Planning for Autonomous Robots

Part 2: *Deterministic Planning*

Professor Nick Hawes
GOALS Research Group
Oxford Robotics Institute, Department of Engineering Science, University of Oxford

Pembroke College

Overview

- Classical Planning: STRIPS / PDDL
- Forward Search
- Backward Search
- Planning Heuristics
- The Planning Graph
- Partial Order Planning

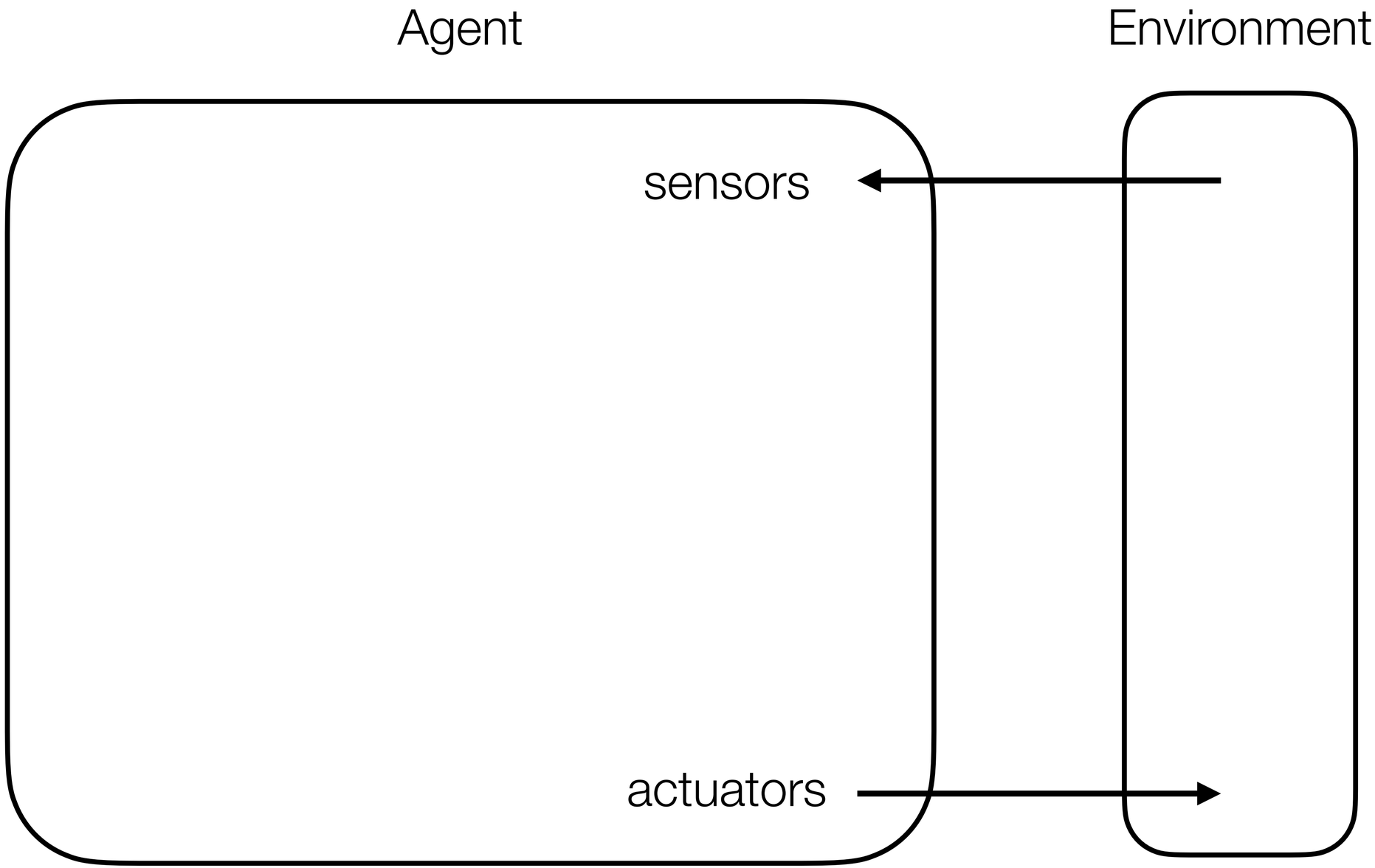
Robot mission **planning** has been a major part of the history of **AI** *and* **robotics**

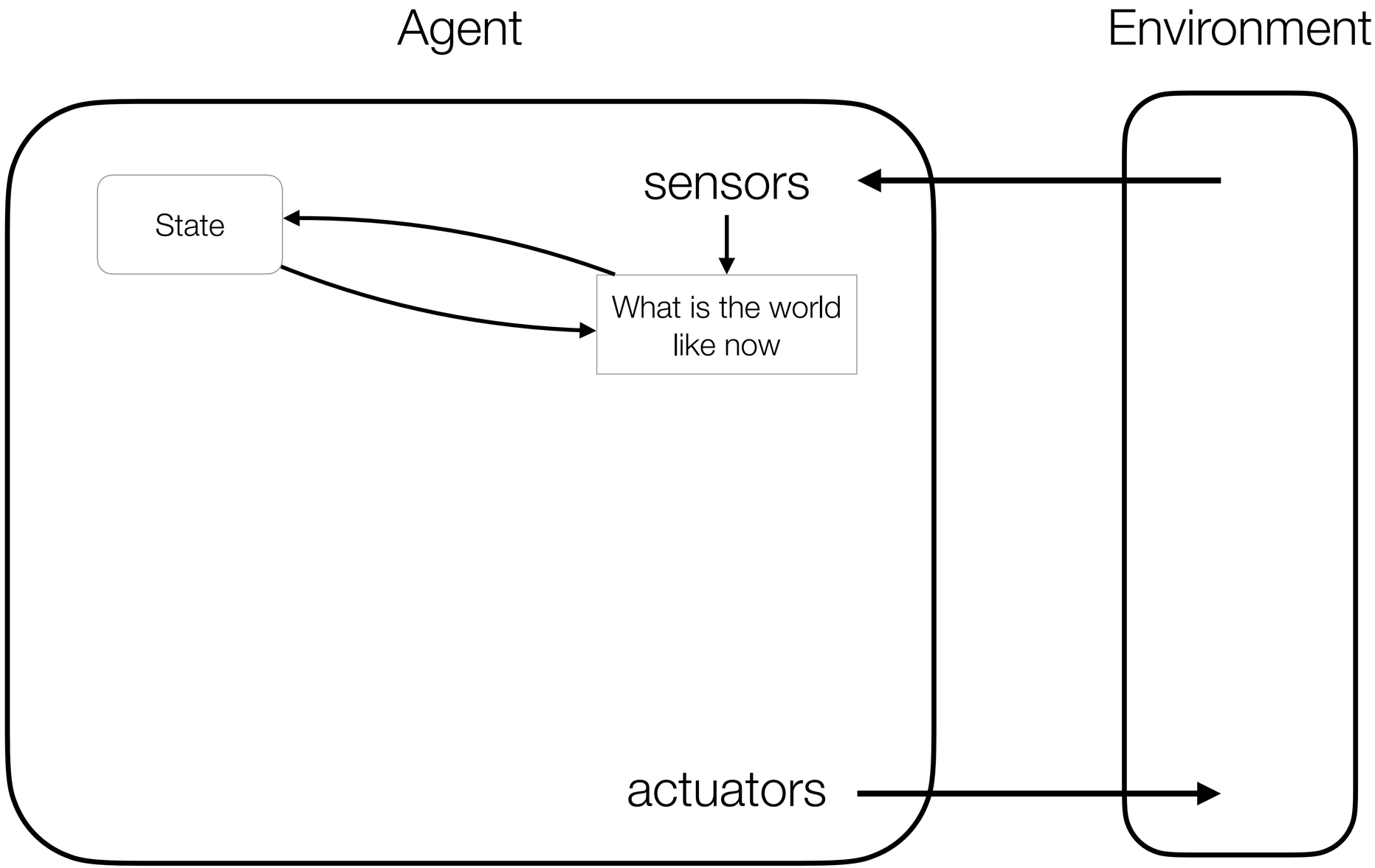
Shakey: SRI, 1966

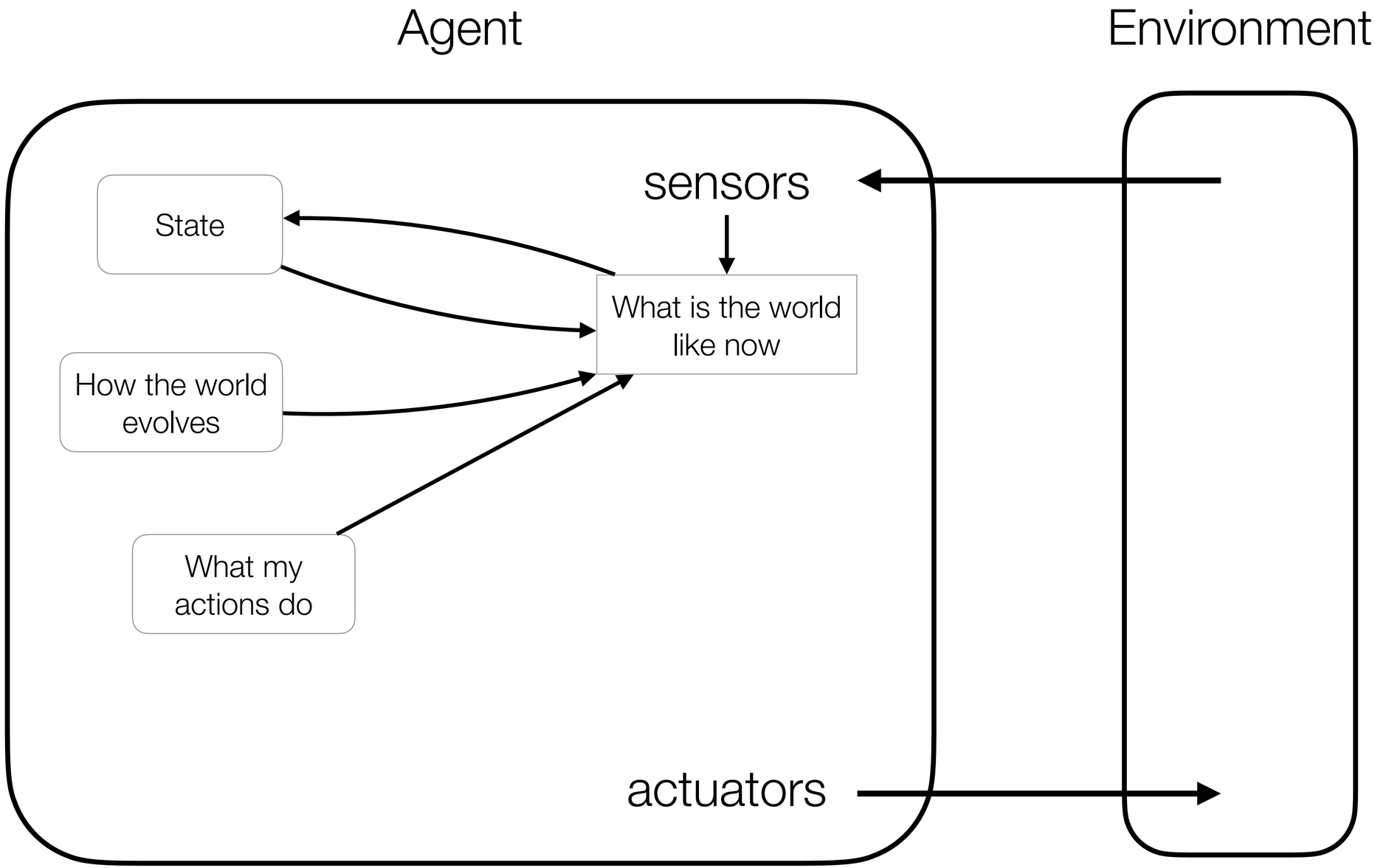
Robot mission **planning** has been a major part of the history of **AI** *and* **robotics**

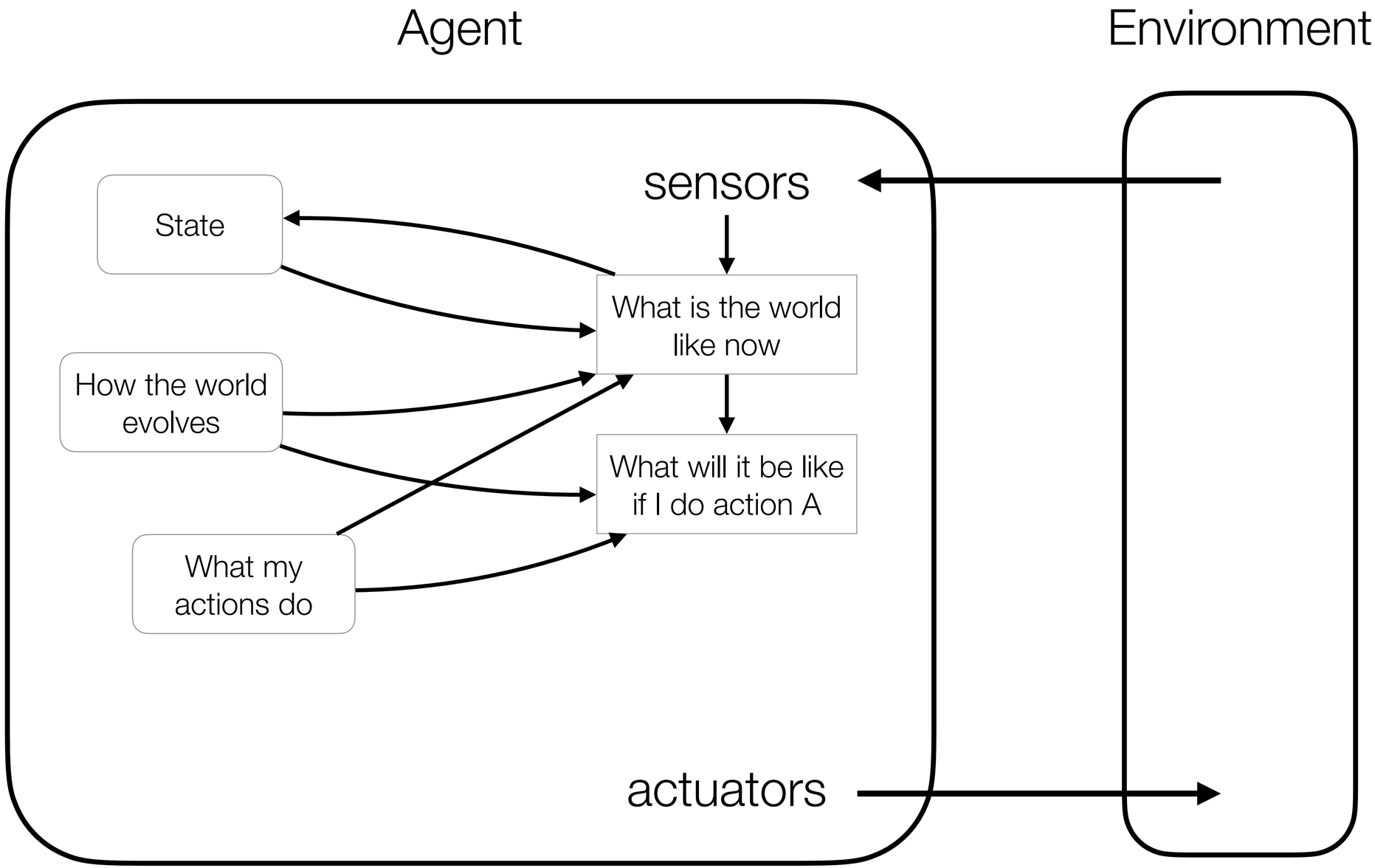
Shakey: SRI, 1966

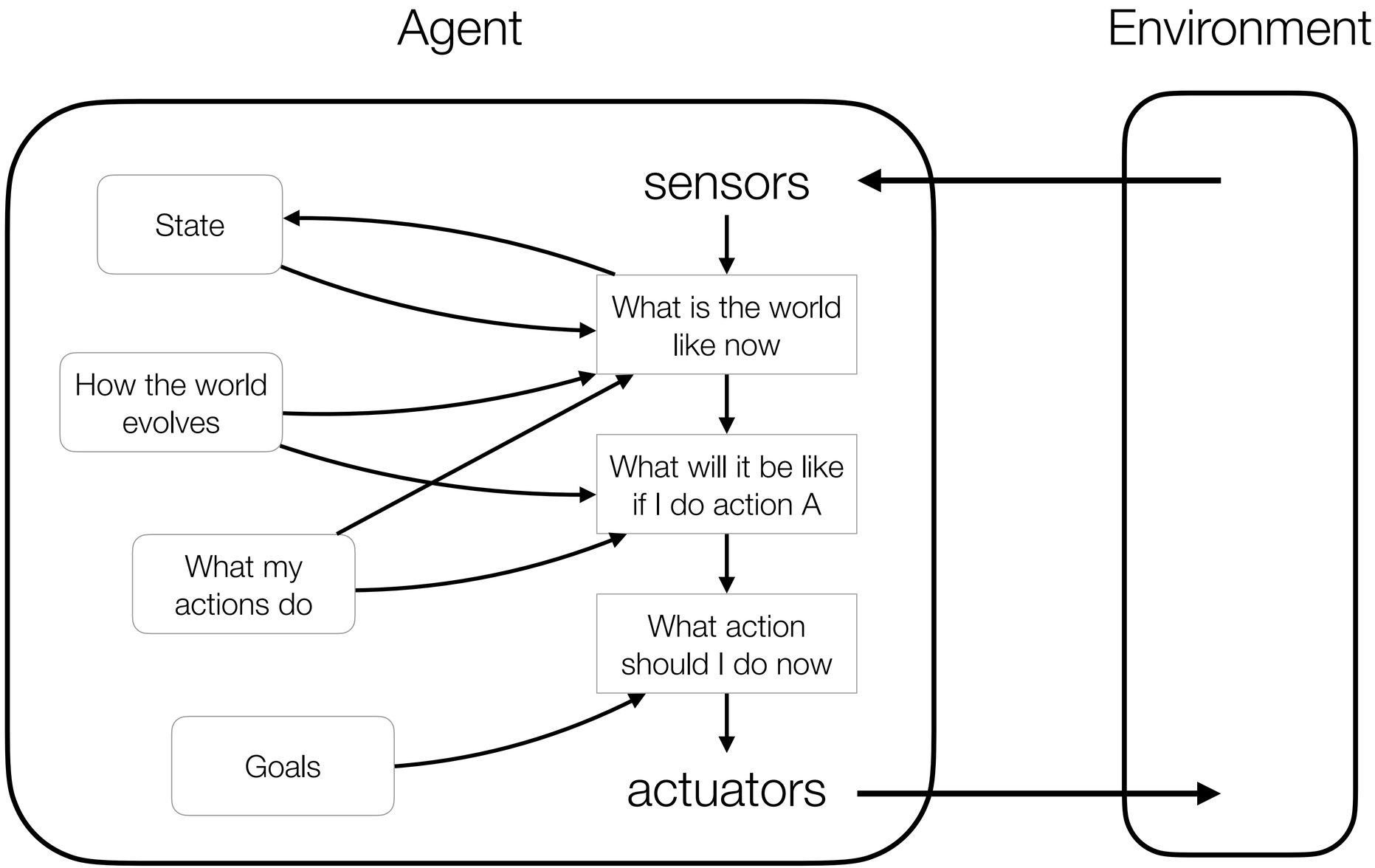

Agent
Environment


Agent
Environment
sensors
actuators


Agent
Environment
sensors
State
What is the world like now
actuators


Agent
Environment
sensors
State
What is the world like now
How the world evolves
What my actions do
actuators


Agent
Environment
sensors
State
What is the world like now
How the world evolves
What will it be like if I do action A
What my actions do
actuators


Agent
Environment
sensors
State
What is the world like now
How the world evolves
What will it be like if I do action A
What my actions do
What action should I do now
Goals
actuators

Planning is the process of creating a **sequence of actions** to achieve an agent's **goals**

Planning is the process of creating a **sequence of actions** to achieve an agent's **goals**

Why is it necessary for an **autonomous system** to have this capability?

Planning is the process of creating a **sequence of actions** to achieve an agent's **goals**

Why is it necessary for an **autonomous system** to have this capability?

- to achieve a **goal** in the face of **online variation**

Planning is the process of creating a **sequence of actions** to achieve an agent's **goals**

Why is it necessary for an **autonomous system** to have this capability?

- to achieve a **goal** in the face of **online variation**
- to achieve a **goal** where simple reactive behaviour leads to poor behaviour, e.g. resource management, local minima

Planning is the process of creating a **sequence of actions** to achieve an agent's **goals**

Why is it necessary for an **autonomous system** to have this capability?

- to achieve a **goal** in the face of **online variation**
- to achieve a **goal** where simple reactive behaviour leads to poor behaviour, e.g. resource management, local minima
- to know what you're going to do **before you do it**: verification, communication, coordination

Planning is the process of creating a **sequence of actions** to achieve an agent's **goals**

Planning is the process of creating a **sequence of actions** to achieve an agent's **goals**

Why not just use **search** or **logical inference**?

Planning is the process of creating a **sequence of actions** to achieve an agent's **goals**

Why not just use **search** or **logical inference**?

Logical inference does not scale well to larger problems

Planning is the process of creating a **sequence of actions** to achieve an agent's **goals**

Why not just use **search** or **logical inference**?

Logical inference does not scale well to larger problems

Search requires **domain-specific heuristics** to be effective

Planning overcomes these problems by using:

Planning overcomes these problems by using:

a **factored representation** — one that represents the world by a collection of variables

Planning overcomes these problems by using:

a **factored representation** — one that represents the world by a collection of variables

an **action representation** — that lifts reasoning to a subset of first-order logic, compactly expressing a range of **successor states**

