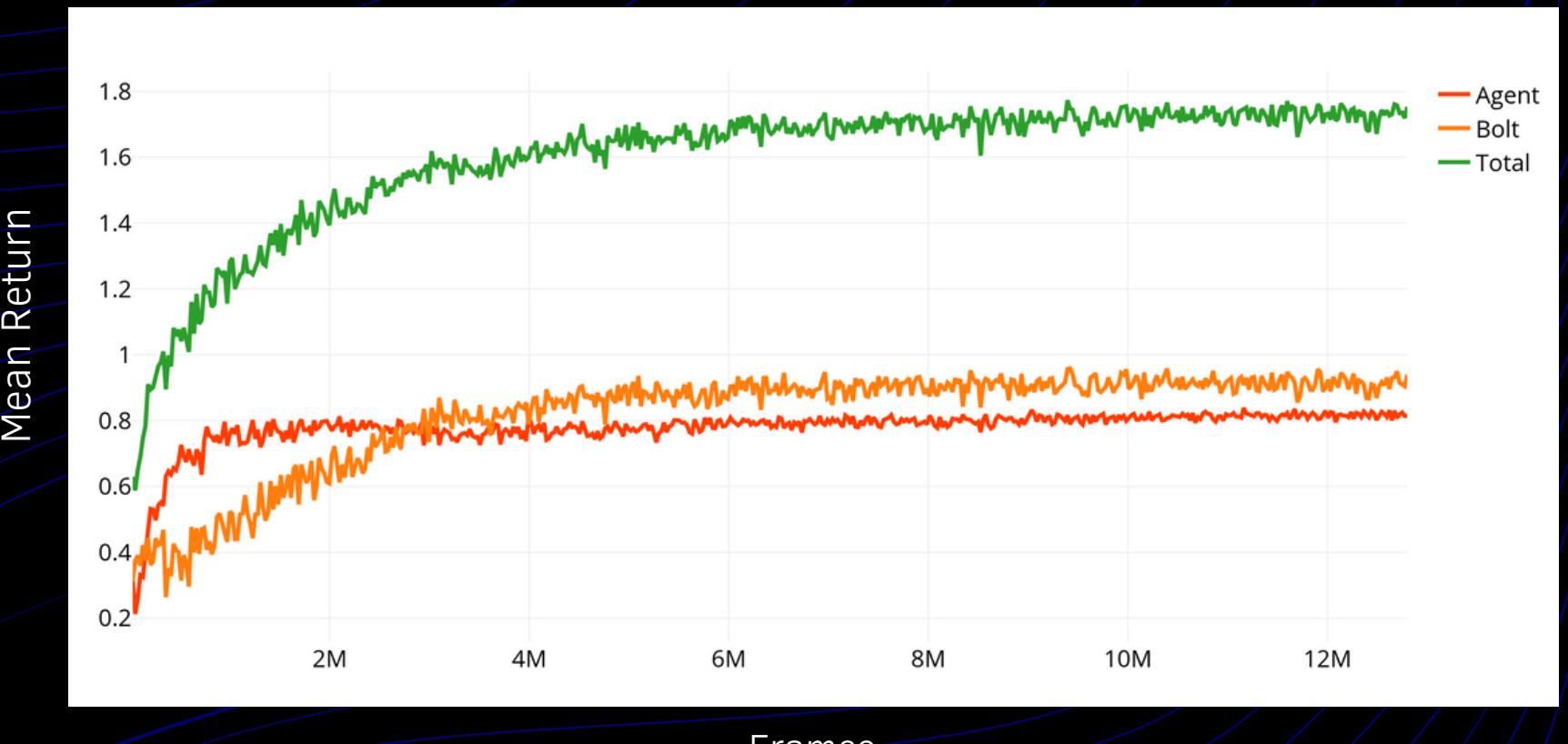
Vanilla agent

Goal: Go to the red ball

RB agent

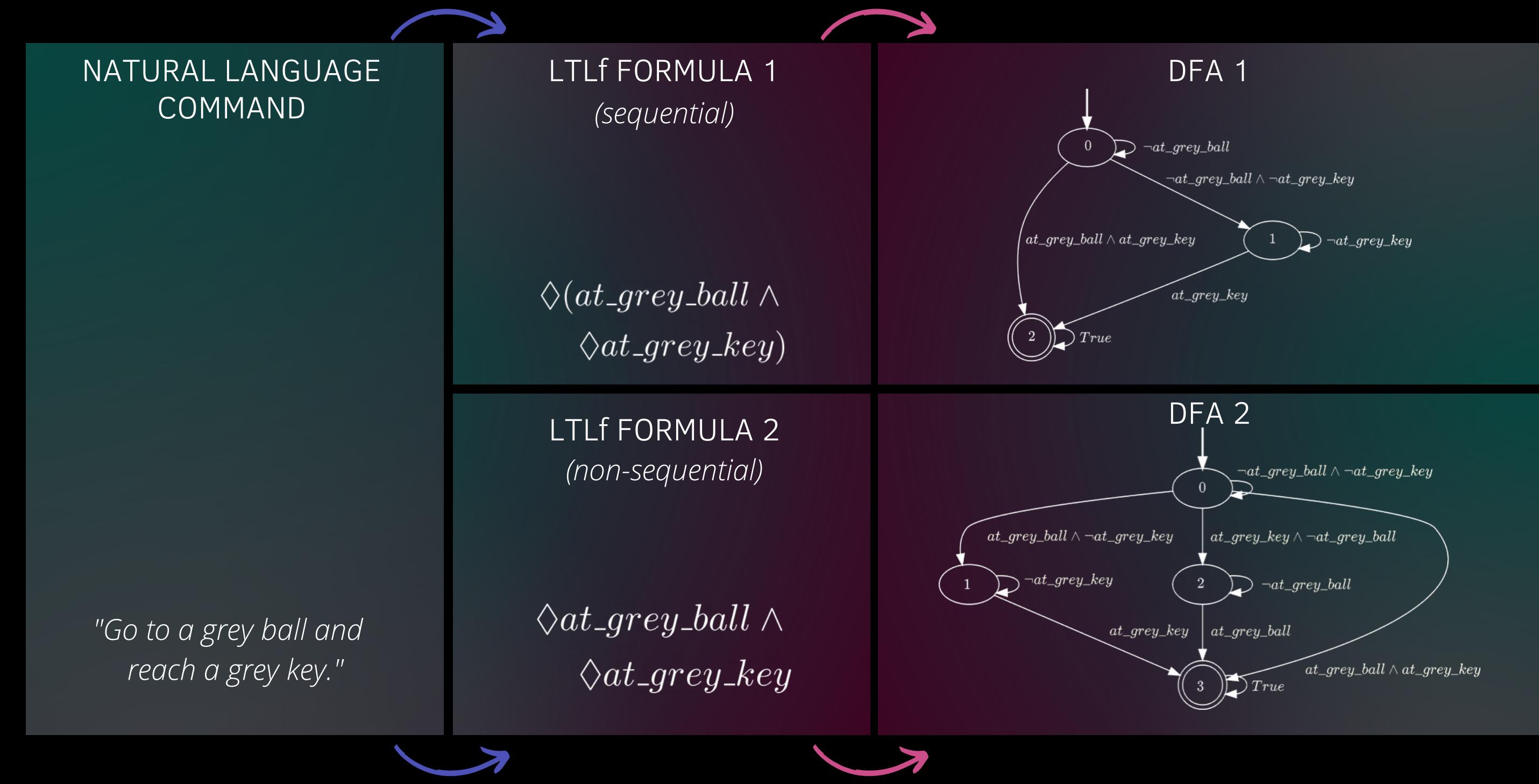
Goal: Go to the red ball +
**Go to a box and
immediately open it**

