# 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

Distinguish between search and planning

Determine how planning can be used in IF

Summarize how planning has appeared in story generation through the years

Explore how Theory of Mind can expand narrative planning in the Sabre planner

# Review: AI Agent Definition

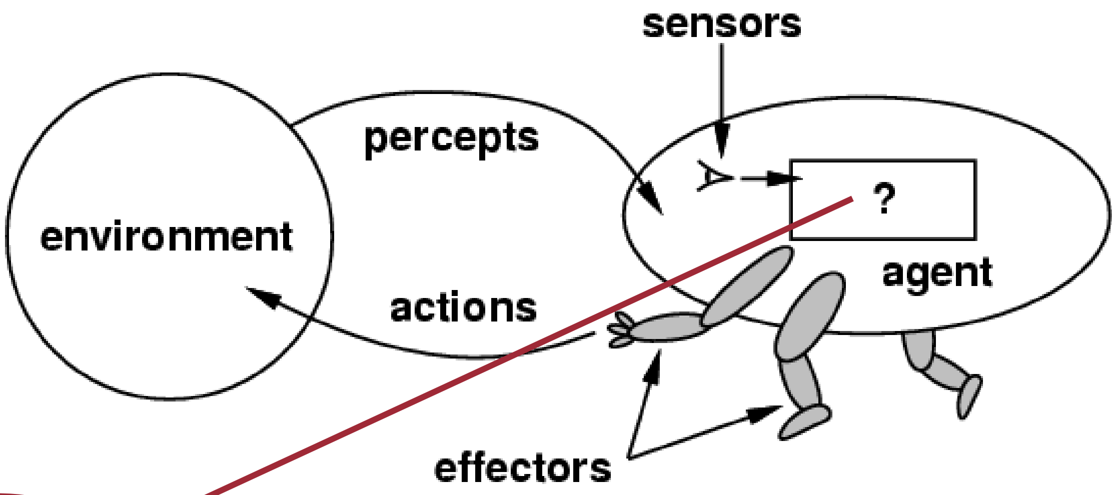
**Agent**: anything that perceives its environment through sensors, and acts on its environment through actuators

**Percept**: input at an instant

**Percept sequence**: history of inputs

**Agent function**: mapping of percept sequence to action

**Agent program**: (concise) implementation of an agent function




*Picture from Dr. Cynthia Matuszek, source unknown*

# Review:
# Formal Definition of a Search Problem

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

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

4. **Transition Model:** ∀ **s**∀ **a**∈**Actions(s)**

   **Result(s, a)** → $\mathbf{s_r}$

   $\mathbf{s_r}$ is called a successor of **s**

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

5. **Path cost** (Performance Measure):
   Must be additive, e.g. sum of distances, number of actions executed, …

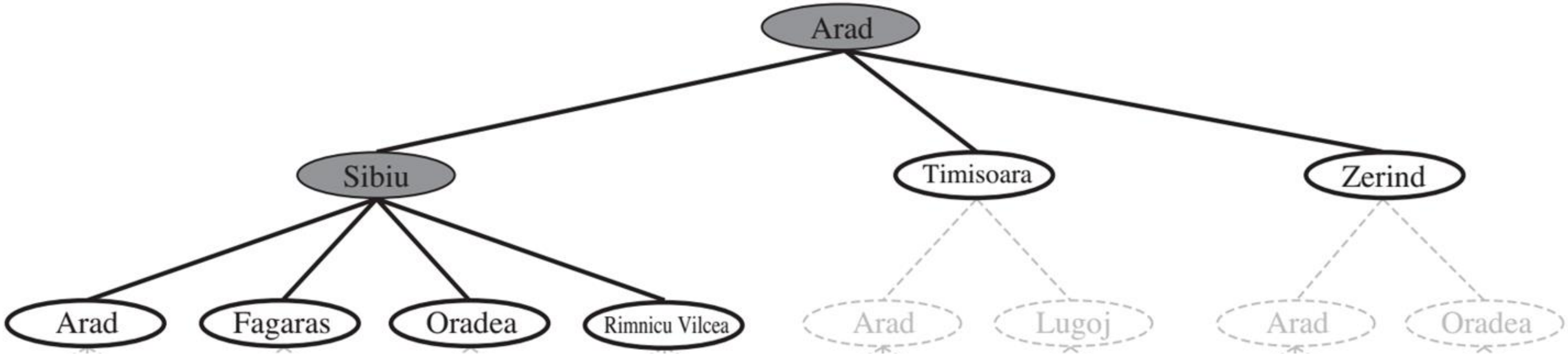
   **c(x,a,y)** is the step cost, assumed ≥ 0

   - (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)**

# Review: Generalized tree search




function TREE-SEARCH(*problem, strategy*) return a solution or failure

Initialize frontier to the *initial state* of the *problem*

do

if the frontier is empty then return *failure*

*choose leaf node for expansion according to strategy & remove from frontier*

if node contains goal state then return *solution*

else expand the node and add resulting nodes to the frontier

**The strategy determines search process!**

# Review: Search Strategies

Several classic search algorithms differ only by the order of how they expand their search trees

You can implement them by using different queue data structures

- **Depth-first search** = LIFO queue
- **Breadth-first search** = FIFO queue
- **Greedy best-first search** or **A* search** = Priority queue

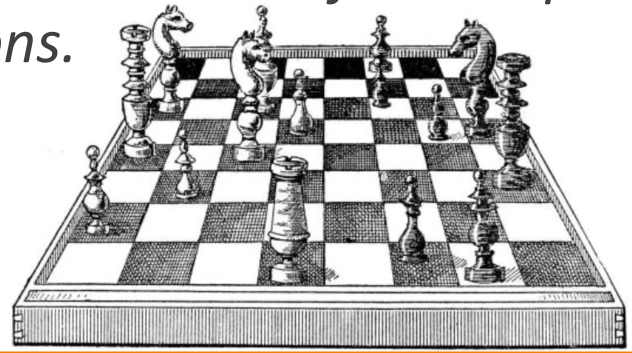
# Classical Planning

The task of finding a sequence of action to accomplish a goal in a deterministic, fully observable, discrete, static environment.

If an environment is:

- **Deterministic**
- **Fully observable**

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



In environments that are

- **Nondeterministic** or
- **Partially observable**

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

# Representation Language

**Planning Domain Definition Language** (PDDL) express **actions** as a **schema**

```
(: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 name

Variables

Preconditions

Preconditions and effects are **conjunctions** of logical sentences

Effects

These logical sentences are **literals** – positive or negated atomic sentences

# State Representation

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

Ground means they contain no variables

Atomic sentences contain just a single predicate

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

**Action Schema**
has variables

```
(: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)))
)
```

**State Representation**
arguments are constants
fluents may change over time

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

- Closed world assumption: any fluent not mentioned is false.
- Unique names are distinct.

# Successor States

A **ground action** is **applicable** if if every positive literal in the precondition is true, and every negative literal in the precondition is false

**Ground Action**
no variables

```
(:action go
   :parameters (out, npc, cottage, gardenpath)
   :precondition (and (at npc cottage) (connected cottage out gardenpath)
                      (not (blocked cottage out gardenpath)))
   :effect (and (at npc gardenpath) (not (at npc cottage)))
)
```

**Initial State**

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

Negative literals in the effects are kept in a **delete list** DEL(), and positive literals are kept in an **add list** ADD()

**Result**
New state reflecting the effect of applying the ground action

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

# Domain

Set of Action Schema

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

# 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)))
)
```

Initial State

Goal

# Algorithms for Classical Planning

We can apply **BFS** to the **initial state** through possible states looking for a **goal**.

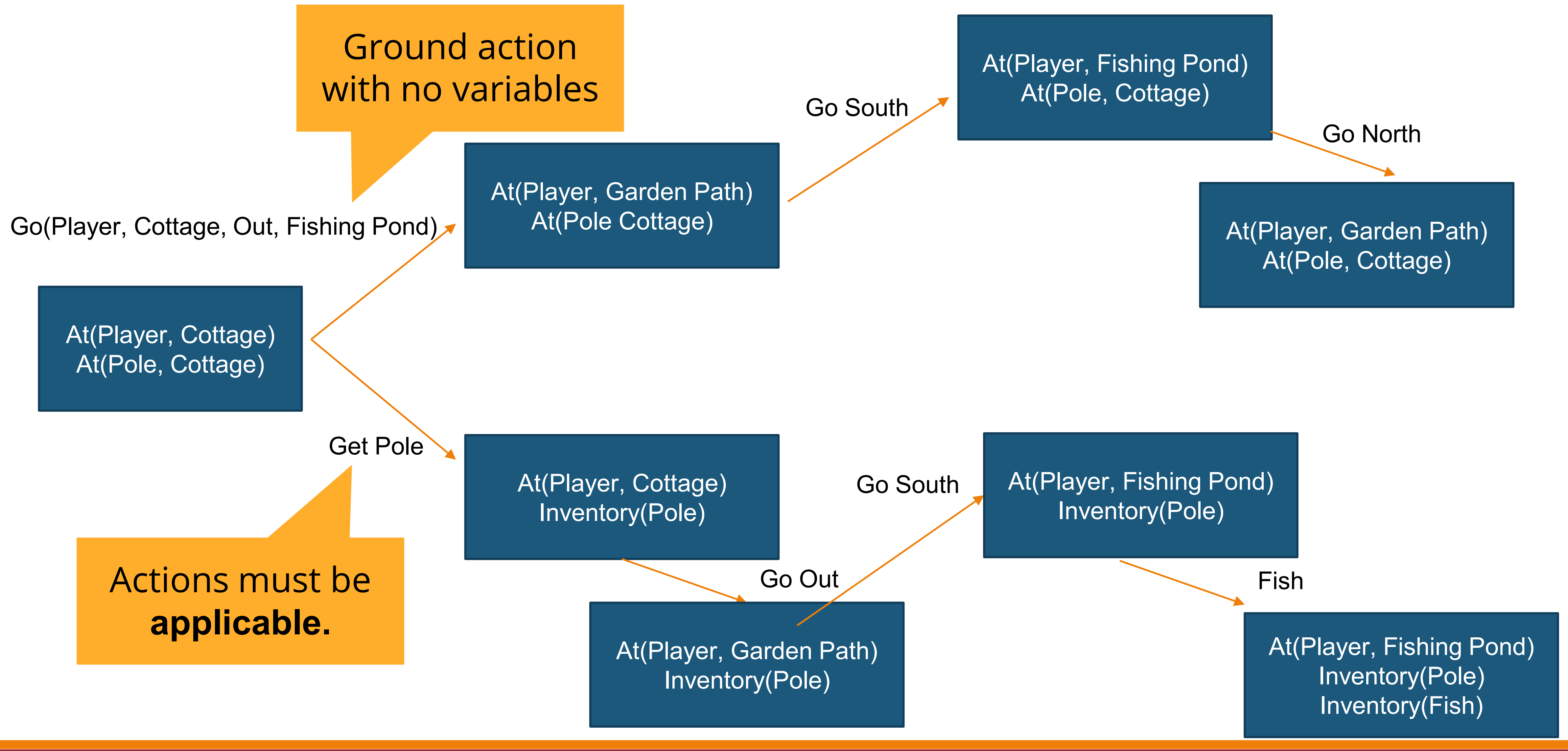
An advantage of the **declarative representation** of action schemas is that we can also **search backwards**.

**Start with a goal** and work backwards towards the initial state.

Why work backwards?

In our Action Castle example, this would help us with the branching problem that the `drop` action introduced. If we work backwards from the goal, then we realize that we don't ever need to drop an item for the correct solution.

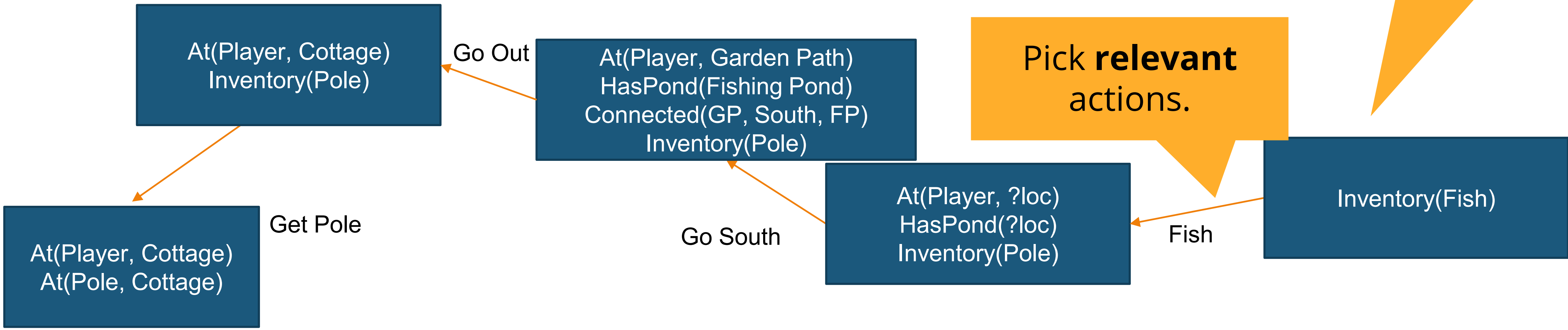
# Forward State-Space Search for Planning



# 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

# Heuristics for Planning

Neither forward nor backward search is efficient without good **heuristics.**

In search, a heuristic function *h(s)* estimates the distance from a state to the goal.

**Admissible heuristics** never over-estimate the true distance and can be used with **A* search** to find optimal solutions.

Admissible heuristics can be derived from a **relaxed problem** (approximation) that is easier to solve.

The "**ignore preconditions" heuristic** relaxes the problem.