Classical Planning

Classical Planning

fully observable

Classical Planning

fully observable

deterministic

Classical Planning

fully observable

deterministic

static environments

Classical Planning

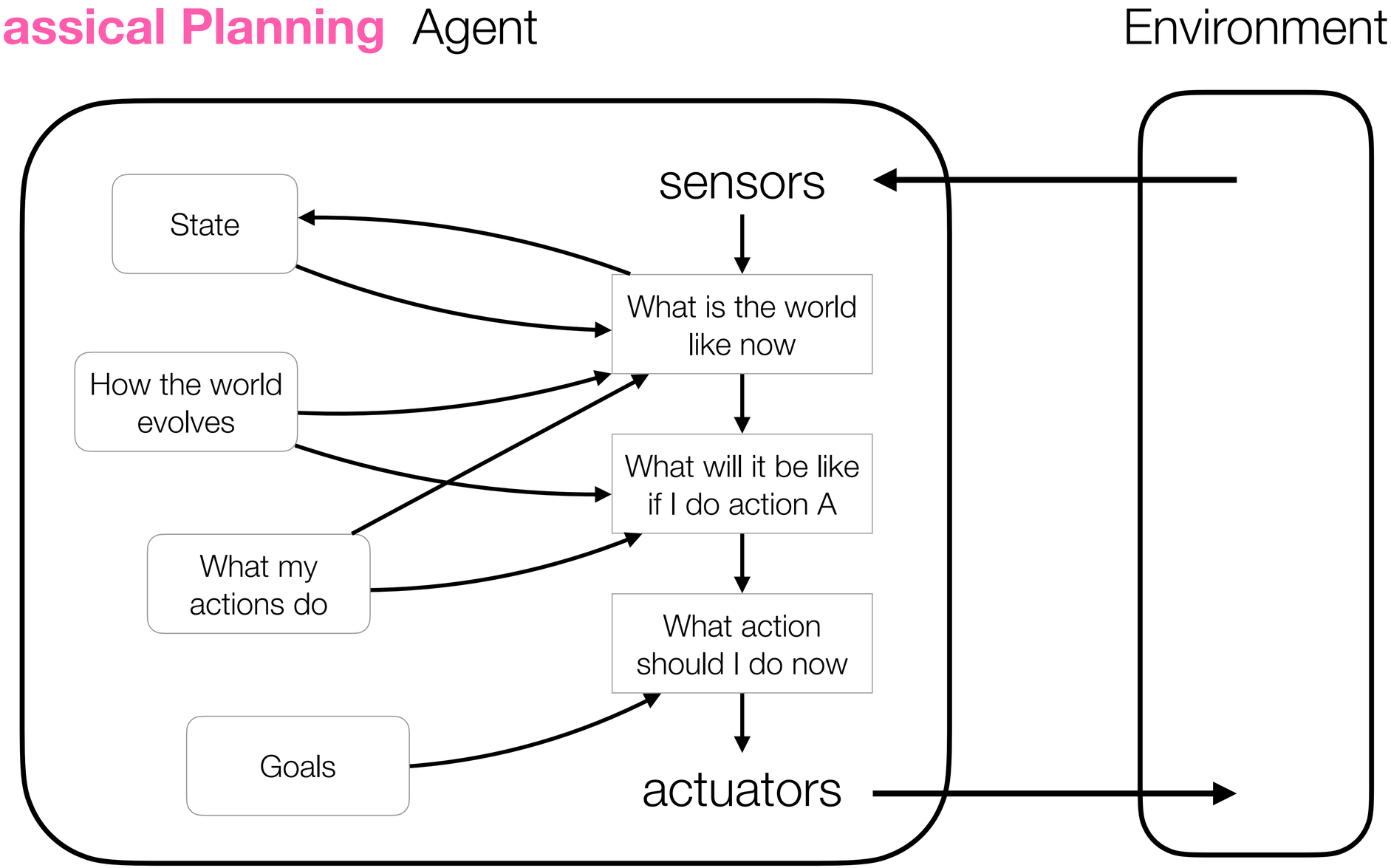
fully observable

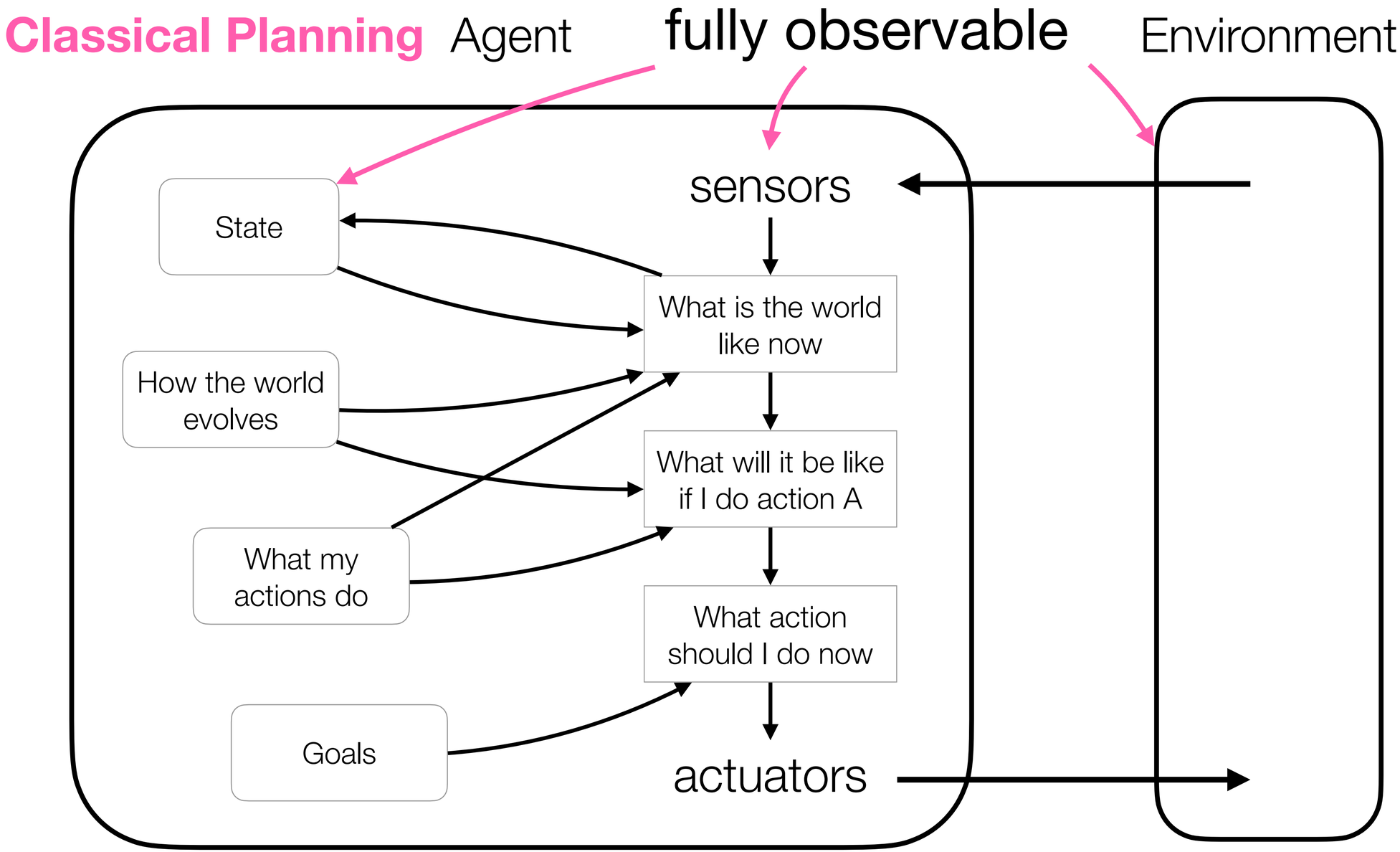
deterministic

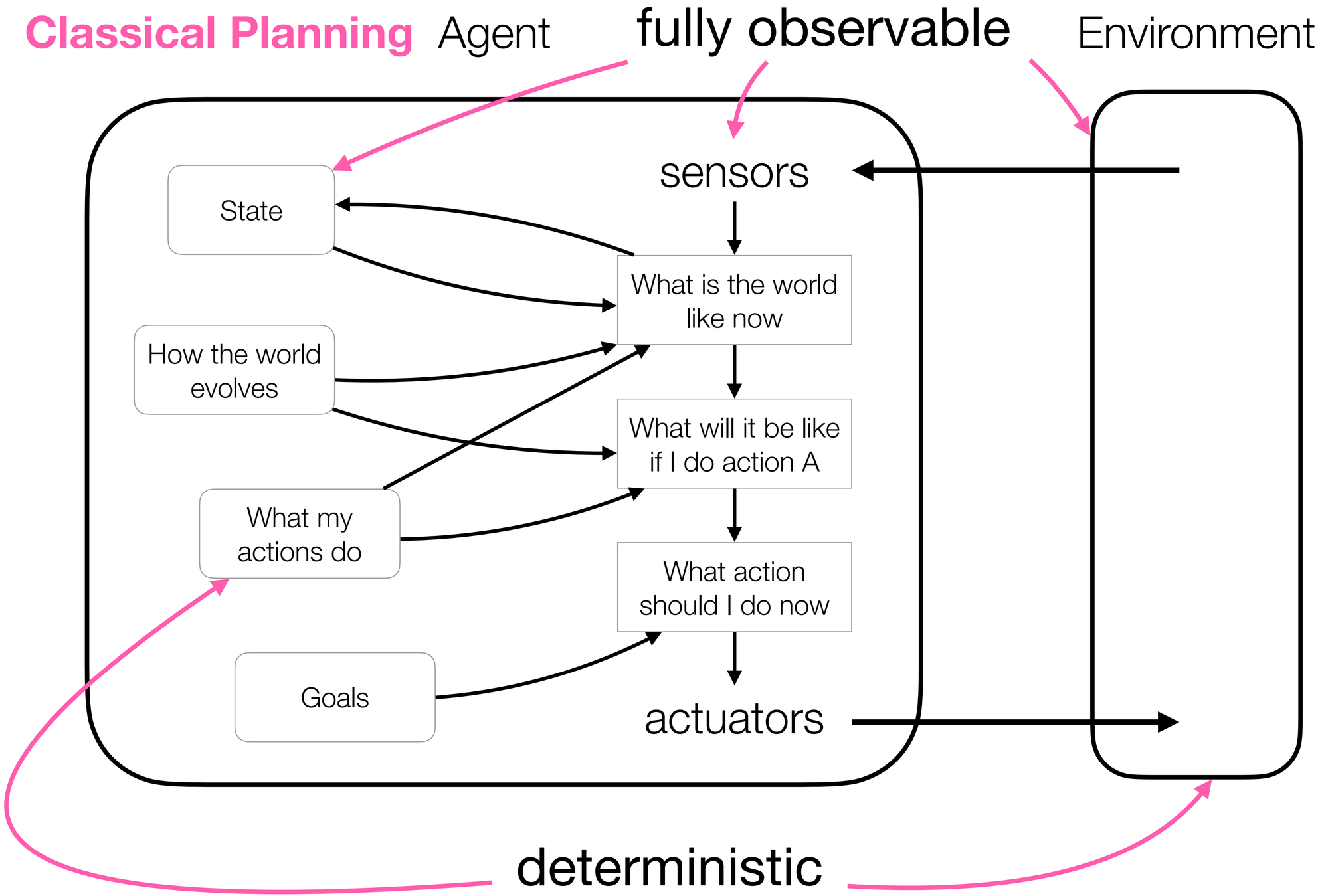
static environments

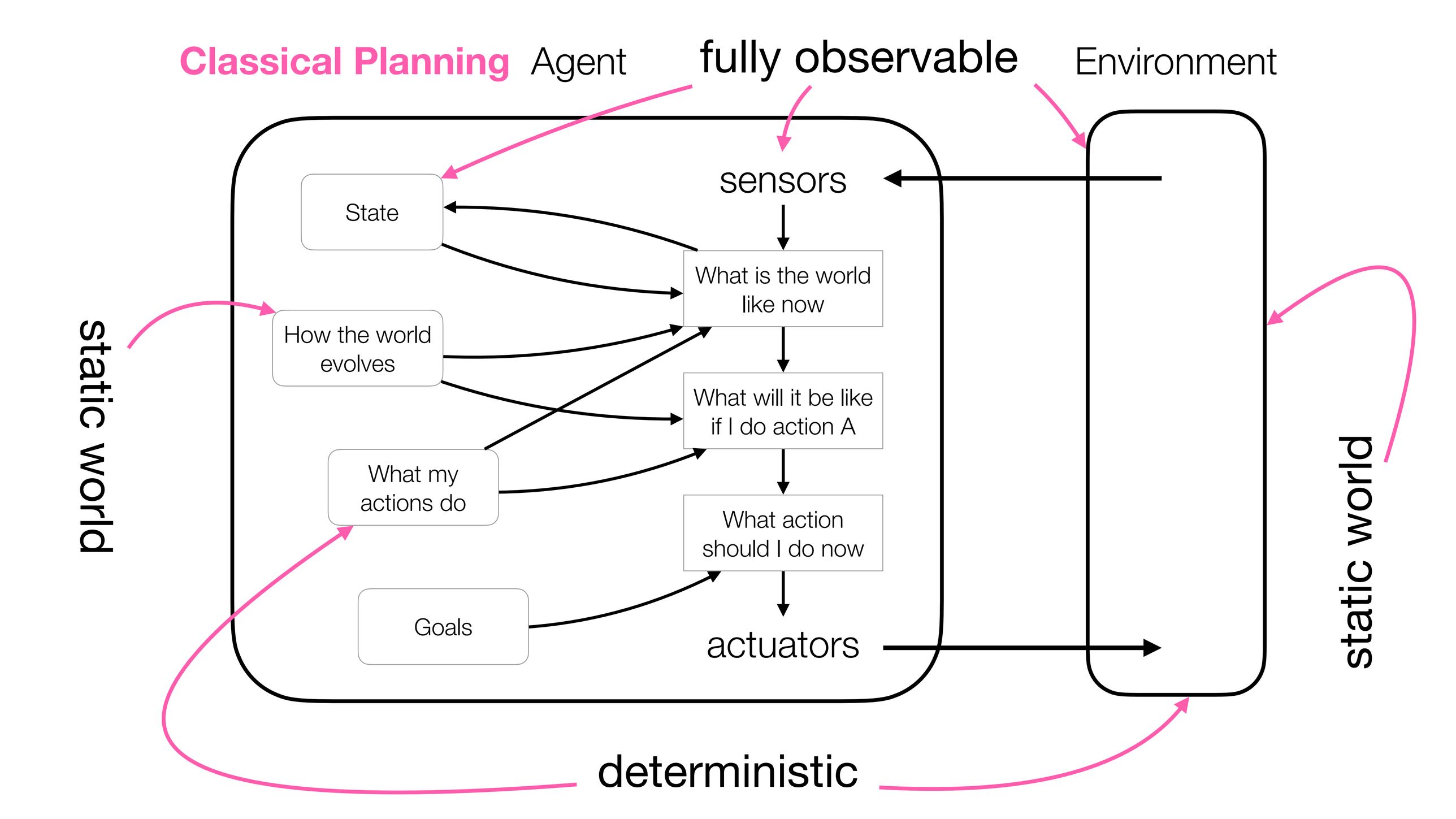
single agent

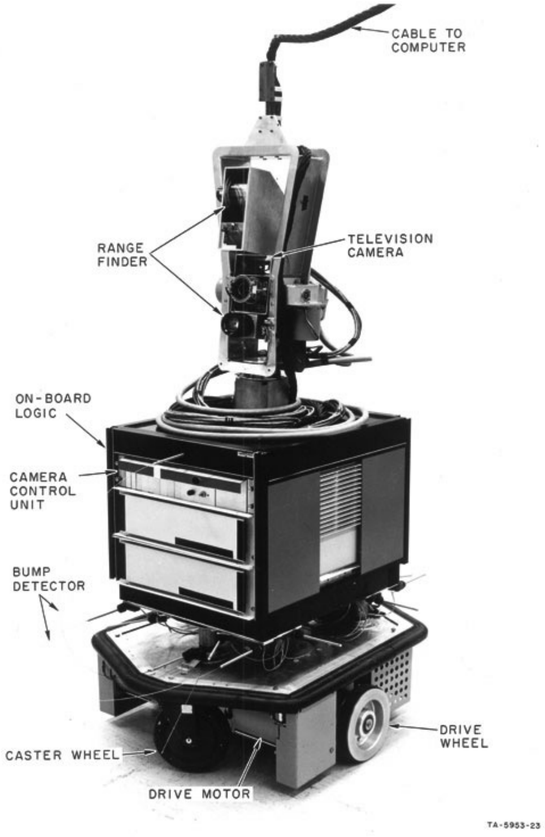
Classical Planning




Classical Planning
Agent
fully observable
Environment
sensors
State
What is the world like now
How the world evolves
What will it be like if I do action A
What my actions do
What action should I do now
Goals
actuators

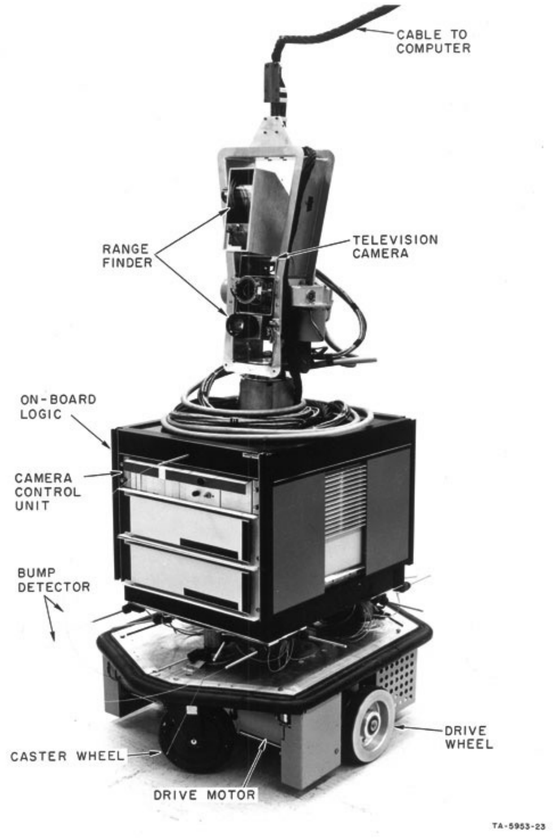

Classical Planning
Agent
fully observable
Environment
sensors
State
What is the world like now
How the world evolves
What will it be like if I do action A
What my actions do
What action should I do now
Goals
actuators
deterministic


Classical Planning
Agent
fully observable
Environment
sensors
State
What is the world like now
How the world evolves
What will it be like if I do action A
What my actions do
What action should I do now
Goals
actuators
static world
static world
deterministic


CABLE TO COMPUTER
RANGE FINDER
TELEVISION CAMERA
ON-BOARD LOGIC
CAMERA CONTROL UNIT
BUMP DETECTOR
DRIVE WHEEL
CASTER WHEEL
DRIVE MOTOR
TA-5953-23

STRIPS: **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver

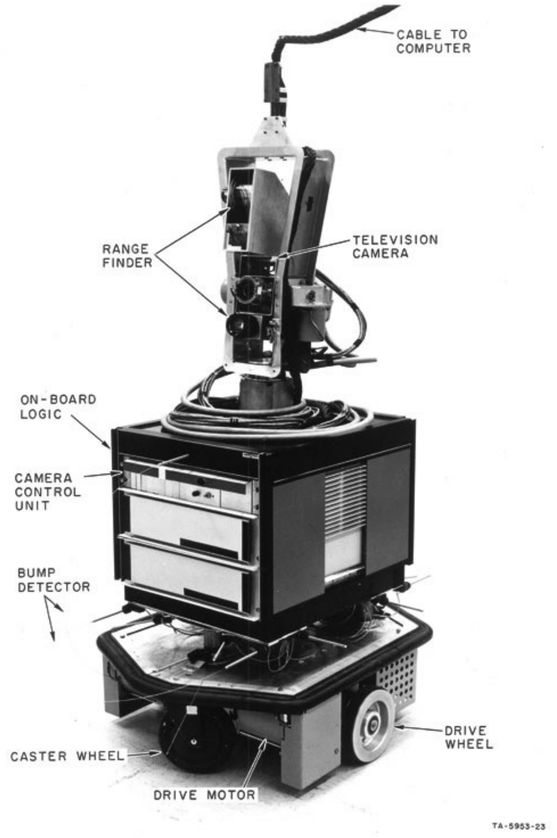
PDDL: **P**lanning **D**omain **D**efinition **L**anguage



STRIPS: **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver

PDDL: **P**lanning **D**omain **D**efinition **L**anguage

States are represented by **conjunctions** of **atoms** (positive literals)

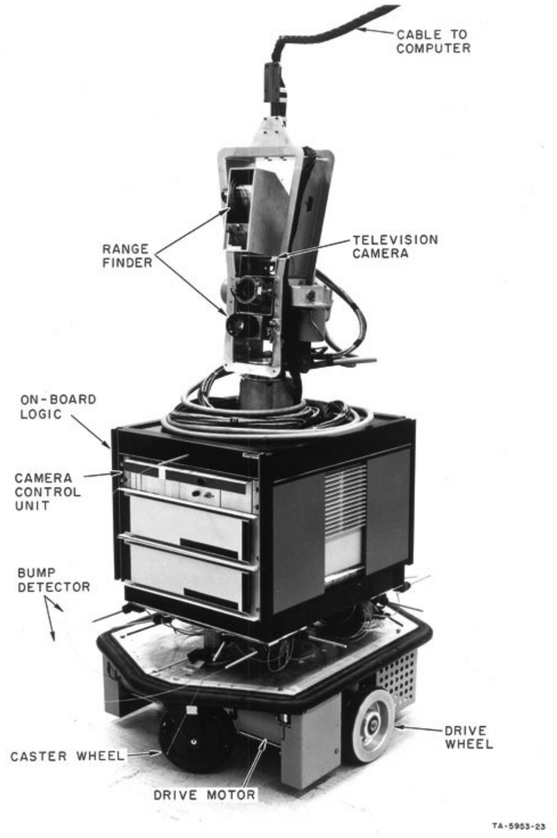


STRIPS: **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver

PDDL: **P**lanning **D**omain **D**efinition **L**anguage

States are represented by **conjunctions** of **atoms** (positive literals)

The **closed-world assumption** means that *anything not positively represented is false*



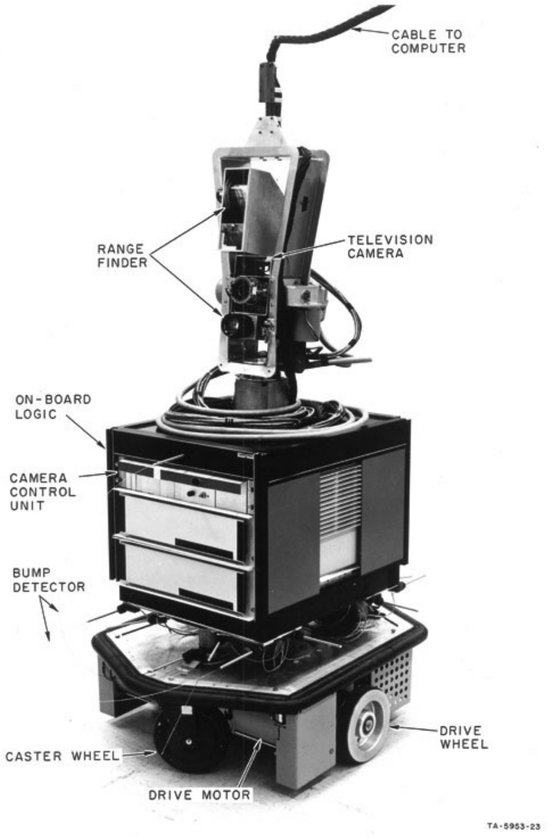
STRIPS: **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver

PDDL: **P**lanning **D**omain **D**efinition **L**anguage

States are represented by **conjunctions** of **atoms** (positive literals)

The **closed-world assumption** means that *anything not positively represented is false*

This state representation is designed to support **efficient processing** using **logical inference** or **set semantics**



E.g. the following **state** from a ***logistics domain***

$$\texttt{Plane}(P_1) \land \texttt{Plane}(P_2) \land$$
$$\texttt{Airport(BHX)} \land \texttt{Airport(FCO)} \land$$
$$\texttt{At}(P_1, \texttt{BHX}) \land \texttt{At}(P_2, \texttt{FCO})$$

… means that the following things are also **true**:

E.g. the following **state** from a ***logistics domain***

$$\begin{gathered}\texttt{Plane}(P_1) \wedge \texttt{Plane}(P_2) \wedge \\ \texttt{Airport(BHX)} \wedge \texttt{Airport(FCO)} \wedge \\ \texttt{At}(P_1, \texttt{BHX}) \wedge \texttt{At}(P_2, \texttt{FCO})\end{gathered}$$

… means that the following things are also **true**:

$$\neg\texttt{Plane(BHX)}$$

$$\neg\texttt{Airport}(P_2)$$

$$\neg\texttt{At(FCO,}\ P_2)$$

etc.

STRIPS: **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver

PDDL: **P**lanning **D**omain **D**efinition **L**anguage

STRIPS: **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver

PDDL: **P**lanning **D**omain **D**efinition **L**anguage

Actions describe only **what changes** in the state when they are applied

STRIPS: **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver

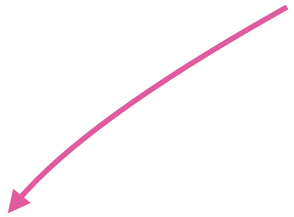
PDDL: **P**lanning **D**omain **D**efinition **L**anguage

Actions describe only **what changes** in the state when they are applied

They use a **schema form** to compactly represent a range of **ground** actions

$$
\begin{aligned}
&\mathit{Action}(\ \mathit{Fly}(p, \mathit{from}, \mathit{to}),\\
&\quad \text{PRECOND: } \mathrm{At}(p, \mathit{from}) \wedge \mathrm{Plane}(p) \wedge\\
&\qquad\qquad \mathrm{Airport}(\mathit{from}) \wedge \mathrm{Airport}(\mathit{to})\\
&\quad \text{EFFECT: } \neg\mathrm{At}(p, \mathit{from}) \wedge \mathrm{At}(p, \mathit{to}))
\end{aligned}
$$

Action **name** and **signature**

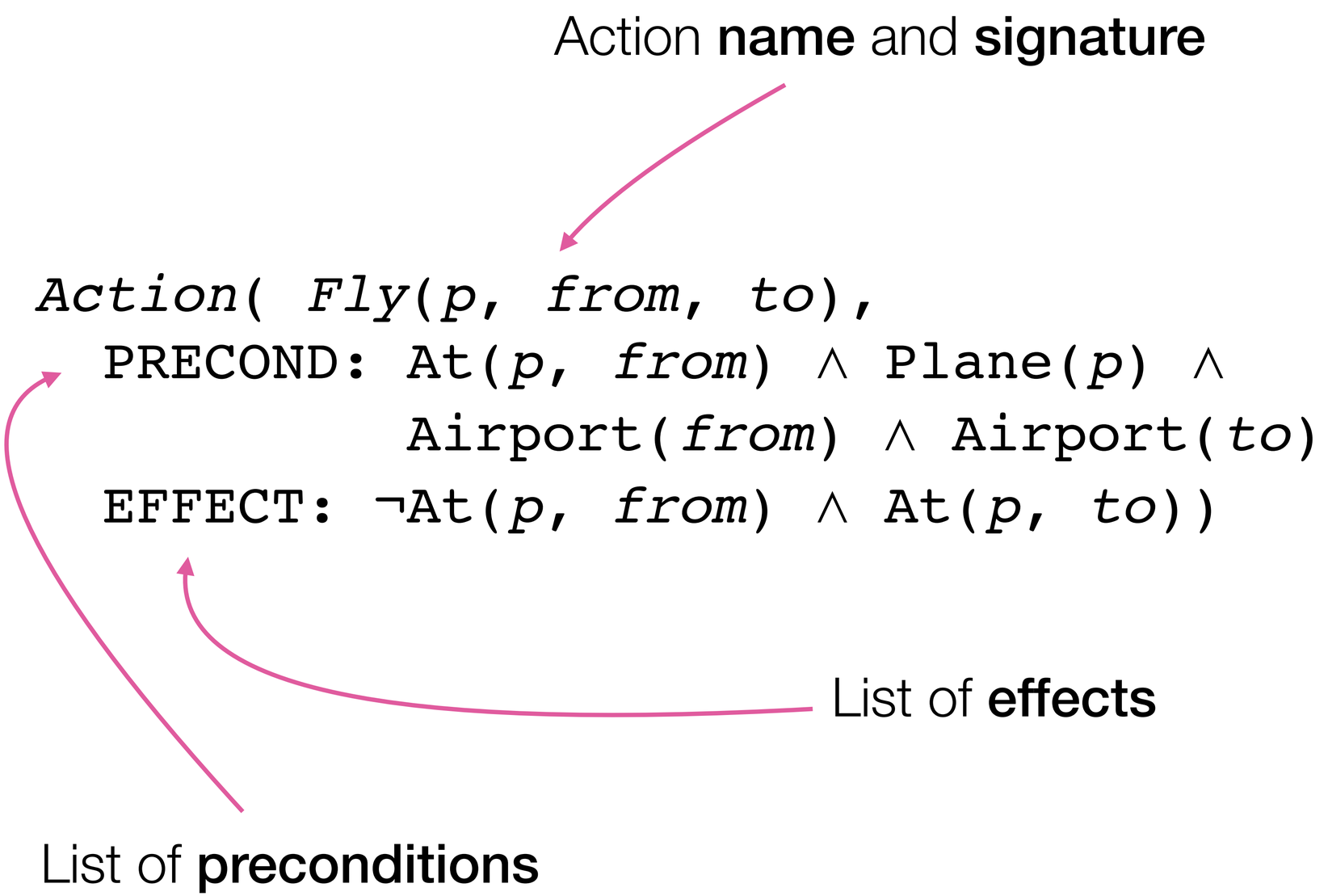


```
Action( Fly(p, from, to),
  PRECOND: At(p, from) ∧ Plane(p) ∧
           Airport(from) ∧ Airport(to)
  EFFECT: ¬At(p, from) ∧ At(p, to))
```

Action **name** and **signature**

```
Action( Fly(p, from, to),
  PRECOND: At(p, from) ∧ Plane(p) ∧
           Airport(from) ∧ Airport(to)
  EFFECT: ¬At(p, from) ∧ At(p, to))
```

List of **preconditions**


Action **name** and **signature**
Action(Fly(p, from, to),
PRECOND: At(p, from) ∧ Plane(p) ∧
Airport(from) ∧ Airport(to)
EFFECT: ¬At(p, from) ∧ At(p, to))
List of **effects**
List of **preconditions**

Actions are **instantiated** by substituting
`Values` for the ***`variables`***

Action(*Fly*(P_1, BHX, FCO),
  PRECOND: At(P_1, BHX) ∧ Plane(P_1) ∧
    Airport(BHX) ∧ Airport(FCO)
  EFFECT: ¬At(P_1, BHX) ∧ At(P_1, FCO))

An **action** a is ***applicable*** in a state s when s *entails the preconditions* of a

$s \models q$ iff every positive literal in q is also in s and every negated literal in q is not

An **action** a is ***applicable*** in a state s when s *entails the preconditions* of a

$s \models q$ iff every positive literal in q is also in s and every negated literal in q is not

$$\texttt{Plane}(P_1) \land \texttt{Plane}(P_2) \land \texttt{Airport}(BHX) \land \texttt{Airport}(FCO) \land \texttt{At}(P_1, BHX) \land \texttt{At}(P_2, FCO)$$

$$\models$$

$$\texttt{At}(P_1, BHX) \land \texttt{Plane}(P_1) \land \texttt{Airport}(BHX) \land \texttt{Airport}(FCO)$$

An **action** ***a*** is ***applicable*** in a state ***s*** when ***s*** *entails the preconditions* of ***a***

s ⊨ ***q*** iff every positive literal in ***q*** is also in ***s*** and every negated literal in ***q*** is not

$$\texttt{Plane}(P_1) \land \texttt{Plane}(P_2) \land \texttt{Airport}(\texttt{BHX}) \land \texttt{Airport}(\texttt{FCO}) \land \texttt{At}(P_1, \texttt{BHX}) \land \texttt{At}(P_2, \texttt{FCO})$$

$$\not\models$$

$$\texttt{At}(P_1, \textbf{FCO}) \land \texttt{Plane}(P_1) \land \texttt{Airport}(\texttt{BHX}) \land \texttt{Airport}(\texttt{FCO})$$

An **action** ***a*** is ***applicable*** in a state ***s*** when ***s*** *entails the preconditions* of ***a***

s ⊨ ***q*** iff every positive literal in ***q*** is also in ***s*** and every negated literal in ***q*** is not

$$\texttt{Plane}(P_1) \land \texttt{Plane}(P_2) \land \texttt{Airport}(\texttt{BHX}) \land \texttt{Airport}(\texttt{FCO}) \land \texttt{At}(P_1, \texttt{BHX}) \land \texttt{At}(P_2, \texttt{FCO})$$

$$\not\models$$

$$\mathbf{\neg At(P_1, BHX)} \land \texttt{Plane}(P_1) \land \texttt{Airport}(\texttt{BHX}) \land \texttt{Airport}(\texttt{FCO})$$

Action(*Fly*(P_1, BHX, FCO),
PRECOND: At(P_1, BHX) ∧ Plane(P_1) ∧
Airport(BHX) ∧ Airport(FCO)
EFFECT: ¬At(P_1, BHX) ∧ At(P_1, FCO))

The **result** of applying action ***a*** in state ***s*** is ***s'***
adding positive effect literals **ADD(*a*)**
and removing negative ones **DEL(*a*)**

s' = RESULT(***s***, ***a***) = (***s*** - DEL(***a***)) U ADD(***a***)

Plane(P_1) ∧
Plane(P_2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P_1, BHX) ∧
At(P_2, FCO)

Action(*Fly*(*p*, *from*, *to*),
PRECOND: At(*p*, *from*) ∧ Plane(*p*) ∧
Airport(*from*) ∧ Airport(*to*)
EFFECT: ¬At(*p*, *from*) ∧ At(*p*, *to*))

Which states result from applying *Fly* to this state?

(a) Plane(P_1) ∧
Plane(P_2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P_2, BHX) ∧
At(P_1, FCO)

(b) Plane(P_1) ∧
Plane(P_2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P_1, FCO) ∧
At(P_2, FCO)

(c) Plane(P_1) ∧
Plane(P_2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P_1, BHX) ∧
At(P_2, BHX)

Poll:	(a)	(b)	(c)	(b) & (c)	(a), (b) & (c)

$\text{Plane}(P_1) \wedge$
$\text{Plane}(P_2) \wedge$
$\text{Airport}(BHX) \wedge$
$\text{Airport}(FCO) \wedge$
$\text{At}(P_1, BHX) \wedge$
$\text{At}(P_2, FCO)$

Action(*Fly*(*p*, *from*, *to*),
PRECOND: At(*p*, *from*) ∧ Plane(*p*) ∧
Airport(*from*) ∧ Airport(*to*)
EFFECT: ¬At(*p*, *from*) ∧ At(*p*, *to*))

Which states result from applying *Fly* to this state?

(a) $\text{Plane}(P_1) \wedge$
$\text{Plane}(P_2) \wedge$
$\text{Airport}(BHX) \wedge$
$\text{Airport}(FCO) \wedge$
$\text{At}(P_2, BHX) \wedge$
$\text{At}(P_1, FCO)$

(b) $\text{Plane}(P_1) \wedge$
$\text{Plane}(P_2) \wedge$
$\text{Airport}(BHX) \wedge$
$\text{Airport}(FCO) \wedge$
$\text{At}(P_1, FCO) \wedge$
$\text{At}(P_2, FCO)$

(c) $\text{Plane}(P_1) \wedge$
$\text{Plane}(P_2) \wedge$
$\text{Airport}(BHX) \wedge$
$\text{Airport}(FCO) \wedge$
$\text{At}(P_1, BHX) \wedge$
$\text{At}(P_2, BHX)$

Poll:				(b) & (c)	

Action(*Fly*(P_1, BHX, FCO),
PRECOND: At(P_1, BHX) ∧ Plane(P_1) ∧
Airport(BHX) ∧ Airport(FCO)
EFFECT: ¬At(P_1, BHX) ∧ At(P_1, FCO))

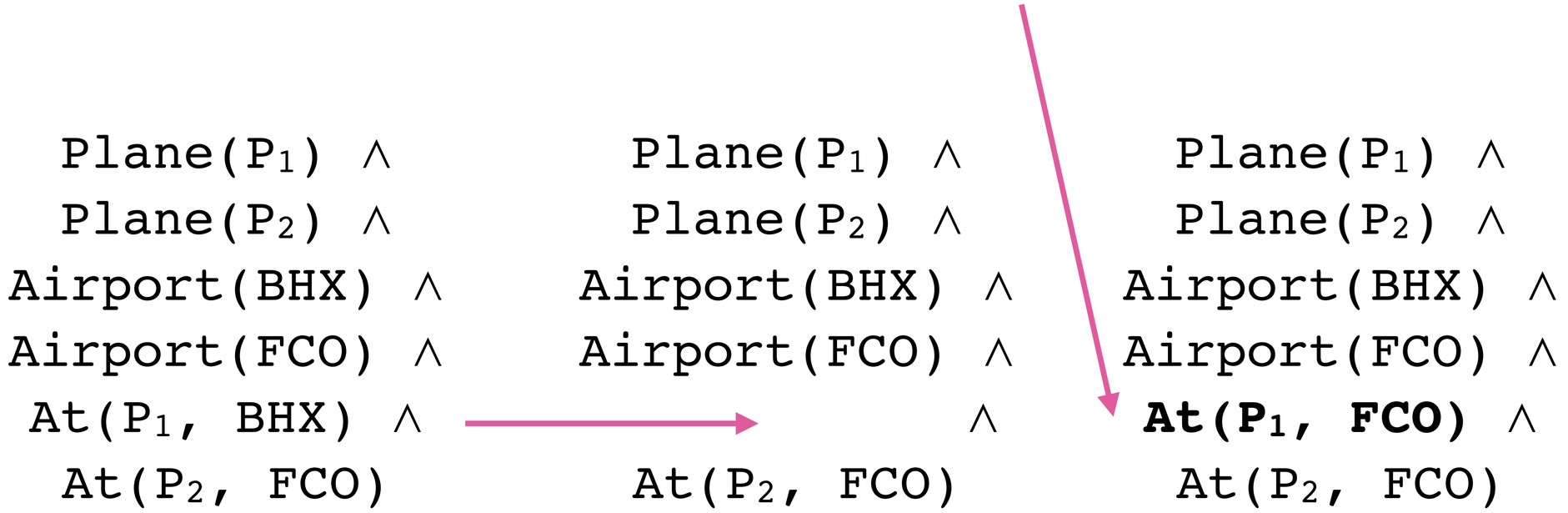
Plane(P_1) ∧
Plane(P_2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P_1, BHX) ∧
At(P_2, FCO)

Action(*Fly*(P_1, BHX, FCO),
PRECOND: At(P_1, BHX) ∧ Plane(P_1) ∧
Airport(BHX) ∧ Airport(FCO)
EFFECT: **¬At(P_1, BHX)** ∧ At(P_1, FCO))

Plane(P_1) ∧
Plane(P_2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P_1, BHX) ∧
At(P_2, FCO)

Plane(P_1) ∧
Plane(P_2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
∧
At(P_2, FCO)

Action(*Fly*(P_1, BHX, FCO),
PRECOND: At(P_1, BHX) ∧ Plane(P_1) ∧
Airport(BHX) ∧ Airport(FCO)
EFFECT: **¬At(P_1, BHX) ∧ At(P_1, FCO)**)



$$\text{Plane}(P_1) \wedge$$
$$\text{Plane}(P_2) \wedge$$
$$\text{Airport}(BHX) \wedge$$
$$\text{Airport}(FCO) \wedge$$
$$\mathbf{At(P_2, BHX)} \wedge$$
$$\mathbf{At(P_1, FCO)}$$

$$\texttt{Plane}(P_1) \wedge$$
$$\texttt{Plane}(P_2) \wedge$$
$$\texttt{Airport(BHX)} \wedge$$
$$\texttt{Airport(FCO)} \wedge$$
$$\mathbf{At(P_2, BHX)} \wedge$$
$$\mathbf{At(P_1, FCO)}$$