Mean Return



Experiment 2

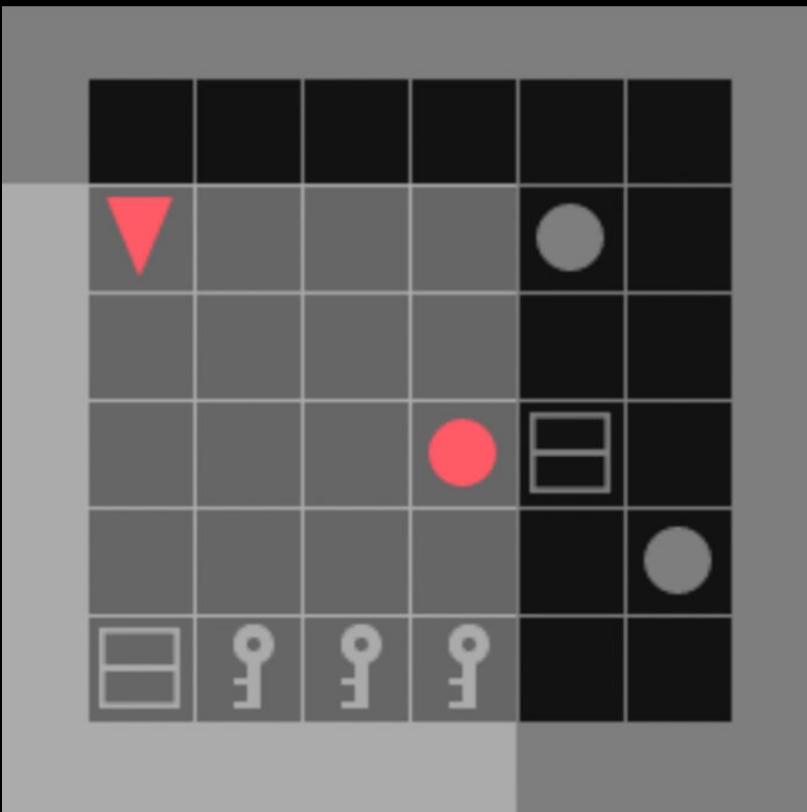
Ambiguity



Experiment 2

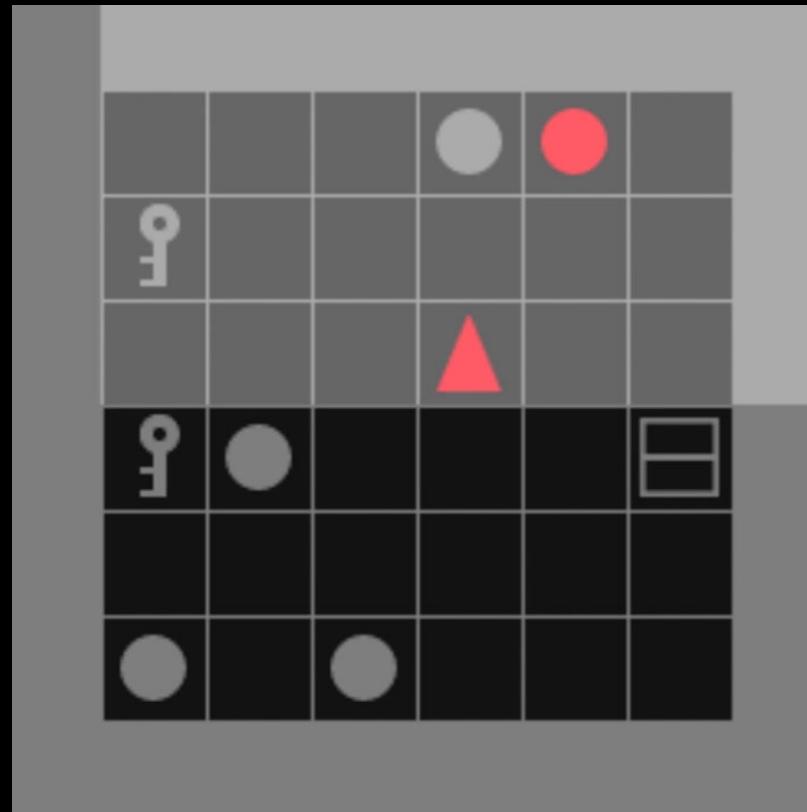
Ambiguity

Demo



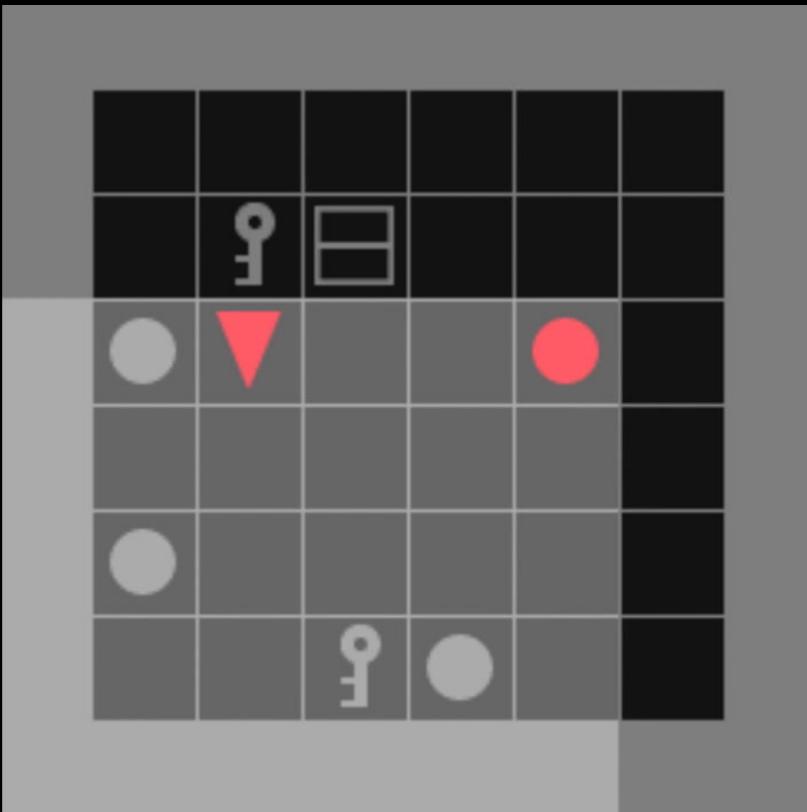
Vanilla agent