# Hierarchical Planning

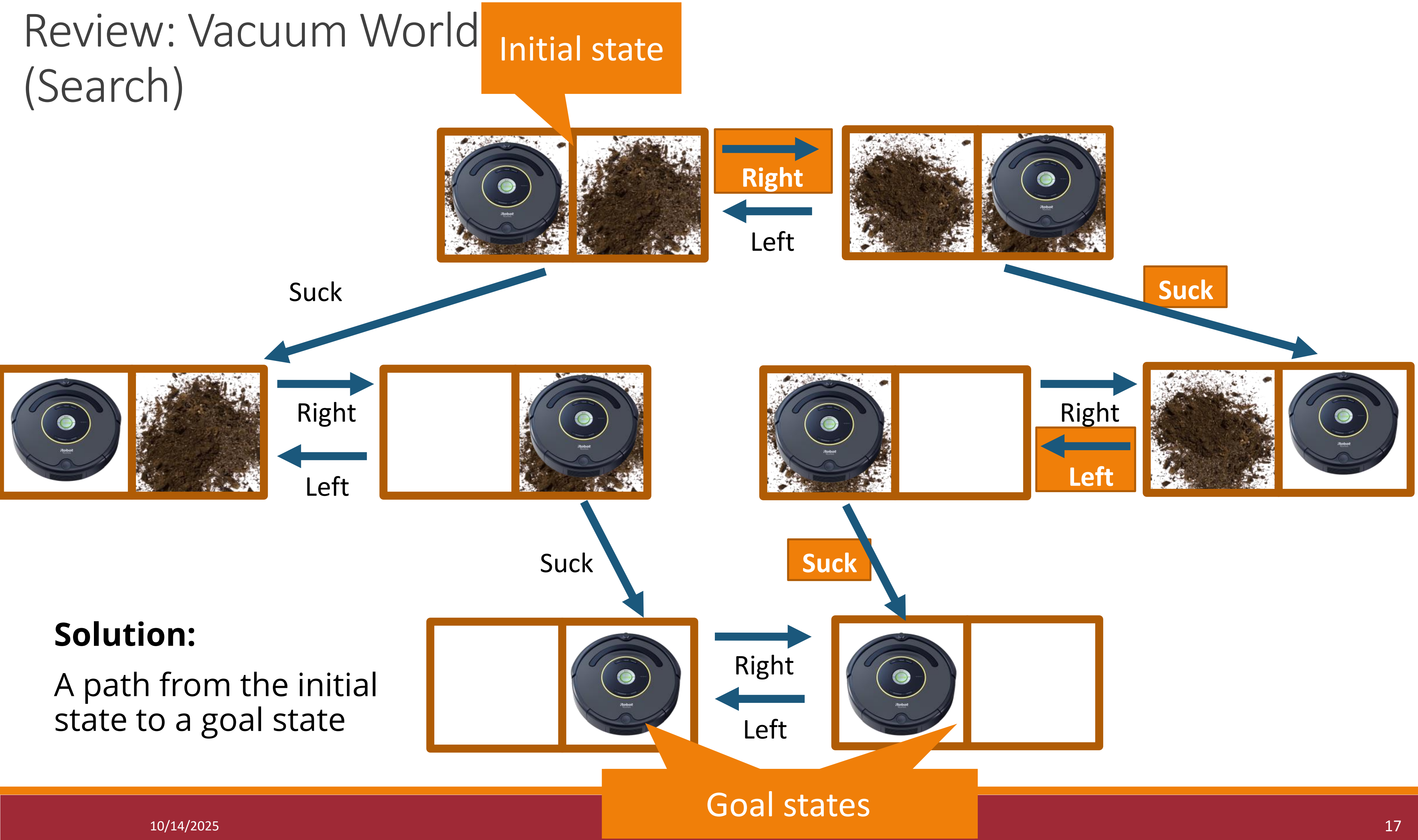
Instead of using atomic actions, we can define actions at **higher levels of abstraction.**

**Hierarchical decomposition** organizes actions into high-level functions, composed of more fine-grained function, composed of atomic actions.

Plan out sequence of high-level actions, reclusively **refine the plan** until we've got atomic actions.

Tricky to ensure that the resulting plan is optimal.

# Review: Vacuum World (Search)

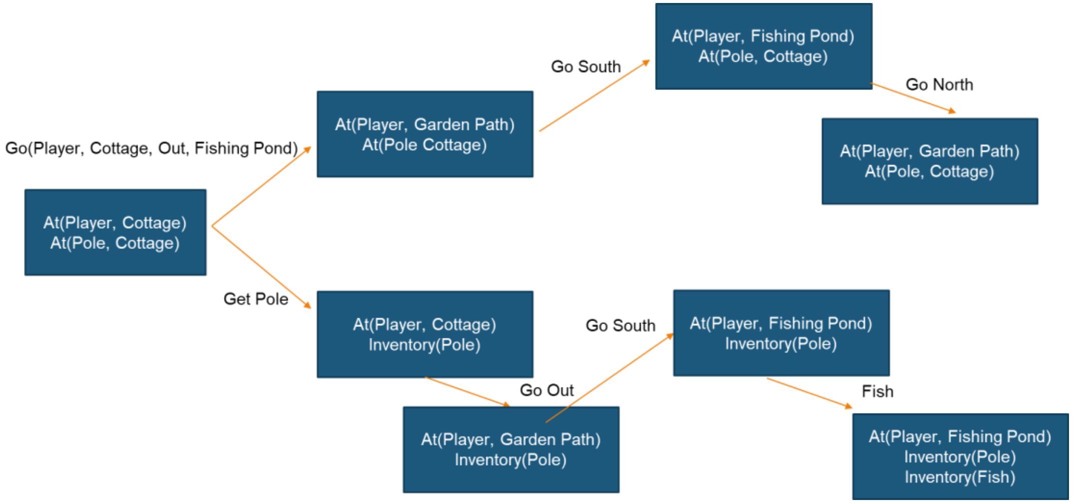



**Solution:**

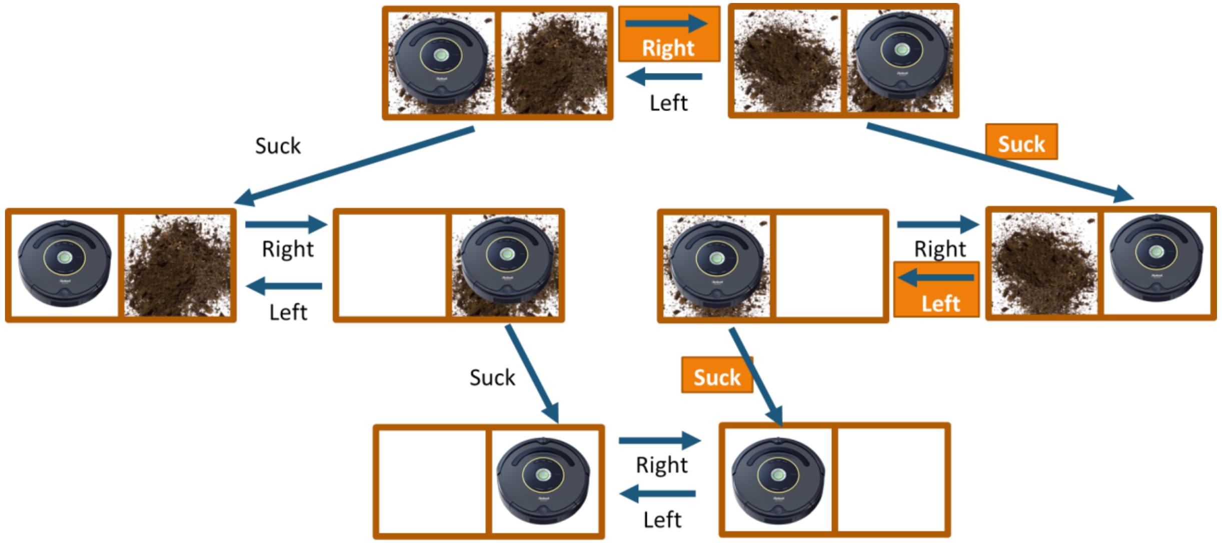
A path from the initial state to a goal state

# Think-Pair-Share: Search vs Planning

What are some of the differences between search vs planning?




Planning




Search

# Planning and Games

# Planning can be used for AI characters

In our current text adventure games, all of the non-player characters are boring!

Why doesn't the princess try to escape the tower and claim the throne herself?

Why doesn't the troll go hunt for food and eat us or the guard?

Why is the ghost of the king stuck in the dungeon?

We could give each of them goals and have them try to plan out and play the game alongside the player.

(Teaser for HW 3)

# Generating Puzzles

In HW1, we were able to generate descriptions of locations and items.

Could we use planning to automatically generate:

1. Puzzles?
2. Special actions?

Let's say a player needs a **sword** and we decide to make the game more challenging by not putting one anywhere in the game.

Could we generate an action that results in the creation of a sword?

**Action: forge a sword**

**Effects: a sword is created**

**Preconditions: molten metal, a cast of a sword, an anvil, a hammer**

# Planning and Stories

# UNIVERSE

Table 2
A typical UNIVERSE plot fragment.

| | | |
|---|---|---|
| PLOT FRAGMENT: | forced-marriage | |
| CHARACTERS: | ?him ?her ?husband ?parent | |
| CONSTRAINTS: | (has-husband ?her) | {the husband character} |
| | (has-parent ?husband) | {the parent character} |
| | ( < (trait-value ?parent ‘niceness) – 5) | |
| | (female-adult ?her) | |
| | (male-adult ?him) | |
| GOALS: | (churn ?him ?her) | {prevent them from being happy} |
| SUBGOALS: | (do-threaten ?parent ?her “forget it”) | {threaten ?her} |
| | (dump-lover ?her ?him) | {have ?her dump ?him} |
| | (worry-about ?him) | {have someone worry about ?him} |
| | (together * ?him) | {get ?him involved with someone else} |
| | (eliminate ?parent) | {get rid of ?parent (breaking threat)} |
| | (do-divorce ?husband ?her) | {end the unhappy marriage} |
| | (or (churn ?him ?her) | {either keep churning or} |
| | (together ?her ?him)) | {try and get ?her and ?him back together} |



M. Lebowitz, “Story-Telling as Planning and Learning,” *Poetics*, vol. 14, no. 6, pp. 483–502, Dec. 1985, doi: 10.1016/0304-422X(85)90015-4.

# UNIVERSE (with multiple goals)

```
*(tell '(((churn JOSHUA FRAN)) ((together JOSHUA VALERIE))))

working on goal -- CHURN JOSHUA FRAN
 -- using plan ACCIDENT-BREAKUP P1/FRAN P2/JOSHUA THIRD-PARTY/VALERIE

working on goal -- DO-DISABLE FRAN
 -- using plan DISABLE PERSON/FRAN

>>> FRAN has a spinal injury and is paralyzed

>>> FRAN doesn't want to ruin JOSHUA's life

>>> FRAN pretends to blame JOSHUA for her malady

working on goal -- DUMP-LOVER FRAN JOSHUA
 -- using plan BREAK-UP DUMPER/FRAN DUMPED/JOSHUA

>>> FRAN tells JOSHUA she doesn't love him

working on goal -- TOGETHER JOSHUA VALERIE

[again, the story continues unhappily for almost all concerned]
```

**Figure 3:** A multi-goal story



M. Lebowitz, “Planning Stories,” *Annual Conference of the Cognitive Science Society (CogSci)*, vol. 1, no. 2.2, pp. 234–242, Jul. 1987, Available: https://cognitivesciencesociety.org/wp-content/uploads/2019/01/cogsci_9.pdf