Swapping the planes requires two action applications

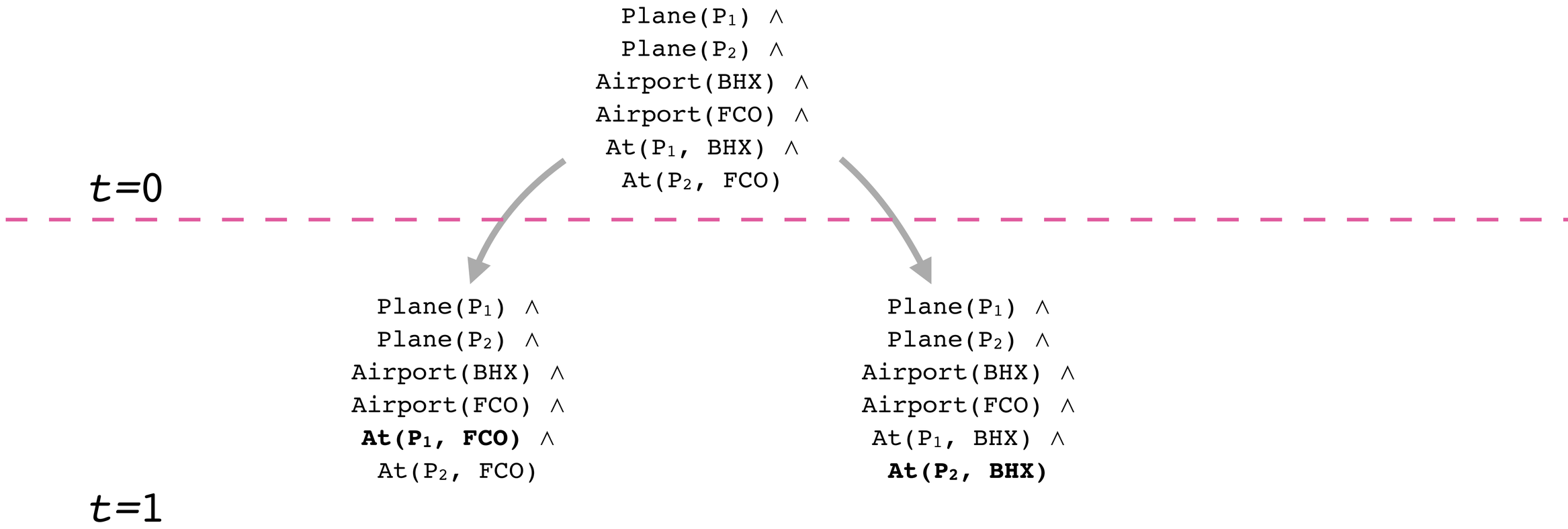
$$\texttt{Plane}(P_1) \wedge$$
$$\texttt{Plane}(P_2) \wedge$$
$$\texttt{Airport(BHX)} \wedge$$
$$\texttt{Airport(FCO)} \wedge$$
$$\textbf{At}(P_2, \textbf{BHX}) \wedge$$
$$\textbf{At}(P_1, \textbf{FCO})$$

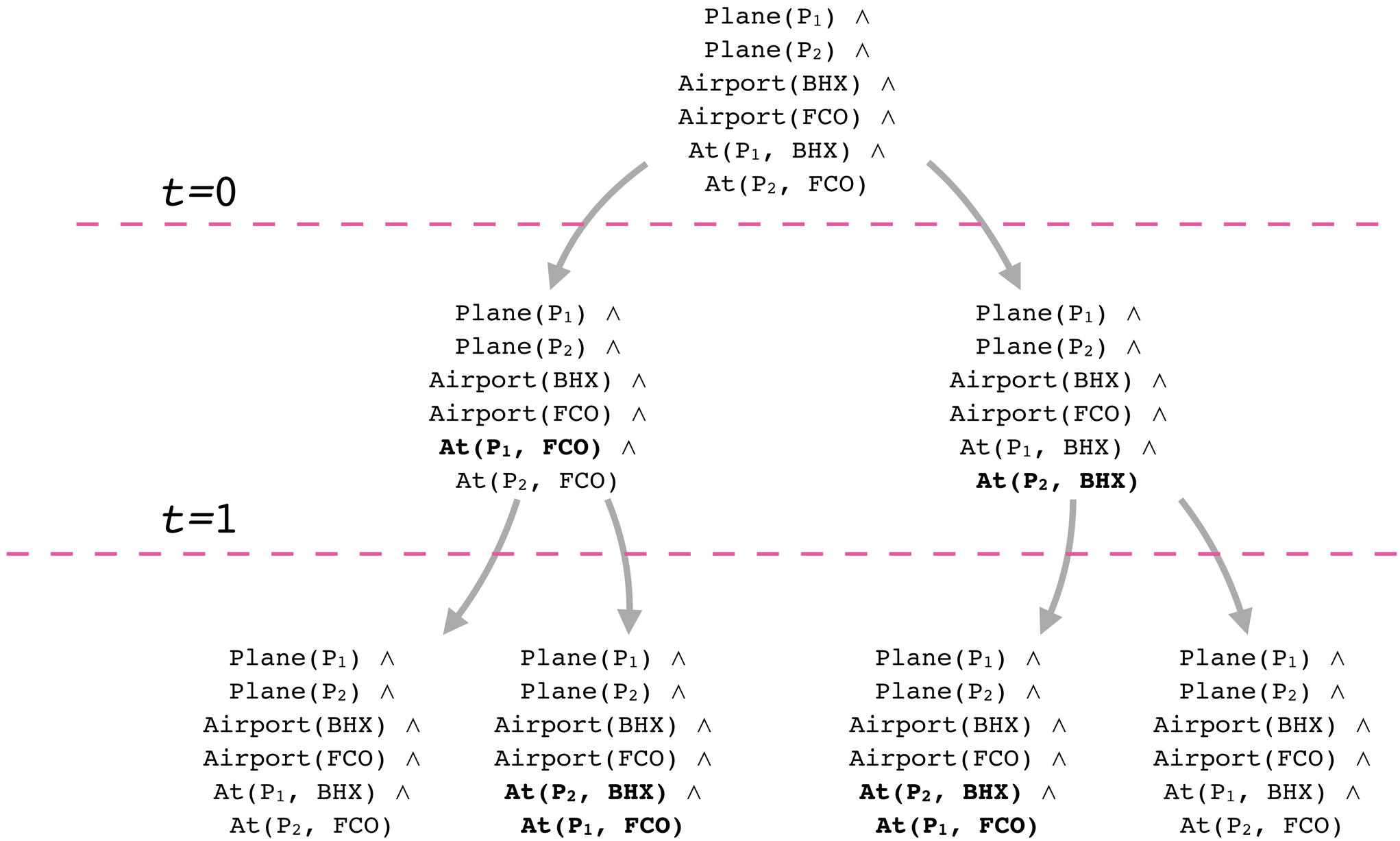
Swapping the planes requires two action applications

Time is **implicit** in *state changes*:
preconditions are at time ***t***
effects are at time ***t+1***

$t=0$

$$
\begin{aligned}
&\text{Plane}(P_1) \wedge \\
&\text{Plane}(P_2) \wedge \\
&\text{Airport}(BHX) \wedge \\
&\text{Airport}(FCO) \wedge \\
&\text{At}(P_1, BHX) \wedge \\
&\text{At}(P_2, FCO)
\end{aligned}
$$


Plane(P1) ∧
Plane(P2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P1, BHX) ∧
At(P2, FCO)
t=0
Plane(P1) ∧
Plane(P2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P1, FCO) ∧
At(P2, FCO)
Plane(P1) ∧
Plane(P2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P1, BHX) ∧
At(P2, BHX)
t=1


Plane(P1) ∧
Plane(P2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P1, BHX) ∧
At(P2, FCO)
t=0
Plane(P1) ∧
Plane(P2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P1, FCO) ∧
At(P2, FCO)
Plane(P1) ∧
Plane(P2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P1, BHX) ∧
At(P2, BHX)
t=1
Plane(P1) ∧
Plane(P2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P1, BHX) ∧
At(P2, FCO)
Plane(P1) ∧
Plane(P2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P2, BHX) ∧
At(P1, FCO)
Plane(P1) ∧
Plane(P2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P2, BHX) ∧
At(P1, FCO)
Plane(P1) ∧
Plane(P2) ∧
Airport(BHX) ∧
Airport(FCO) ∧
At(P1, BHX) ∧
At(P2, FCO)
t=2

The *Logistics* planning **domain** (Russell & Norvig, Figure 10.1)

$$
\begin{aligned}
&Action(Load(c,\ p,\ a),\\
&\quad \textsc{Precond}: At(c,\ a)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)\\
&\quad \textsc{Effect}: \neg\, At(c,\ a)\ \wedge\ In(c,\ p))\\
&Action(Unload(c,\ p,\ a),\\
&\quad \textsc{Precond}: In(c,\ p)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)\\
&\quad \textsc{Effect}: At(c,\ a)\ \wedge\ \neg\, In(c,\ p))\\
&Action(Fly(p,\ from,\ to),\\
&\quad \textsc{Precond}: At(p,\ from)\ \wedge\ Plane(p)\ \wedge\ Airport(from)\ \wedge\ Airport(to)\\
&\quad \textsc{Effect}: \neg\, At(p,\ from)\ \wedge\ At(p,\ to))
\end{aligned}
$$

The *Logistics* planning **domain** (Russell & Norvig, Figure 10.1)

$$
\begin{aligned}
&Action(Load(c,\ p,\ a),\\
&\quad \textsc{Precond}: At(c,\ a)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)\\
&\quad \textsc{Effect}: \neg\ At(c,\ a)\ \wedge\ In(c,\ p))\\
&Action(Unload(c,\ p,\ a),\\
&\quad \textsc{Precond}: In(c,\ p)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)\\
&\quad \textsc{Effect}: At(c,\ a)\ \wedge\ \neg\ In(c,\ p))\\
&Action(Fly(p,\ from,\ to),\\
&\quad \textsc{Precond}: At(p,\ from)\ \wedge\ Plane(p)\ \wedge\ Airport(from)\ \wedge\ Airport(to)\\
&\quad \textsc{Effect}: \neg\ At(p,\ from)\ \wedge\ At(p,\ to))
\end{aligned}
$$

A *Logistics* planning **problem** (Russell & Norvig, Figure 10.1)

$$
\begin{aligned}
&Init(At(C_1,\ SFO)\ \wedge\ At(C_2,\ JFK)\ \wedge\ At(P_1,\ SFO)\ \wedge\ At(P_2,\ JFK)\\
&\quad \wedge\ Cargo(C_1)\ \wedge\ Cargo(C_2)\ \wedge\ Plane(P_1)\ \wedge\ Plane(P_2)\\
&\quad \wedge\ Airport(JFK)\ \wedge\ Airport(SFO))\\
&Goal(At(C_1,\ JFK)\ \wedge\ At(C_2,\ SFO))
\end{aligned}
$$

The **initial state** is *fully specified*

$$\textit{Init}(\ \text{At}(C_1, \text{SFO}) \land \text{At}(C_2, \text{JFK}) \land$$
$$\text{At}(P_1, \text{SFO}) \land \text{At}(P_2, \text{JFK}) \land$$
$$\text{Cargo}(C_1) \land \text{Cargo}(C_2) \land$$
$$\text{Plane}(P_1) \land \text{Plane}(P_2) \land$$
$$\text{Airport}(\text{JFK}) \land \text{Airport}(\text{SFO}))$$

The **initial state** is ***fully specified***

$$Init(\ At(C_1, SFO) \land At(C_2, JFK) \land$$
$$At(P_1, SFO) \land At(P_2, JFK) \land$$
$$Cargo(C_1) \land Cargo(C_2) \land$$
$$Plane(P_1) \land Plane(P_2) \land$$
$$Airport(JFK) \land Airport(SFO))$$

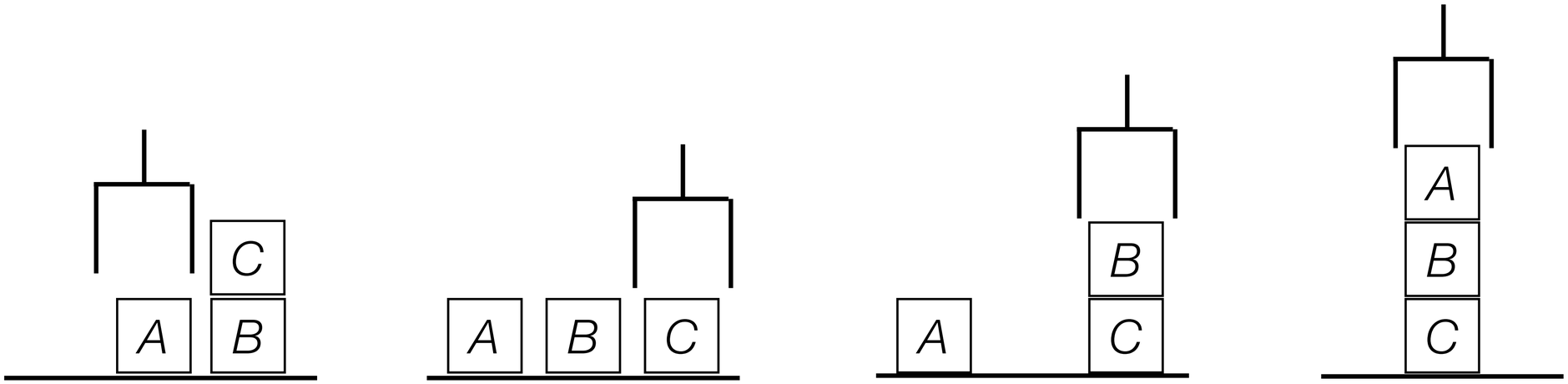
The **goal state** is ***partially specified***.
Like a *precondition* it can contain variables
and is checked by *entailment*

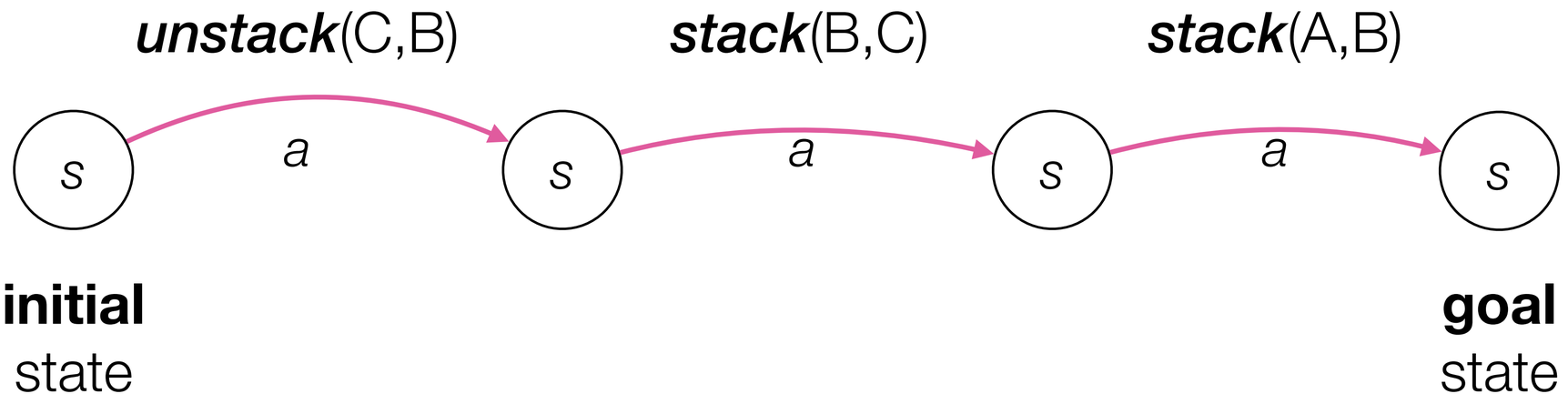
$$Goal(\ At(C_1, JFK) \land At(C_2, SFO))$$

$$Goal(\ At(p, JFK) \land Plan(p, SFO))$$

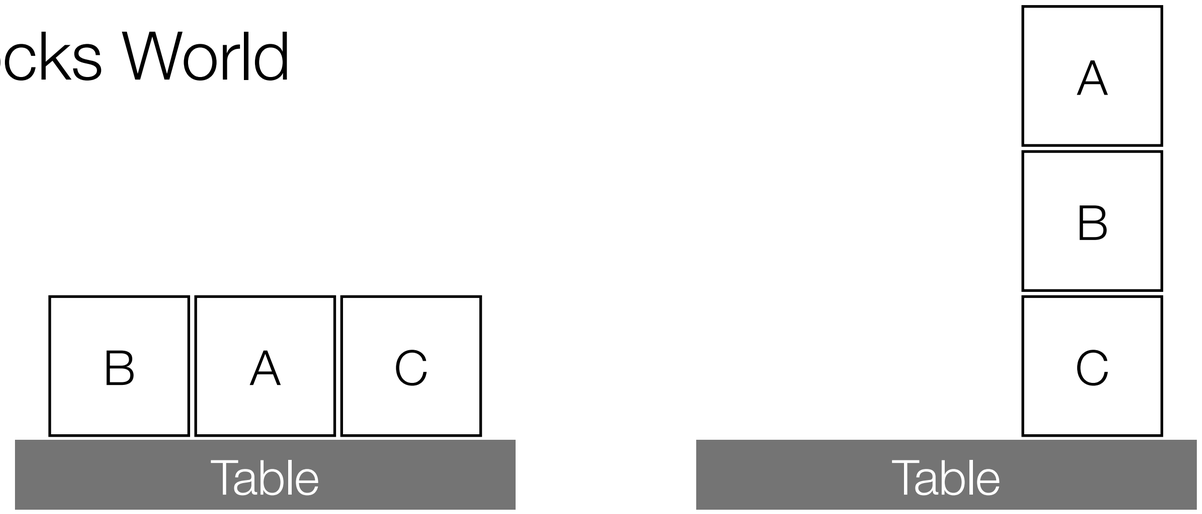
(variables are existentially quantified)

unstack ***stack***





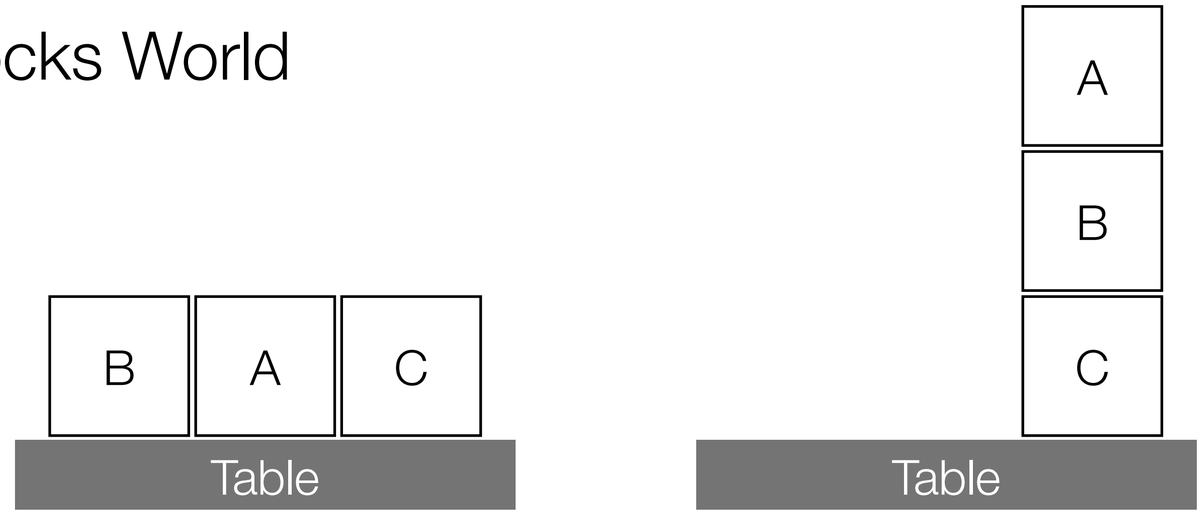
The Blocks World



A *partial* initial state description:

```
... Block(A) ∧ Block(B) Block(C) ...
```

The Blocks World



A *partial* initial state description:

```
... Block(A) ∧ Block(B) Block(C) ...
```

Task: complete the initial and goal states, define the actions, produce a plan for the goal state

Rules: only one block can be *on* another block, the *table* is infinitely wide, a block can only be picked up if it's *clear*

$Action(Move(b, x, y),$
PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge$
$(b \neq x) \wedge (b \neq y) \wedge (x \neq y),$
EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$
$Action(MoveToTable(b, x),$
PRECOND: $On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x),$
EFFECT: $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

Overview

- Classical Planning: STRIPS / PDDL
- **Forward vs Backward Search**
- Planning Heuristics
- The Planning Graph
- Partial Order Planning

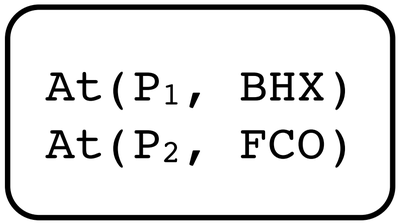
The **PDDL** *representation* does not specify an
algorithm for producing plans

There are many different algorithms we can use
to create plans. Many are based on
domain independent search.

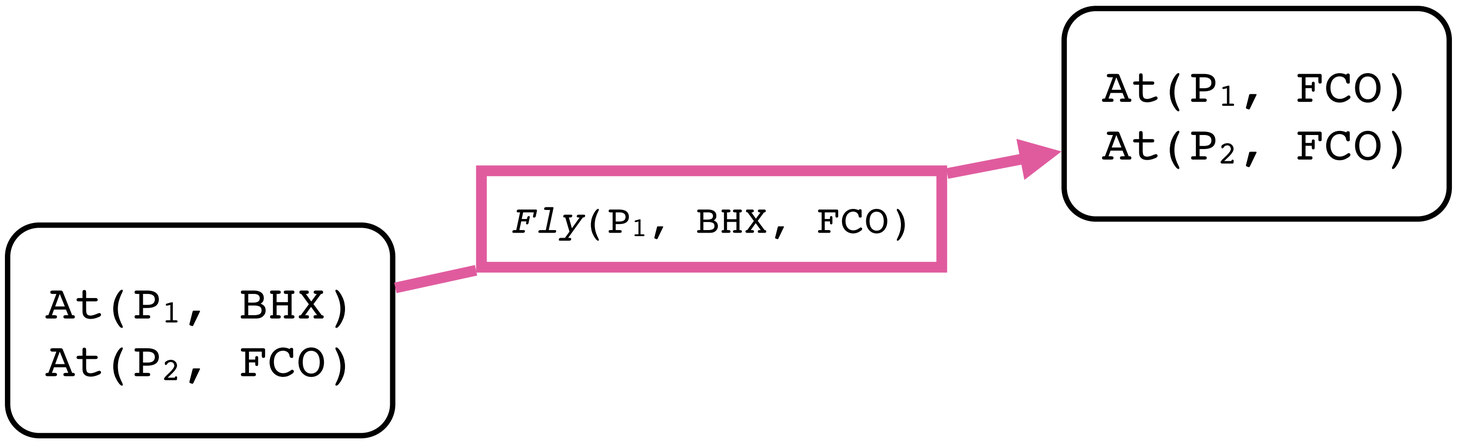
Planning algorithms are often described as either
state-space search
or
plan-space search

Forward (*progression*) state-space search

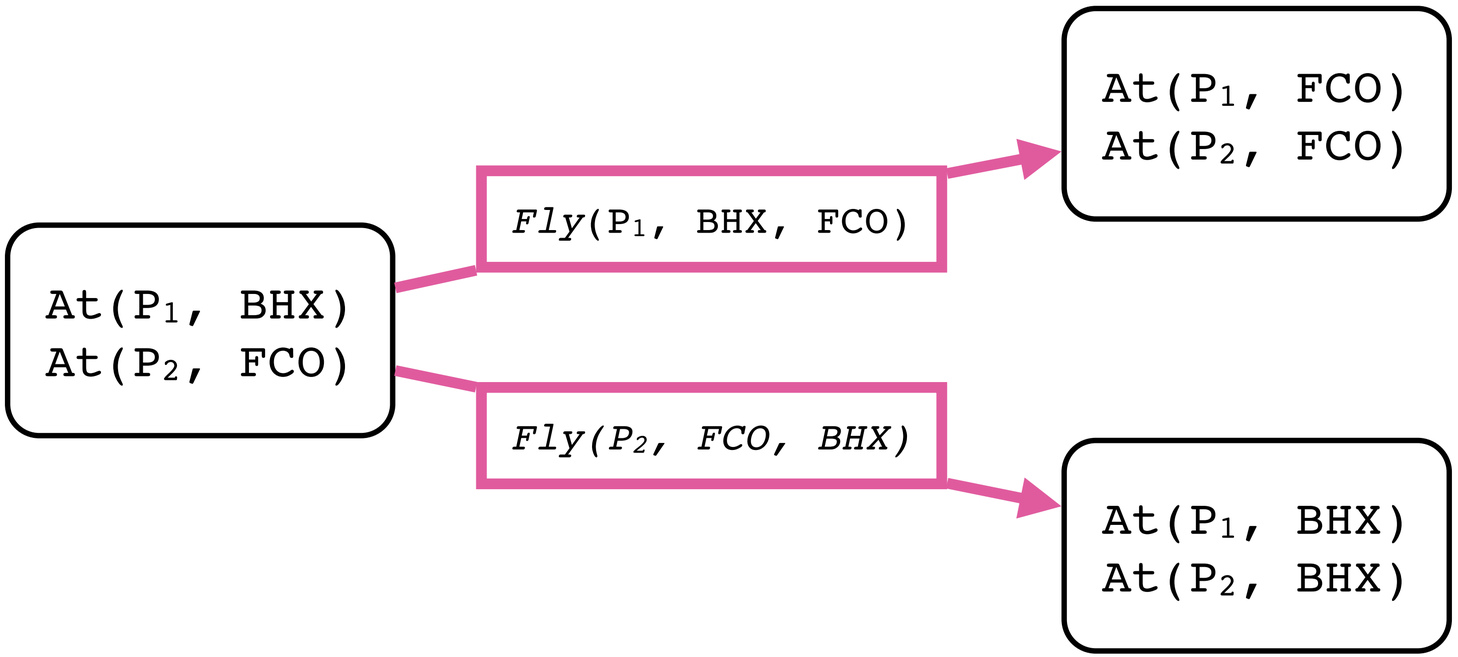
Forward (*progression*) state-space search



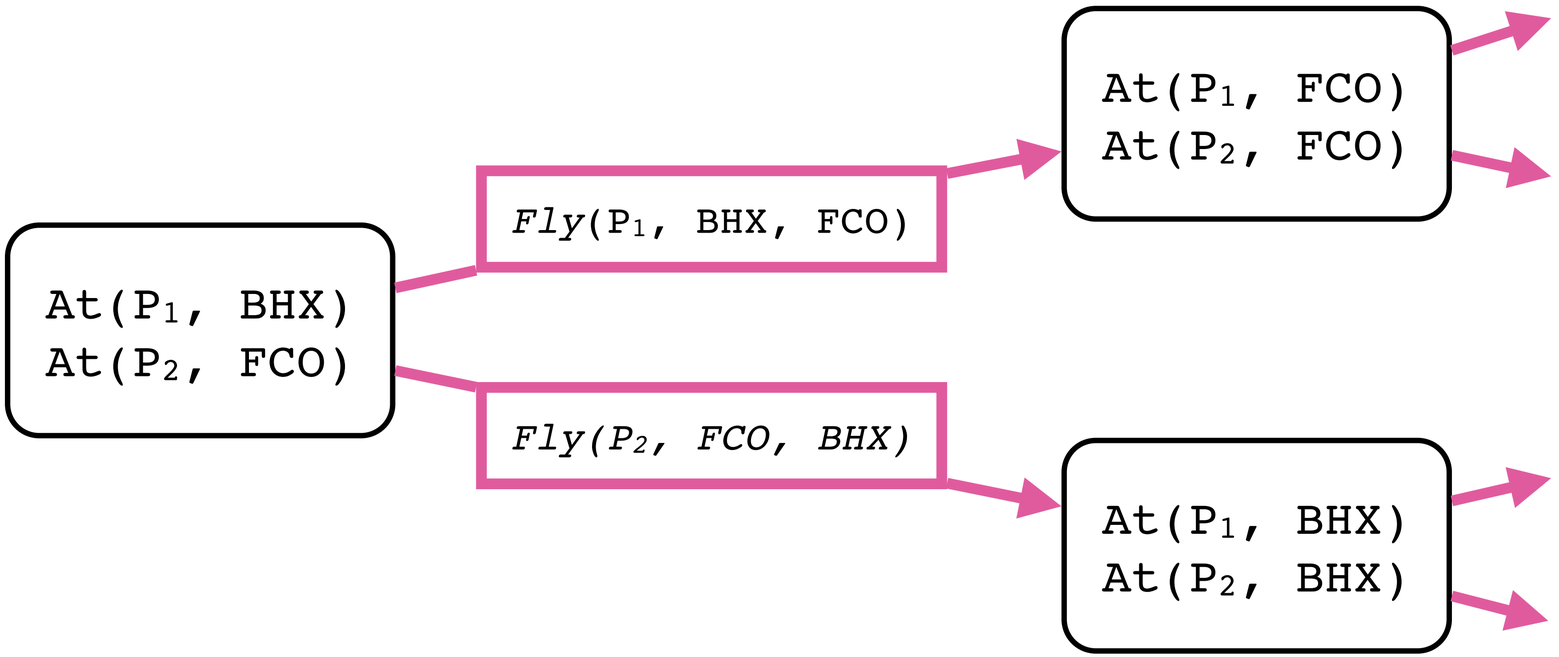
Forward (*progression*) state-space search



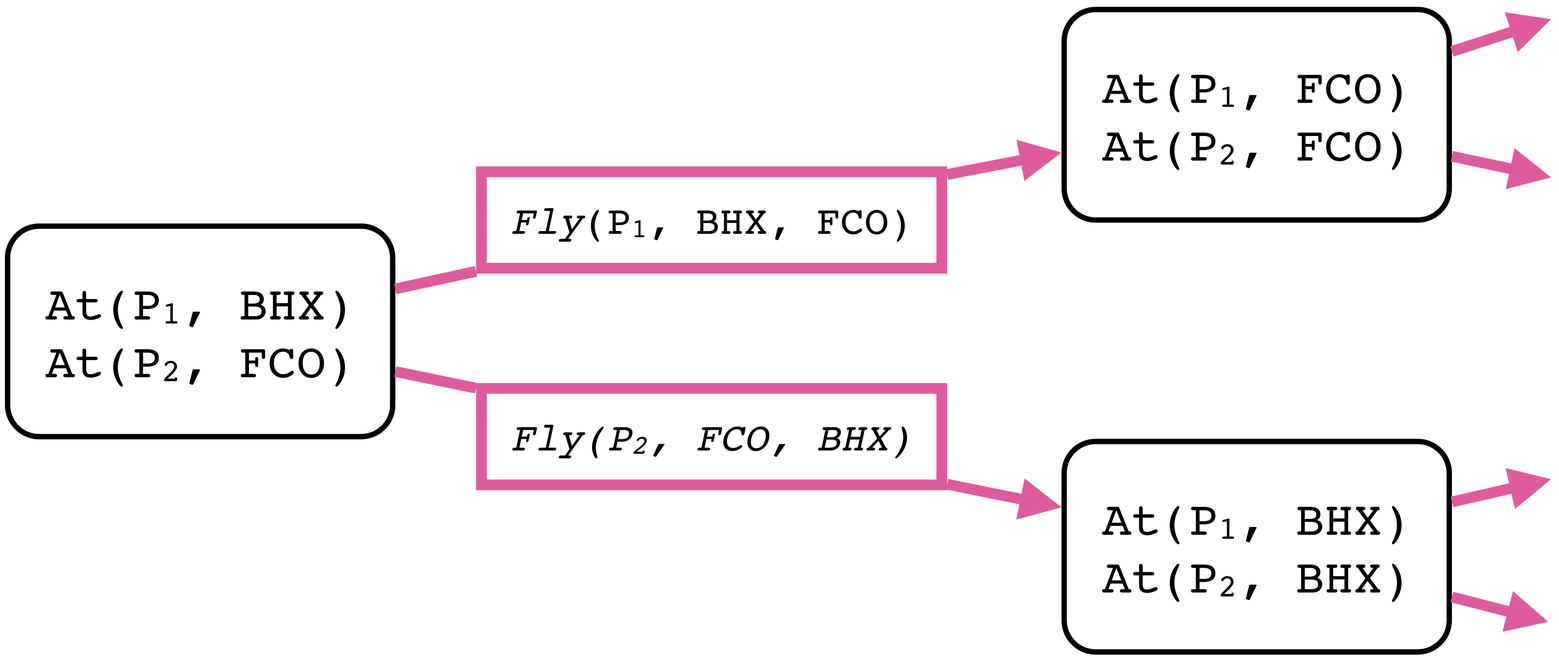
Forward (*progression*) state-space search



Forward (*progression*) state-space search



Forward (*progression*) state-space search



Is this a good approach?