Goal: *Go to the red ball*



**RB agent
(non-sequential)**

Goal: *Go to the red ball +
Go to a grey ball **and** reach a grey key*

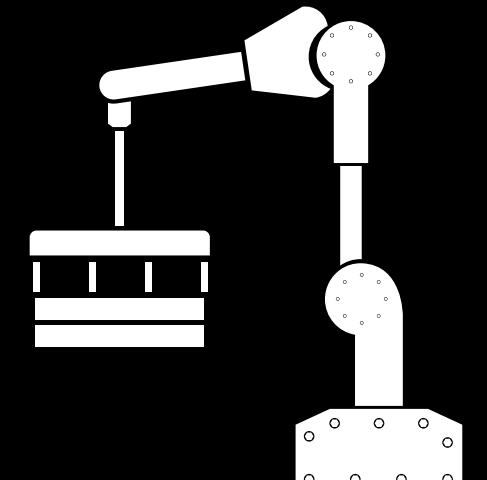
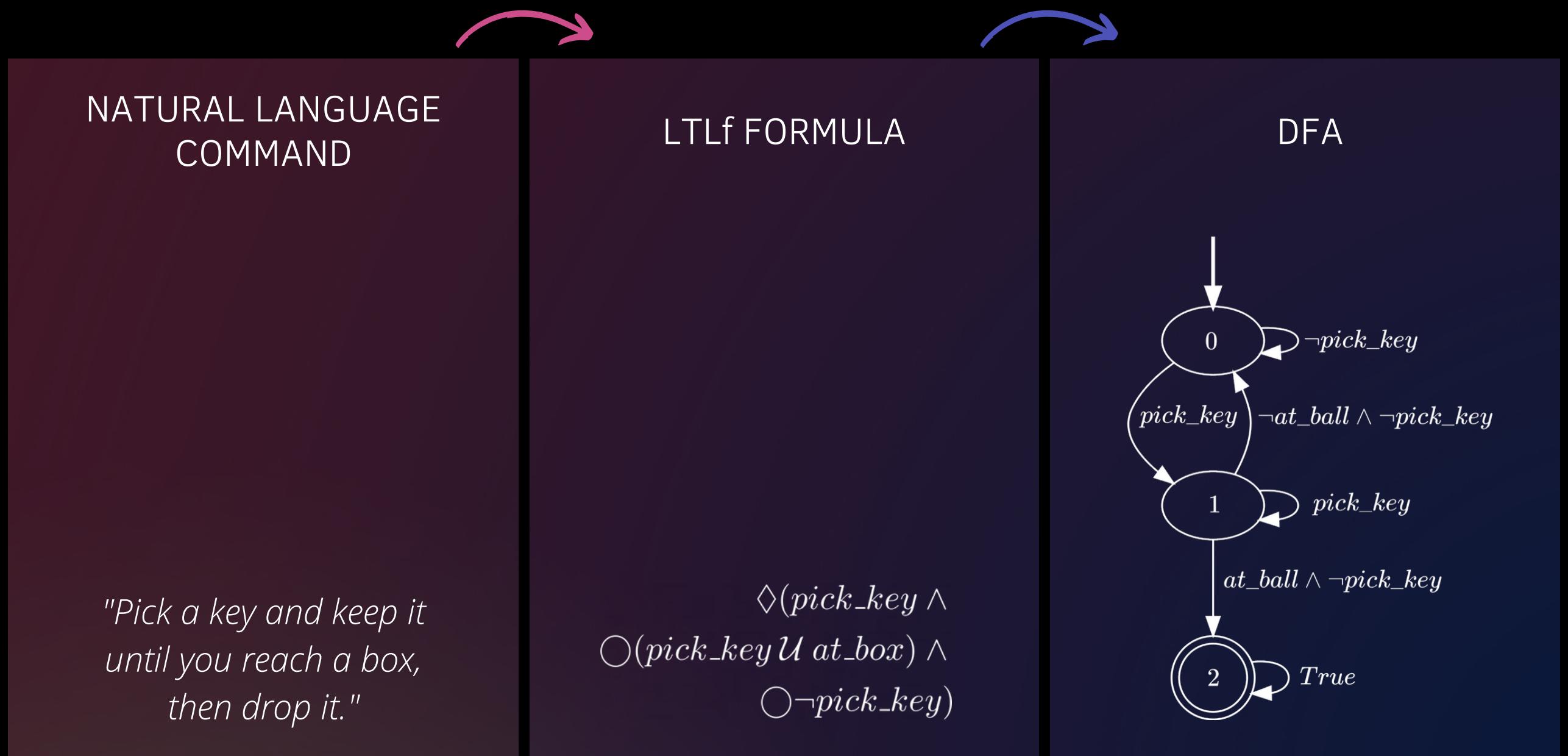