# Partial Order Causal Link (POCL) planning

Conflict POCL

Figure 1: Example CPOCL Problem and Domain

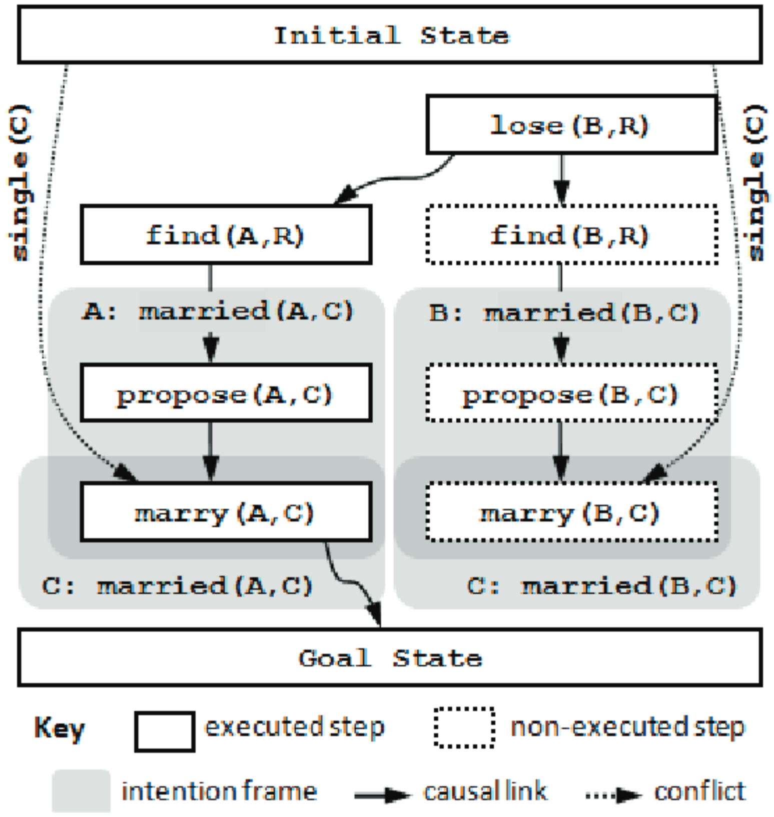
```
Initial:    single(A)∧single(B)∧single(C)∧
            loves(A,C)∧intends(A,married(A,C))∧
            loves(B,C)∧intends(B,married(B,C))∧has(B,R)
Goal:       married(A,C)

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.

# Sabre

## A Narrative Planner Supporting Intention and Deep Theory of Mind

Stephen G. Ware

Cory Siler

Narrative Intelligence LAB

University of Kentucky

GLAIVE

NARRATIVE PLANNER

SABRE

NARRATIVE PLANNER

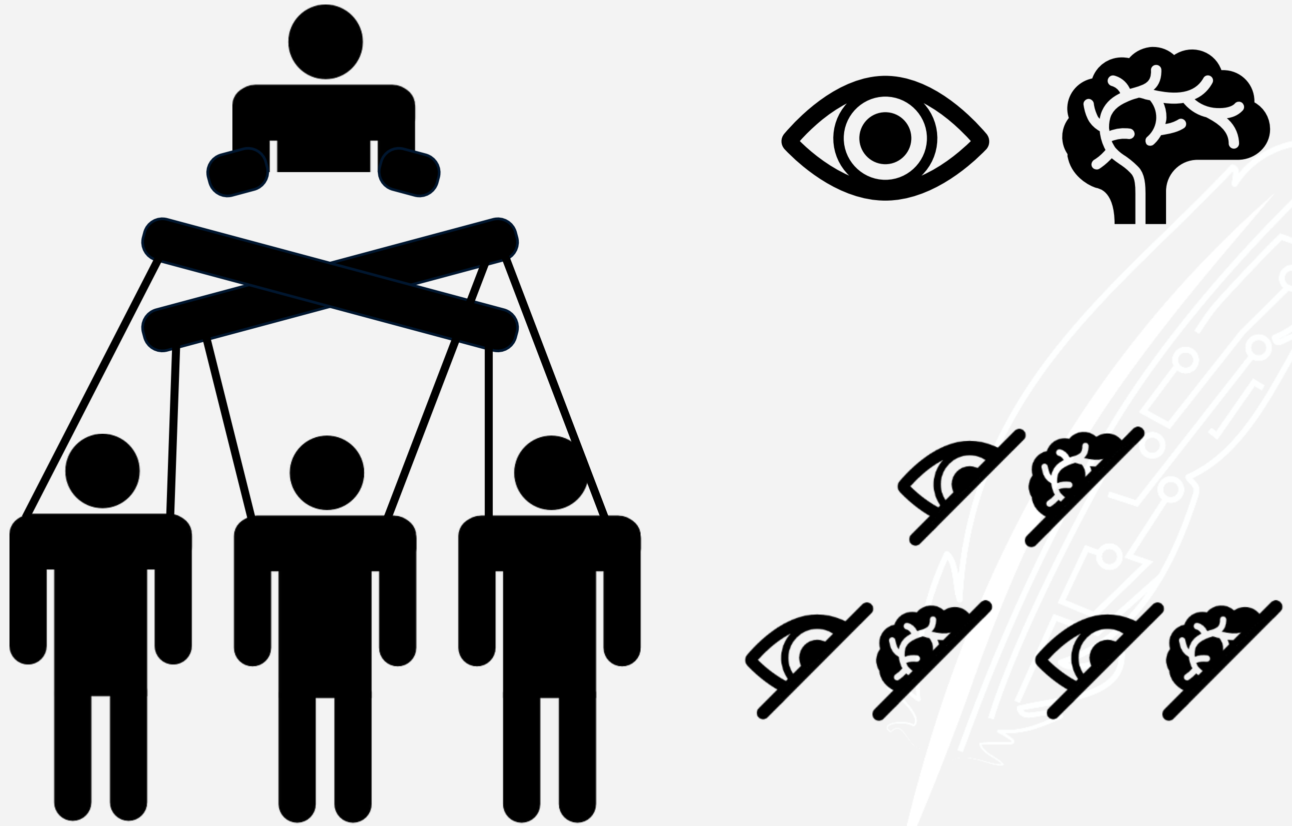
NIL

UK

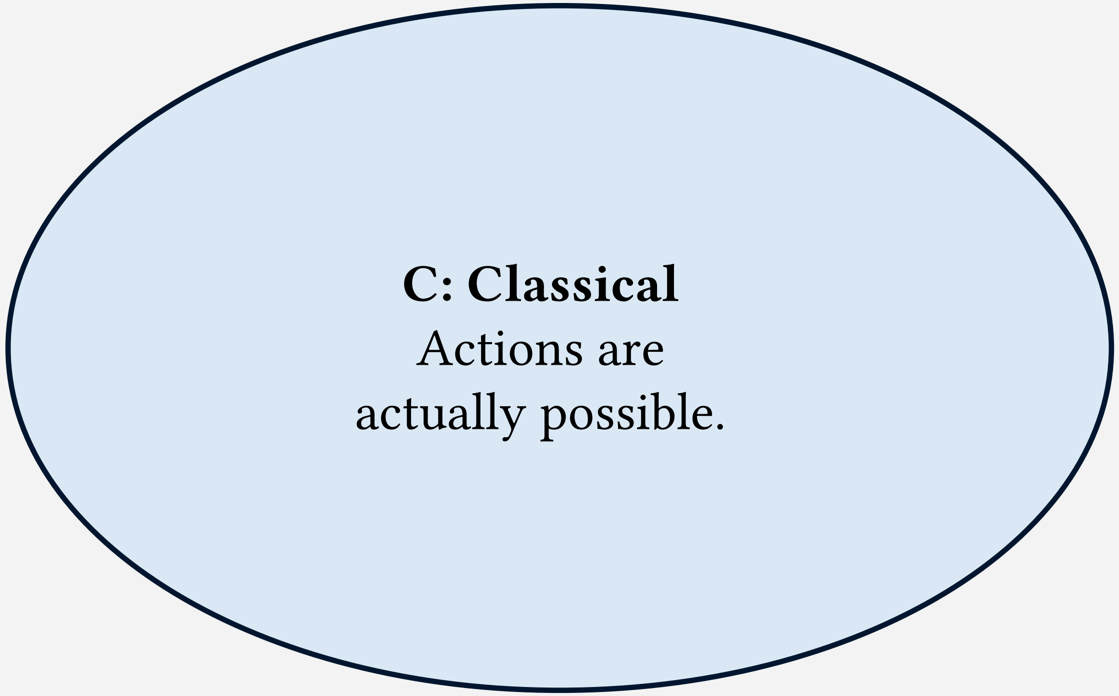
# Narrative Planning

A single decision maker

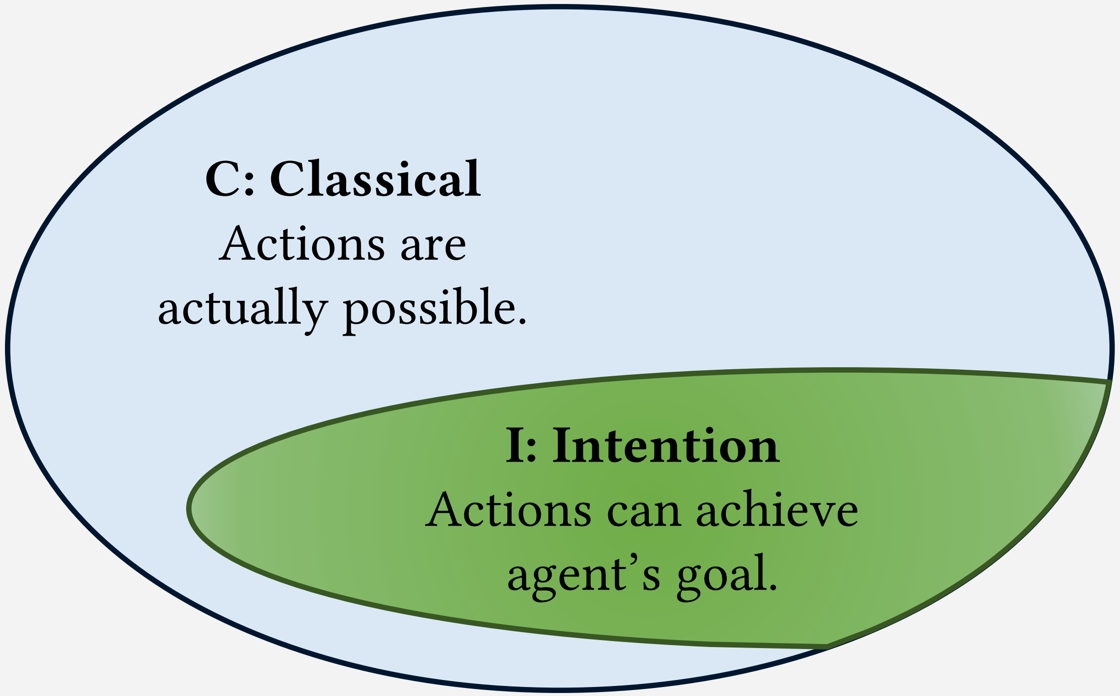
creates the appearance of a multi-agent system.



# Intentions and Beliefs

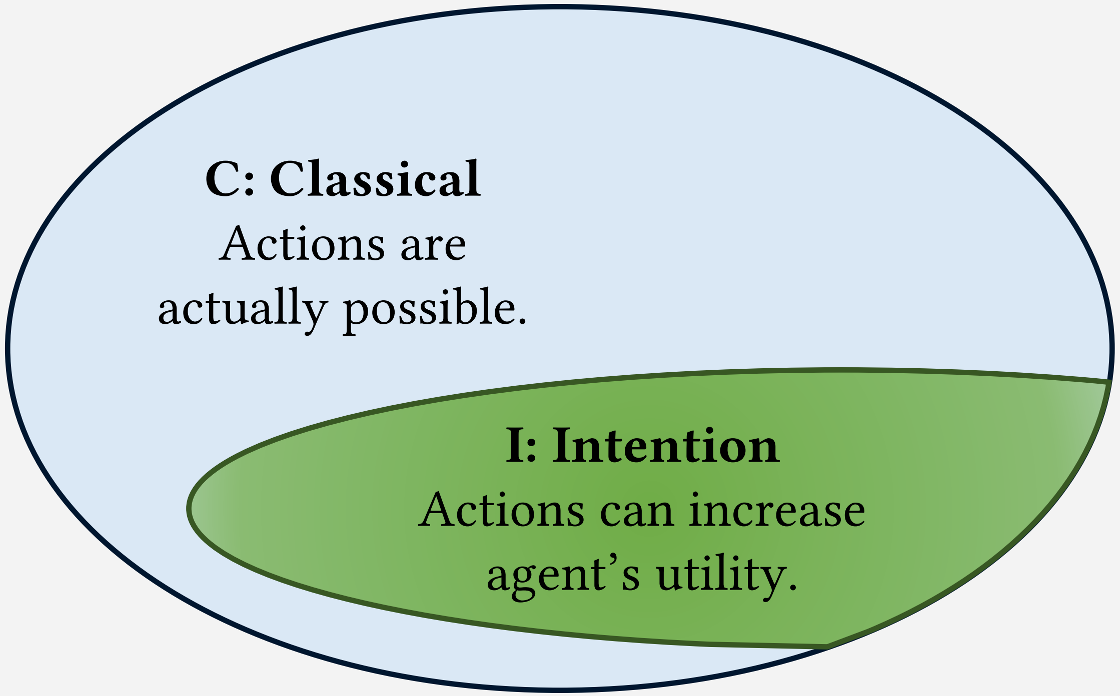


# Intentions and Beliefs

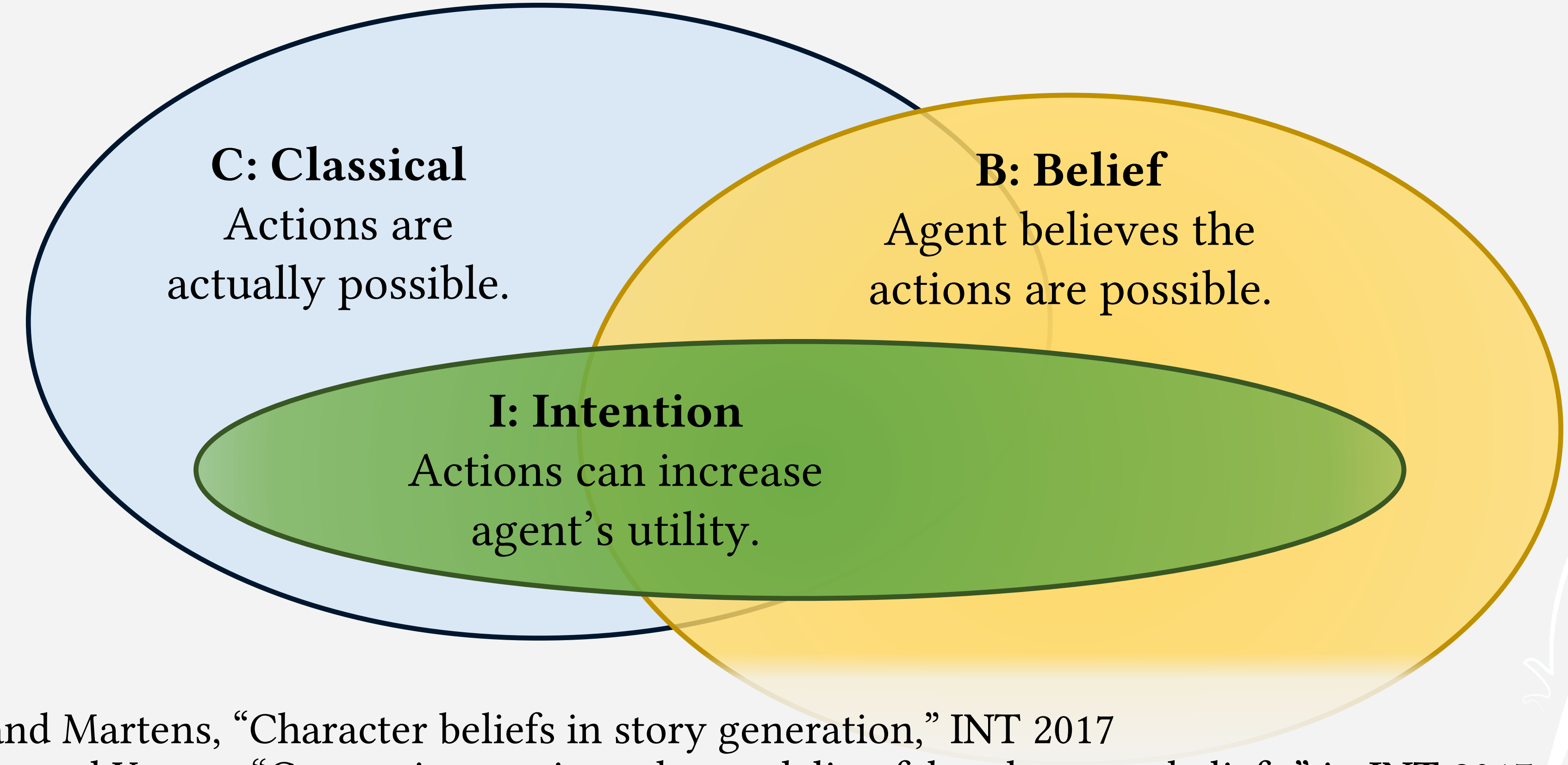



- Riedl and Young, "Narrative planning: balancing plot and character," in JAIR 2010
- Teutenberg and Porteous, "Efficient intent-based narrative generation...," in AAMAS 2013
- Ware and Young, "Glaive: a state-space narrative planner...," in AIIDE 2014

# Intentions and Beliefs



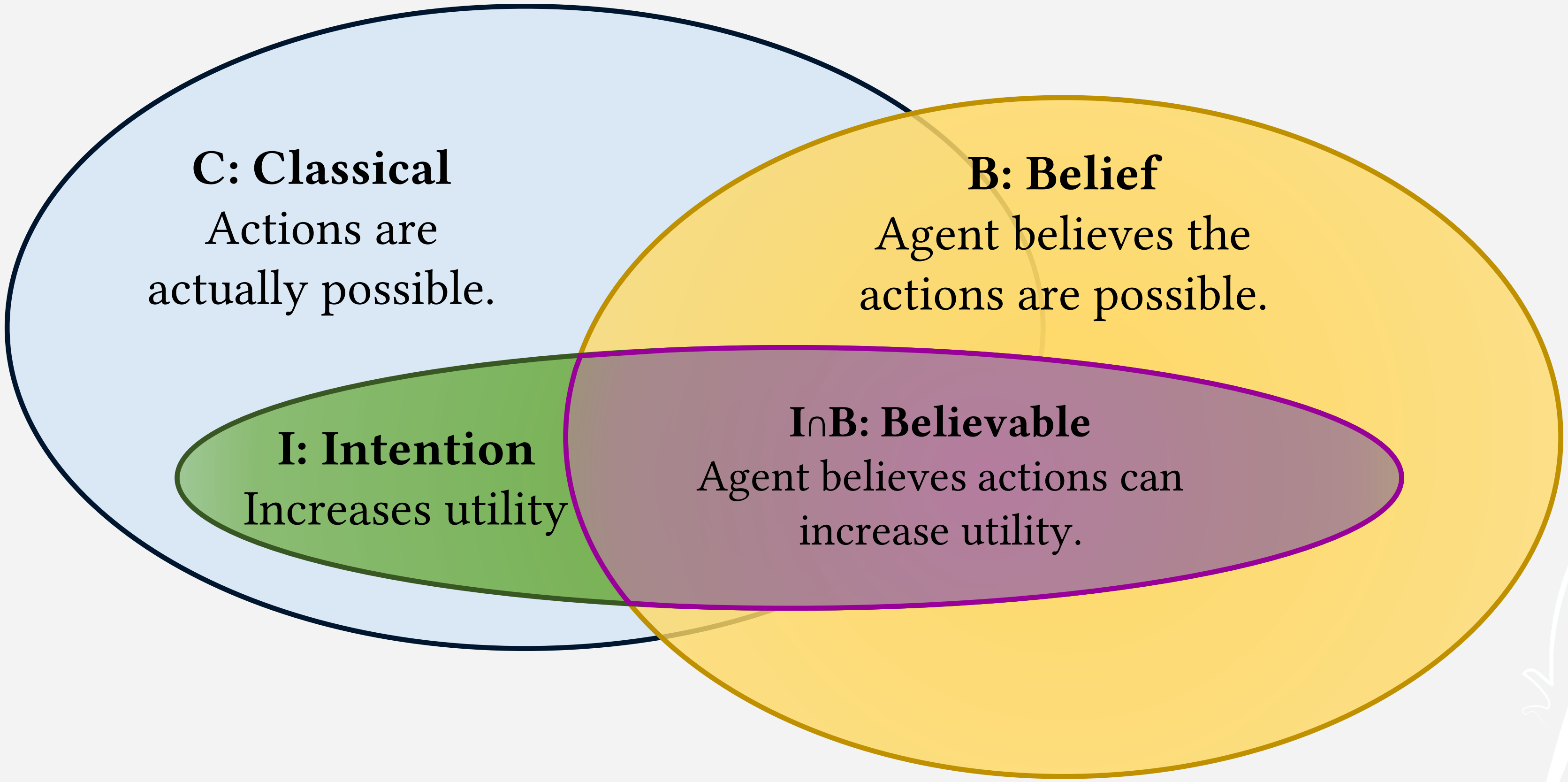
# Intentions and Beliefs





- Eger and Martens, "Character beliefs in story generation," INT 2017
- Thorne and Young, "Generating stories … by modeling false character beliefs," in INT 2017
- Shirvani, Ware, and Farrell, "A possible worlds model of belief…," in AIIDE 2017

# Intentions and Beliefs




- Shirvani, Farrell, and Ware, "Combining intentionality and belief…," in AIIDE 2018

# Syntax and Features

# Fluents

$$at(Tom) =$$

Helmert, "The Fast Downward planning system," in JAIR 2006

# Fluents

$$at(Tom) = Cottage$$

Helmert, "The Fast Downward planning system," in JAIR 2006

# Fluents

$$at(Tom) = Cottage$$

$$path(Cottage, Market) = \top$$

# Fluents

$$at(Tom) = Cottage$$

$$path(Cottage, Market) = \top$$

$$wealth(Merchant) = 3$$

NIL

# Fluents

$$at(Tom) = Cottage$$

$$path(Cottage, Market) = \top$$

$$wealth(Merchant) = 3$$

$$believes(Tom, wealth(Merchant)) = 2$$

# Fluents

$$at(Tom) = Cottage$$

$$path(Cottage, Market) = \top$$

$$wealth(Merchant) = 3$$

$$believes(Tom, wealth(Merchant)) = 2$$

$$believes(Merchant, believes(Tom, wealth(Merchant))) = 3$$

# Theory of Mind

- Arbitrarily deep

  what $x$ believes $y$ believes $z$ believes…
- No uncertainty

  Everyone commits to beliefs, which can be wrong.

# Other Syntactical Features

- Negation
- Disjunction
- Conditional Effects
- First Order Quantifiers

# Actions

$$buy(Tom, Potion, Merchant, Market)$$

# Actions

$$a\text{:}\ \ \boldsymbol{buy(Tom, Potion, Merchant, Market)}$$

# Actions

$$\boldsymbol{a}\text{:}\ \ \boldsymbol{buy(Tom, Potion, Merchant, Market)}$$

$\text{PRE}(a)$:

# Actions

$$\boldsymbol{a}\text{:}\quad \boldsymbol{buy(Tom, Potion, Merchant, Market)}$$

$$\text{PRE}(a)\text{:}\quad at(Tom) = Market$$

# Actions

$$\boldsymbol{a}\text{:}\ \ \boldsymbol{buy(Tom, Potion, Merchant, Market)}$$