Consider the *Logistics* domain with **10 airports.** Each airport has **5 planes** and **20 cargo**. Goal to move 20 items from airport **A** to airport **B**.

Consider the *Logistics* domain with **10 airports.** Each airport has **5 planes** and **20 cargo**. Goal to move 20 items from airport **A** to airport **B**.

Consider the *Logistics* domain with **10 airports.** Each airport has **5 planes** and **20 cargo**. Goal to move 20 items from airport **A** to airport **B**.

Minimum **450** applicable actions (all planes at airport without cargo: 10 airports * 5 planes * 10 - 1 destinations).

Consider the *Logistics* domain with **10 airports.** Each airport has **5 planes** and **20 cargo**. Goal to move 20 items from airport **A** to airport **B**.

Minimum **450** applicable actions (all planes at airport without cargo: 10 airports * 5 planes * 10 - 1 destinations).

Consider the *Logistics* domain with **10 airports.** Each airport has **5 planes** and **20 cargo**. Goal to move 20 items from airport **A** to airport **B**.

Minimum **450** applicable actions (all planes at airport without cargo: 10 airports * 5 planes * 10 - 1 destinations).

Maximum **10,450** applicable actions (all packages and planes at the same airport: 200 packages * 50 planes = 10,000 then add 450)

Consider the *Logistics* domain with **10 airports.** Each airport has **5 planes** and **20 cargo**. Goal to move 20 items from airport **A** to airport **B**.

Minimum **450** applicable actions (all planes at airport without cargo: 10 airports * 5 planes * 10 - 1 destinations).

Maximum **10,450** applicable actions (all packages and planes at the same airport: 200 packages * 50 planes = 10,000 then add 450)

Consider the *Logistics* domain with **10 airports.** Each airport has **5 planes** and **20 cargo**. Goal to move 20 items from airport **A** to airport **B**.

Minimum **450** applicable actions (all planes at airport without cargo: 10 airports * 5 planes * 10 - 1 destinations).

Maximum **10,450** applicable actions (all packages and planes at the same airport: 200 packages * 50 planes = 10,000 then add 450)

Shallowest solution at **41** actions (20 loads, 1 fly, 20 unloads)

Consider the *Logistics* domain with **10 airports.** Each airport has **5 planes** and **20 cargo**. Goal to move 20 items from airport **A** to airport **B**.

Minimum **450** applicable actions (all planes at airport without cargo: 10 airports * 5 planes * 10 - 1 destinations).

Maximum **10,450** applicable actions (all packages and planes at the same airport: 200 packages * 50 planes = 10,000 then add 450)

Shallowest solution at **41** actions (20 loads, 1 fly, 20 unloads)

Consider the *Logistics* domain with **10 airports.** Each airport has **5 planes** and **20 cargo**. Goal to move 20 items from airport **A** to airport **B**.

Minimum **450** applicable actions (all planes at airport without cargo: 10 airports * 5 planes * 10 - 1 destinations).

Maximum **10,450** applicable actions (all packages and planes at the same airport: 200 packages * 50 planes = 10,000 then add 450)

Shallowest solution at **41** actions (20 loads, 1 fly, 20 unloads)

Assuming average branching factor of **2000**, we have $\mathbf{2000^{41}}$ nodes in the search.

Consider the *Logistics* domain with **10 airports.** Each airport has **5 planes** and **20 cargo**. Goal to move 20 items from airport **A** to airport **B**.

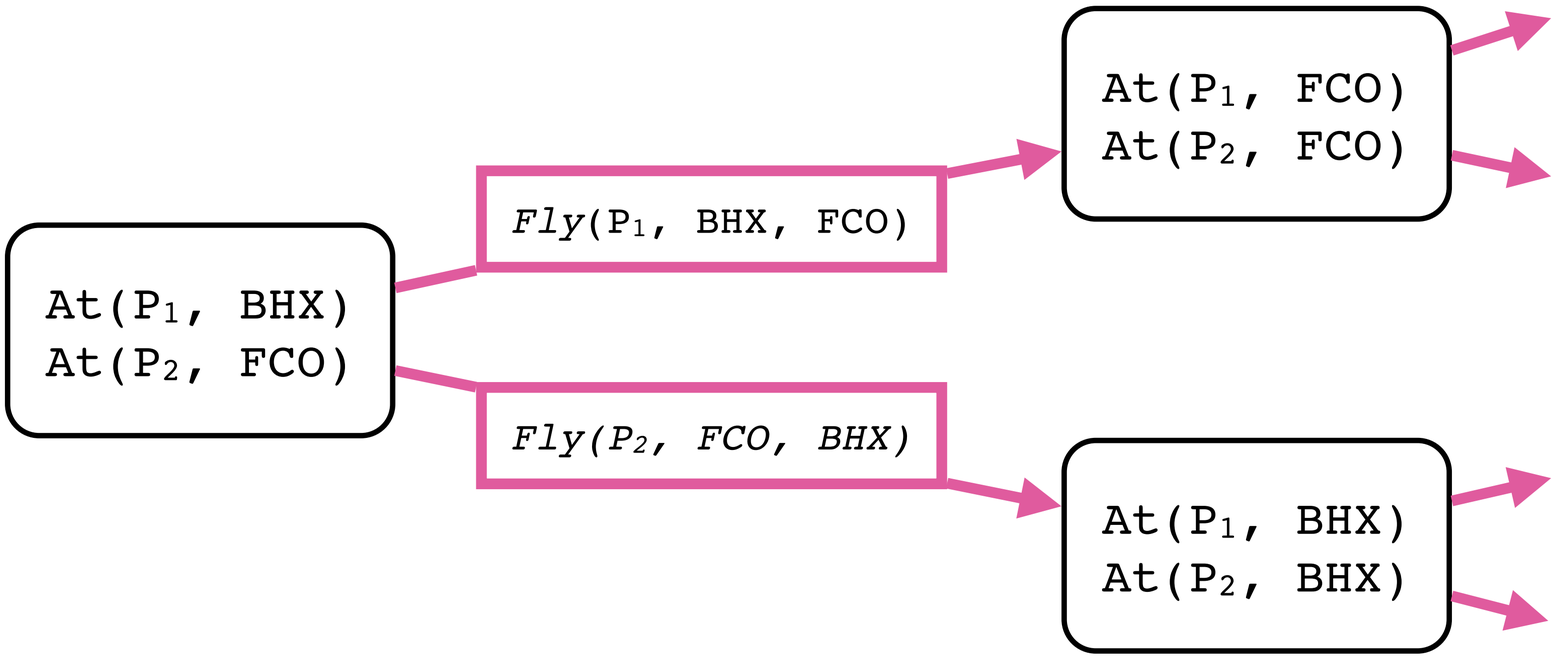
Minimum **450** applicable actions (all planes at airport without cargo: 10 airports * 5 planes * 10 - 1 destinations).

Maximum **10,450** applicable actions (all packages and planes at the same airport: 200 packages * 50 planes = 10,000 then add 450)

Shallowest solution at **41** actions (20 loads, 1 fly, 20 unloads)

Assuming average branching factor of **2000**, we have $\mathbf{2000^{41}}$ nodes in the search.

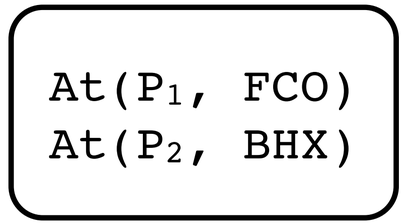
Forward (*progression*) state-space search



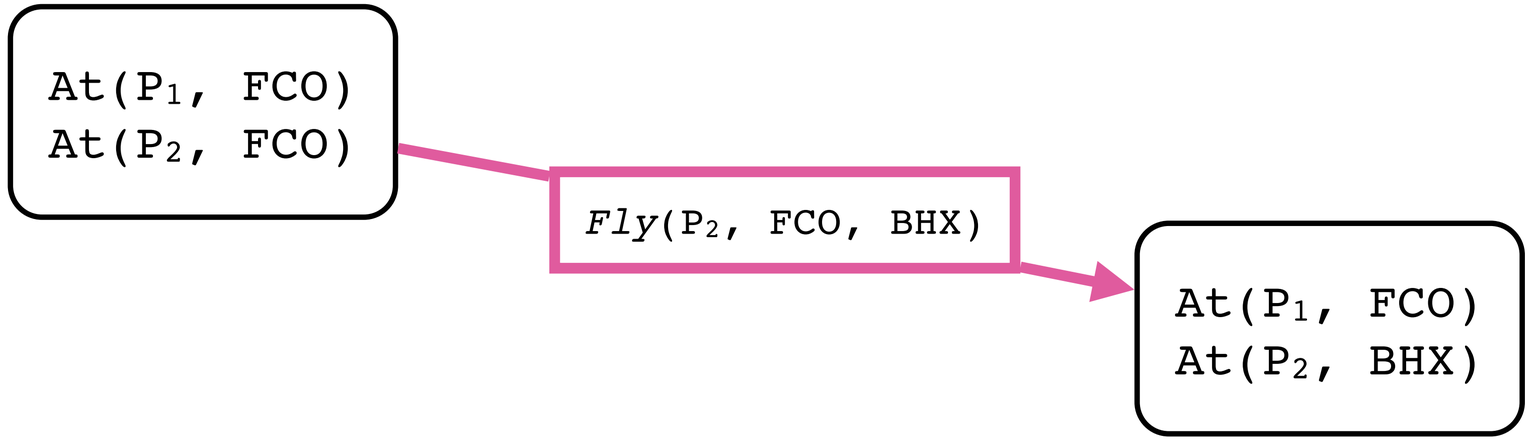
Uniformed forward search must look at too many ***irrelevant states*** to be a good approach to find plans

Backward (*regression*) state-space search

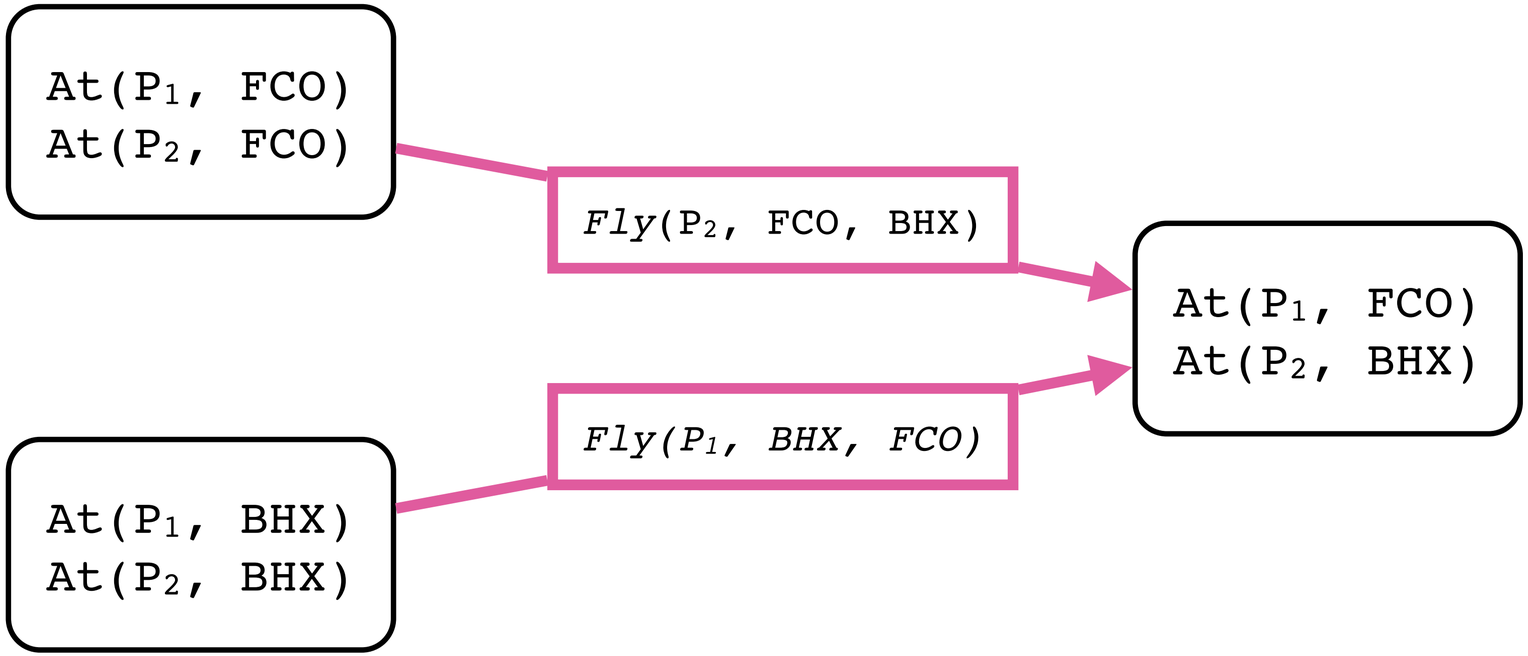
Backward (*regression*) state-space search



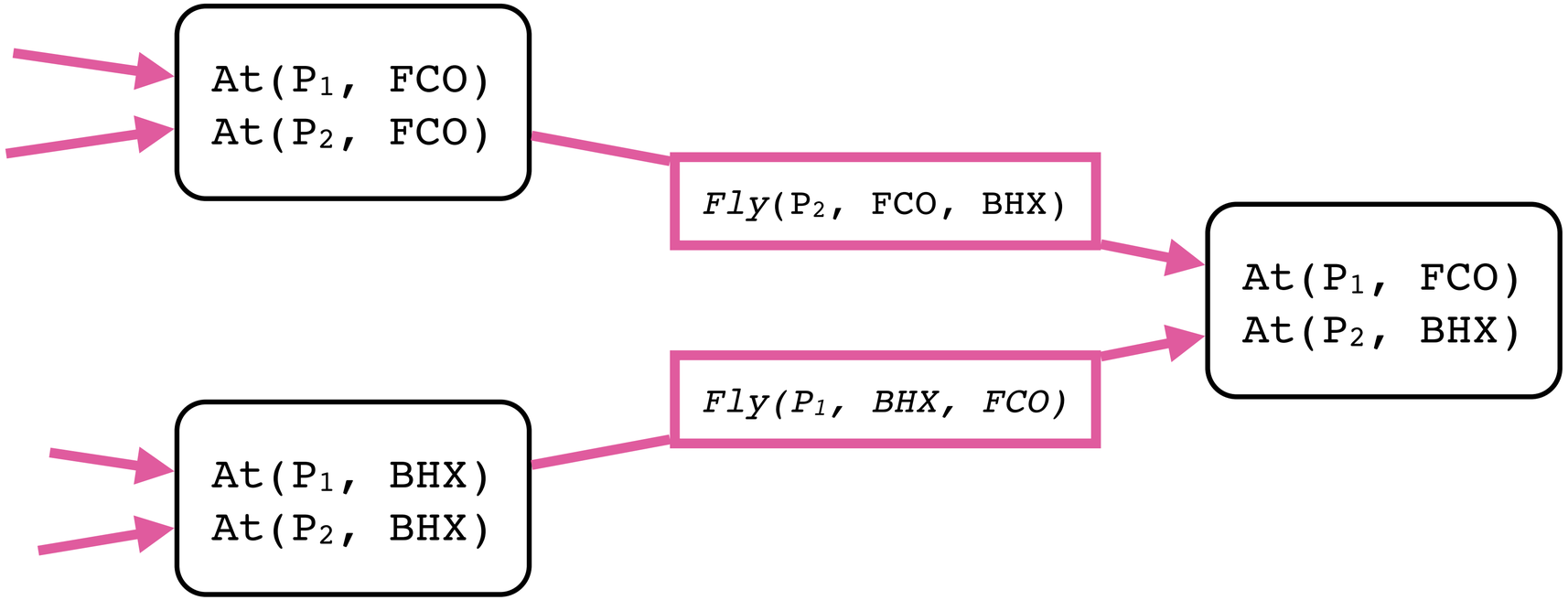
Backward (*regression*) state-space search



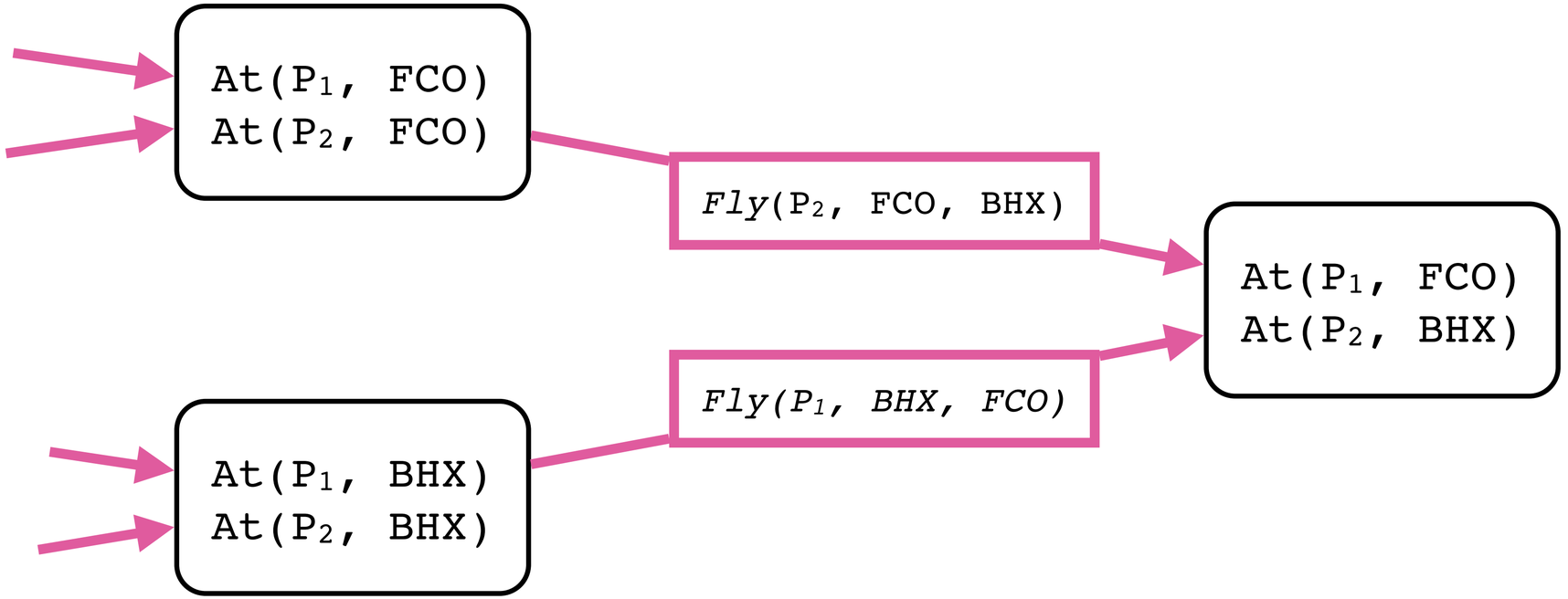
Backward (*regression*) state-space search



Backward (*regression*) state-space search



Backward (*regression*) state-space search



This is also called ***relevant state search*** as it only looks at states ***relevant*** to the goal

Regress from ***goal*** *g* to *g'* via action *a*

$$g' = (g - \text{ADD}(a)) \cup \text{PRECOND}(a)$$

Regress from ***goal*** *g* to *g'* via action *a*

$$g' = (g - \text{ADD}(a)) \cup \text{PRECOND}(a)$$

The *positive effects* of *a*
were not necessarily true in
the preceding state

Regress from ***goal*** *g* to *g'* via action *a*

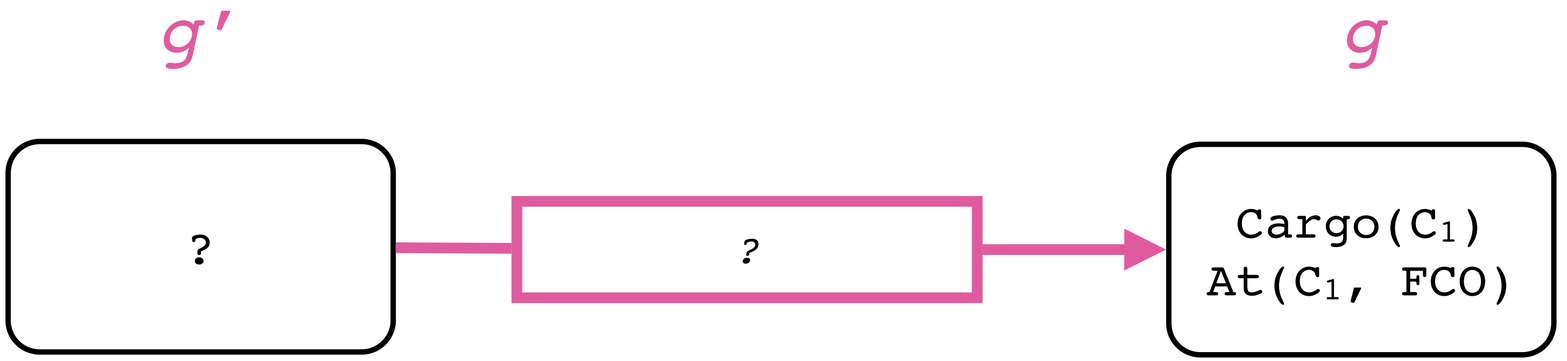
$$g' = (g - \text{ADD}(a)) \cup \text{PRECOND}(a)$$

The *positive effects* of *a* were not necessarily true in the preceding state

The *preconditions* of *a* must have been true in the preceding state

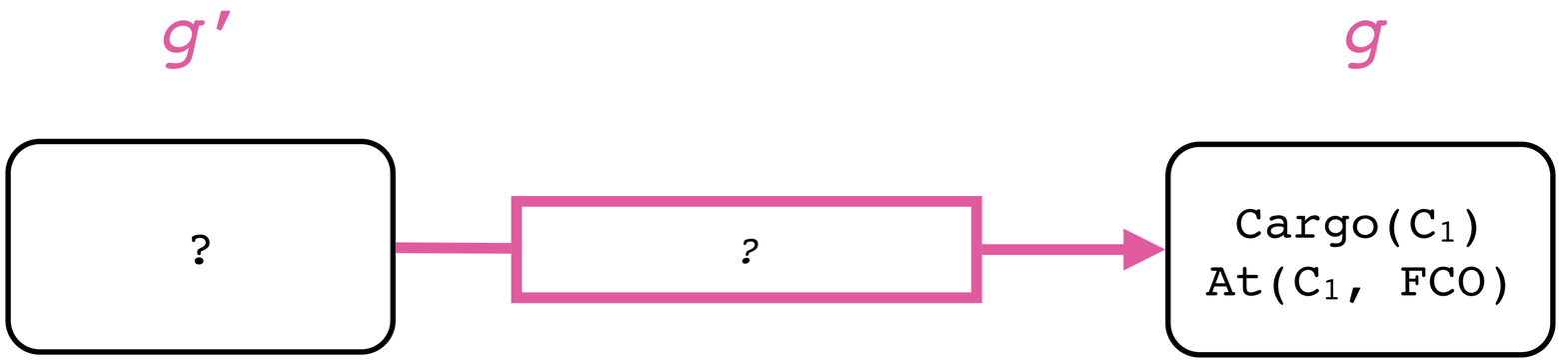
The power of goal regression is that we're are searching **sets of states**, as the goal state is ***partially specified***

(if we enumerated all states we'd be back to the problems of forward search)



The power of goal regression is that we're are searching **sets of states**, as the goal state is ***partially specified***

(if we enumerated all states we'd be back to the problems of forward search)



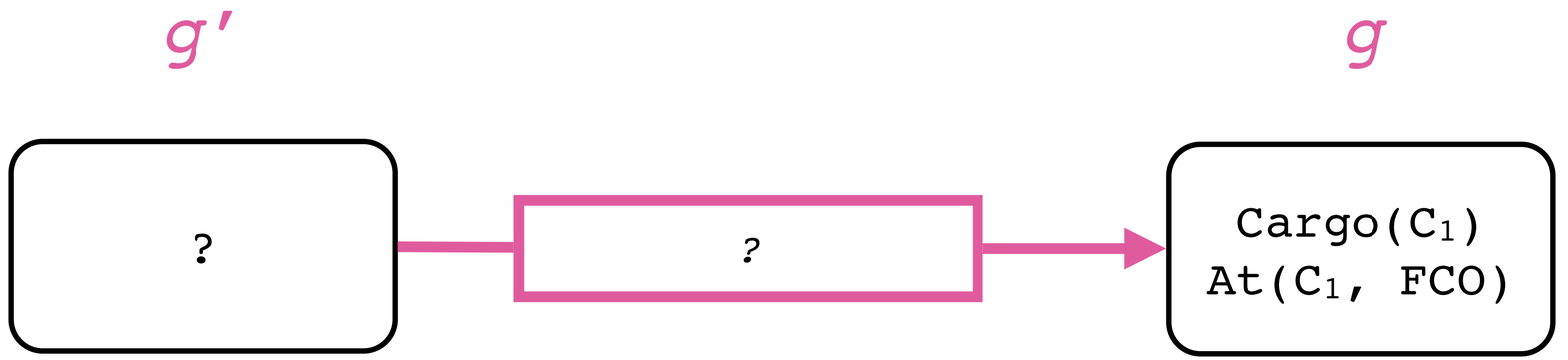
`Action( Unload(c, p, a) )`

We need to ***Unload*** a plane.

(assuming that the goal is not already true)

The power of goal regression is that we're are searching **sets of states**, as the goal state is ***partially specified***

(if we enumerated all states we'd be back to the problems of forward search)



Action(Unload(c, p, a))

Action(Unload(C1, p, FCO))

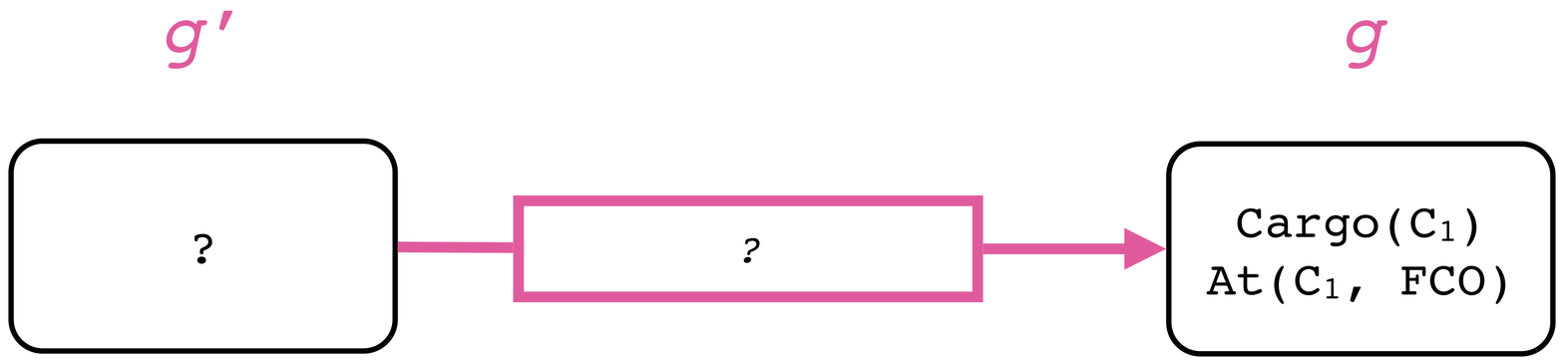
We need to *Unload* a plane.

(assuming that the goal is not already true)

But which one?

The power of goal regression is that we're are searching **sets of states**, as the goal state is ***partially specified***

(if we enumerated all states we'd be back to the problems of forward search)



Action(Unload(c, p, a))

*Action(Unload(C1, **p'**, FCO))*

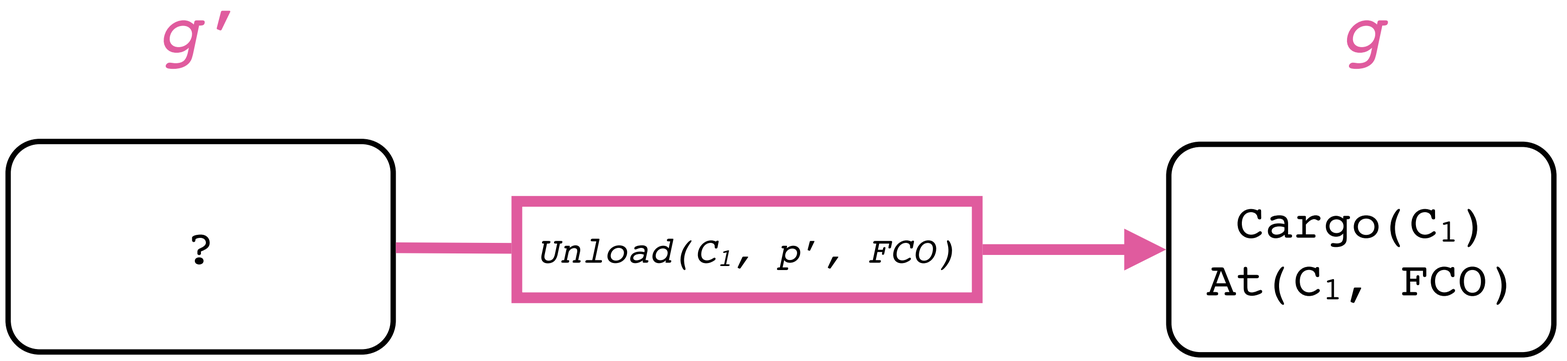
We need to *Unload* a plane.

(assuming that the goal is not already true)

Any one will do!

But which one?