**RB agent
(sequential)**

Goal: *Go to the red ball +
Go to a grey ball **and (then)**
reach a grey key*

Experiment 3

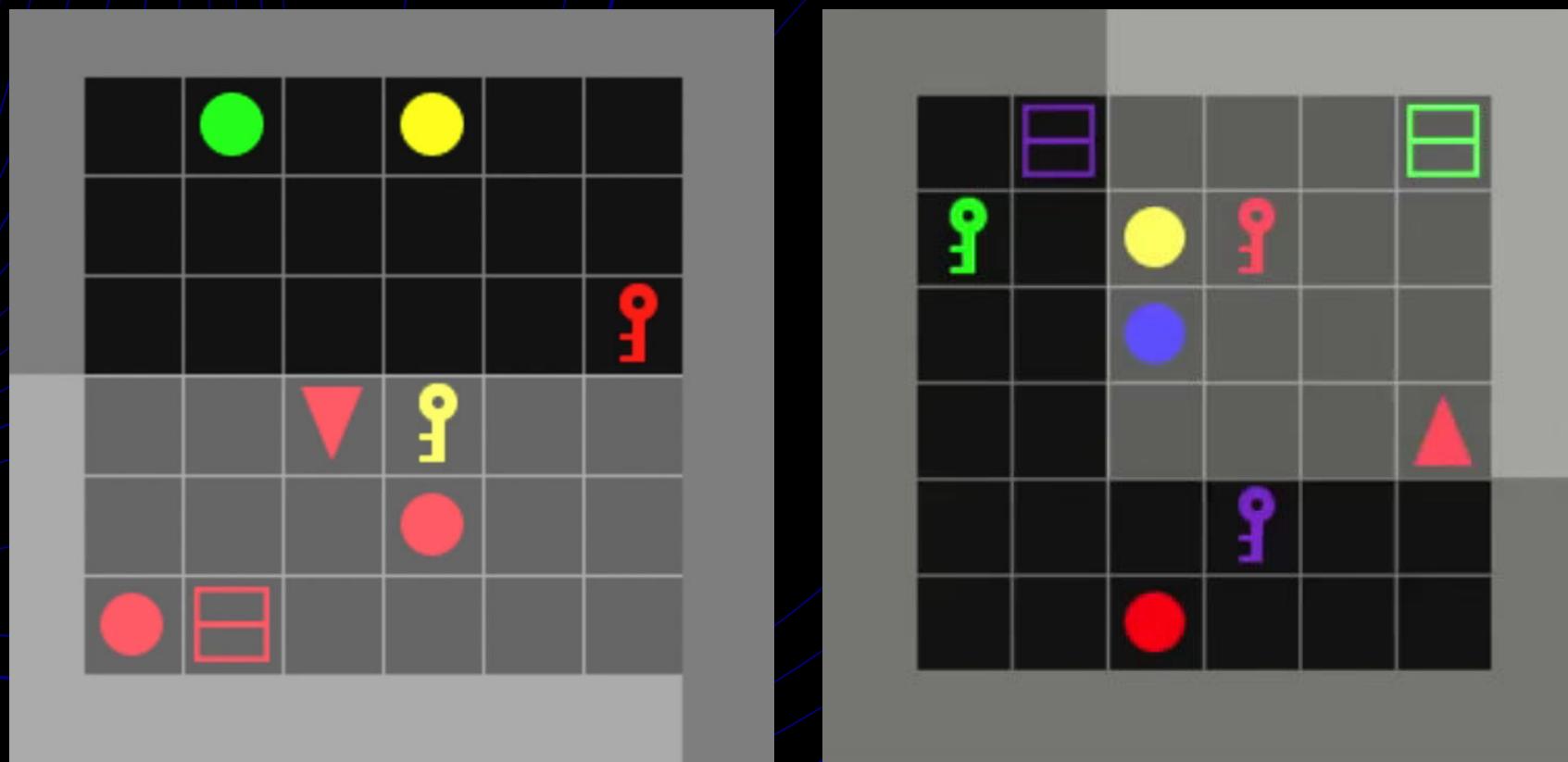
Pick and Place



Experiment 3

Pick and Place

Demo



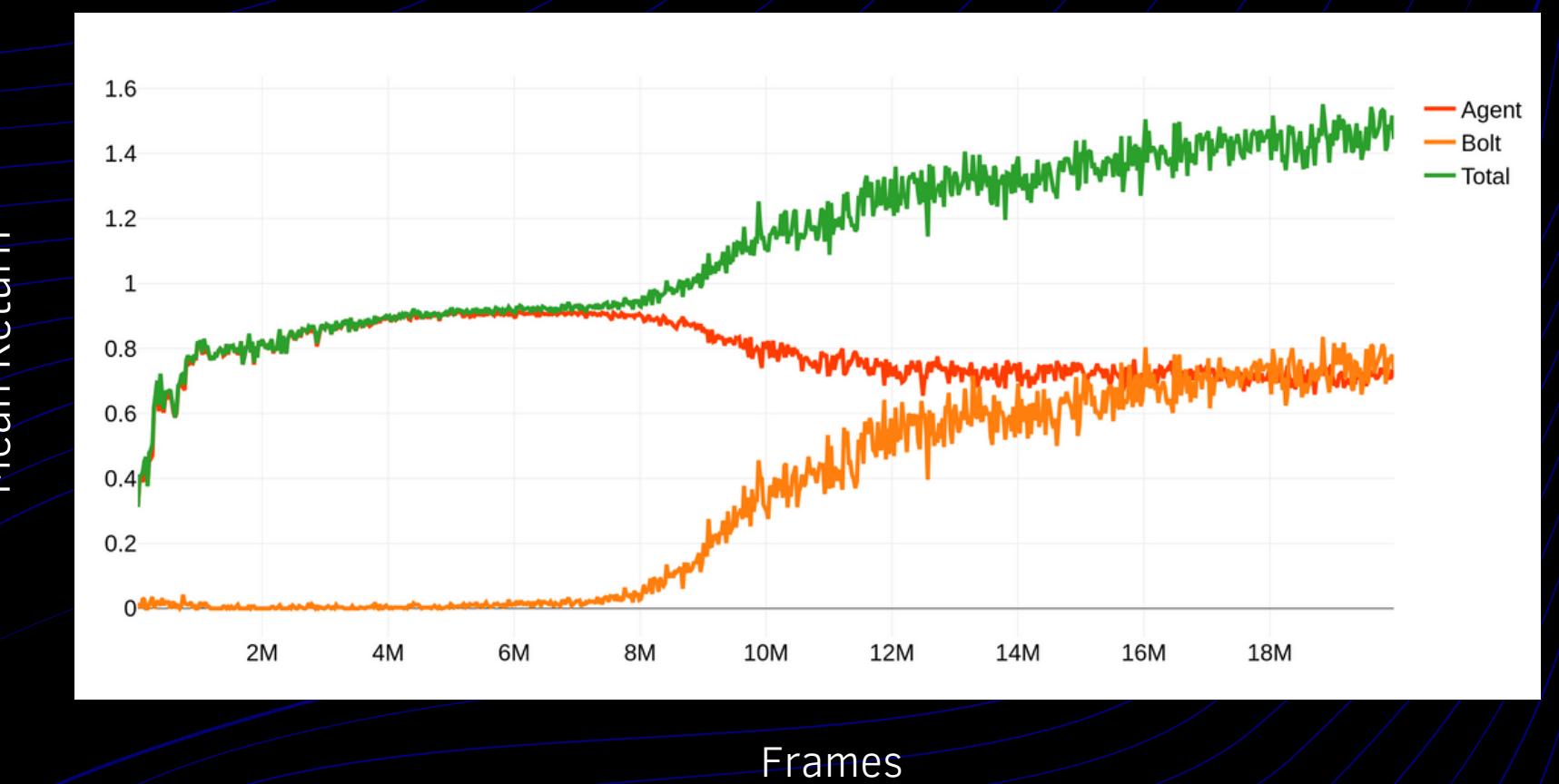
Vanilla agent

Goal: Go to the red ball

RB agent

Goal: Go to the red ball +
**Pick a key and keep it until you reach
a box, then drop it**

Mean Return



Conclusions

LANGUAGE TRANSLATION

We developed a modular system able to translate a sentence in NL into an LTLf formula using three different approaches.

RESTRAINING BOLTS INTEGRATION IN BABYAI

We explored the way in which the restraining bolts can be connected with the BabyAI platform and we validated the work with several experiments, obtaining interesting results.

Future work

ROBUSTNESS OF LANGUAGE TRANSLATION

Implementing a Deep Word Sense Disambiguation system to detect the correct verb sense and make the frame extraction easier and more flexible.

END-TO-END PIPELINE

Composing the translator, the automata tool and the Fluent Restraining Bolt, an end-to-end architecture is achievable. The only requirement is the set of fluents and the mapping between world states and fluents.

AUTOMATIC FLUENT DETECTION

The explicit mapping between world states and fluents could be delegated to a visual relation classifier. This is useful in continuous domains such as robotics, where the mapping is difficult to implement. More interestingly, this component would decouple a bolt not only from an underlying learning agent, but from an underlying environment too.