$$\text{PRE}(a)\text{:}\ \ at(Tom) = Market \wedge at(Merchant) = Market$$

# Actions

$$\boldsymbol{a}\text{:}\ \ \boldsymbol{buy(Tom, Potion, Merchant, Market)}$$

$$\text{PRE}(a)\text{:}\ \ at(Tom) = Market \land at(Merchant) = Market \land$$

$$at(Potion) = Merchant$$

# Actions

$$\boldsymbol{a}\text{:}\quad \boldsymbol{buy(Tom, Potion, Merchant, Market)}$$

$$\text{PRE}(a)\text{:}\quad at(Tom) = Market \land at(Merchant) = Market \land$$

$$at(Potion) = Merchant \land wealth(Tom) \geq 1$$

# Actions

$$\boldsymbol{a}\text{:}\quad \boldsymbol{buy(Tom, Potion, Merchant, Market)}$$

$$\text{PRE}(a)\text{:}\quad at(Tom) = Market \land at(Merchant) = Market \land$$

$$at(Potion) = Merchant \land wealth(Tom) \geq 1$$

$$\text{EFF}(a)\text{:}$$

# Actions

$$\boldsymbol{a}\text{:}\ \ \boldsymbol{buy(Tom, Potion, Merchant, Market)}$$

$$\text{PRE}(a)\text{:}\ \ at(Tom) = Market \wedge at(Merchant) = Market \wedge$$
$$at(Potion) = Merchant \wedge wealth(Tom) \geq 1$$

$$\text{EFF}(a)\text{:}\ \ at(Potion) = Tom$$

# Actions

$$\boldsymbol{a}\text{:}\ \ \boldsymbol{buy(Tom, Potion, Merchant, Market)}$$

$$\text{PRE}(a)\text{:}\ \ at(Tom) = Market \land at(Merchant) = Market \land$$
$$at(Potion) = Merchant \land wealth(Tom) \geq 1$$

$$\text{EFF}(a)\text{:}\ \ at(Potion) = Tom \land wealth(Merchant) \mathrel{+}= 1$$

# Actions

$$\boldsymbol{a}\text{:}\ \ \boldsymbol{buy(Tom, Potion, Merchant, Market)}$$

$$\text{PRE}(a)\text{:}\ \ at(Tom) = Market \land at(Merchant) = Market \land$$
$$at(Potion) = Merchant \land wealth(Tom) \geq 1$$

$$\text{EFF}(a)\text{:}\ \ at(Potion) = Tom \land wealth(Merchant) \mathrel{+}= 1 \land$$
$$wealth(Tom) \mathrel{-}= 1$$

# Actions

$\boldsymbol{a}$: $\boldsymbol{buy(Tom, Potion, Merchant, Market)}$

PRE$(a)$: $at(Tom) = Market \wedge at(Merchant) = Market \wedge$
$at(Potion) = Merchant \wedge wealth(Tom) \geq 1$

EFF$(a)$: $at(Potion) = Tom \wedge wealth(Merchant) \mathrel{+}= 1 \wedge$
$wealth(Tom) \mathrel{-}= 1$

CON$(a)$:

# Actions

$$\boldsymbol{a}\text{:}\quad \boldsymbol{buy(Tom, Potion, Merchant, Market)}$$

$$\text{PRE}(a)\text{:}\quad at(Tom) = Market \wedge at(Merchant) = Market \wedge$$
$$at(Potion) = Merchant \wedge wealth(Tom) \geq 1$$

$$\text{EFF}(a)\text{:}\quad at(Potion) = Tom \wedge wealth(Merchant) += 1 \wedge$$
$$wealth(Tom) -= 1$$

$$\text{CON}(a)\text{:}\quad \{Tom, Merchant\}$$

# Actions

$\boldsymbol{a}$: $\boldsymbol{buy(Tom, Potion, Merchant, Market)}$

PRE$(a)$: $at(Tom) = Market \wedge at(Merchant) = Market \wedge$
$at(Potion) = Merchant \wedge wealth(Tom) \geq 1$

EFF$(a)$: $at(Potion) = Tom \wedge wealth(Merchant) \mathrel{+}= 1 \wedge$
$wealth(Tom) \mathrel{-}= 1$

CON$(a)$: $\{Tom, Merchant\}$

OBS$(a, c)$:

# Actions

$$\boldsymbol{a}\text{:}\ \ \boldsymbol{buy}(\boldsymbol{Tom}, \boldsymbol{Potion}, \boldsymbol{Merchant}, \boldsymbol{Market})$$

$$\text{PRE}(a)\text{:}\ \ at(Tom) = Market \wedge at(Merchant) = Market \wedge$$
$$at(Potion) = Merchant \wedge wealth(Tom) \geq 1$$

$$\text{EFF}(a)\text{:}\ \ at(Potion) = Tom \wedge wealth(Merchant) \mathrel{+}= 1 \wedge$$