Action(*Unload*(C_1, *p'*, FCO),
  PRECOND: In(C_1, *p'*) ∧ At(P_1, FCO) ∧
    Cargo(C_1) ∧ Plane(*p'*) ∧ Airport(FCO)
  EFFECT: At(C_1, FCO) ∧ ¬In(P_1, C_1))



g' = (g - ADD(a)) U PRECOND(a)

g' = In(C_1, p') ∧ At(P_1, FCO) ∧
  Cargo(C_1) ∧ Plane(p') ∧
  Airport(FCO)

Regress from ***goal*** g to g' via action a

Regress from ***goal*** g to g' via action a

Search proceeds by choosing only ***relevant actions*** which ***unify*** with an element of g

… and which could be ***the last step*** in a plan leading to g

Regress from ***goal*** g to g' via action ***a***

Search proceeds by choosing only ***relevant actions*** which ***unify*** with an element of g

… and which could be ***the last step*** in a plan leading to ***g***

$$g = A \wedge B \wedge C$$

Regress from ***goal*** *g* to *g'* via action ***a***

Search proceeds by choosing only ***relevant actions*** which ***unify*** with an element of *g*

… and which could be ***the last step*** in a plan leading to ***g***

$$g = A \wedge B \wedge C$$

Action(A1, EFFECT: A)

Regress from ***goal*** *g* to *g'* via action ***a***

Search proceeds by choosing only ***relevant actions*** which ***unify*** with an element of *g*

… and which could be ***the last step*** in a plan leading to ***g***

$$g = A \wedge B \wedge C$$

Action(A1, EFFECT: A)

relevant

Regress from ***goal*** *g* to *g'* via action ***a***

Search proceeds by choosing only ***relevant actions*** which ***unify*** with an element of *g*

… and which could be ***the last step*** in a plan leading to ***g***

$$g = A \wedge B \wedge C$$

Action(A1, EFFECT: A)

Action(A2, EFFECT: A ∧ B)

relevant

Regress from ***goal*** *g* to *g'* via action ***a***

Search proceeds by choosing only ***relevant actions*** which ***unify*** with an element of *g*

… and which could be ***the last step*** in a plan leading to ***g***

$$g = A \wedge B \wedge C$$

Action(A1, EFFECT: A) — *relevant*

Action(A2, EFFECT: A ∧ B) — *relevant*

Regress from ***goal*** *g* to *g'* via action ***a***

Search proceeds by choosing only ***relevant actions*** which ***unify*** with an element of *g*

… and which could be ***the last step*** in a plan leading to ***g***

$$g = A \wedge B \wedge C$$

Action(*A1*, EFFECT: A) *relevant*

Action(*A2*, EFFECT: A ∧ B) *relevant*

Action(*A3*, EFFECT: A ∧ B ∧ ¬C)

Regress from ***goal*** *g* to *g'* via action ***a***

Search proceeds by choosing only ***relevant actions*** which ***unify*** with an element of *g*

… and which could be ***the last step*** in a plan leading to ***g***

$$g = A \wedge B \wedge C$$

Action(A1, EFFECT: A)	*relevant*
Action(A2, EFFECT: A ∧ B)	*relevant*
Action(A3, EFFECT: A ∧ B ∧ ¬C)	**not** *relevant*

g = In(C_1, p') ∧ At(p', FCO) ∧
Cargo(C_1) ∧ Plane(p') ∧
Airport(FCO)

g' = (g - ADD(a)) U PRECOND(a)

$Action(Load(c,\ p,\ a)$,
PRECOND: $At(c,\ a) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg\, At(c,\ a) \wedge In(c,\ p))$
$Action(Unload(c,\ p,\ a)$,
PRECOND: $In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c,\ a) \wedge \neg\, In(c,\ p))$
$Action(Fly(p,\ from,\ to)$,
PRECOND: $At(p,\ from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg\, At(p,\ from) \wedge At(p,\ to))$

Question: How many actions are ***relevant*** to g?

Poll:	0	1	2	3

g = In(C_1, p′) ∧ At(p′, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

g' = (g - ADD(a)) U PRECOND(a)

$Action(Load(c,\ p,\ a)$,
PRECOND: $At(c,\ a) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg\, At(c,\ a) \wedge In(c,\ p))$
$Action(Unload(c,\ p,\ a)$,
PRECOND: $In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c,\ a) \wedge \neg\, In(c,\ p))$
$Action(Fly(p,\ from,\ to)$,
PRECOND: $At(p,\ from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg\, At(p,\ from) \wedge At(p,\ to))$

g = In(C_1, p′) ∧ At(p′, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

g' = (g - ADD(a)) U PRECOND(a)

$Action(Load(c,\ p,\ a)$,
PRECOND: $At(c,\ a) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg\, At(c,\ a) \wedge In(c,\ p))$
$Action(Unload(c,\ p,\ a)$,
PRECOND: $In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c,\ a) \wedge \neg\, In(c,\ p))$
$Action(Fly(p,\ from,\ to)$,
PRECOND: $At(p,\ from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg\, At(p,\ from) \wedge At(p,\ to))$

g = In(C_1, p′) ∧ At(p′, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

g' = (g - ADD(a)) U PRECOND(a)

$Action(Load(c,\ p,\ a)$,
PRECOND: $At(c,\ a) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg\, At(c,\ a) \wedge In(c,\ p))$
$Action(Unload(c,\ p,\ a)$,
PRECOND: $In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c,\ a) \wedge \neg\, In(c,\ p))$
$Action(Fly(p,\ from,\ to)$,
PRECOND: $At(p,\ from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg\, At(p,\ from) \wedge At(p,\ to))$

g = In(C_1, p′) ∧ At(p′, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

g' = (g - ADD(a)) U PRECOND(a)

$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$

g = In(C_1, p′) ∧ At(p′, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

g' = (g - ADD(a)) U PRECOND(a)

$Action(Load(c,\ p,\ a)$,
PRECOND: $At(c,\ a) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg\, At(c,\ a) \wedge In(c,\ p))$
$Action(Unload(c,\ p,\ a)$,
PRECOND: $In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c,\ a) \wedge \neg\, In(c,\ p))$
$Action(Fly(p,\ from,\ to)$,
PRECOND: $At(p,\ from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg\, At(p,\ from) \wedge At(p,\ to))$

g = In(C_1, p′) ∧ At(p′, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

g' = (g - ADD(a)) U PRECOND(a)

$Action(Load(c,\ p,\ a)$,
PRECOND: $At(c,\ a) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg\, At(c,\ a) \wedge In(c,\ p))$
$Action(Unload(c,\ p,\ a)$,
PRECOND: $In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c,\ a) \wedge \neg\, In(c,\ p))$
$Action(Fly(p,\ from,\ to)$,
PRECOND: $At(p,\ from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg\, At(p,\ from) \wedge At(p,\ to))$

g = In(C_1, p′) ∧ At(p′, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

g' = (g − ADD(a)) U PRECOND(a)

$Action(Load(c,\ p,\ a)$,
PRECOND: $At(c,\ a) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg\, At(c,\ a) \wedge In(c,\ p))$ ✓
$Action(Unload(c,\ p,\ a)$,
PRECOND: $In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c,\ a) \wedge \neg\, In(c,\ p))$ ✗
$Action(Fly(p,\ from,\ to)$,
PRECOND: $At(p,\ from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg\, At(p,\ from) \wedge At(p,\ to))$

g = In(C_1, p') ∧ At(p', FCO) ∧
Cargo(C_1) ∧ Plane(p') ∧
Airport(FCO)

g' = (g - ADD(a)) U PRECOND(a)

$Action(Load(c, p, a)$,
PRECOND: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg At(c, a) \wedge In(c, p))$
$Action(Unload(c, p, a)$,
PRECOND: $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c, a) \wedge \neg In(c, p))$
$Action(Fly(p, from, to)$,
PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg At(p, from) \wedge At(p, to))$

g = In(C_1, p') ∧ At(p', FCO) ∧
Cargo(C_1) ∧ Plane(p') ∧
Airport(FCO)

g' = (g - ADD(a)) U PRECOND(a)

$Action(Load(c,\ p,\ a)$,
PRECOND: $At(c,\ a) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg\, At(c,\ a) \wedge In(c,\ p))$ ✓
$Action(Unload(c,\ p,\ a)$,
PRECOND: $In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c,\ a) \wedge \neg\, In(c,\ p))$ ✗
$Action(Fly(p,\ from,\ to)$,
PRECOND: $At(p,\ from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg\, At(p,\ from) \wedge At(p,\ to))$

g = In(C_1, p′) ∧ At(p′, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

g' = (g − ADD(a)) U PRECOND(a)

$Action(Load(c,\ p,\ a)$,
PRECOND: $At(c,\ a) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg\, At(c,\ a) \wedge In(c,\ p))$ ✓
$Action(Unload(c,\ p,\ a)$,
PRECOND: $In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c,\ a) \wedge \neg\, In(c,\ p))$ ✗
$Action(Fly(p,\ from,\ to)$,
PRECOND: $At(p,\ from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg\, At(p,\ from) \wedge At(p,\ to))$ ✓

g = In(C_1, p') ∧ At(p', FCO) ∧
Cargo(C_1) ∧ Plane(p') ∧
Airport(FCO)

g' = (g - ADD(a)) U PRECOND(a)

$Action(Load(c, p, a)$,
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$ ✓
$Action(Unload(c, p, a)$,
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$ ✗
$Action(Fly(p, from, to)$,
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$ ✓

Question: How many actions are ***relevant*** to g?

Poll:			2	

g = In(C_1, p′) ∧ At(P_1, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

Load(C_1, p', FCO)

$Action(Load(c,\ p,\ a),$
PRECOND: $At(c,\ a)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)$
EFFECT: $\neg\ At(c,\ a)\ \wedge\ In(c,\ p))$
$Action(Unload(c,\ p,\ a),$
PRECOND: $In(c,\ p)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)$
EFFECT: $At(c,\ a)\ \wedge\ \neg\ In(c,\ p))$
$Action(Fly(p,\ from,\ to),$
PRECOND: $At(p,\ from)\ \wedge\ Plane(p)\ \wedge\ Airport(from)\ \wedge\ Airport(to)$
EFFECT: $\neg\ At(p,\ from)\ \wedge\ At(p,\ to))$

g = In(C_1, p′) ∧ At(P_1, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

Load(C_1, p', FCO)

g' = At(C_1, FCO) ∧ At(P_1, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

$Action(Load(c,\ p,\ a),$
PRECOND: $At(c,\ a)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)$
EFFECT: $\neg\ At(c,\ a)\ \wedge\ In(c,\ p))$
$Action(Unload(c,\ p,\ a),$
PRECOND: $In(c,\ p)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)$
EFFECT: $At(c,\ a)\ \wedge\ \neg\ In(c,\ p))$
$Action(Fly(p,\ from,\ to),$
PRECOND: $At(p,\ from)\ \wedge\ Plane(p)\ \wedge\ Airport(from)\ \wedge\ Airport(to)$
EFFECT: $\neg\ At(p,\ from)\ \wedge\ At(p,\ to))$

g = In(C_1, p′) ∧ At(P_1, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

Fly(p′, a′, FCO)

$Action(Load(c,\ p,\ a)$,
PRECOND: $At(c,\ a)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)$
EFFECT: $\neg\ At(c,\ a)\ \wedge\ In(c,\ p))$
$Action(Unload(c,\ p,\ a)$,
PRECOND: $In(c,\ p)\ \wedge\ At(p,\ a)\ \wedge\ Cargo(c)\ \wedge\ Plane(p)\ \wedge\ Airport(a)$
EFFECT: $At(c,\ a)\ \wedge\ \neg\ In(c,\ p))$
$Action(Fly(p,\ from,\ to)$,
PRECOND: $At(p,\ from)\ \wedge\ Plane(p)\ \wedge\ Airport(from)\ \wedge\ Airport(to)$
EFFECT: $\neg\ At(p,\ from)\ \wedge\ At(p,\ to))$

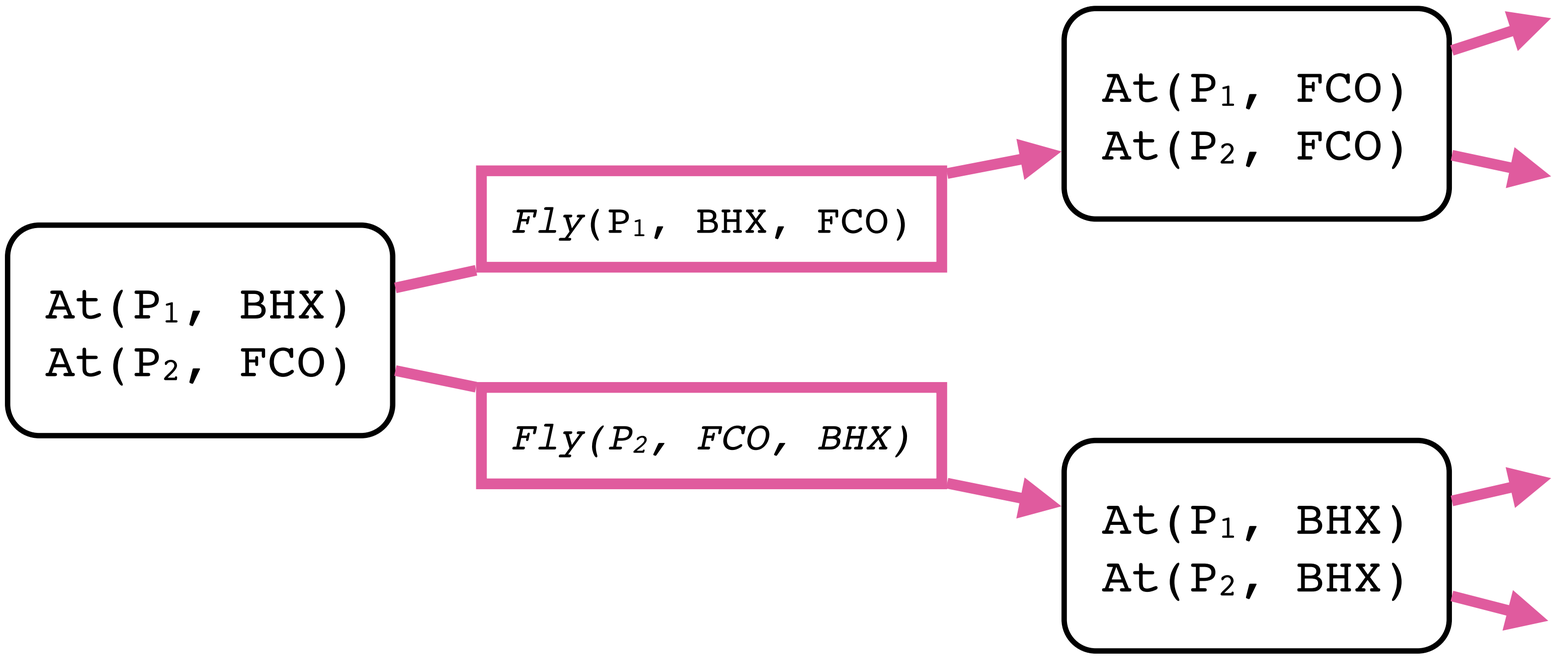
g = In(C_1, p′) ∧ At(P_1, FCO) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

Fly(*p′*, *a′*, FCO)

g' = At(P_1, *a′*) ∧ Airport(*a′*)
In(C_1, p′) ∧
Cargo(C_1) ∧ Plane(p′) ∧
Airport(FCO)

$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$

Forward (*progression*) state-space search



Uniformed forward search must look at too many ***irrelevant states*** to be a good approach to find plans

Overview

- Classical Planning: STRIPS / PDDL
- Forward vs Backward Search
- **Planning Heuristics**
- The Planning Graph
- Partial Order Planning

Informed state-space search

Informed state-space search

For informed search we need a **heuristic.**

Informed state-space search

For informed search we need a **heuristic.**

If we have an **admissible heuristic** then A* will find **optimal solutions**

Informed state-space search

For informed search we need a **heuristic.**

If we have an **admissible heuristic** then A* will find **optimal solutions**

For domain independent planning we need a **domain independent heuristic**

Planning **heuristics** are often produced by *creating* and *solving* a **relaxed problem**

Planning **heuristics** are often produced by *creating* and *solving* a **relaxed problem**

A **relaxed problem** is an *easier* version of the original problem

Planning **heuristics** are often produced by *creating* and *solving* a **relaxed problem**

A **relaxed problem** is an *easier* version of the original problem

The cost value for the **relaxed problem** is then used as the **heuristic** estimate in the full problem

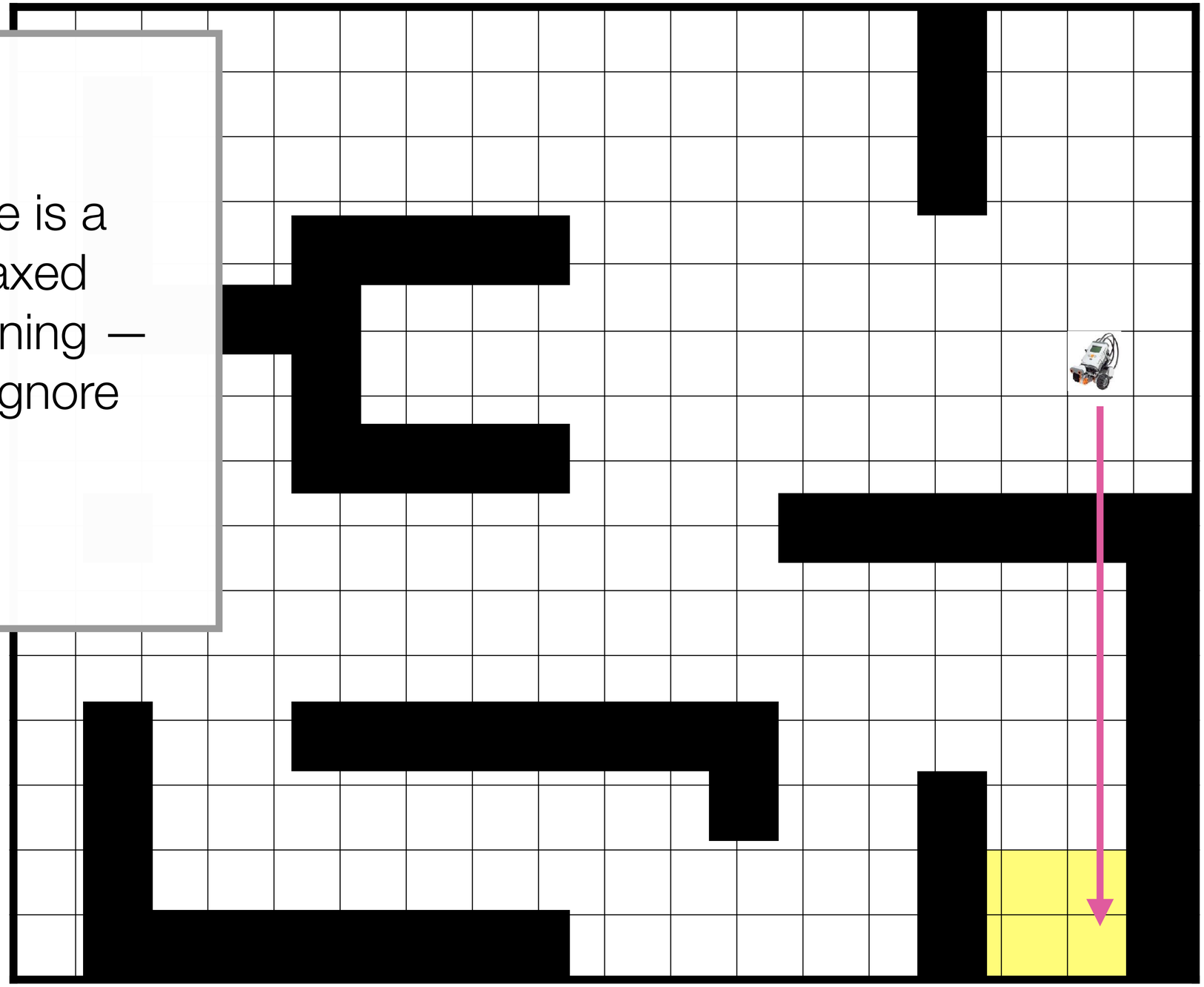
Planning **heuristics** are often produced by *creating* and *solving* a **relaxed problem**

A **relaxed problem** is an *easier* version of the original problem

The cost value for the **relaxed problem** is then used as the **heuristic** estimate in the full problem

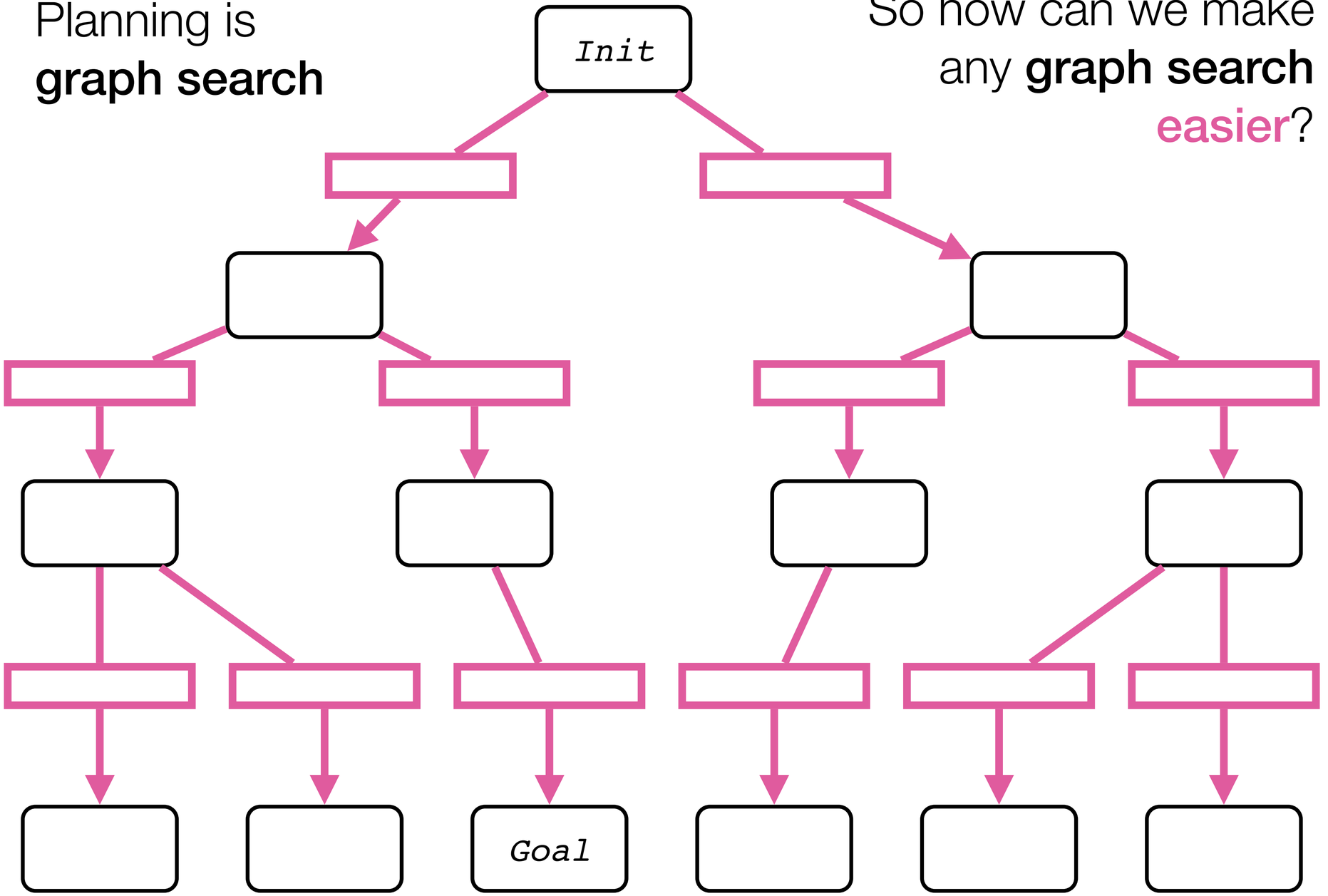
If it's *easier* then a solution to to the **relaxed problem** should be *cheaper* than for the original problem, thus the **heuristic** should be **admissible**

Manhattan distance is a solution to the relaxed problem of path planning — the relaxation is to ignore obstacles



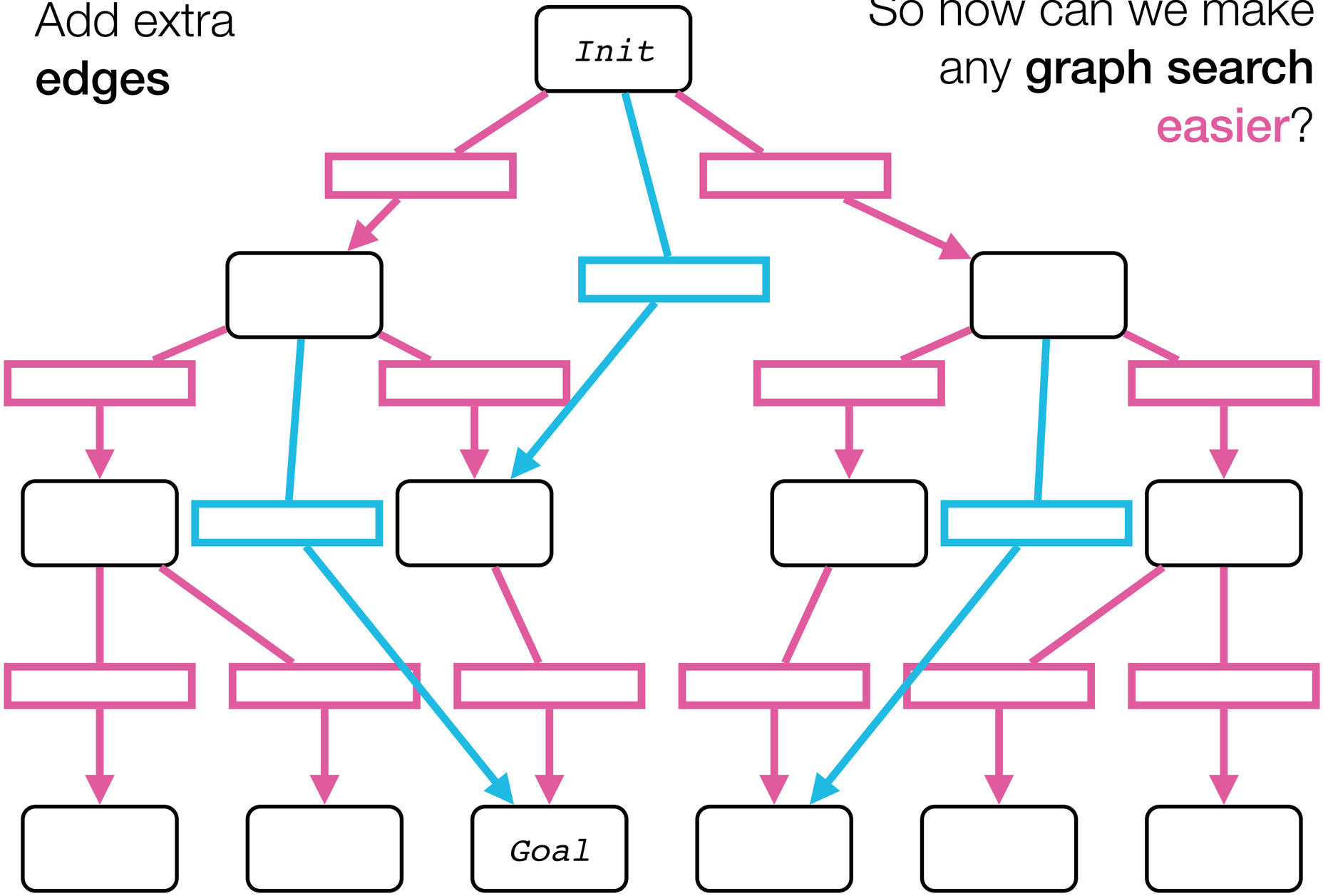
Planning is
graph search

So how can we make any **graph search** easier?



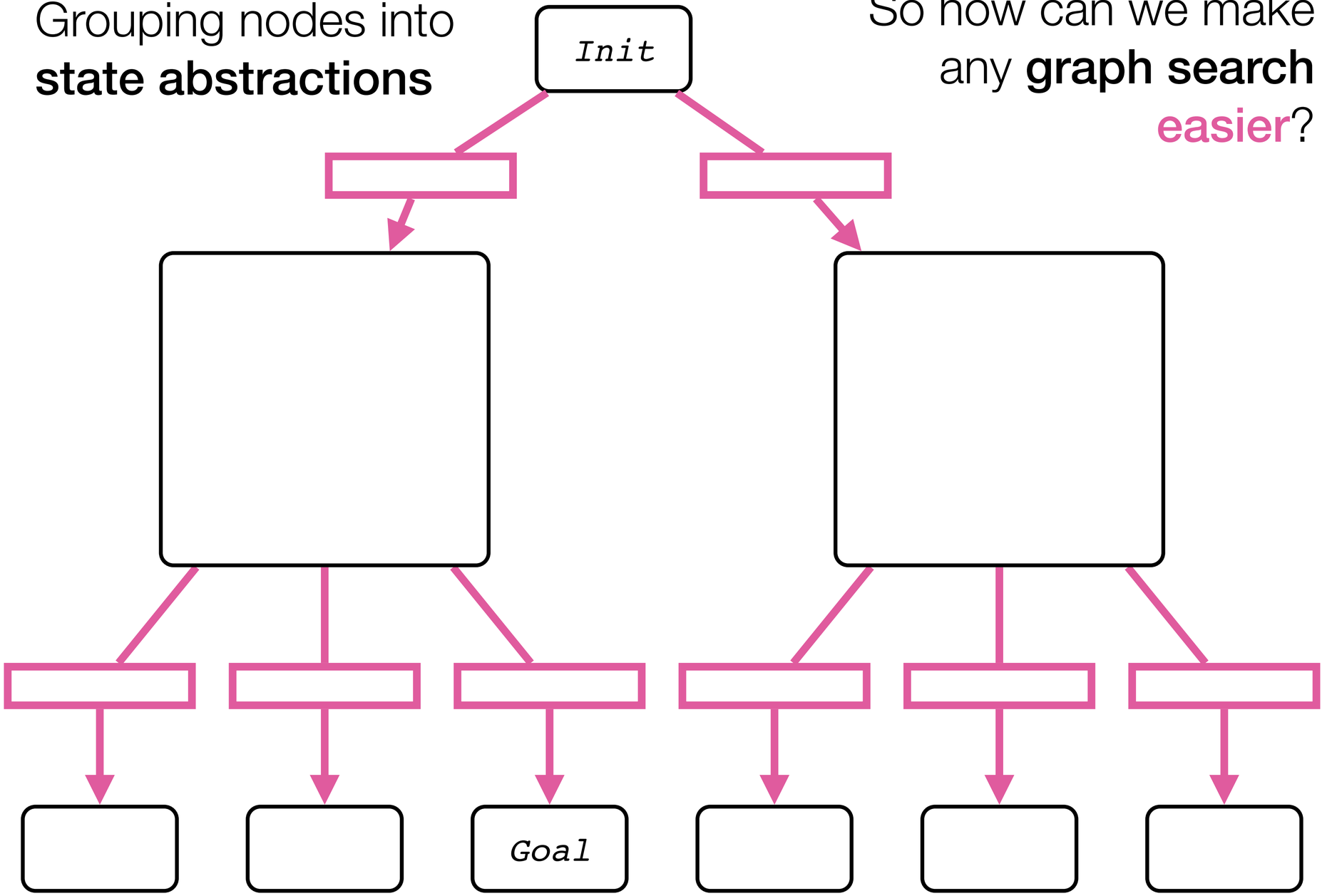
Add extra
edges

So how can we make any **graph search** easier?



Grouping nodes into **state abstractions**

So how can we make any **graph search** easier?



Add extra
edges

ignore delete lists
heuristic

Add extra
edges

ignore delete lists
heuristic

Assume goals are all **positively stated**
(*any domain can be rewritten to achieve this*)

Add extra
edges

ignore delete lists
heuristic

Assume goals are all **positively stated**
(*any domain can be rewritten to achieve this*)

Also remove the `DEL(a)` from `EFFECTS(a)`
(*prevents undoing of progress to goal*)

Add extra
edges

ignore delete lists
heuristic

Assume goals are all **positively stated**
(*any domain can be rewritten to achieve this*)

Also remove the `DEL(a)` from `EFFECTS(a)`
(*prevents undoing of progress to goal*)

This problem is still NP-hard but good solutions
can be found by hill-climbing

Add extra
edges

ignore delete lists
heuristic

Assume goals are all **positively stated**
(*any domain can be rewritten to achieve this*)

Also remove the `DEL(a)` from `EFFECTS(a)`
(*prevents undoing of progress to goal*)

This problem is still NP-hard but good solutions
can be found by hill-climbing

Solve for **every state you expand**!