References

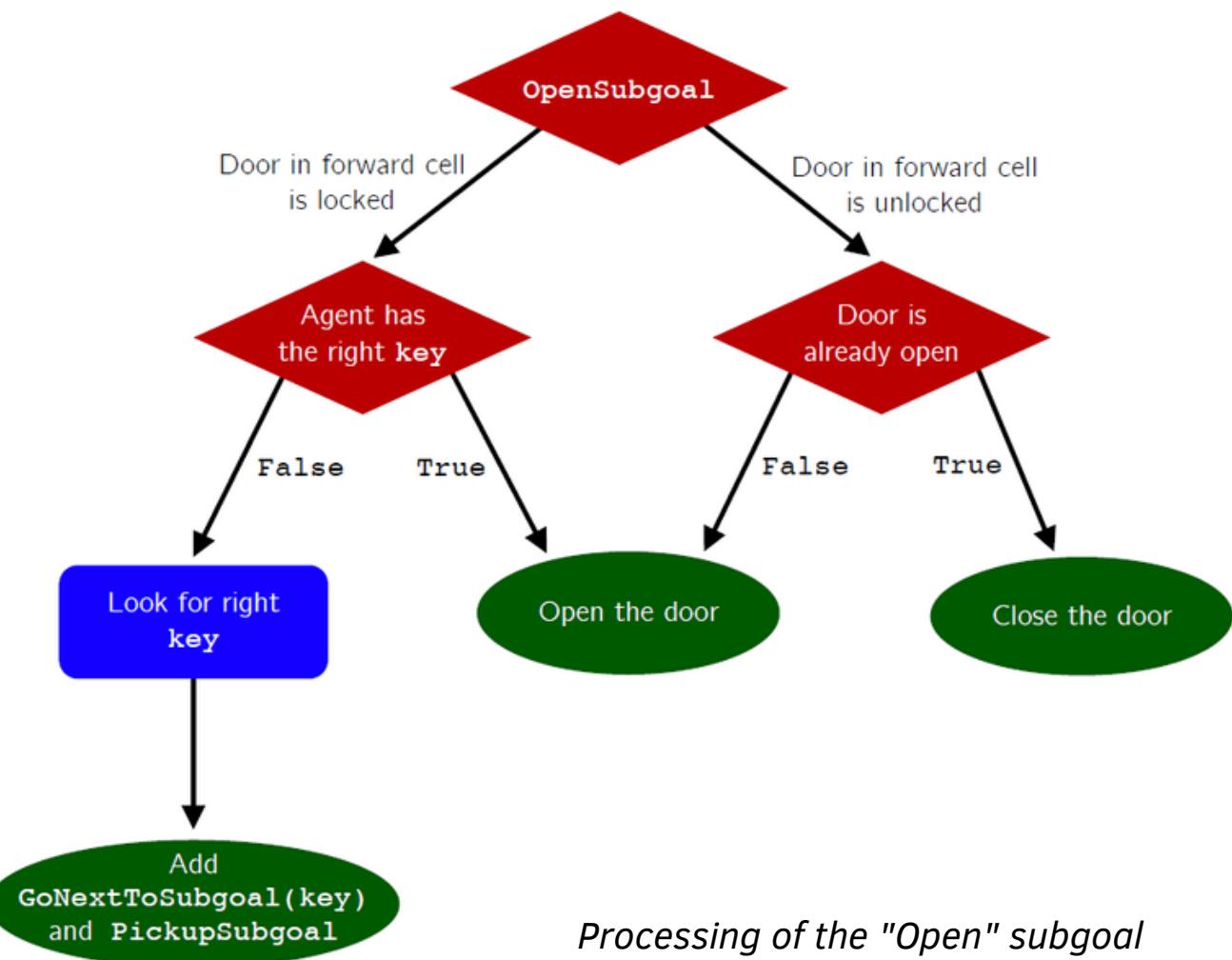
- [1] Chevalier-Boisvert, M., Bahdanau, D., Lahou, S., Willems, L., Saharia, C., Nguyen, T. H., & Bengio, Y. (2018, September). *BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning*. In International Conference on Learning Representations.
- [2] Brunello, A., Montanari, A., & Reynolds, M. (2019). *Synthesis of LTL Formulas from Natural Language Texts: State of the Art and Research Directions*. In 26th International Symposium on Temporal Representation and Reasoning (TIME 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- [3] De Giacomo, G., Iocchi, L., Favorito, M., & Patrizi, F. (2019, July). Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In Proceedings of the International Conference on Automated Planning and Scheduling (Vol. 29, No. 1, pp. 128-136).
- [4] J. Dzifcak, M. Scheutz, C. Baral and P. Schermerhorn, *What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution*. 2009 IEEE International Conference on Robotics and Automation, Kobe, 2009, pp. 4163-4168, doi: 10.1109/ROBOT.2009.5152776.
- [5] Lignos, C., Raman, V., Finucane, C. et al. *Provably correct reactive control from natural language*. Auton Robot 38, pp. 89–105 (2015). <https://doi.org/10.1007/s10514-014-9418-8>.
- [6] G. Sturla, 2017 (May, 26). *A Two-Phased Approach for Natural Language Parsing into Formal Logic* (Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts). pp. 18-44. Retrieved from <https://dspace.mit.edu/bitstream/handle/1721.1/113294/1016164771-MIT.pdf?sequence=1>.
- [7] M. Chen, (2018). *Translating Natural Language into Linear Temporal Logic* (RUCS publication, University of Toronto, Toronto, Ontario). Retrieved from <https://ruc.s.ca/assets/2018/submissions/chen.pdf>.
- [8] C. Lu, R. Krishna, M. Bernstein and L. Fei-Fei, 2016. *Visual Relationship Detection with Language Priors*. European Conference on Computer Vision. Retrieved from <https://cs.stanford.edu/people/ranjaykrishna/vrd/>.

References