$$wealth(Tom) \mathrel{-}= 1$$

$$\text{CON}(a)\text{:}\ \ \{Tom, Merchant\}$$

$$\text{OBS}(a, c)\text{:}\ \ at(c) = Market$$

# Triggers

$$\boldsymbol{t}\text{:}\;\; \boldsymbol{see(Tom, Merchant, Market)}$$

PRE$(t)$:

EFF$(t)$:

# Triggers

$\boldsymbol{t}$: $\boldsymbol{see(Tom, Merchant, Market)}$

PRE$(t)$: $at(Tom) = Market$

EFF$(t)$:

# Triggers

$$\boldsymbol{t}\text{:}\ \ \boldsymbol{see(Tom, Merchant, Market)}$$

$$\text{PRE}(t)\text{:}\ \ at(Tom) = Market \land at(Merchant) = Market$$

$$\text{EFF}(t)\text{:}$$

# Triggers

$$\boldsymbol{t}\text{:}\ \ \boldsymbol{see(Tom, Merchant, Market)}$$

$$\text{PRE}(t)\text{:}\ \ at(Tom) = Market \land at(Merchant) = Market \land$$

$$believes(Tom, at(Merchant)) \neq Market$$

$$\text{EFF}(t)\text{:}$$

# Triggers

$$\boldsymbol{t}\text{:}\quad \boldsymbol{see(Tom, Merchant, Market)}$$

$$\text{PRE}(t)\text{:}\quad at(Tom) = Market \wedge at(Merchant) = Market \wedge$$
$$believes(Tom, at(Merchant)) \neq Market$$

$$\text{EFF}(t)\text{:}\quad believes\big(Tom, at(Merchant)\big) = Market$$

# Pre-Processing

- Make action and trigger results explicit
- Detect and remove immutable fluents
- Detect and remove impossible actions and triggers

# Results of an Event

After Tom buys the potion from the merchant…

- Tom has the potion.
- Tom knows he has the potion.
- The merchant knows Tom has the potion.
- Tom know that the merchant knows that he has the potion.
- … and so on.

# Example Trigger: Two-Way Paths

$$\boldsymbol{t}\text{:}\ \ \boldsymbol{add\_path}(\boldsymbol{y}, \boldsymbol{x})$$

$$\text{PRE}(t)\ \ path(x, y) = \top \wedge path(y, x) = \bot$$

$$\text{EFF}(t)\text{:}\ \ path(y, x) = \top$$

# Example Trigger: Two-Way Paths

$$\begin{aligned}
\boldsymbol{t}&: \boldsymbol{add_path}(\boldsymbol{Market}, \boldsymbol{Cottage}) \\
\text{PRE}(t)&: path(Cottage, Market) = \top \wedge \\
&\quad\, path(Market, Cottage) = \bot \\
\text{EFF}(t)&: path(Market, Cottage) = \top
\end{aligned}$$

NIL

UK

# Example Action: Walk

$$\boldsymbol{a}: \ \boldsymbol{walk(Tom, Market, Cottage)}$$

$$\text{PRE}(a): \ at(Tom) = Market \wedge$$

$$path(Market, Cottage) = \top$$

$$\text{EFF}(a): \ at(Tom) = Cottage$$

$$\text{CON}(a): \ \{Tom\}$$

$$\text{OBS}(a, c): \ at(c) = Market \vee at(c) = Cottage$$

# Example Action: Walk

$$\begin{aligned}
\boldsymbol{a}:&\ \boldsymbol{walk(Tom, Market, Cottage)} \\
\text{PRE}(a):&\ at(Tom) = Market \wedge \\
&\ path(Market, Cottage) = \top \\
\text{EFF}(a):&\ at(Tom) = Cottage \\
\text{CON}(a):&\ \{Tom\} \\
\text{OBS}(a, c):&\ at(c) = Market \vee at(c) = Cottage
\end{aligned}$$

# Example Action: Walk

$$\boldsymbol{a}\text{:}\quad \boldsymbol{walk(Tom, Market, Cottage)}$$

$$\text{PRE}(a)\text{:}\quad at(Tom) = Market$$

$$\text{EFF}(a)\text{:}\quad at(Tom) = Cottage$$

$$\text{CON}(a)\text{:}\quad \{Tom\}$$

$$\text{OBS}(a, c)\text{:}\quad at(c) = Market \vee at(c) = Cottage$$

# Search

**Algorithm 1** The Sabre algorithm

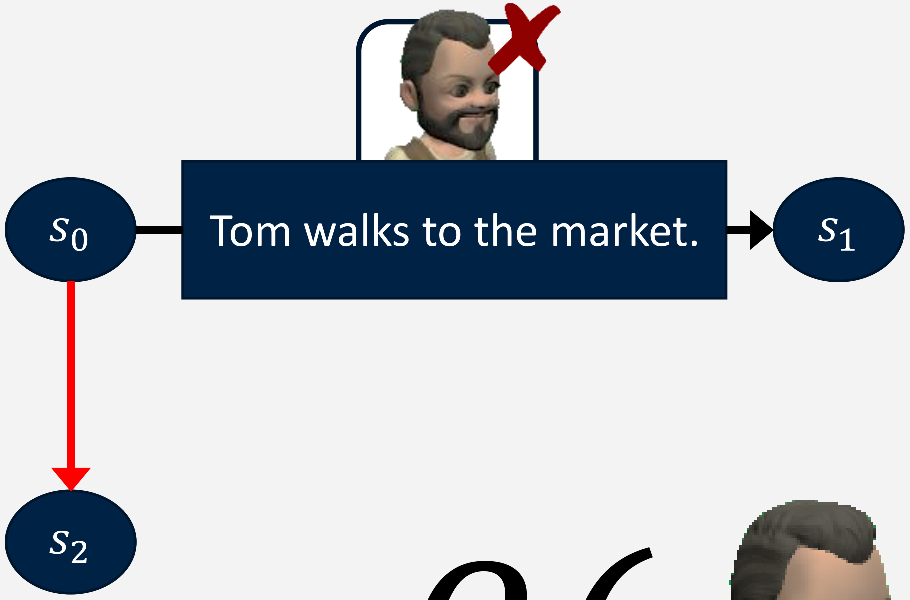
Let $\mathcal{A}$ be the set of all actions defined in the domain.
$\textsc{Sabre}(c_{author}, s_0, \emptyset, s_0)$