Add extra
edges

ignore delete lists
heuristic

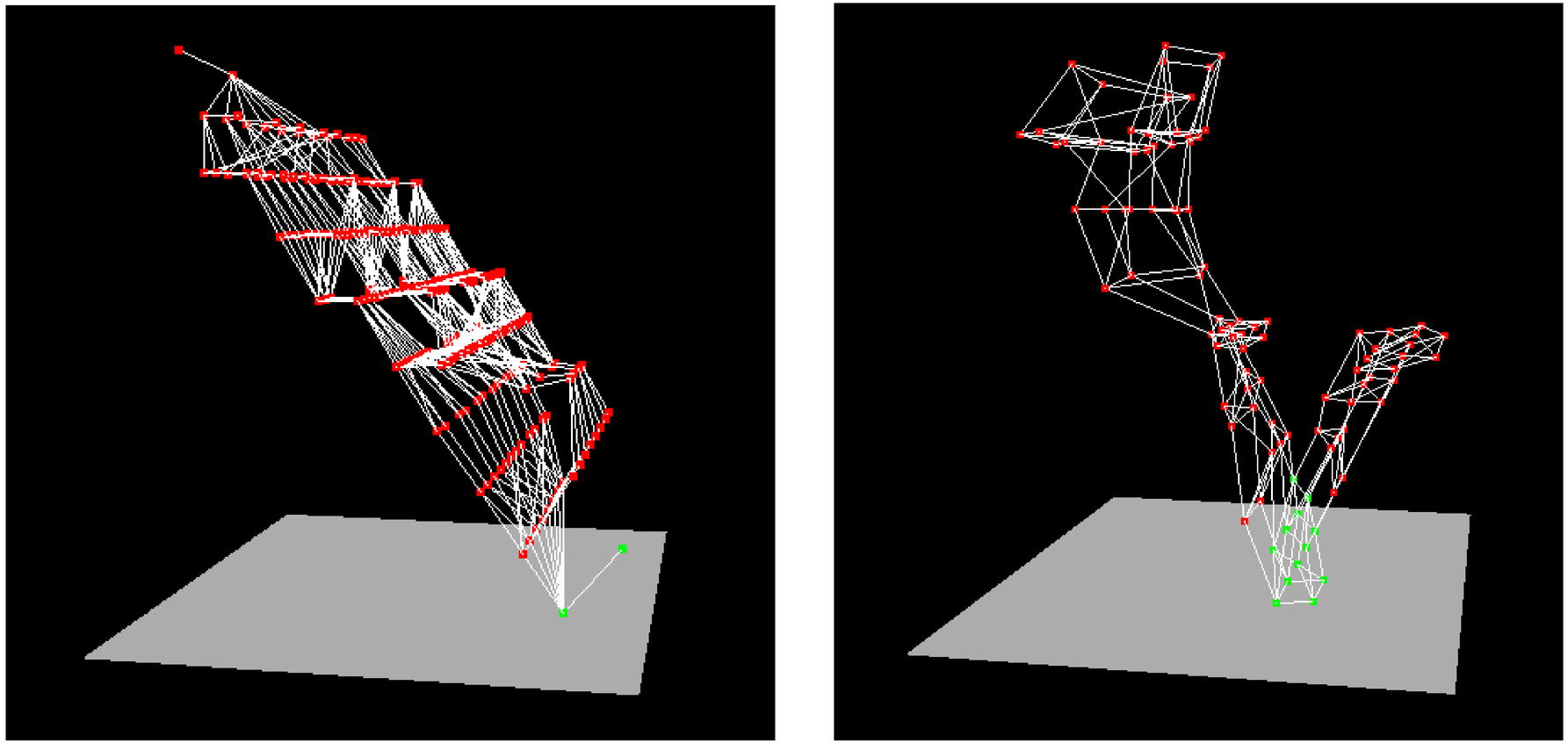
Assume goals are all **positively stated**
(*any domain can be rewritten to achieve this*)

Also remove the `DEL(a)` from `EFFECTS(a)`
(*prevents undoing of progress to goal*)

This problem is still NP-hard but good solutions
can be found by hill-climbing

Solve for **every state you expand**!

Use **the cost of the relaxed solution** (*number of actions to goal*) as the **heuristic value of that state**.



(a) (b)

Figure 1: Visualized state space under h^+ of (a) a Gripper and (b) a Logistics instance.

Grouping nodes into
state abstractions

Relaxing actions does not reduce the number of
states in a problem

Grouping nodes into
state abstractions

Relaxing actions does not reduce the number of **states** in a problem

Instead, **ignore irrelevant predicates**

Grouping nodes into
state abstractions

Relaxing actions does not reduce the number of **states** in a problem

Instead, **ignore irrelevant predicates**

For example, in the *Logistics* domain, ignore all `At` predicates that are not in goal or initial state

Grouping nodes into
state abstractions

Relaxing actions does not reduce the number of **states** in a problem

Instead, **ignore irrelevant predicates**

For example, in the *Logistics* domain, ignore all `At` predicates that are not in goal or initial state

This problem is easier to solve and can be used to guide search in a similar way

Overview

- Classical Planning: STRIPS / PDDL
- Forward vs Backward Search
- Planning Heuristics
- **The Planning Graph**
- Partial Order Planning

A **planning graph** is a data structure that can be used for both **improved heuristic estimates** and **planning**

A **planning graph** is a data structure that can be used for both **improved heuristic estimates** and **planning**

It allows us to determine if G can be reached from S_0 by lookup (i.e. *quickly*)

A **planning graph** is a data structure that can be used for both **improved heuristic estimates** and **planning**

It allows us to determine if G can be reached from S_0 by lookup (i.e. *quickly*)

It is a **polynomial-size approximation** of the full problem

- ➡ **admissible** estimate of number of steps from S_0 to G
- ➡ **definitive** answer if G is *not* reachable from S_0
- ➡ non-**definitive** answer if G is reachable from S_0

$Action(Eat(Cake)$

PRECOND: $Have(Cake)$

EFFECT: $\neg\ Have(Cake)\ \wedge\ Eaten(Cake))$

$Action(Bake(Cake)$

PRECOND: $\neg\ Have(Cake)$

EFFECT: $Have(Cake))$

$Init(Have(Cake))$
$Goal(Have(Cake) \ \wedge \ Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg \ Have(Cake) \ \wedge \ Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg \ Have(Cake)$
EFFECT: $Have(Cake))$

Planning graph **levels**

S_0 A_0 S_1 A_1 S_2

Planning graph **levels**

S_0 A_0 S_1 A_1 S_2

S levels - *approximately* all the **literals** that *could* hold at time t

Planning graph **levels**

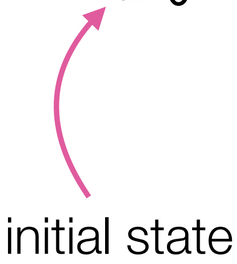
S_0 A_0 S_1 A_1 S_2

initial state

S levels - *approximately* all the **literals** that *could* hold at time t

Planning graph **levels**

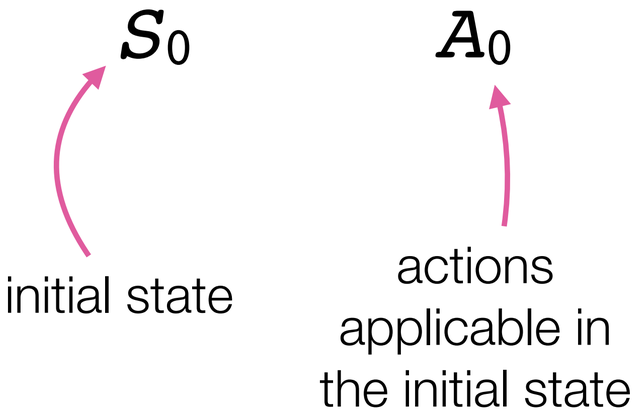
S_0 A_0 S_1 A_1 S_2



S levels - *approximately* all the **literals** that *could* hold at time t

A levels - *approximately* all the **actions** that *could* have their preconditions satisfied at time t

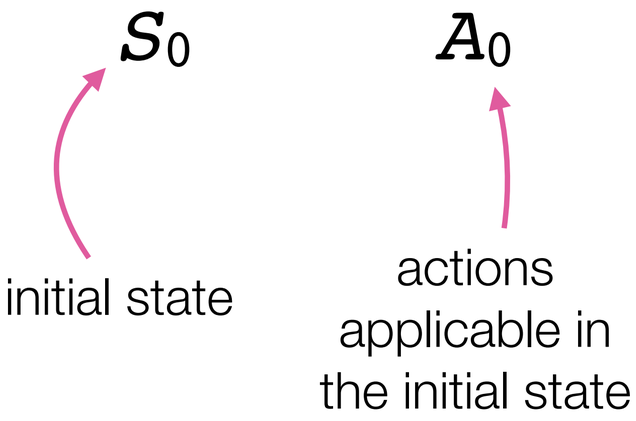
Planning graph **levels**



S levels - *approximately* all the **literals** that *could* hold at time t

A levels - *approximately* all the **actions** that *could* have their preconditions satisfied at time t

Planning graph **levels**

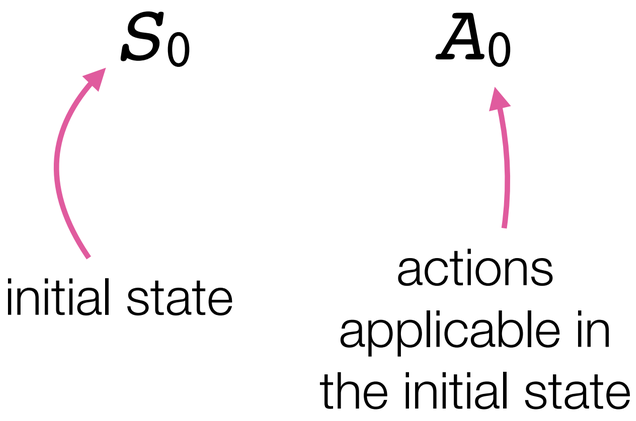


S levels - *approximately* all the **literals** that *could* hold at time t

A levels - *approximately* all the **actions** that *could* have their preconditions satisfied at time t

"approximately" because only some negative interactions are considered
(remember, it's an approximation)

Planning graph **levels**



The **level** at which a **literal** appears is a good estimate of its cost

The graph will **never** *over estimate*
(it's an easier problem)

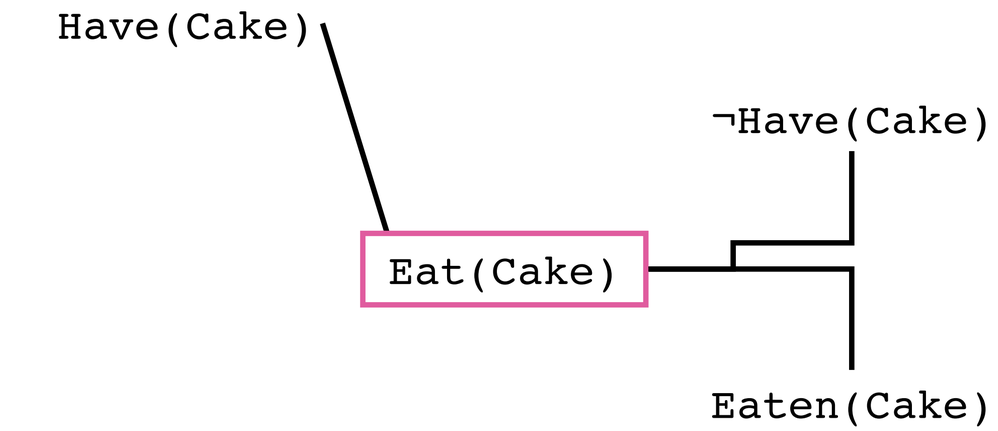
"approximately" because only some negative interactions are considered
(remember, it's an approximation)

S_0 A_0 S_1 A_1 S_2

Have(Cake)

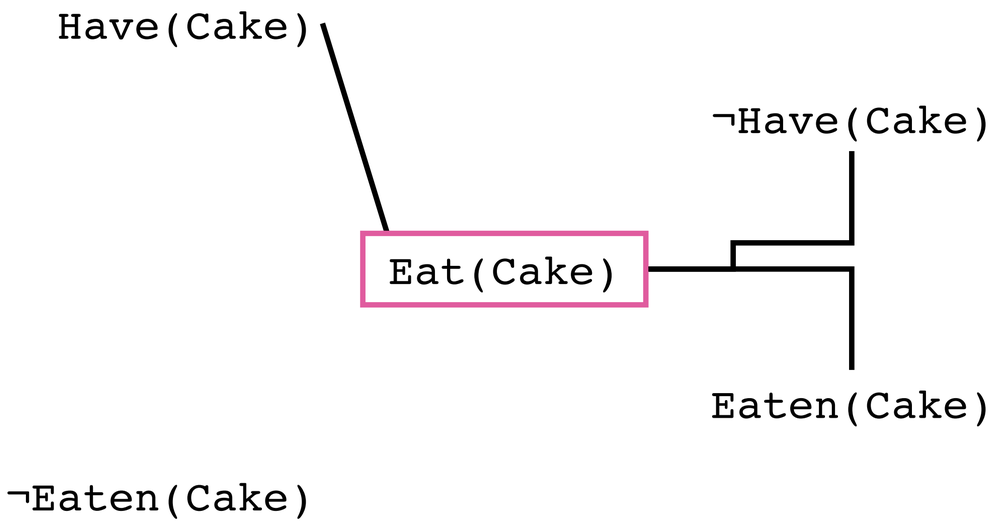
¬Eaten(Cake)

S_0 A_0 S_1 A_1 S_2



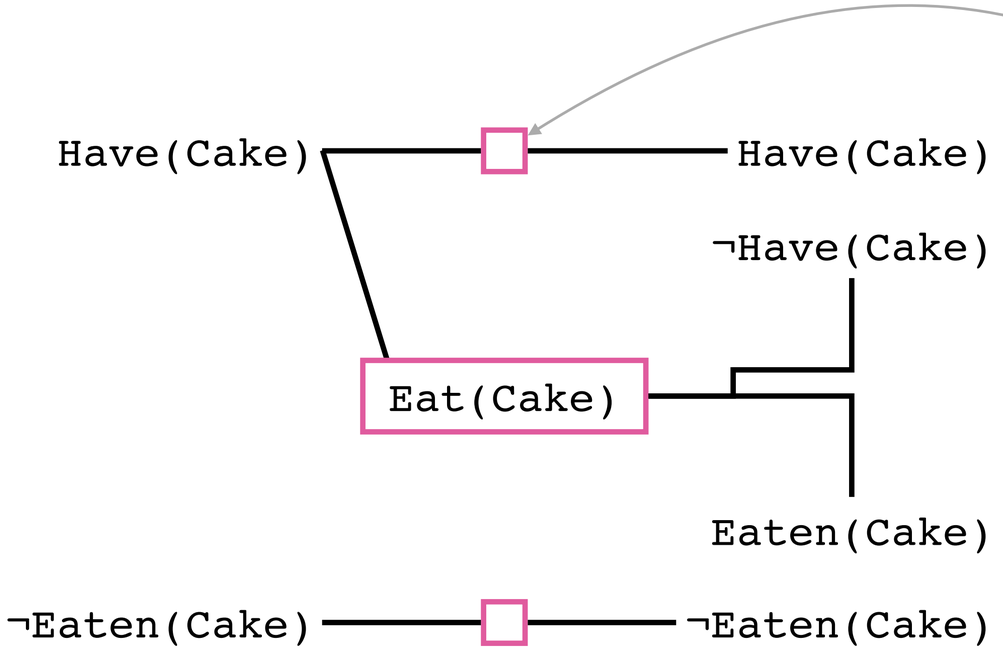
¬Eaten(Cake)

S_0 A_0 S_1 A_1 S_2



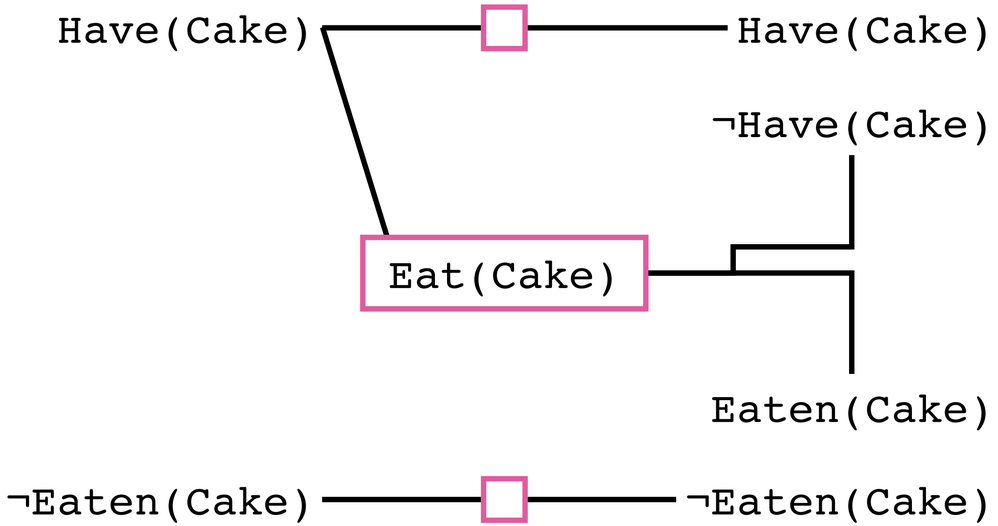
What about the things that ***haven't*** changed?

S_0 A_0 S_1 A_1 S_2



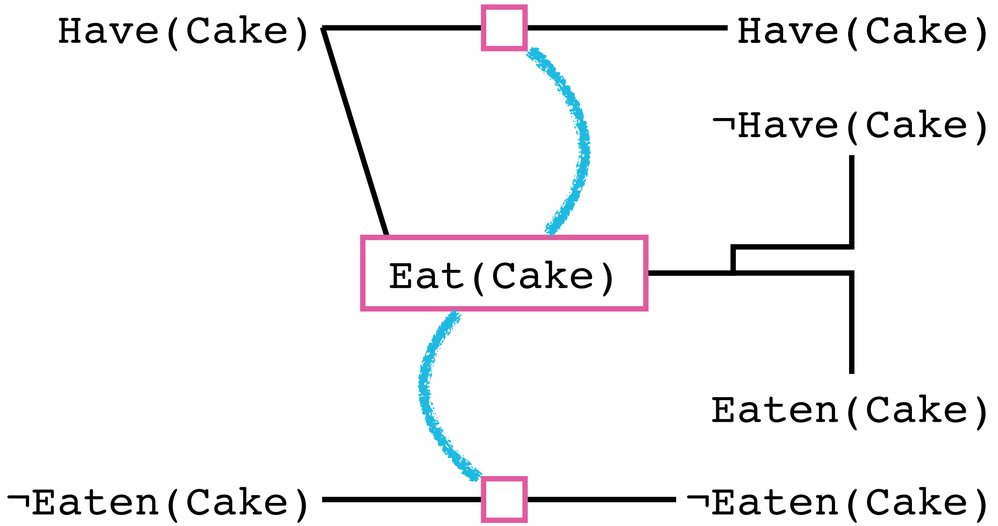
persistence actions show the effects of an action not affecting a literal

S_0 A_0 S_1 A_1 S_2



Not all actions or literals can occur together

S_0 A_0 S_1 A_1 S_2

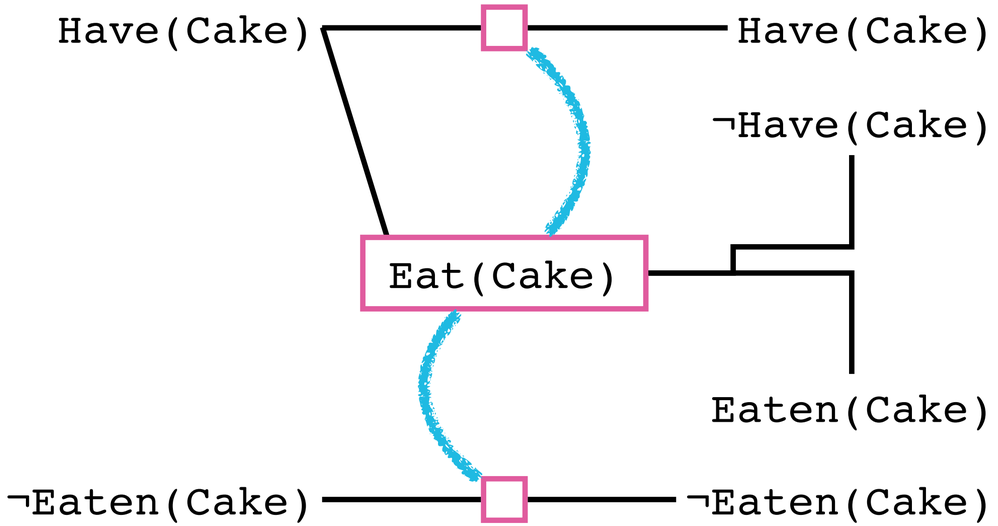


Not all actions or literals can occur together

mutex relations in the graph record which actions and literals are **mutually exclusive**

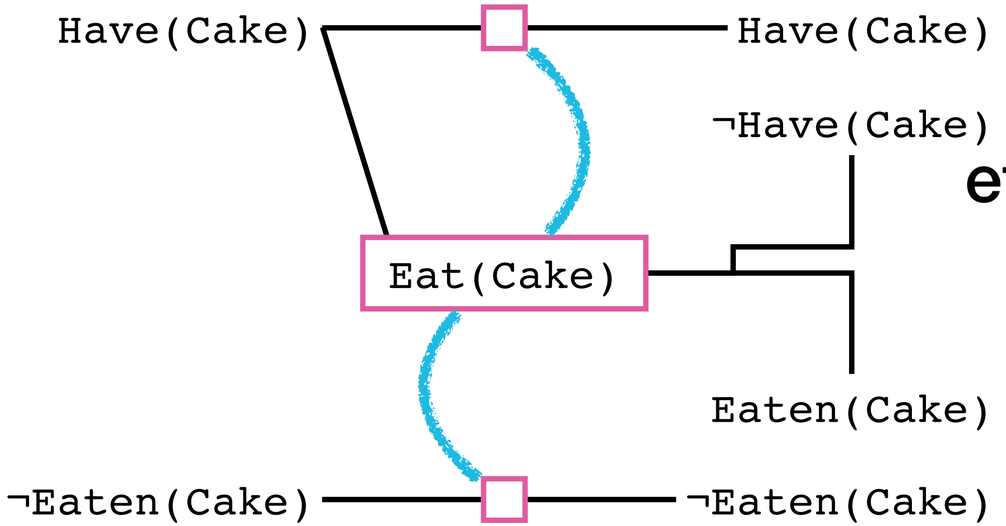
S_0 A_0 S_1 A_1 S_2

these actions are **mutex** because



S_0 A_0 S_1 A_1 S_2

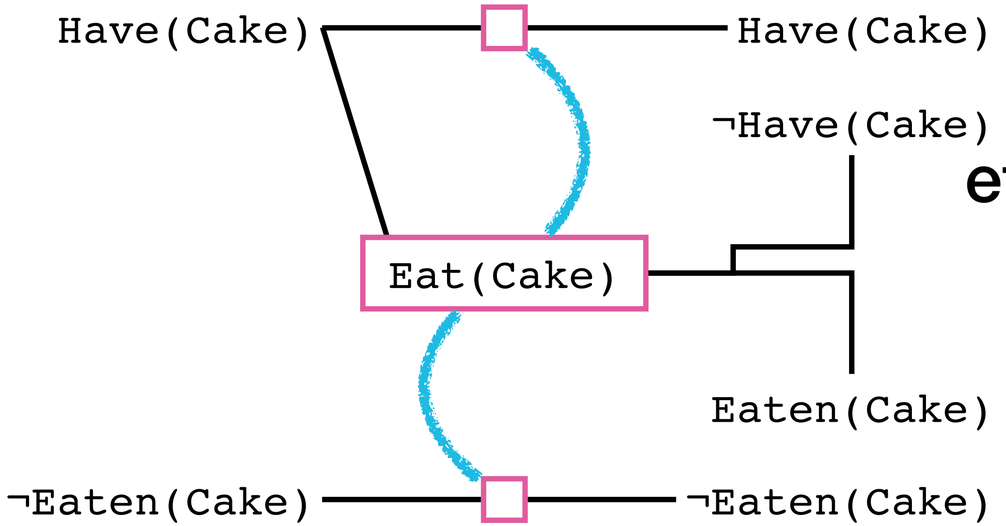
these actions are **mutex** because



they have **inconsistent effects** (their effects disagree)

S_0 A_0 S_1 A_1 S_2

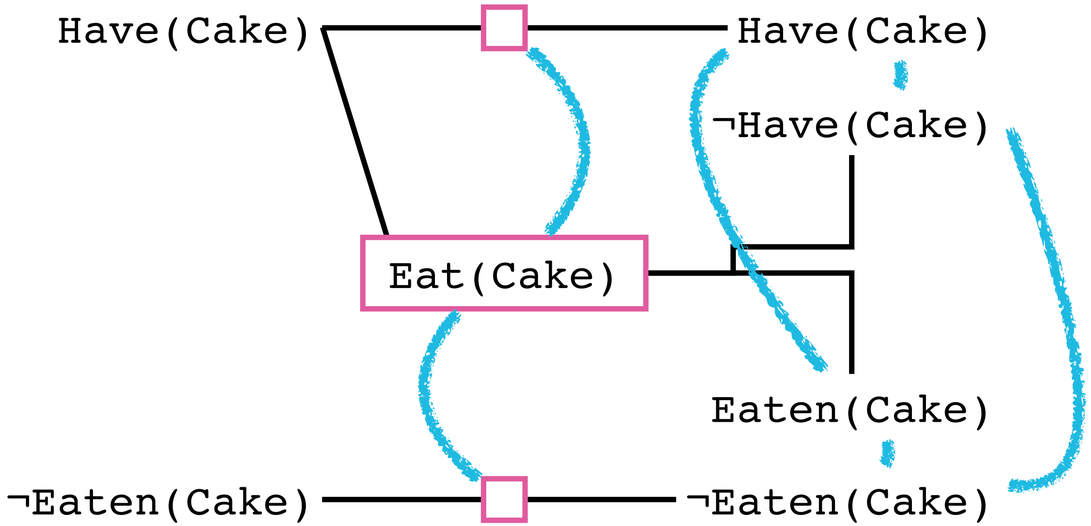
these actions are **mutex** because



they have **inconsistent effects** (their effects disagree)

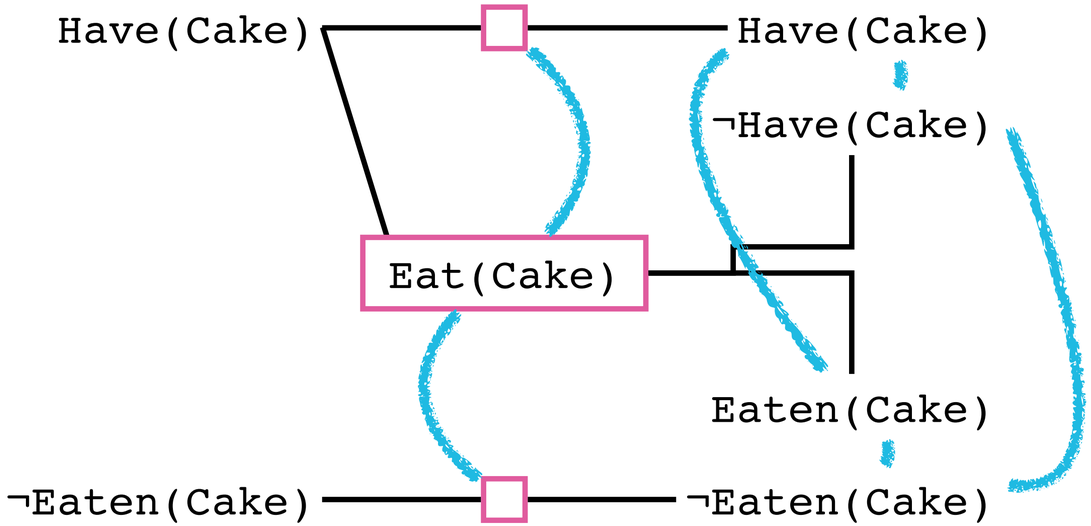
they **interfere** (an effect of one negates the a precondition of another)

S_0 A_0 S_1 A_1 S_2



literals are **mutex**
if the **actions**
which produce
them are **mutex**

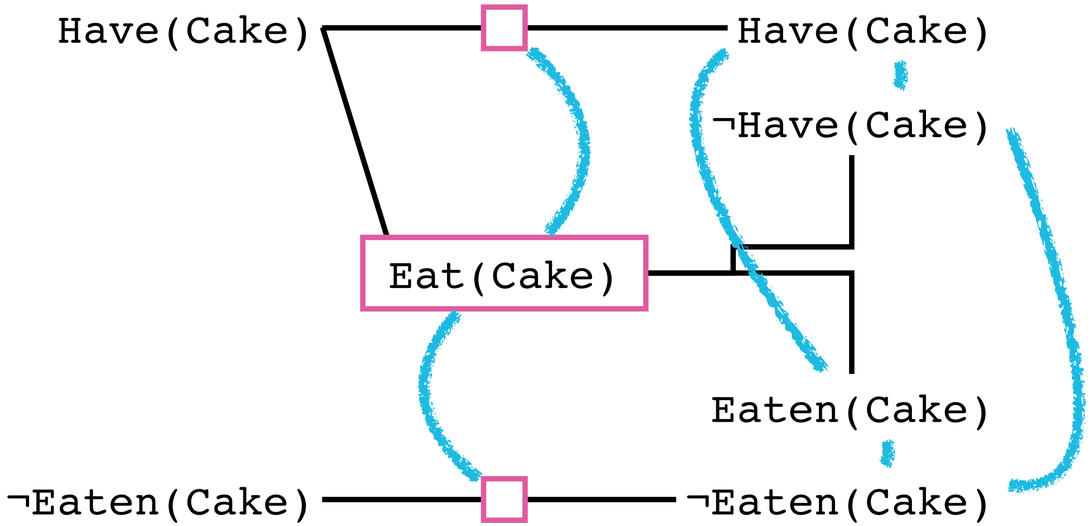
S_0 A_0 S_1 A_1 S_2



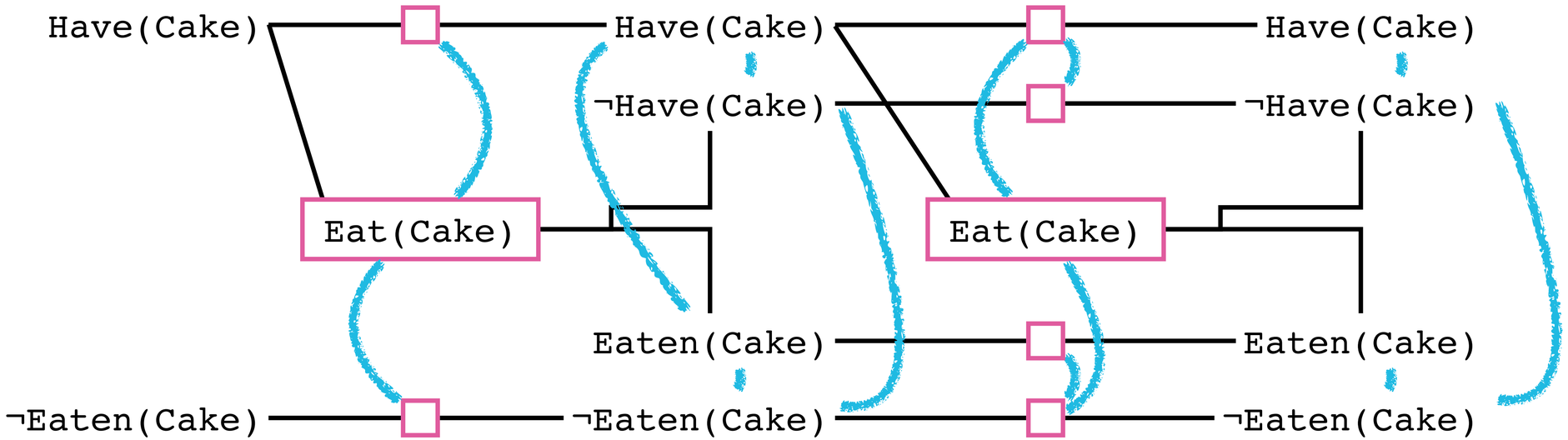
literals are **mutex**
if the **actions**
which produce
them are **mutex**