- [9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov (2017). *Proximal Policy Optimization Algorithms*. OpenAI. Retrieved from <https://arxiv.org/abs/1707.06347>.

The BOT

- Simulates a human teacher by helping the agent during learning
- Direct access to a tree representation of instructions
- Uses a stack machine with possibly recursive subgoals
- Implements the shortest path search routine



Implemented subgoals:

- Open: open a door that is in front of the agent
- Close: close a door that is in front of the agent
- Pickup: execute pickup action
- Drop: execute drop action
- GoNextTO: go next to an object matching a given description or next to a cell at a given position
- Explore: uncover unseen parts of the environment

Competencies of levels

Appendix 2

- **Room Navigation (ROOM)**: navigate a 6x6 room.
- **Ignoring Distracting Boxes (DISTR-BOX)**: navigate the environment even when there are multiple distracting grey box objects in it.
- **Ignoring Distractors (DISTR)**: same as DISTR-BOX, but distractor objects can be boxes, keys or balls of any color.
- **Maze Navigation (MAZE)**: navigate a 3x3 maze of 6x6 rooms, randomly inter-connected by doors.
- **Unblocking the Way (UNBLOCK)**: navigate the environment even when it requires moving objects out of the way.
- **Unlocking Doors (UNLOCK)**: to be able to find the key and unlock the door if the instructions requires this explicitly.
- **Guessing to Unlock Doors (IMP-UNLOCK)**: to solve levels that require unlocking a door, even if this is not explicitly stated in the instruction.
- **Go To Instructions (GOTO)**: understand "go to" instructions.
- **Open Instructions (OPEN)**: understand "open" instructions.
- **Pickup Instructions (PICKUP)**: understand "pick up" instructions.
- **Put Instructions (PUT)**: understand "put" instructions.
- **Location Language (LOC)**: understand instructions where objects are referred to by relative location as well as their shape and color.
- **Sequences of Commands (SEQ)**: understand composite instructions requiring an agent to execute a sequence of instruction clauses.