**function** $\textsc{Sabre}(c, r, \pi, s)$
    **Input:** character $c$, start state $r$, plan $\pi$, current state $s$
    **if** $u(c, s) > u(c, r)$ and $\pi$ is non-redundant **then**
        **return** $\pi$
    Choose an action $a \in \mathcal{A}$ such that $s \models \textsc{pre}(a)$.
    **for all** $c' \in \textsc{con}(a)$ such that $c' \neq c$ **do**
        Let state $b = \alpha(a, \beta(c', s))$.
        **if** $b$ is undefined **then return** failure.
        **else if** $\textsc{Sabre}(c', b, \emptyset, b)$ fails **then return** failure.
    **return** $\textsc{Sabre}(c, r, \pi \cup a, \alpha(a, s))$

$s_0$

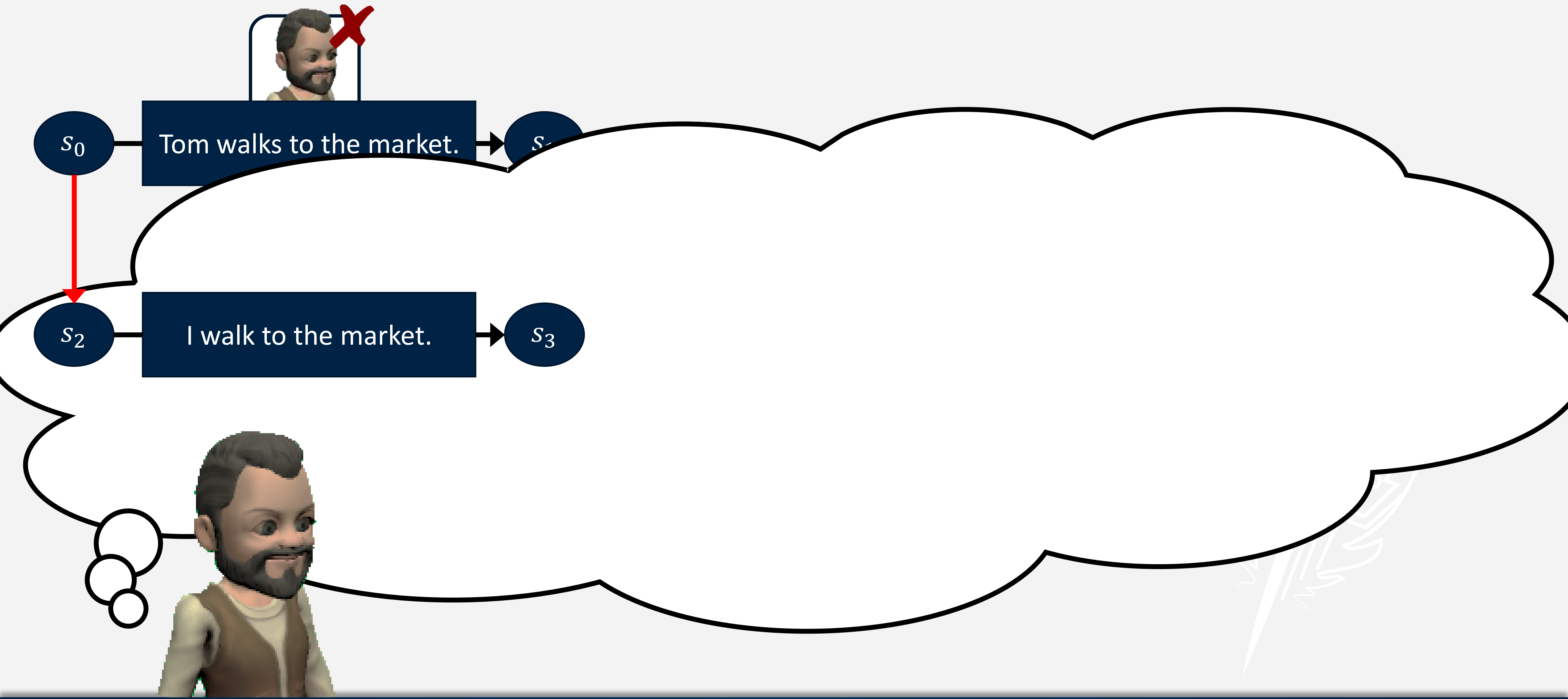
$s_0$

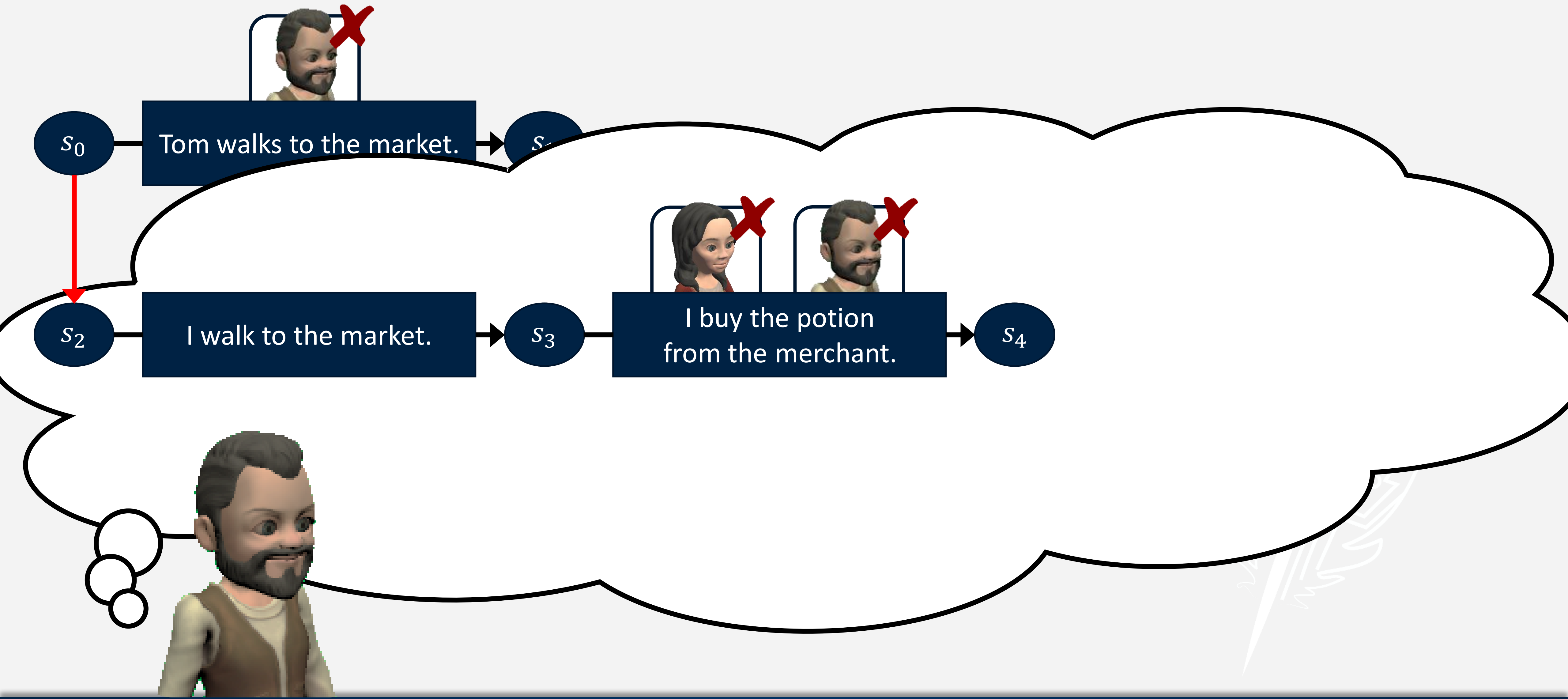
$s_0$ — Tom walks to the market. → $s_1$

$$\alpha(\boxed{\text{Tom walks to the market.}}, s_0) = s_1$$

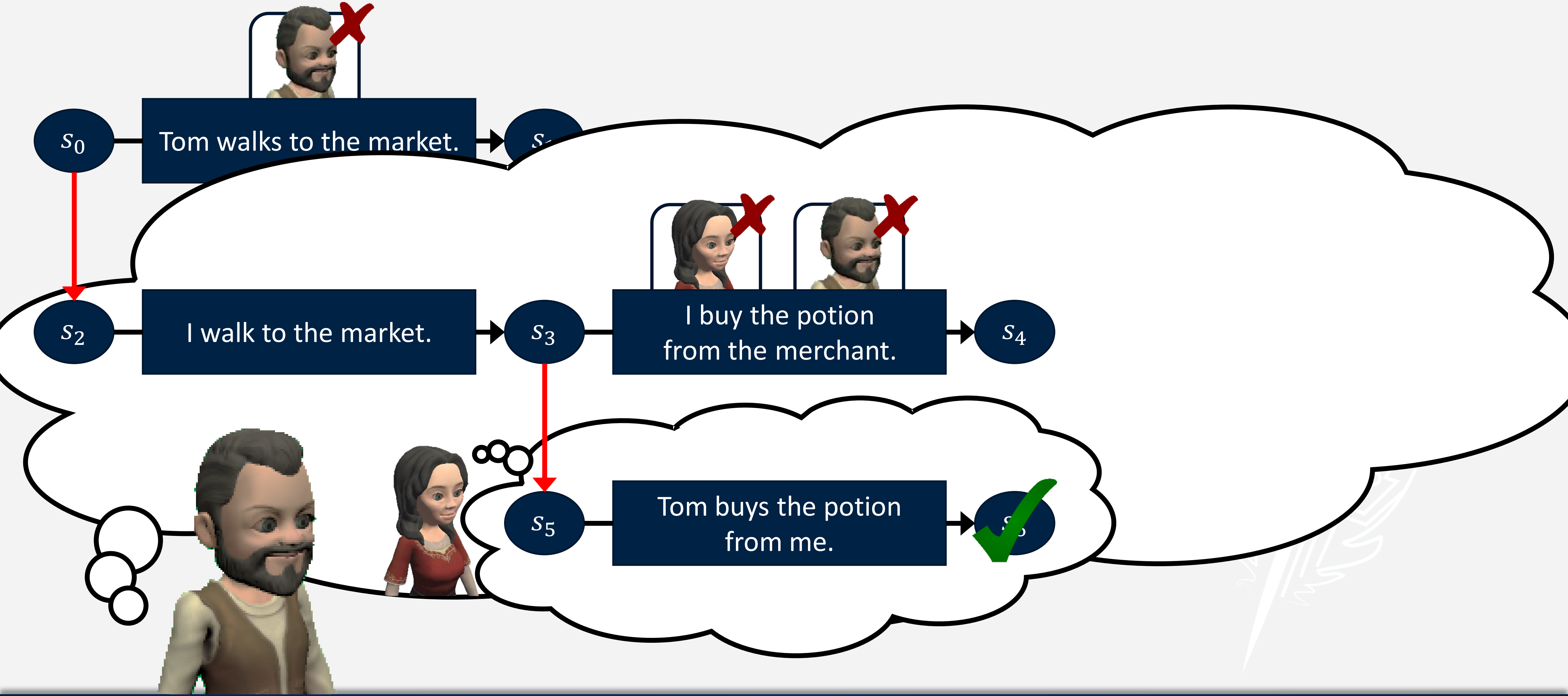


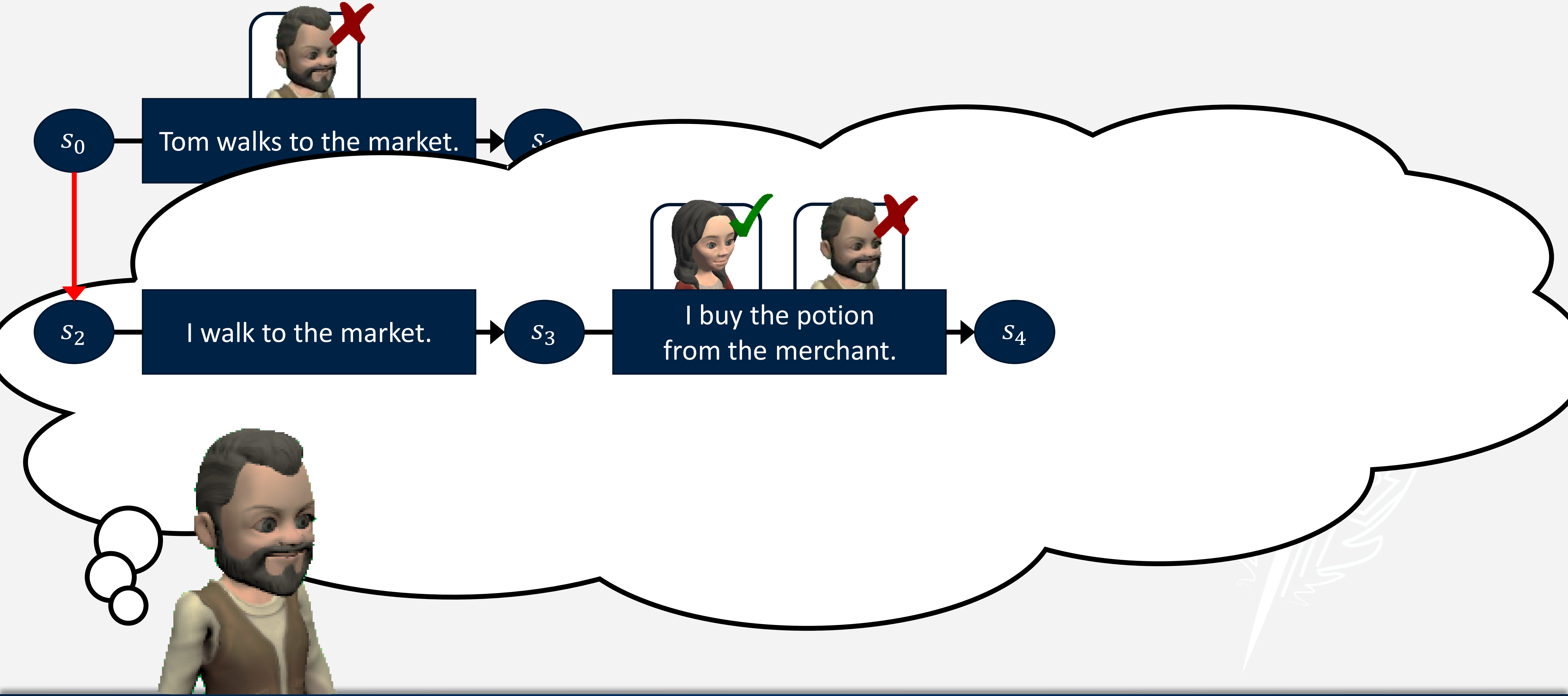

$\beta($  $, s_0) = s_2$


$s_0$
Tom walks to the market.
$s_2$
I walk to the market.
$s_3$

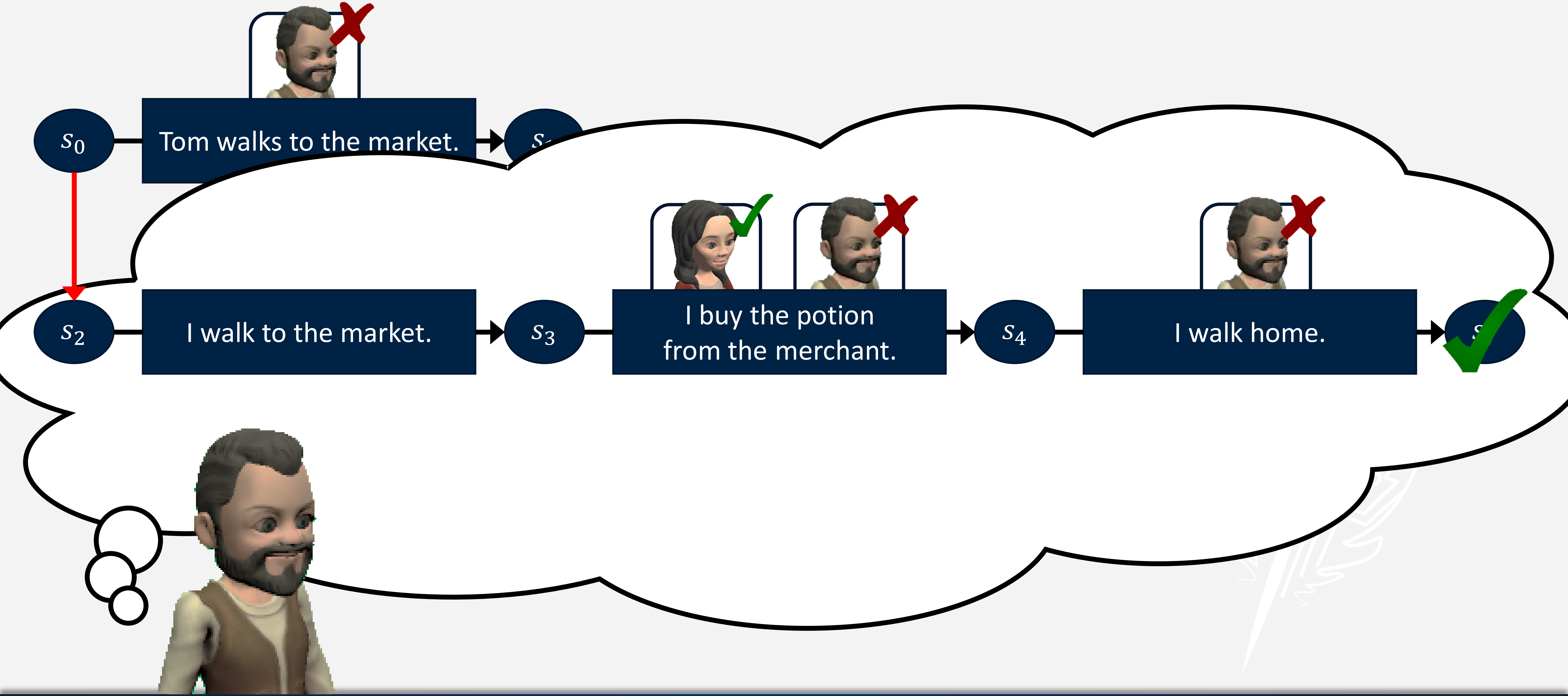


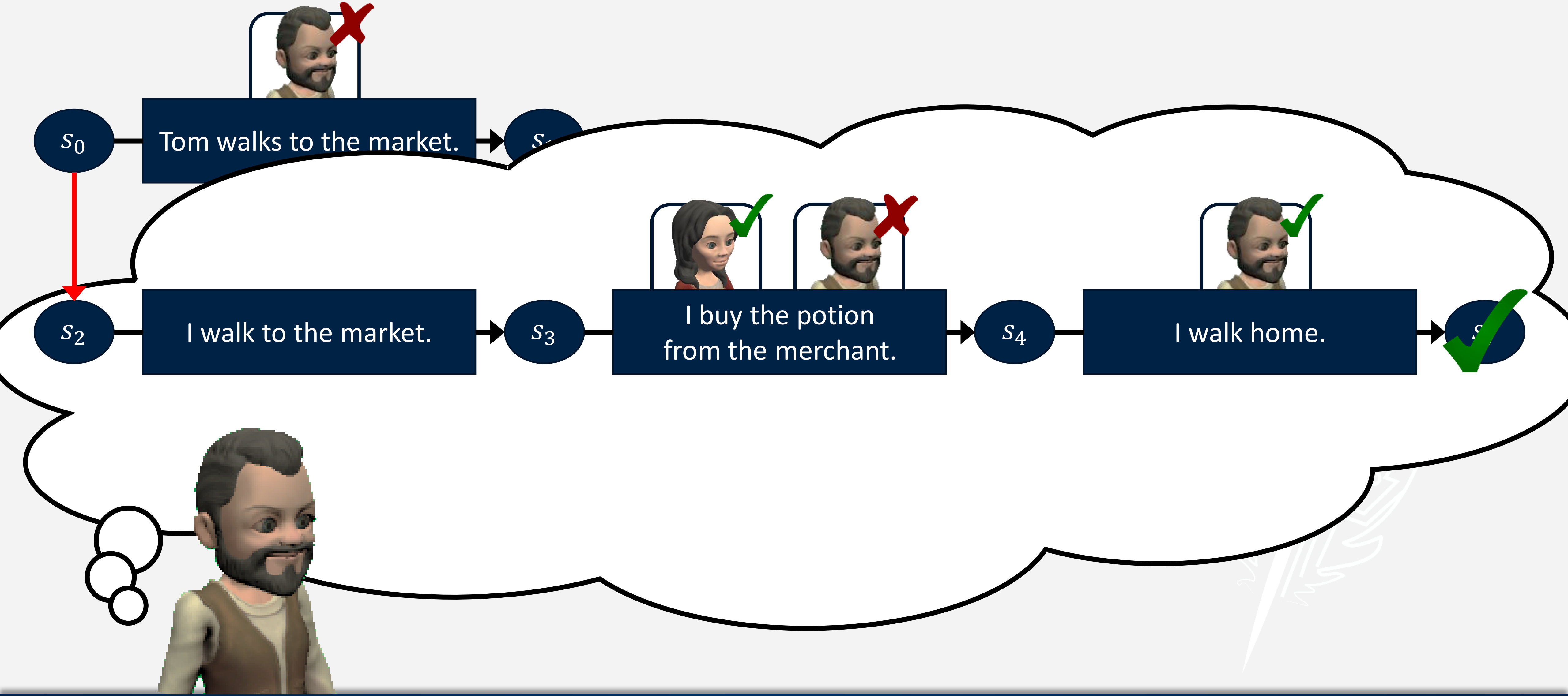
$s_0$
Tom walks to the market.
$s_2$
I walk to the market.
$s_3$
I buy the potion
from the merchant.
$s_4$


$s_0$
Tom walks to the market.
$s_2$
I walk to the market.
$s_3$
I buy the potion from the merchant.
$s_4$
$s_5$
Tom buys the potion from me.

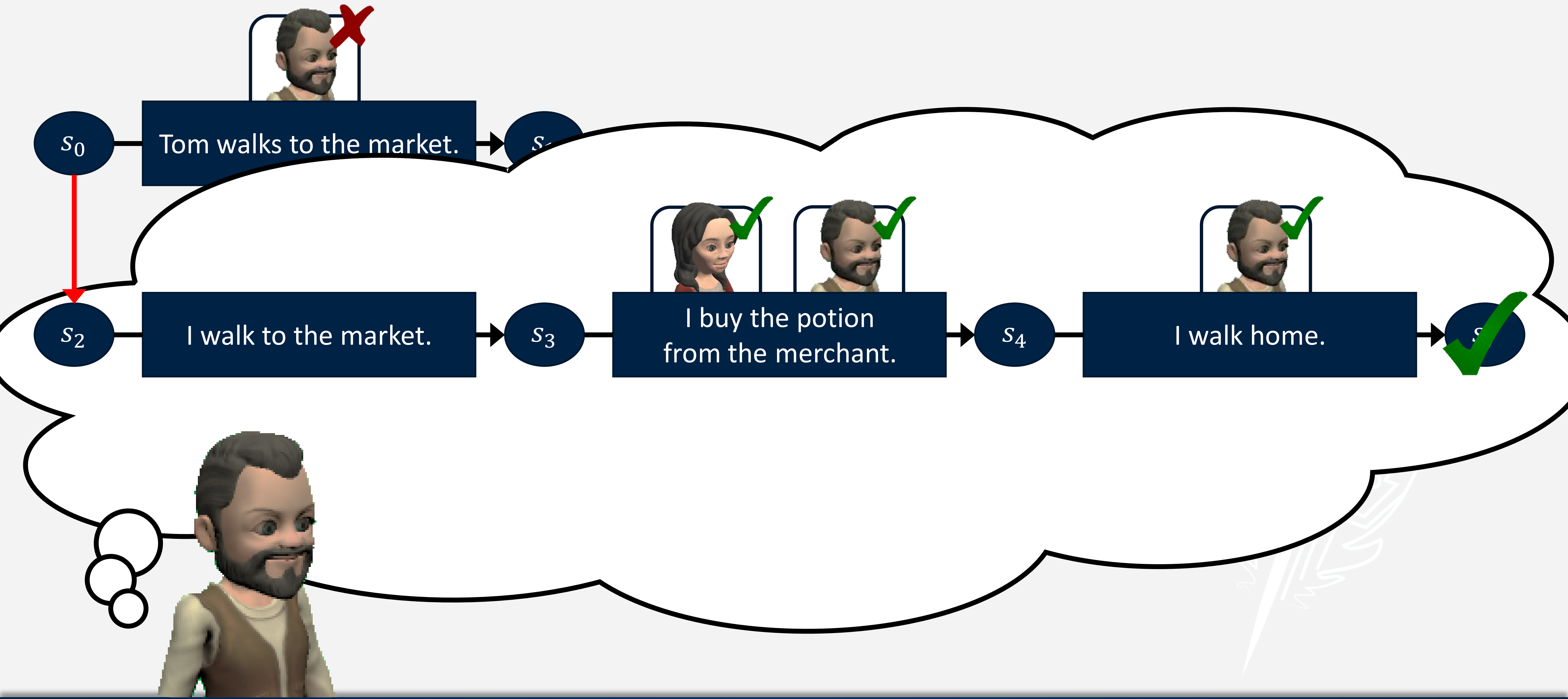


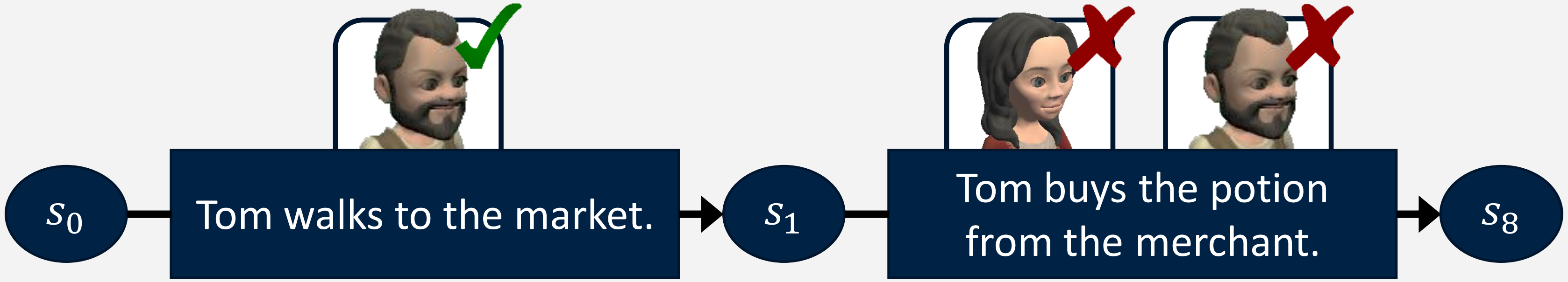
$s_0$
Tom walks to the market.
$s_2$
I walk to the market.
$s_3$
I buy the potion from the merchant.
$s_4$

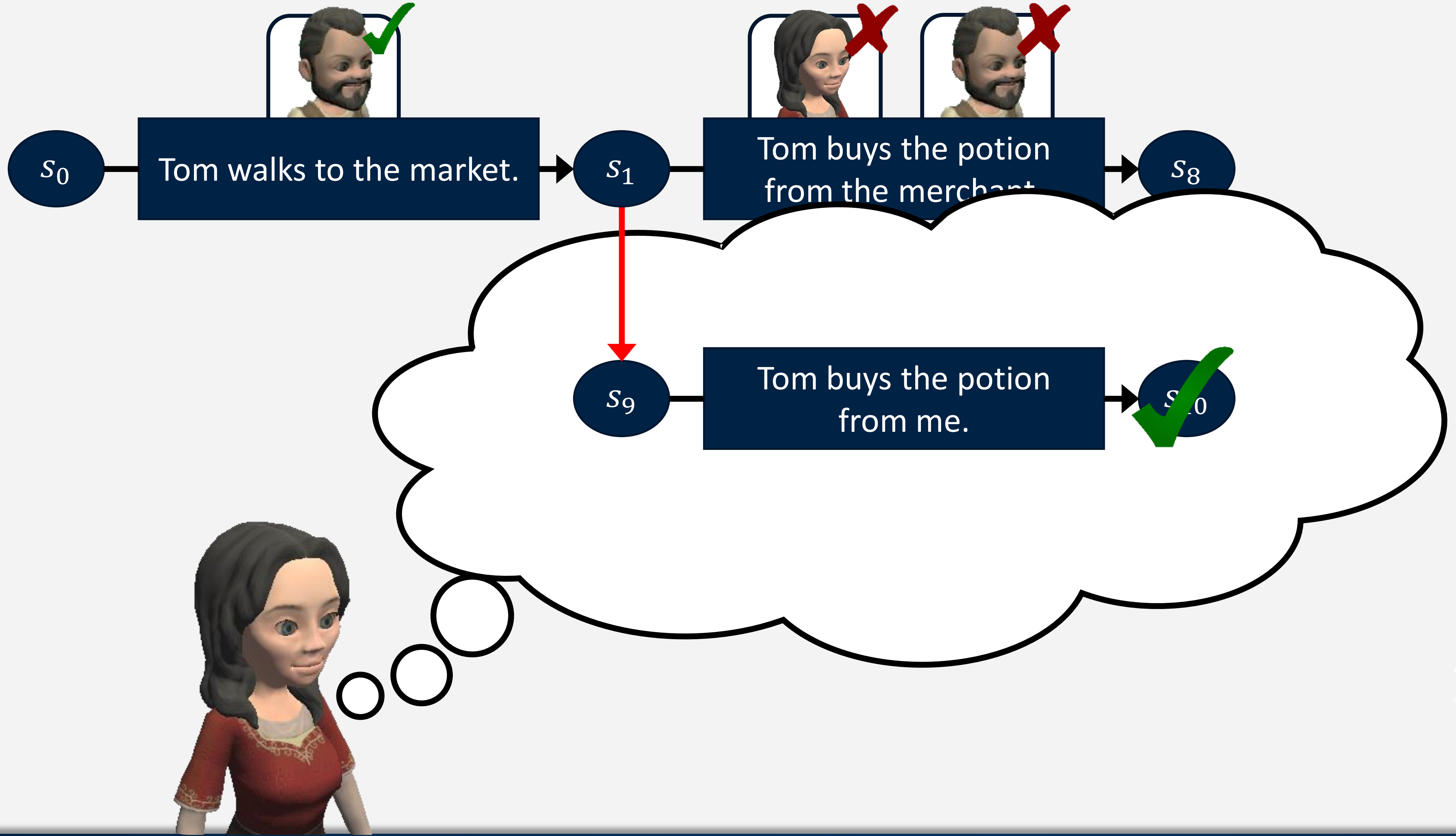

$s_0$
Tom walks to the market.
$s_2$
I walk to the market.
$s_3$
I buy the potion from the merchant.
$s_4$
I walk home.



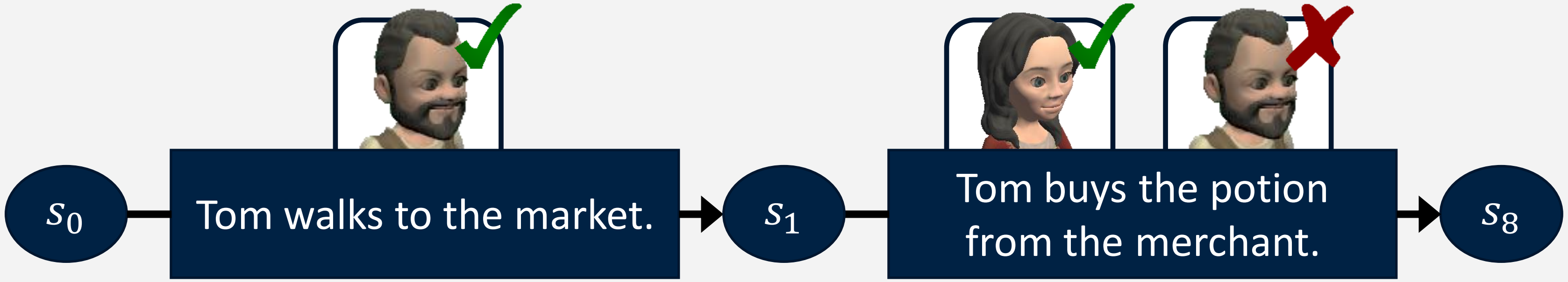
$s_0$
Tom walks to the market.
$s_2$
I walk to the market.
$s_3$
I buy the potion from the merchant.
$s_4$
I walk home.

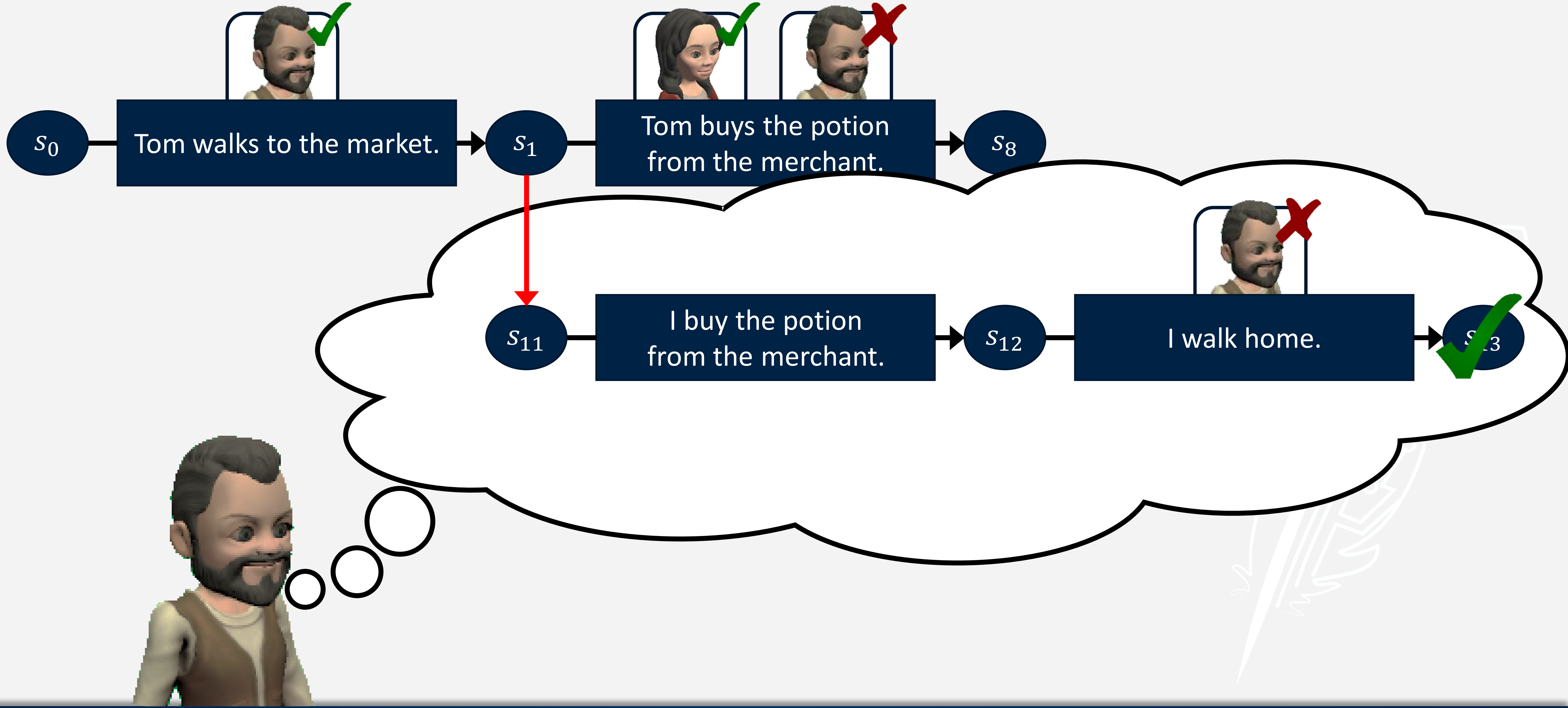

$s_0$
Tom walks to the market.
$s_2$
I walk to the market.
$s_3$
I buy the potion from the merchant.
$s_4$
I walk home.

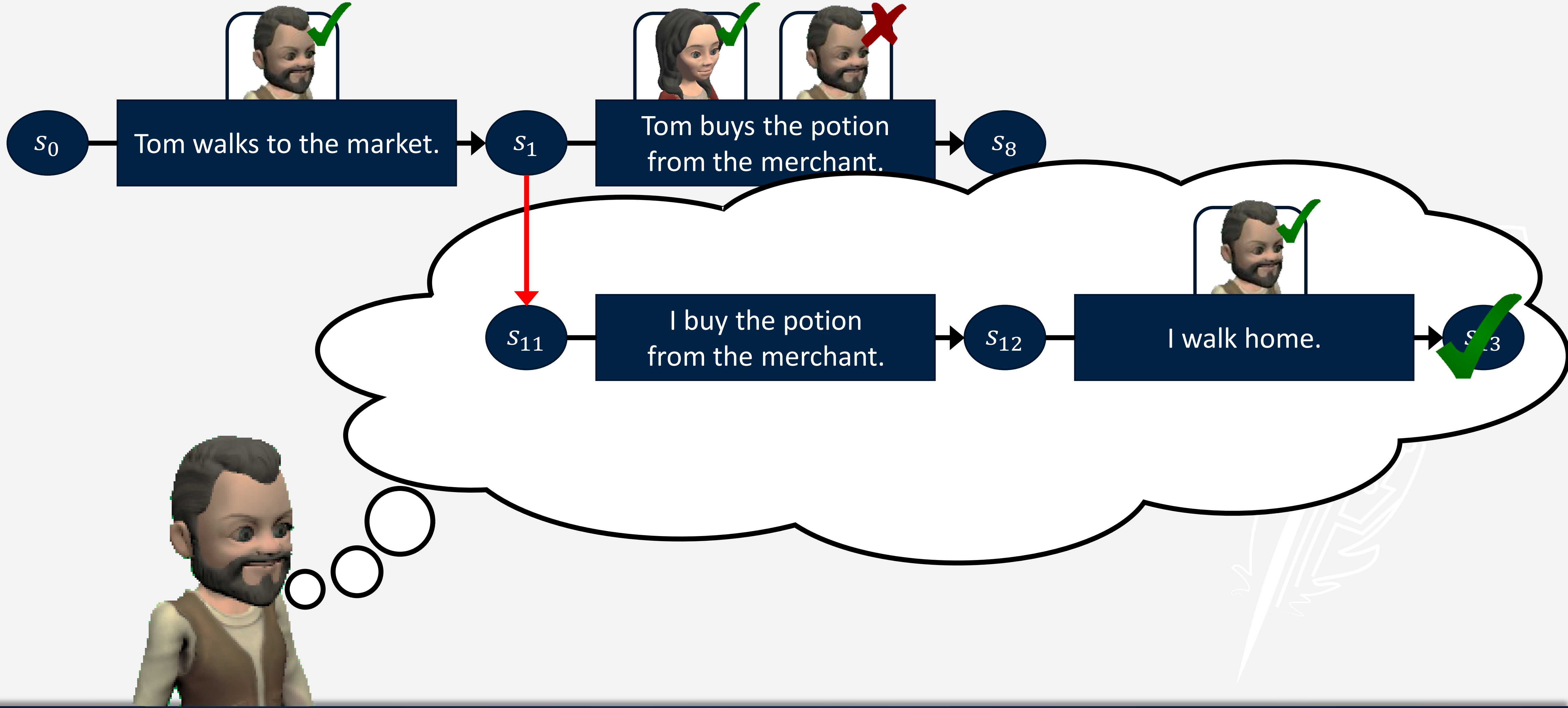

$s_0$
Tom walks to the market.
$s_1$
Tom buys the potion from the merchant.
$s_8$

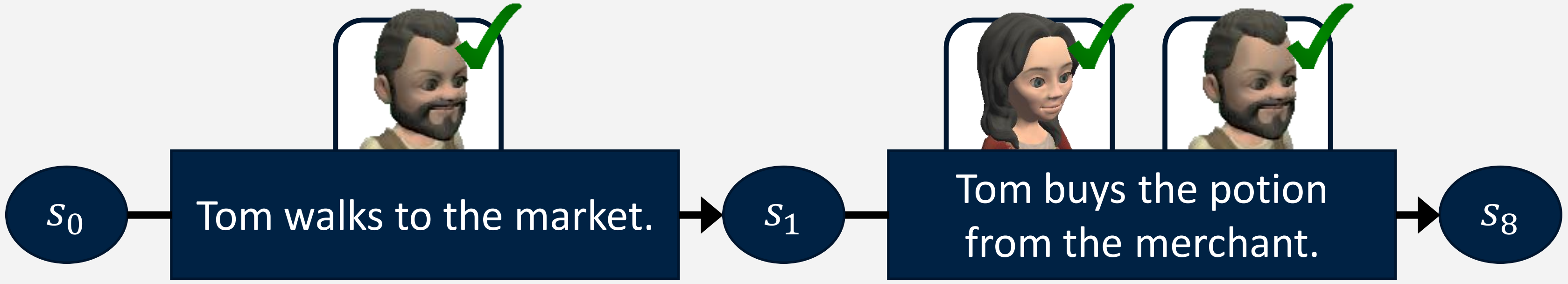


$s_0$
Tom walks to the market.
$s_1$
Tom buys the potion from the merchant.
$s_8$
$s_9$
Tom buys the potion from me.

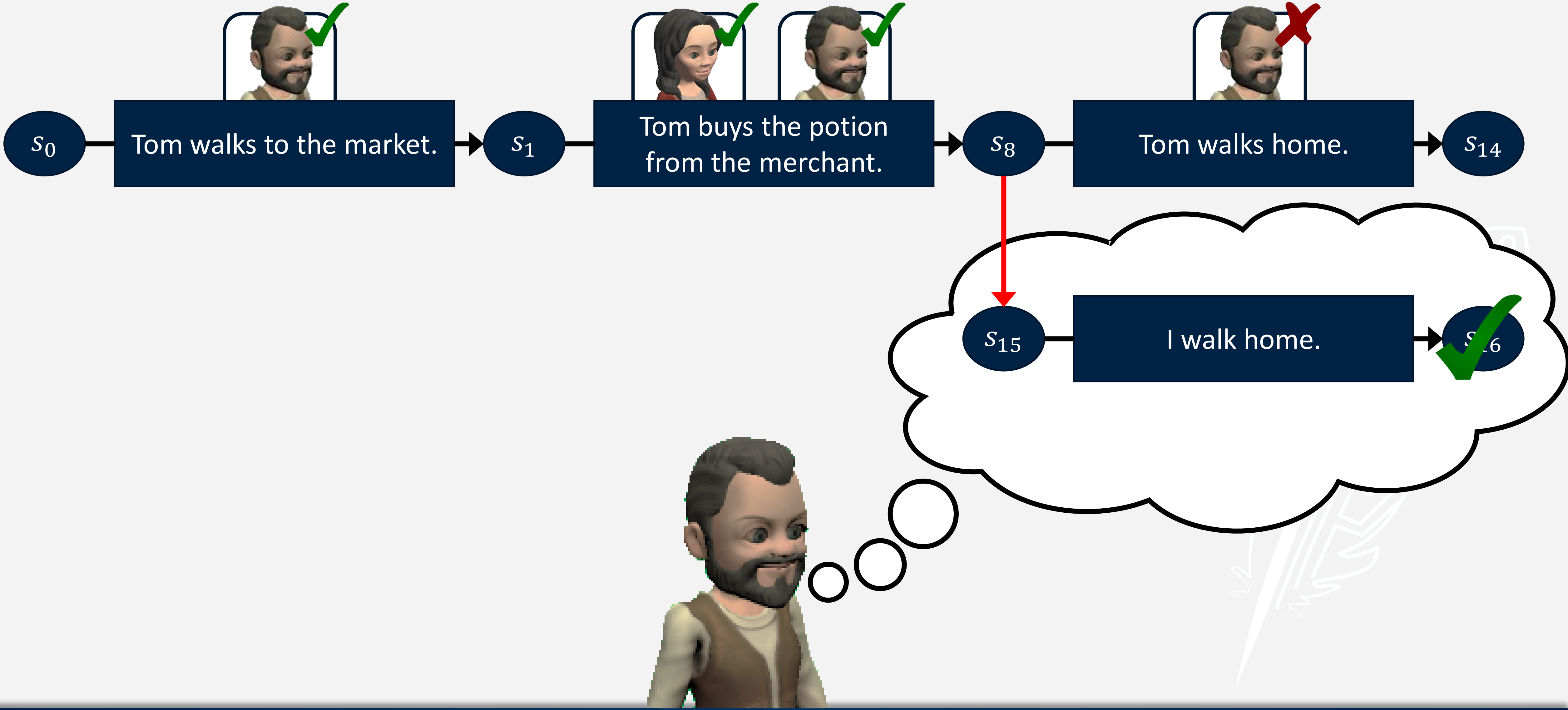

$s_0$
Tom walks to the market.
$s_1$
Tom buys the potion from the merchant.
$s_8$


$s_0$
Tom walks to the market.
$s_1$
Tom buys the potion from the merchant.
$s_8$
$s_{11}$
I buy the potion from the merchant.
$s_{12}$
I walk home.


$s_0$
Tom walks to the market.
$s_1$
Tom buys the potion from the merchant.
$s_8$
$s_{11}$
I buy the potion from the merchant.
$s_{12}$
I walk home.


$s_0$
Tom walks to the market.
$s_1$
Tom buys the potion from the merchant.
$s_8$


$s_0$
Tom walks to the market.
$s_1$
Tom buys the potion from the merchant.
$s_8$
Tom walks home.
$s_{14}$



$s_0$
Tom walks to the market.
$s_1$
Tom buys the potion from the merchant.
$s_8$
Tom walks home.
$s_{14}$
$s_{15}$
I walk home.
$s_{16}$


$s_0$
Tom walks to the market.