Only legal states are: `Have(Cake)` ∧ `¬Eaten(Cake)`
`¬Have(Cake)` ∧ `Eaten(Cake)`

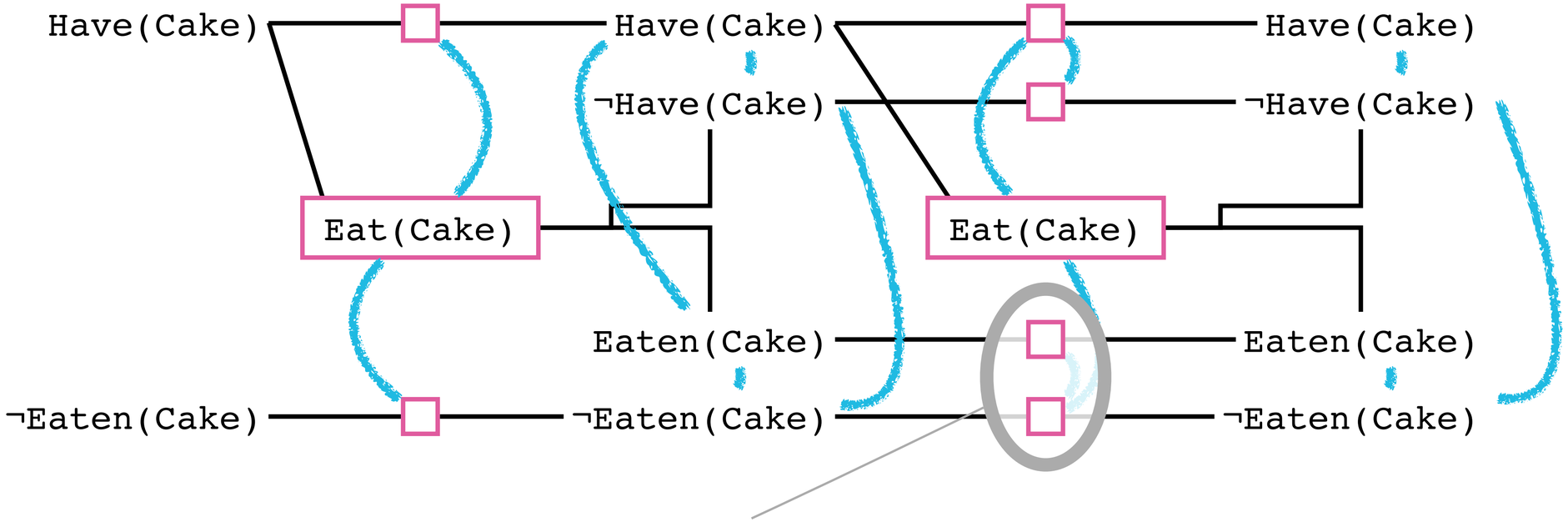
S_0 A_0 S_1 A_1 S_2


Have(Cake)
Have(Cake)
¬Have(Cake)
Eat(Cake)
Eaten(Cake)
¬Eaten(Cake)
¬Eaten(Cake)

S_0 A_0 S_1 A_1 S_2

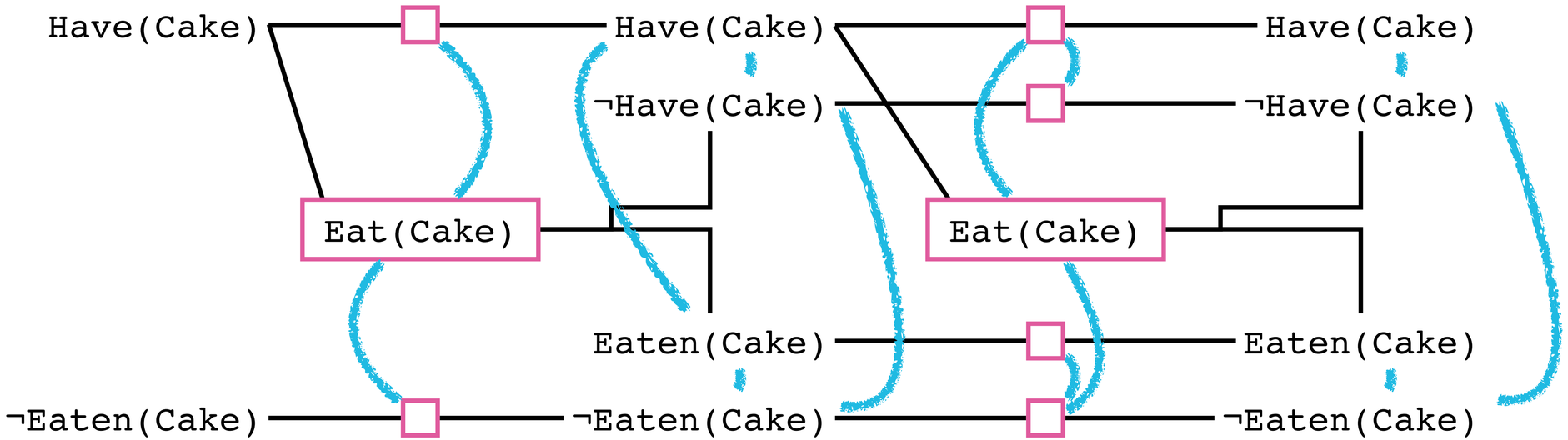

Have(Cake)
¬Have(Cake)
Eat(Cake)
Eaten(Cake)
¬Eaten(Cake)
Have(Cake)
¬Have(Cake)
Eat(Cake)
Eaten(Cake)
¬Eaten(Cake)
Have(Cake)
¬Have(Cake)
Eaten(Cake)
¬Eaten(Cake)

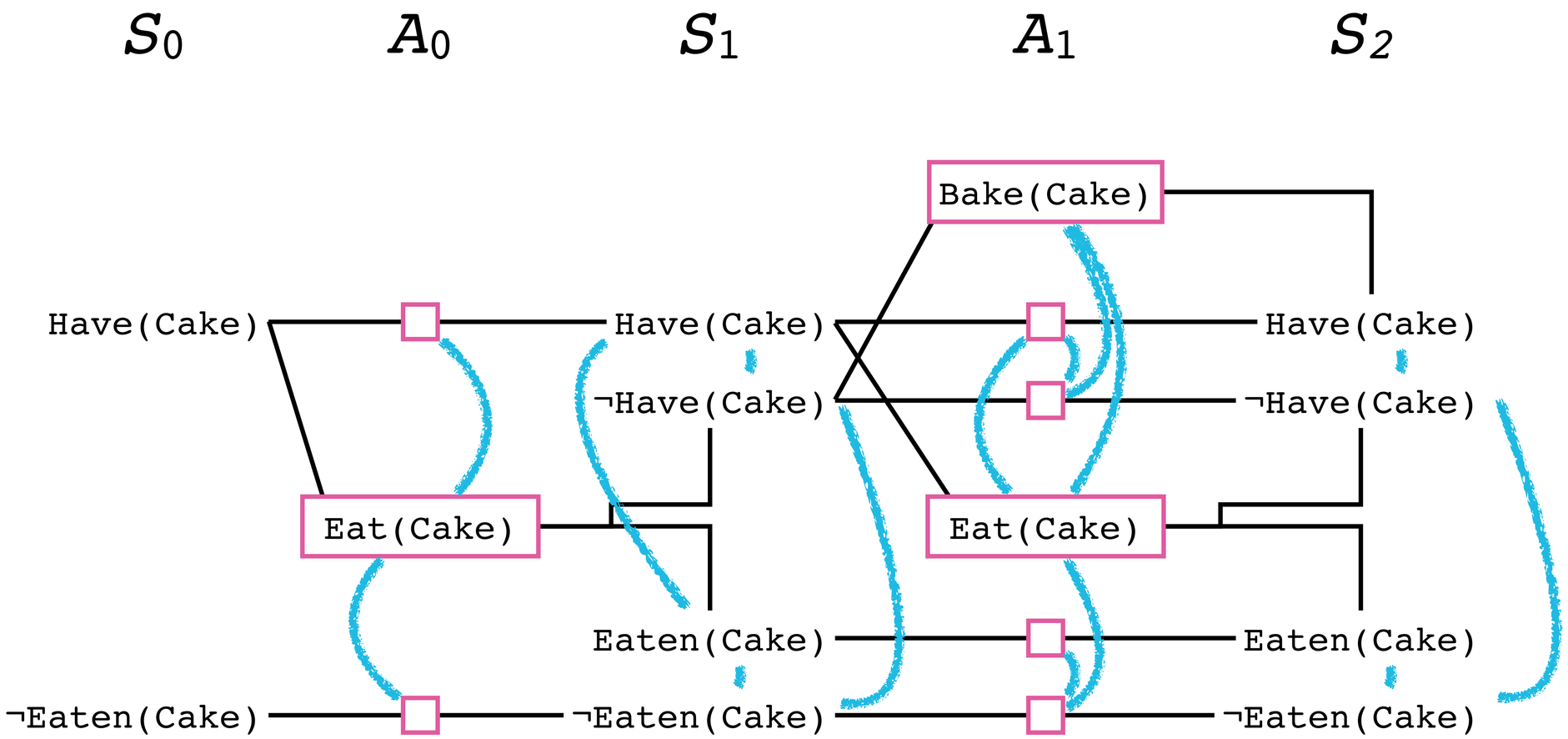
S_0 A_0 S_1 A_1 S_2



they have **competing needs** (their preconditions are **mutex**)

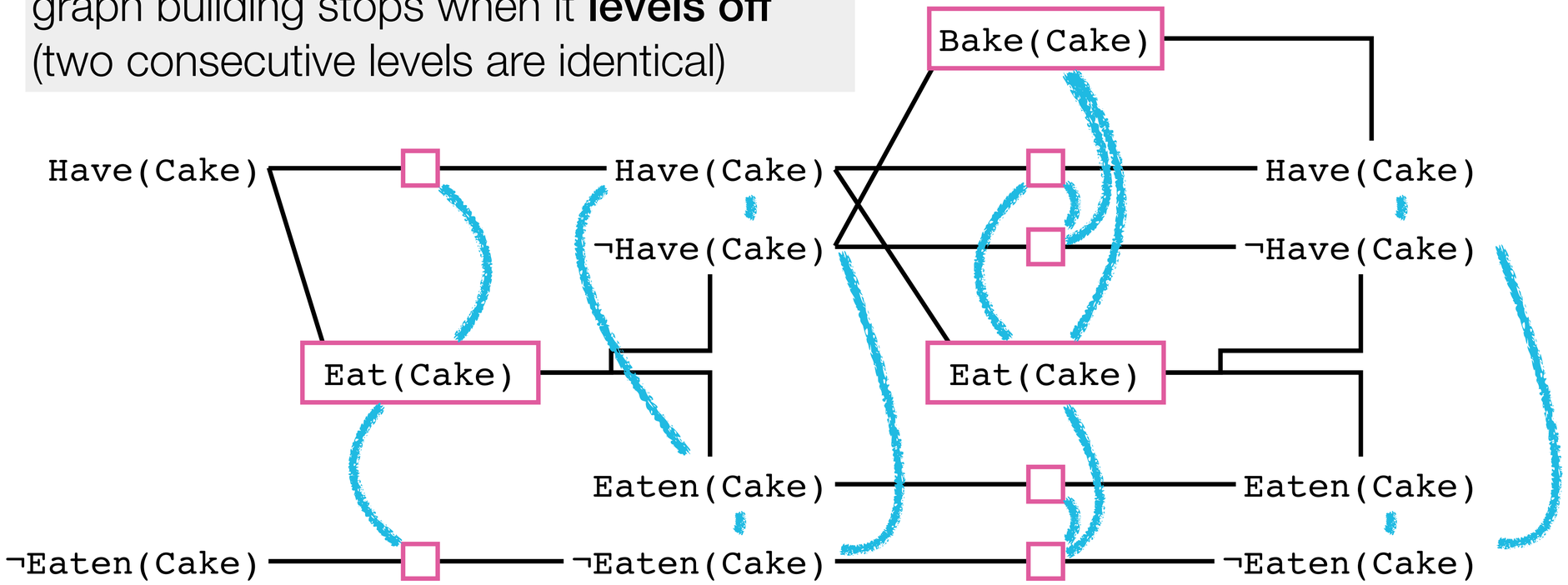
S_0 A_0 S_1 A_1 S_2


Have(Cake)
¬Have(Cake)
Eat(Cake)
Eaten(Cake)
¬Eaten(Cake)
Have(Cake)
¬Have(Cake)
Eat(Cake)
Eaten(Cake)
¬Eaten(Cake)
Have(Cake)
¬Have(Cake)
Eaten(Cake)
¬Eaten(Cake)


S_0
A_0
S_1
A_1
S_2
Bake(Cake)
Have(Cake)
Have(Cake)
Have(Cake)
¬Have(Cake)
¬Have(Cake)
Eat(Cake)
Eat(Cake)
Eaten(Cake)
Eaten(Cake)
¬Eaten(Cake)
¬Eaten(Cake)
¬Eaten(Cake)

S_0 A_0 S_1 A_1 S_2

graph building stops when it **levels off**
(two consecutive levels are identical)



graph building does not require **search**

time and memory: $O(\ n(a+1)^2\ )$

Init(At(C_1, BHX) ∧ At(P_1, BHX) ∧
Cargo(C_1) ∧ Plane(P_1) ∧
Airport(BHX) ∧ Airport(FCO))

Goal(At(C_1, FCO))

Task: Build the planning graph, ignoring invariants (**`Cargo, Plane, Airport`**)

actions are **mutex** because:

- they have **inconsistent effects** (their effects disagree)
- they **interfere** (an effect of one negates the a precondition of another)
- they have **competing needs** (their preconditions are **mutex**)

A **planning graph** is a data structure that can be used for both **improved heuristic estimates** and **planning**

A **planning graph** is a data structure that can be used for both

improved heuristic estimates

A **planning graph** is a data structure that can be used for both

improved heuristic estimates

reachability of goal literal g_i

A **planning graph** is a data structure that can be used for both
improved heuristic estimates

reachability of goal literal g_i

level cost of g_i
(better with a serial planning graph)

A **planning graph** is a data structure that can be used for both
improved heuristic estimates

reachability of goal literal g_i

level cost of g_i
(better with a serial planning graph)

level sum of goal conjunction of G

A **planning graph** is a data structure that can be used for both
improved heuristic estimates

reachability of goal literal g_i

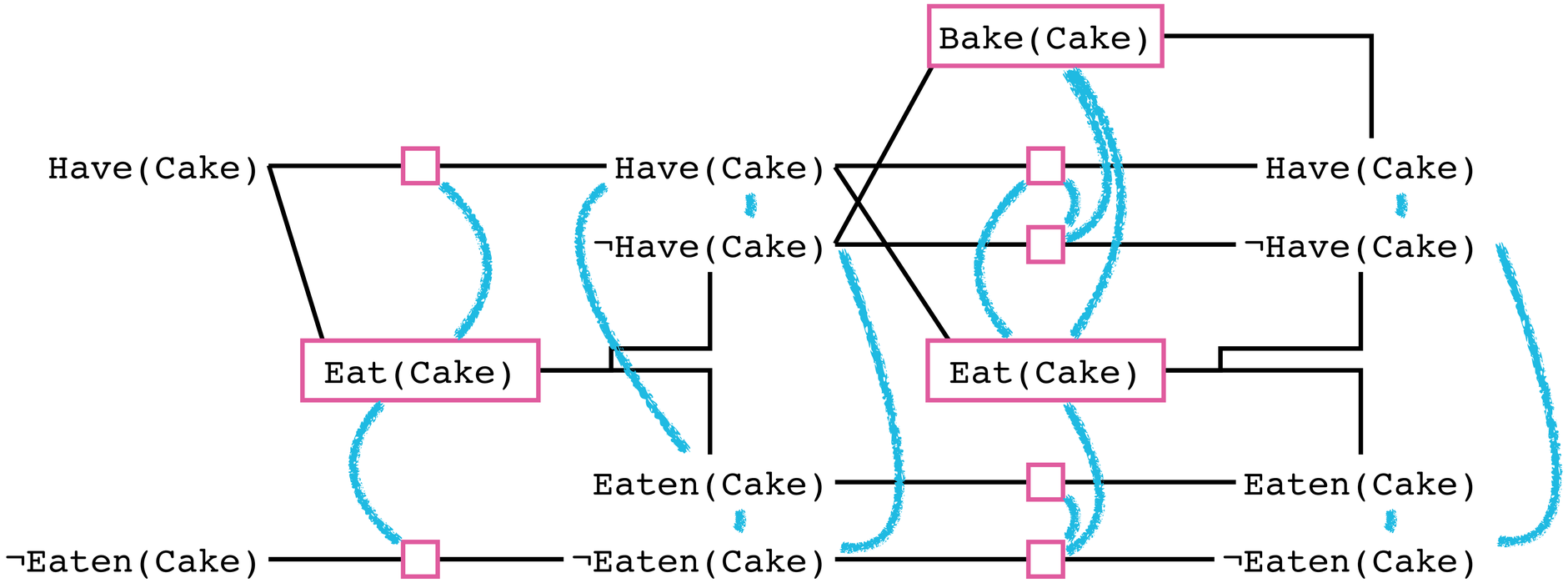
level cost of g_i
(better with a serial planning graph)

level sum of goal conjunction of G

level set of goal conjunction of G

$Goal($ Have(Cake) $\wedge$ Eaten(Cake))

S_0 A_0 S_1 A_1 S_2



$$On(A, Table) \land On(B, Table) \land On(C, Table)$$

$$On(A, B) \land On(B, C) \land On(C, A)$$

$$On(A, Table) \land On(B, Table) \land On(C, Table)$$

$$On(A, B) \land On(B, C) \land On(C, A)$$

mutexes are only calculated for **pairs**, so this goal looks possible to a planning graph

A **planning graph** is a data structure that can be used for both **improved heuristic estimates** and **planning**

A **planning graph** is a data structure that can be used for both

planning

```
for t = 0 to inf do:
    if all of goals non-mutex in S_t of graph then
      solution = ExtractSolution(graph, goals)
      if solution != failure then
        return solution
    if graph and nogoods have both leveled off then
      return failure
    graph = ExpandGraph(graph)
```

```
ExtractSolution(graph, goals)
```

`ExtractSolution(graph, goals)`

Initial state $\mathbf{S_n}$ is the last level of the *graph*,
plus the *goals* to achieve

`ExtractSolution(graph, goals)`

Initial state $\mathbf{S_n}$ is the last level of the *graph*,
plus the *goals* to achieve

Successors any non-**mutex** **set** of *actions*
from $\mathbf{A_{n-1}}$ which cover the *goals*

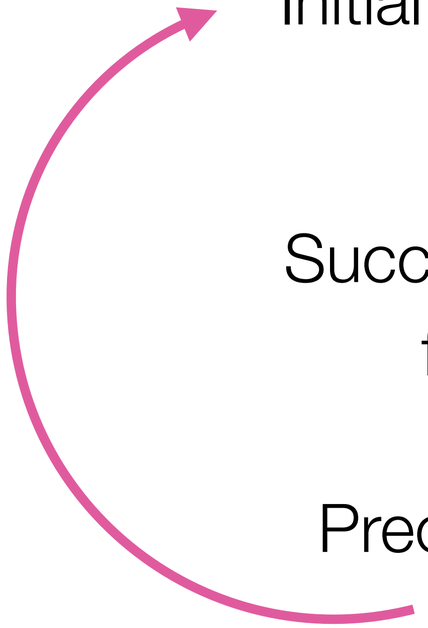
`ExtractSolution(graph, goals)`

Initial state $\mathbf{S_n}$ is the last level of the *graph*,
plus the *goals* to achieve

Successors any non-**mutex** **set** of *actions*
from $\mathbf{A_{n-1}}$ which cover the *goals*

Preconditions of this *action* **set** become
the *goals* for $\mathbf{S_{n-1}}$

`ExtractSolution(graph, goals)`

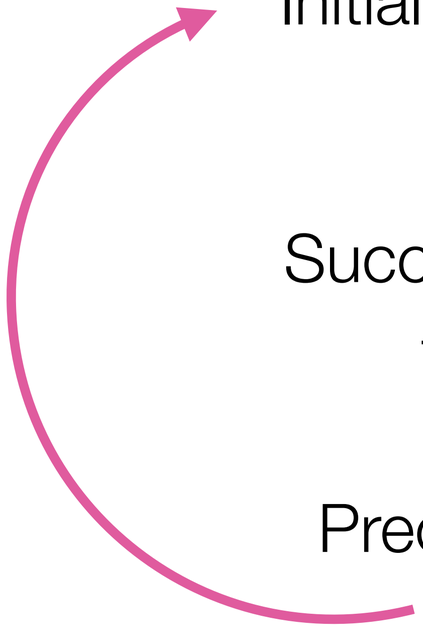


Initial state $\mathbf{S_n}$ is the last level of the *graph*,
plus the *goals* to achieve

Successors any non-**mutex** **set** of *actions*
from $\mathbf{A_{n-1}}$ which cover the *goals*

Preconditions of this *action* **set** become
the *goals* for $\mathbf{S_{n-1}}$

`ExtractSolution(graph, goals)`



Initial state $\mathbf{S_n}$ is the last level of the *graph*,
plus the *goals* to achieve

Successors any non-**mutex** **set** of *actions*
from $\mathbf{A_{n-1}}$ which cover the *goals*

Preconditions of this *action* **set** become
the *goals* for $\mathbf{S_{n-1}}$

Aim to terminate at $\mathbf{S_0}$
with all *goals* having been satisfied

$Goal($ Have(Cake) $\wedge$ Eaten(Cake))

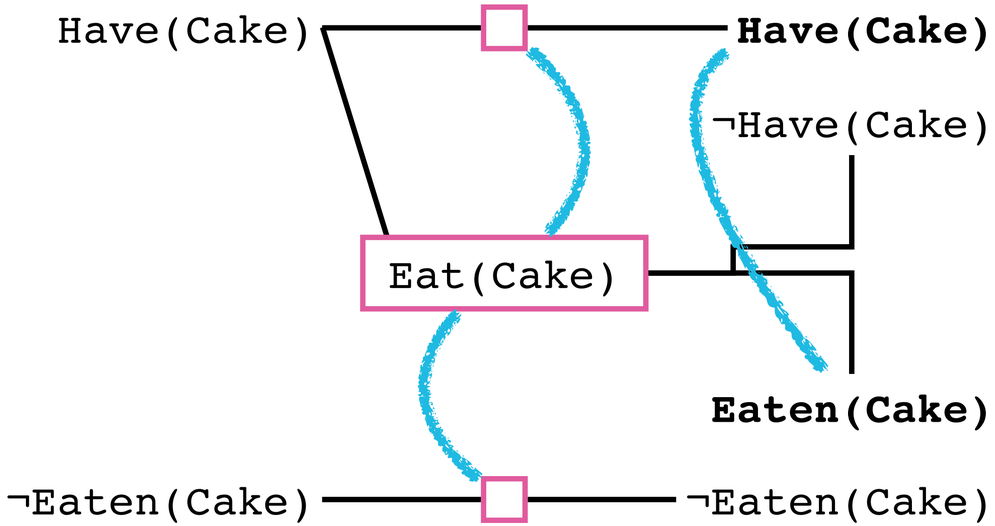
S_0 A_0 S_1 A_1 S_2

Have(Cake)

¬Eaten(Cake)

$Goal($ Have(Cake) $\wedge$ Eaten(Cake))

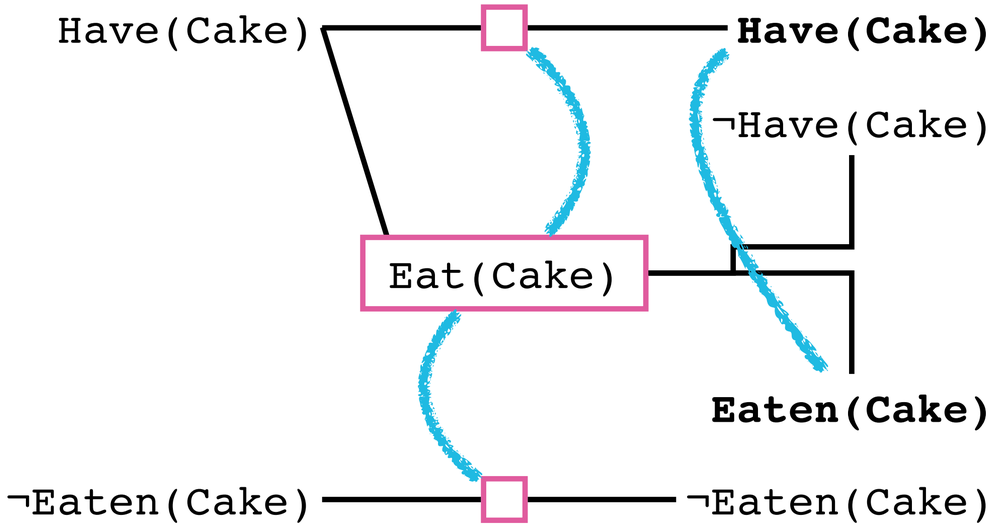
S_0 A_0 S_1 A_1 S_2



if all of goals non-mutex in S_t of graph then

$Goal($ Have(Cake) $\wedge$ Eaten(Cake))

S_0 A_0 S_1 A_1 S_2

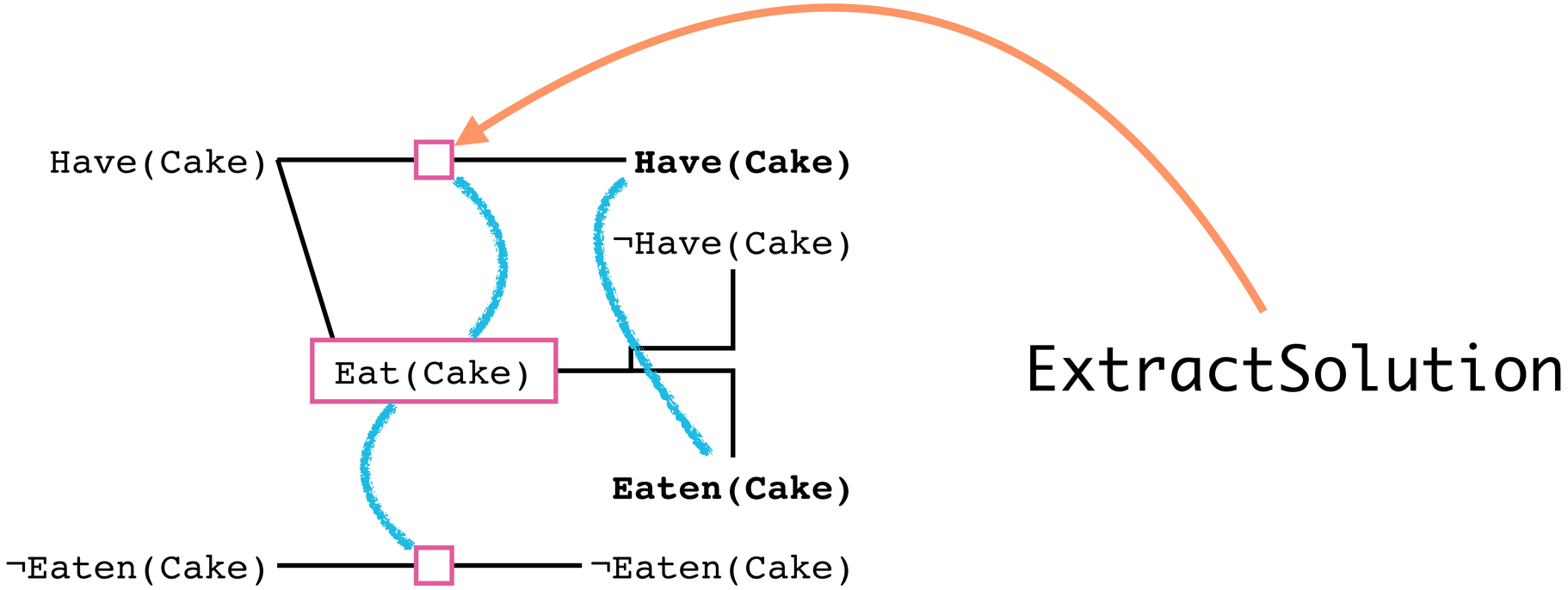


ExtractSolution

if all of goals non-mutex in S_t of graph then

$Goal($ Have(Cake) $\wedge$ Eaten(Cake))

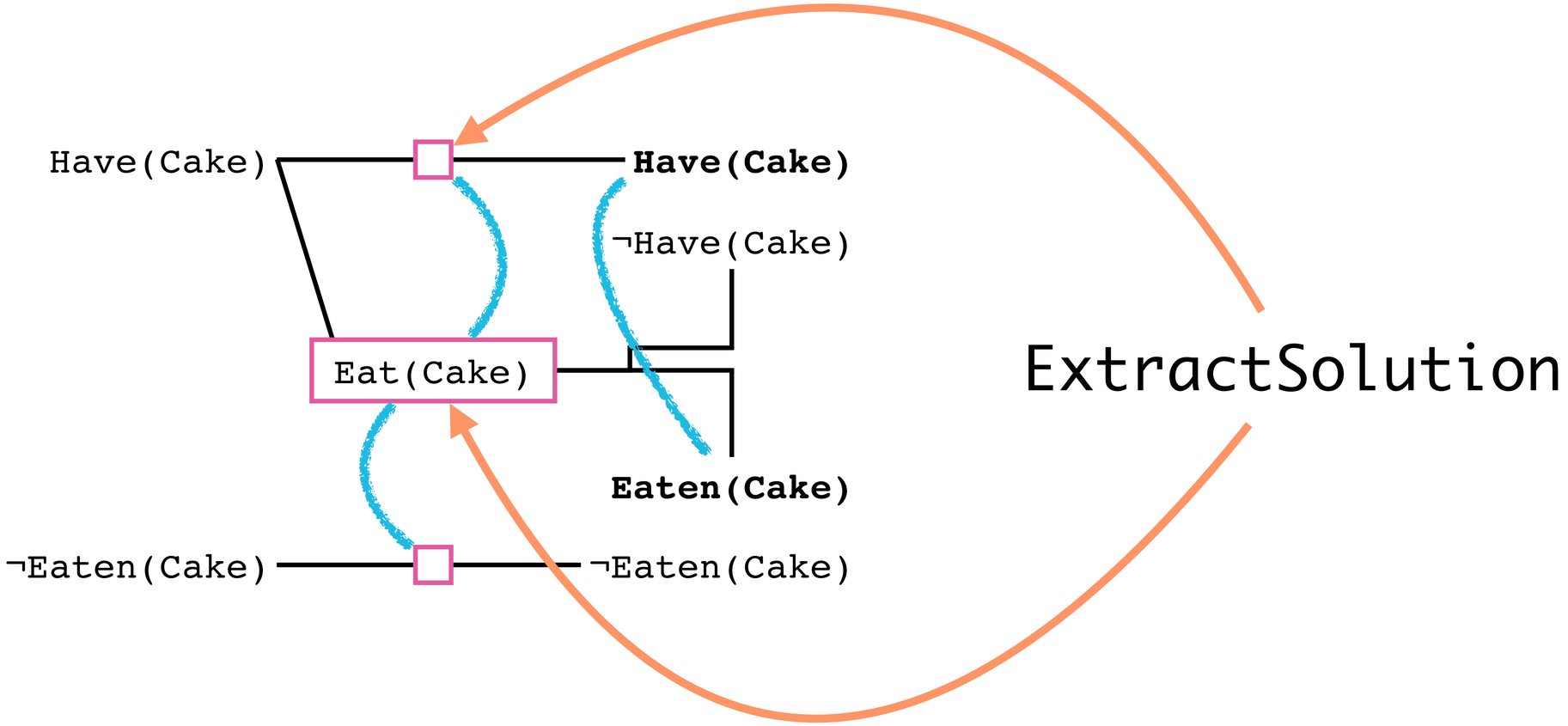
S_0 A_0 S_1 A_1 S_2



if all of goals non-mutex in S_t of graph then

Goal(Have(Cake) ∧ Eaten(Cake))