Markovian Decision Processes

Markov Decision Process (MDP)

- It contains a set of states S , a set of actions A , a transition function $Tr : S \times A \rightarrow Prob(S)$, that returns for every state s and action a , a probability distribution over the next state, and a reward function $R : S \times A \times S \rightarrow \mathbb{R}$ that specifies the reward received by the agent when transitioning from state s to state s' by applying action a .
- The solution of an MDP is a function called *policy*, assigning an action to each state. The value of a policy p at state s is the expected sum of rewards when starting at state s and selecting actions based on p .
- Reinforcement Learning is the task of learning a possibly optimal policy, from an initial state s_0 , on an MDP where only S and A are known, while Tr and R are not.

Appendix 3

Non Markovian Reward Decision Process (NMRDP)

- It is a tuple $M = (S, A, Tr, \hat{R})$, where S, A, Tr are as in an MDP, but the reward \hat{R} is a real-valued function over finite state-action sequences (known as traces) as: $\hat{R} : (S \times A)^* \rightarrow \mathbb{R}$
- The policy is also non-Markovian $\hat{p} : S^* \rightarrow A$
- It is possible to specify \hat{R} implicitly, using a set of pairs $\{(\varphi_i, r_i)\}_{i=1}^m$, with φ_i an LTLf formula selecting the traces to reward and r_i the reward assigned to those traces, where the atomic propositions, so the fluents, of φ_i , correspond to boolean features or boolean propositions.

RB Theory: Lemmas and Thorem

Appendix 4

Lemma 4

RL with LTLf/LDLf restraining specifications $M_{ag}^{rb} = (M_{ag}, RB)$ with $M_{ag} = (S, A, Tr_{ag}, R_{ag})$ and $RB = (L, \{(\varphi_i, r_i)\}_{i=1}^m)$ can be reduced to RL over the NMRDP $M_{ag}^n = (S \times L, A, Tr_{ag}^{rb}, \{(\varphi_i, r_i)\}_{i=1}^m) \cup \{(\varphi_s, R_{ag}(s, a, s'))\}_{s \in S, a \in A, s' \in S}$, by restricting the policy to learn to have the form $\hat{p} : (Q_1 \times \dots \times Q_m \times S)^* \rightarrow A$.

Lemma 5

RL with LTLf/LDLf restraining specifications $M_{ag}^{rb} = (M_{ag}, RB)$ with $M_{ag} = (S, A, Tr_{ag}, R_{ag})$ and $RB = (L, \{(\varphi_i, r_i)\}_{i=1}^m)$ can be reduced to RL over the MDP $M_{ag}' = (S', A, Tr_{ag}', R_{ag}')$, by restricting the policy to learn to have the form $(Q_1 \times \dots \times Q_n \times S) \rightarrow A$.

Theorem 6

RL with LTLf/LDLf restraining specifications $M_{ag}^{rb} = (M_{ag}, RB)$ with $M_{ag} = (S, A, Tr_{ag}, R_{ag})$ and $RB = (L, \{(\varphi_i, r_i)\}_{i=1}^m)$ can be reduced to RL over the MDP $M_{ag}^q = (Q_1 \times \dots \times Q_m \times S, A, Tr_{ag}'', R_{ag}'')$, and optimal policies p_{ag}^{new} for M_{ag}^{rb} can be obtained by learning corresponding optimal policies for M_{ag}^q .

- We can express the goals of Examples 1 and 3 in an alternative way by employing multiple bolts that act simultaneously, one for each natural language command.
- In our implementation we embed multiple bolts in a single container class called Multibolt. In this way the interaction with the BabyAI code is minimal, since a Multibolt exposes the interface of a normal Restraining Bolt.
- The reward of a multibolt is the sum of the rewards of the contained bolts.
- The state of a multibolt is computed from the states of the contained bolts like in TaskExecutor from RLGames [3].

Example 1

"Go at a box."
 $\Diamond \text{at_box}$
 $+$
 "When you are at a box open the box."
 $\Box(\text{at_box} \rightarrow \bigcirc \text{open_box})$

Example 3

"Pick a key and keep it until you reach a box."
 $\Diamond(\text{pick_key} \wedge \bigcirc(\text{pick_key} \cup \text{at_box}))$
 $+$
 "When you are at a box drop the key."
 $\Box(\text{at_key} \rightarrow \bigcirc \neg \text{pick_key})$