$s_1$
Tom buys the potion from the merchant.
$s_8$
Tom walks home.

# Evaluation

# Comparing Sabre to Other Planners

| | Centralized | Intentions | Beliefs | Uncertainty |
|---|---|---|---|---|
| Sabre | ✓ | ✓ | ✓ | ✗ |

# Comparing Sabre to Other Planners

| | Centralized | Intentions | Beliefs | Uncertainty |
|---|---|---|---|---|
| Sabre | ✓ | ✓ | ✓ | ✗ |
| Glaive | ✓ | ✓ | ✗ | ✗ |


- Riedl and Young, "Narrative planning: balancing plot and character," in JAIR 2010
- Ware and Young, "CPOCL: a narrative planner supporting conflict," in AIIDE 2011
- Teutenberg and Porteous, "Efficient intent-based narrative generation...," in AAMAS 2013
- Ware and Young, "Glaive: a state-space narrative planner...," in AIIDE 2014

# Comparing Sabre to Other Planners

| | Centralized | Intentions | Beliefs | Uncertainty |
|---|---|---|---|---|
| Sabre | ✓ | ✓ | ✓ | ✗ |
| Glaive | ✓ | ✓ | ✗ | ✗ |
| HeadSpace | ✓ | ✗ | ~✓ | ✗ |

- Thorne and Young, “Generating stories … by modeling false character beliefs,” in INT 2017

# Comparing Sabre to Other Planners

| | Centralized | Intentions | Beliefs | Uncertainty |
|---|---|---|---|---|
| Sabre | ✓ | ✓ | ✓ | ✗ |
| Glaive | ✓ | ✓ | ✗ | ✗ |
| HeadSpace | ✓ | ✗ | ~✓ | ✗ |
| IMPRACTical | ✓ | ✓ | ~✓ | ✗ |

- Teutenberg and Porteous, "Incorporating global and local knowledge…," in AAMAS 2015

# Comparing Sabre to Other Planners

| | Centralized | Intentions | Beliefs | Uncertainty |
|---|---|---|---|---|
| Sabre | ✓ | ✓ | ✓ | ✗ |
| Glaive | ✓ | ✓ | ✗ | ✗ |
| HeadSpace | ✓ | ✗ | ~✓ | ✗ |
| IMPRACTical | ✓ | ✓ | ~✓ | ✗ |
| Thespian | ✗ | ✓ | ✓ | ✓ |

- Ryan, Summerville, Mateas, and Wardrip-Fruin, “Toward characters who observe…,” in EXAG 2015
- Si and Marsella, “Encoding Theory of Mind in character design…,” in AHCI 2014

# Comparing Sabre to Other Planners

| | Centralized | Intentions | Beliefs | Uncertainty |
|---|---|---|---|---|
| Sabre | ✓ | ✓ | ✓ | ✗ |
| Glaive | ✓ | ✓ | ✗ | ✗ |
| HeadSpace | ✓ | ✗ | ~✓ | ✗ |
| IMPRACTical | ✓ | ✓ | ~✓ | ✗ |
| Thespian | ✗ | ✓ | ✓ | ✓ |
| Ostari | ✓ | ✓ | ✓ | ✓ |

- Eger and Martens, “Practical specification of belief manipulation in games,” in AIIDE 2017

# Test Problems

- *Raiders*
- *Space*

- Ware and Young, “Glaive: a state-space narrative planner…,” in AIIDE 2014

# Test Problems

- *Raiders*
- *Space*
- *Treasure*
- *Lovers*
- *Hubris*


- Farrell and Ware, "Narrative planning for belief and intention recognition," in AIIDE 2020
- Shirvani, Farrell, and Ware, "Combining intentionality and belief …," in AIIDE 2018
- Christensen, Nelson, and Cardona-Rivera, "Using domain compilation to add belief …," in AIIDE 2020

# Test Problems

- *Raiders*
- *Space*
- *Treasure*
- *Lovers*
- *Hubris*
- *BearBirdJr*


- Sack, “Micro-TaleSpin, a story generator,” 1992
- Meehan, “TALE-SPIN, an interactive program that writes stories,” in AAAI 1977

# Test Problems

- *Raiders*
- *Space*
- *Treasure*
- *Lovers*
- *Hubris*
- *BearBirdJr*
- ***Grandma***

- Ware, Garcia, Shirvani, and Farrell, “Multi-agent experience management …,” in AIIDE 2019

# Results

| Domain | Nodes Generated | Time |
|---|---:|---:|
| *Raiders* | 17,815 | 1.4 s |
| *Space* | 192 | 6 ms |
| *Treasure* | 288 | 1 ms |
| *Lovers* | 5,198,414 | 40.3 m |
| *Hubris* | 831 | 47 ms |
| *BearBirdJr* | 34,084,068 | 14.0 m |
| *Grandma* | 105,178,466 | 6.2 h |

# Conclusion

# Limitations

- No true uncertainty
- $h^+$ heuristic often performs poorly[1]

1. Bonet and Geffner, "Planning as heuristic search," in AI, 2001

# Future Work

- More search methods

**Algorithm 1** The Sabre algorithm

Let $\mathcal{A}$ be the set of all actions defined in the domain.
$\textsc{Sabre}(c_{author}, s_0, \emptyset, s_0)$
**function** $\textsc{Sabre}(c, r, \pi, s)$
    **Input:** character $c$, start state $r$, plan $\pi$, current state $s$
    **if** $u(c, s) > u(c, r)$ and $\pi$ is non-redundant **then**
        **return** $\pi$
    Choose an action $a \in \mathcal{A}$ such that $s \models \textsc{pre}(a)$
    **for all** $c' \in \textsc{con}(a)$ such that $c' \neq c$ **do**
        Let state $b = \alpha(a, \beta(c', s))$.
        **if** $b$ is undefined **then return** failure.
        **else if** $\textsc{Sabre}(c', b, \emptyset, b)$ fails **then return** failure.
    **return** $\textsc{Sabre}(c, r, \pi \cup a, \alpha(a, s))$

# Future Work

- More search methods

**Algorithm 1** The Sabre algorithm

Let $\mathcal{A}$ be the set of all actions defined in the domain.
SABRE($c_{author}$, $s_0$, $\emptyset$, $s_0$)
**function** SABRE($c$, $r$, $\pi$, $s$)
  **Input:** character $c$, start state $r$, plan $\pi$, current state $s$
  **if** $u(c, s) > u(c, r)$ and $\pi$ is non-redundant **then**
    **return** $\pi$
  Choose an action $a \in \mathcal{A}$ such that $s \models$ PRE($a$).
  **for all** $c' \in$ CON($a$) such that $c' \neq c$ **do**
    Let state $b = \alpha(a, \beta(c', s))$.
    **if** $b$ is undefined **then return** failure.
    **else if** SABRE($c'$, $b$, $\emptyset$, $b$) fails **then return** failure.
  **return** $\boxed{\text{SABRE}(c, r, \pi \cup a, \alpha(a, s))}$

**Algorithm 2** The Sabre algorithm

Let $\mathcal{A}$ be the set of all actions defined in the domain.
SABRE($c_{author}$, $s_0$, $\emptyset$, $s_0$)
**function** SABRE($c$, $r$, $\pi$, $s$)
  **Input:** character $c$, start state $r$, plan $\pi$, current state $s$
  **if** $u(c, s) > u(c, r)$ and $\pi$ is non-redundant **then**
    **return** $\pi$
  Choose an action $a \in \mathcal{A}$ such that $s \models$ PRE($a$).
  $\boxed{\textbf{if } \text{SABRE}(c, r, \pi \cup a, \alpha(a, s)) \text{ fails } \textbf{then return} \text{ failure.}}$
  **for all** $c' \in$ CON($a$) such that $c' \neq c$ **do**
    Let state $b = \alpha(a, \beta(c', s))$.
    **if** $b$ is undefined **then return** failure.
    **else if** SABRE($c'$, $b$, $\emptyset$, $b$) fails **then return** failure.
  **return** $\pi$

# Future Work

- More search methods
- Better heuristics
- Agent emotions and personalities[1]

1. Shirvani and Ware, "A formalization of emotional planning for strong-story systems," in AIIDE 2020

SABRE
NARRATIVE PLANNER

http://cs.uky.edu/~sgware/projects/sabre

Background Music: https://www.bensound.com