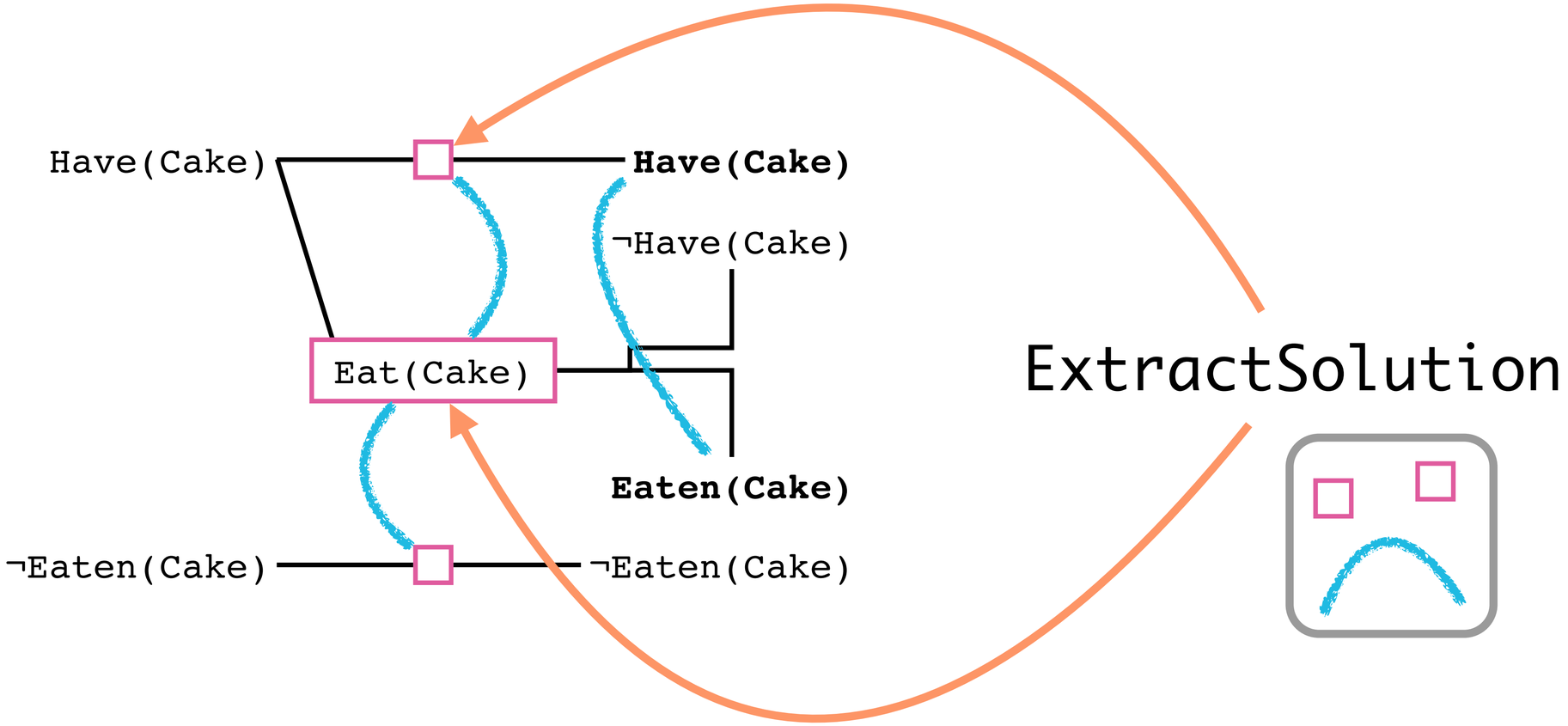
S_0 A_0 S_1 A_1 S_2



if all of goals non-mutex in S_t of graph then

Goal(Have(Cake) ∧ Eaten(Cake))

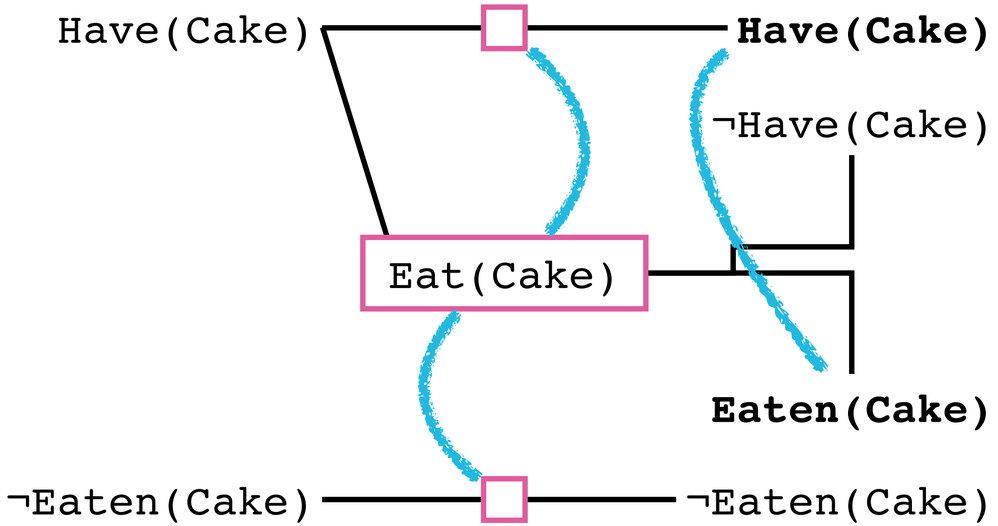
S_0 A_0 S_1 A_1 S_2



if all of goals non-mutex in S_t of graph then

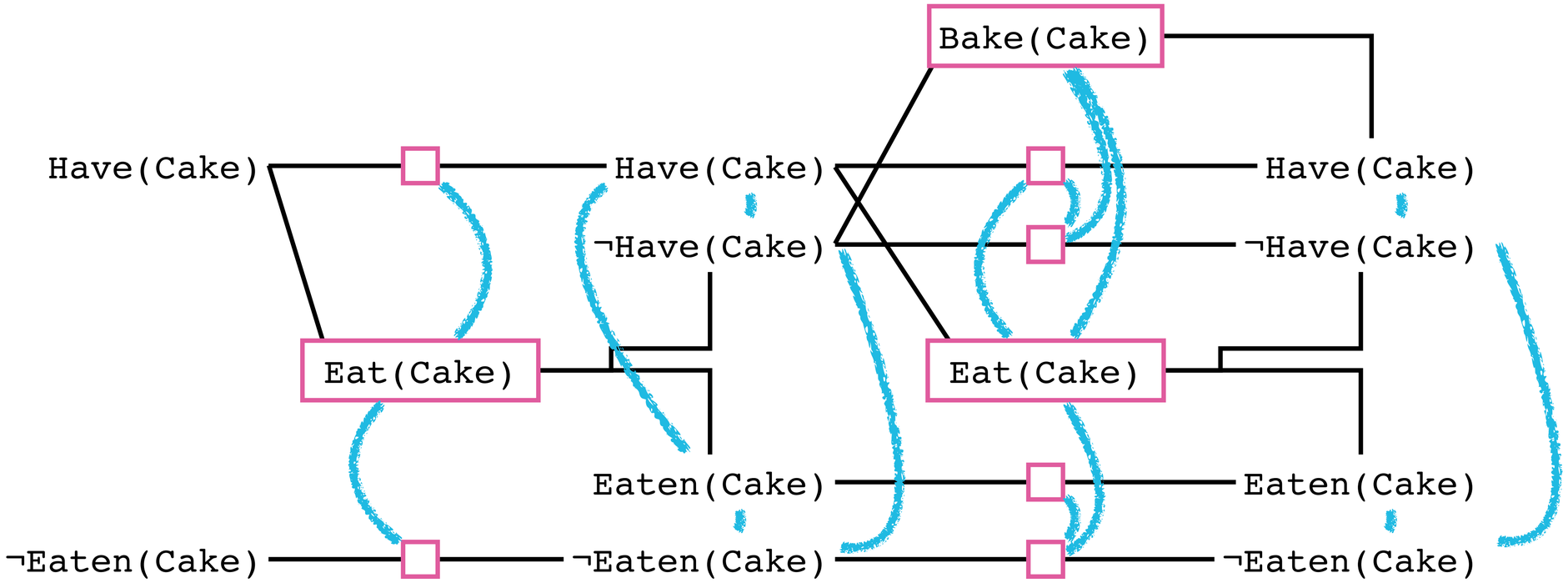
$$Goal(\ \text{Have(Cake)} \wedge \text{Eaten(Cake)})$$

S_0 A_0 S_1 A_1 S_2



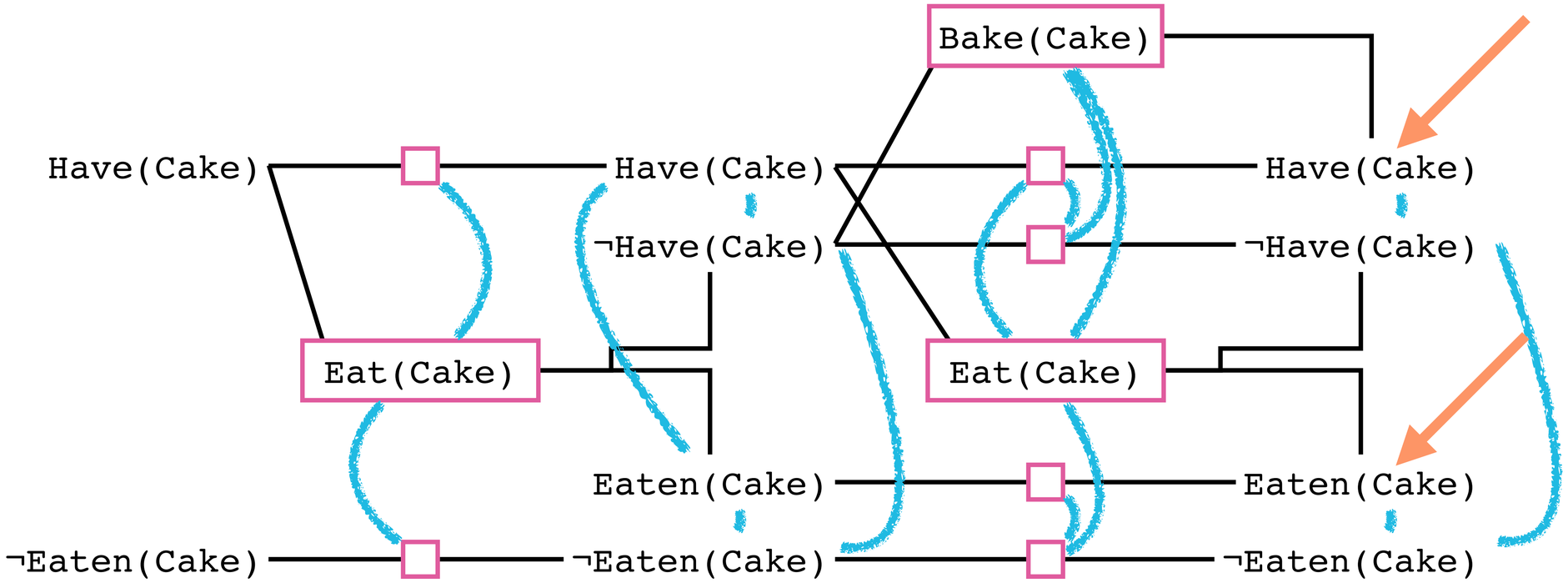
$Goal($ Have(Cake) $\wedge$ Eaten(Cake))

S_0 A_0 S_1 A_1 S_2



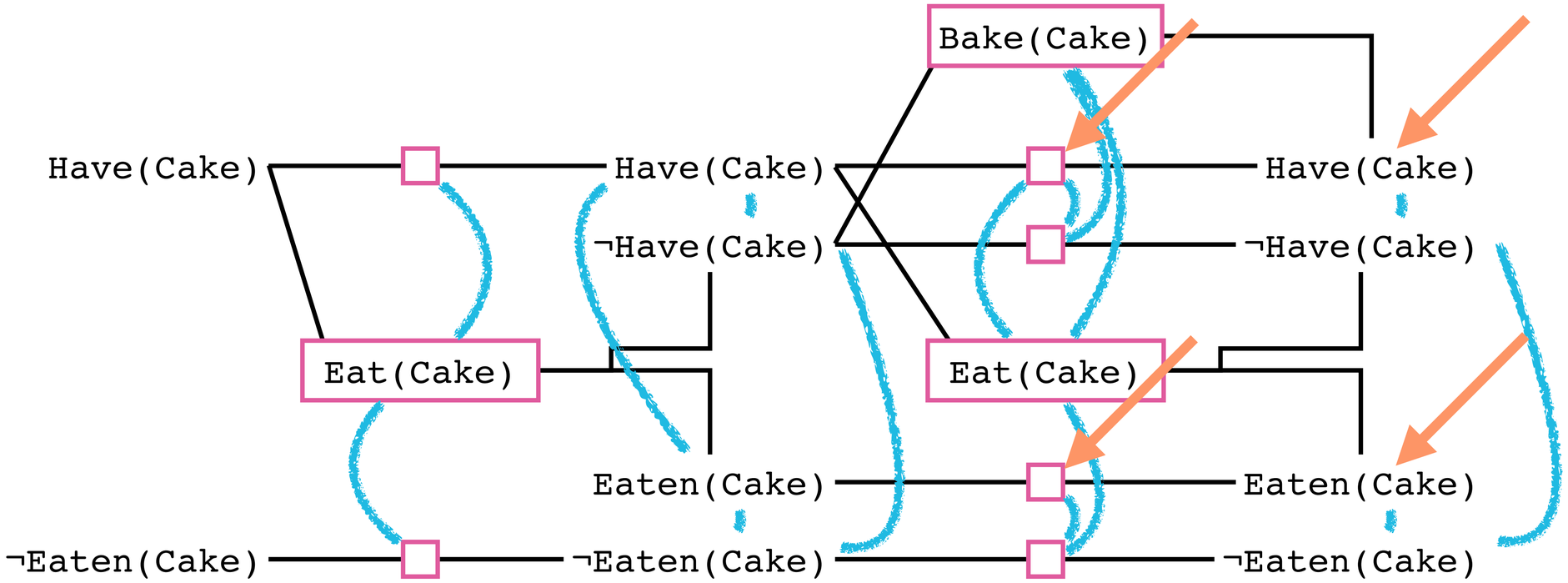
$Goal($ Have(Cake) $\wedge$ Eaten(Cake))

S_0 A_0 S_1 A_1 S_2



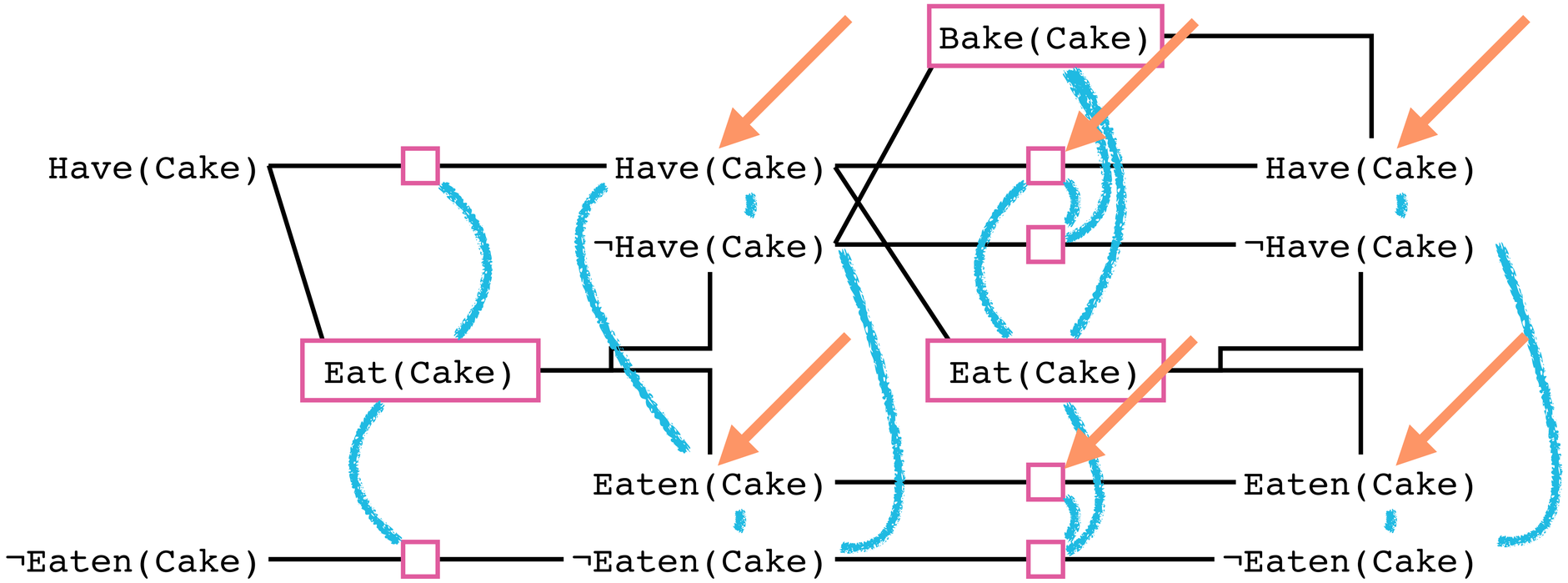
$Goal($ Have(Cake) $\wedge$ Eaten(Cake))

S_0 A_0 S_1 A_1 S_2



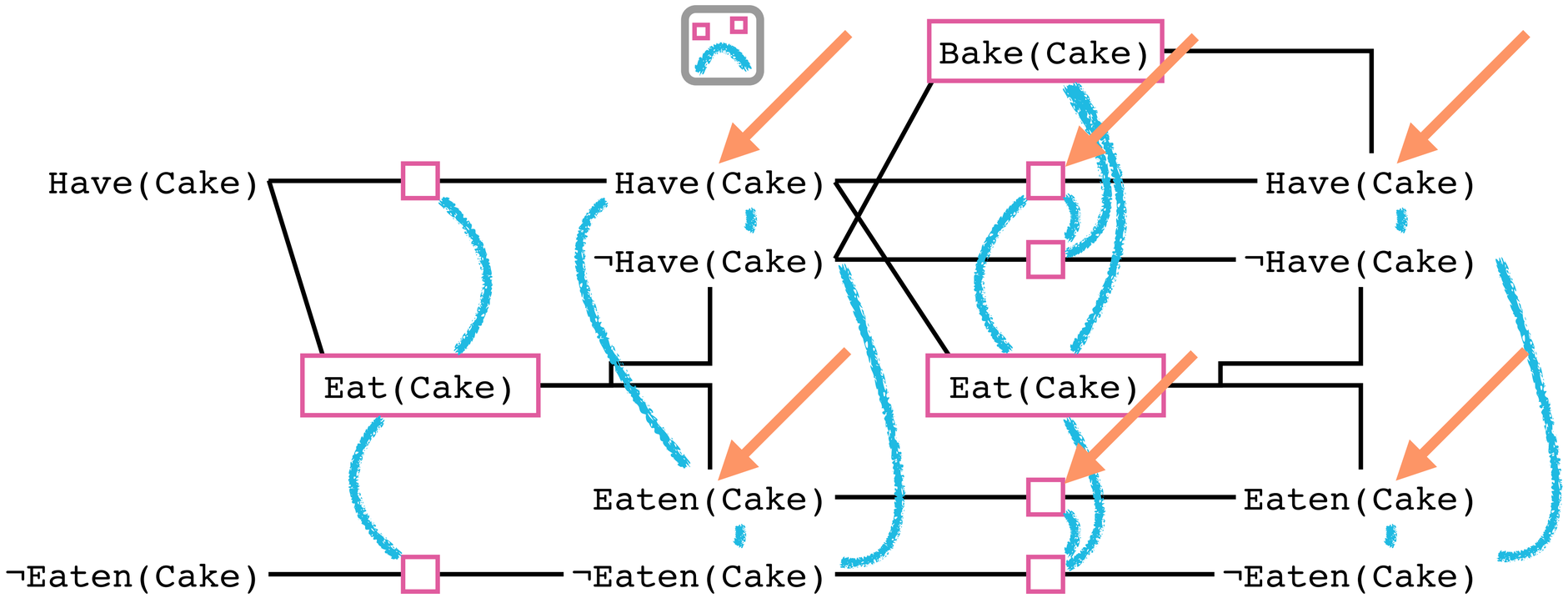
$Goal($ Have(Cake) $\wedge$ Eaten(Cake))

S_0 A_0 S_1 A_1 S_2



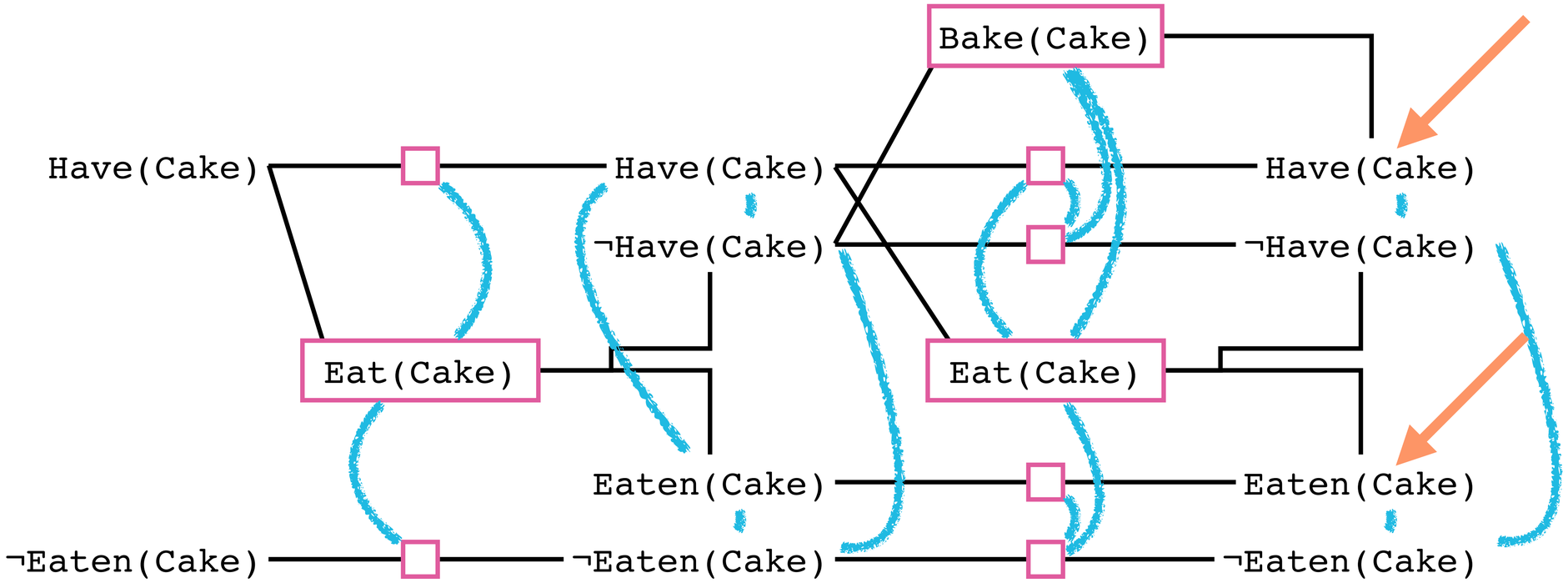
$Goal(\ \text{Have(Cake)} \wedge \text{Eaten(Cake)})$

S_0 A_0 S_1 A_1 S_2

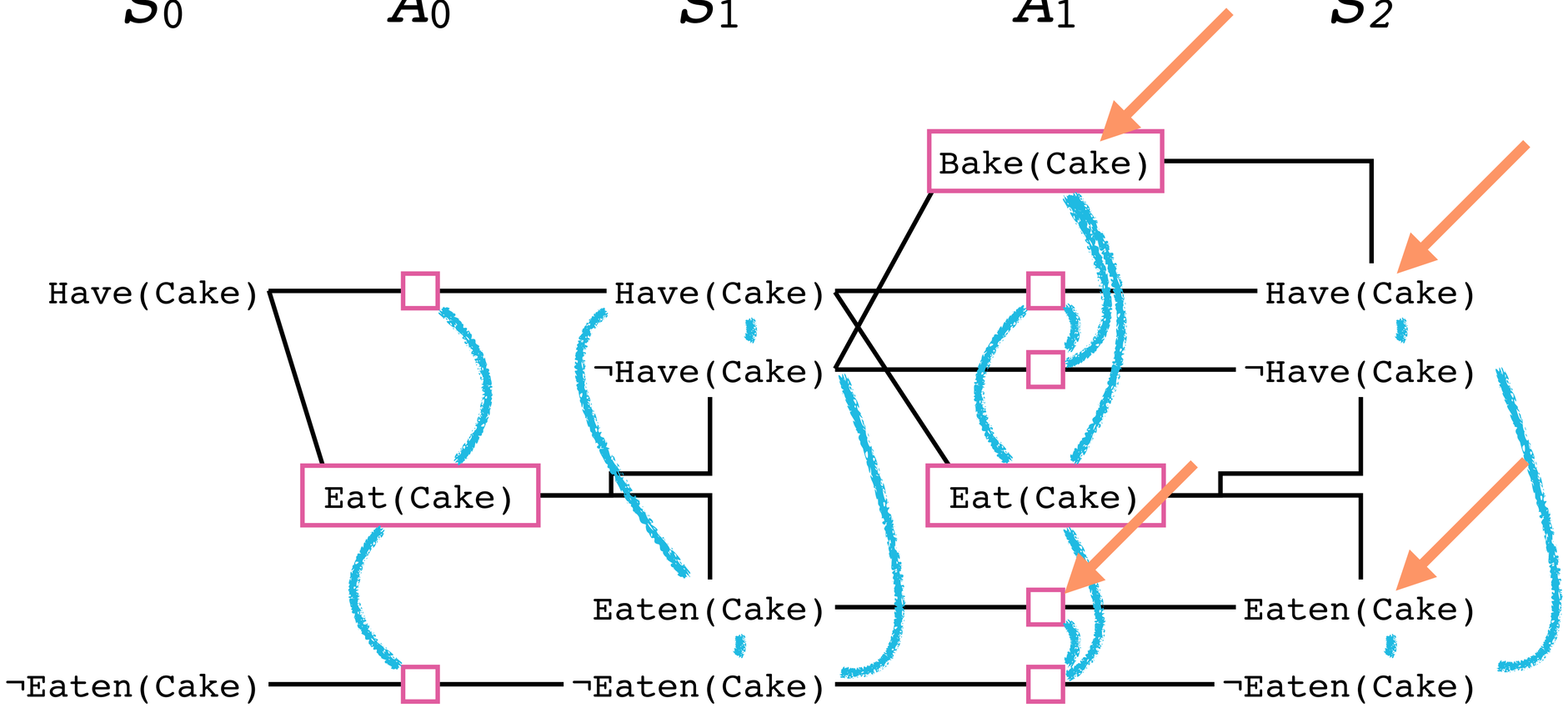


$Goal($ Have(Cake) $\wedge$ Eaten(Cake))

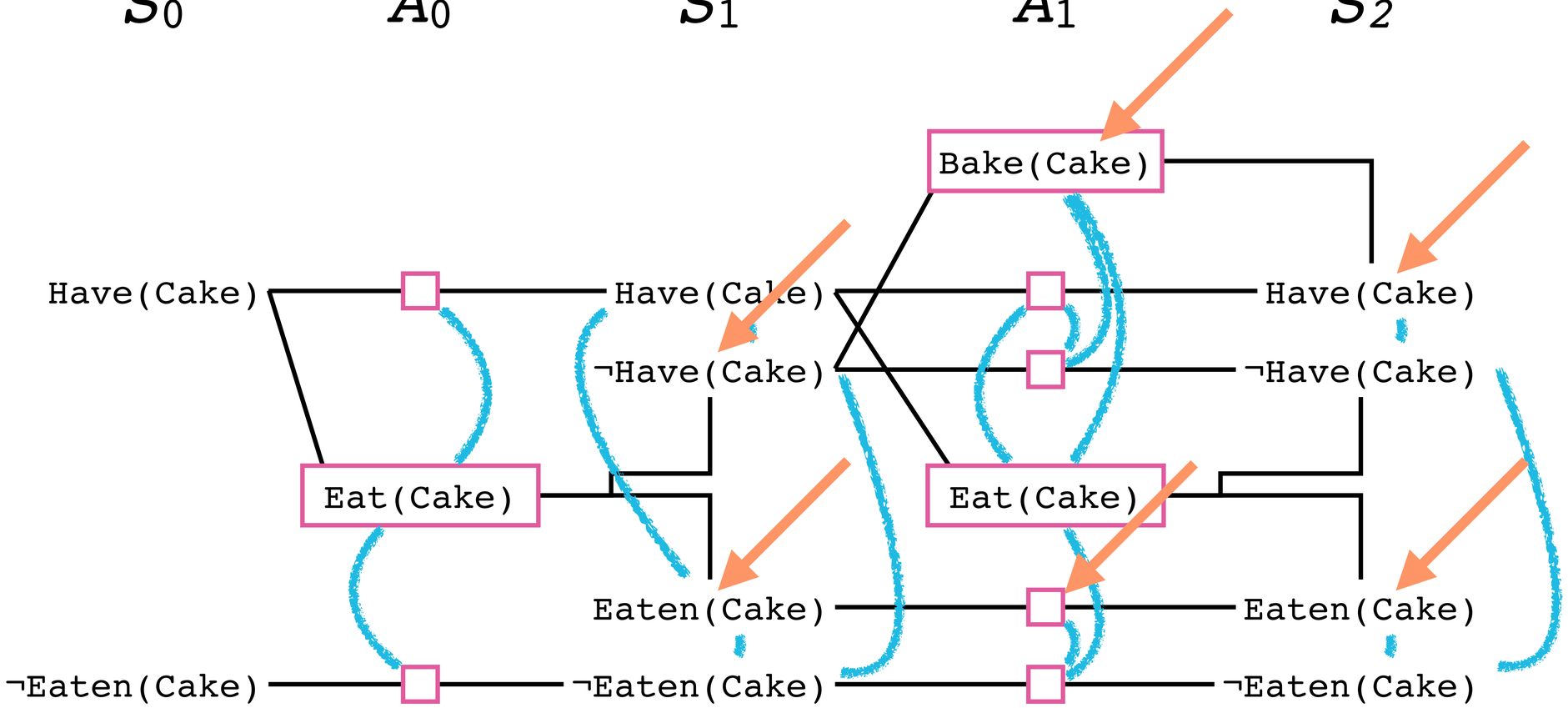
S_0 A_0 S_1 A_1 S_2



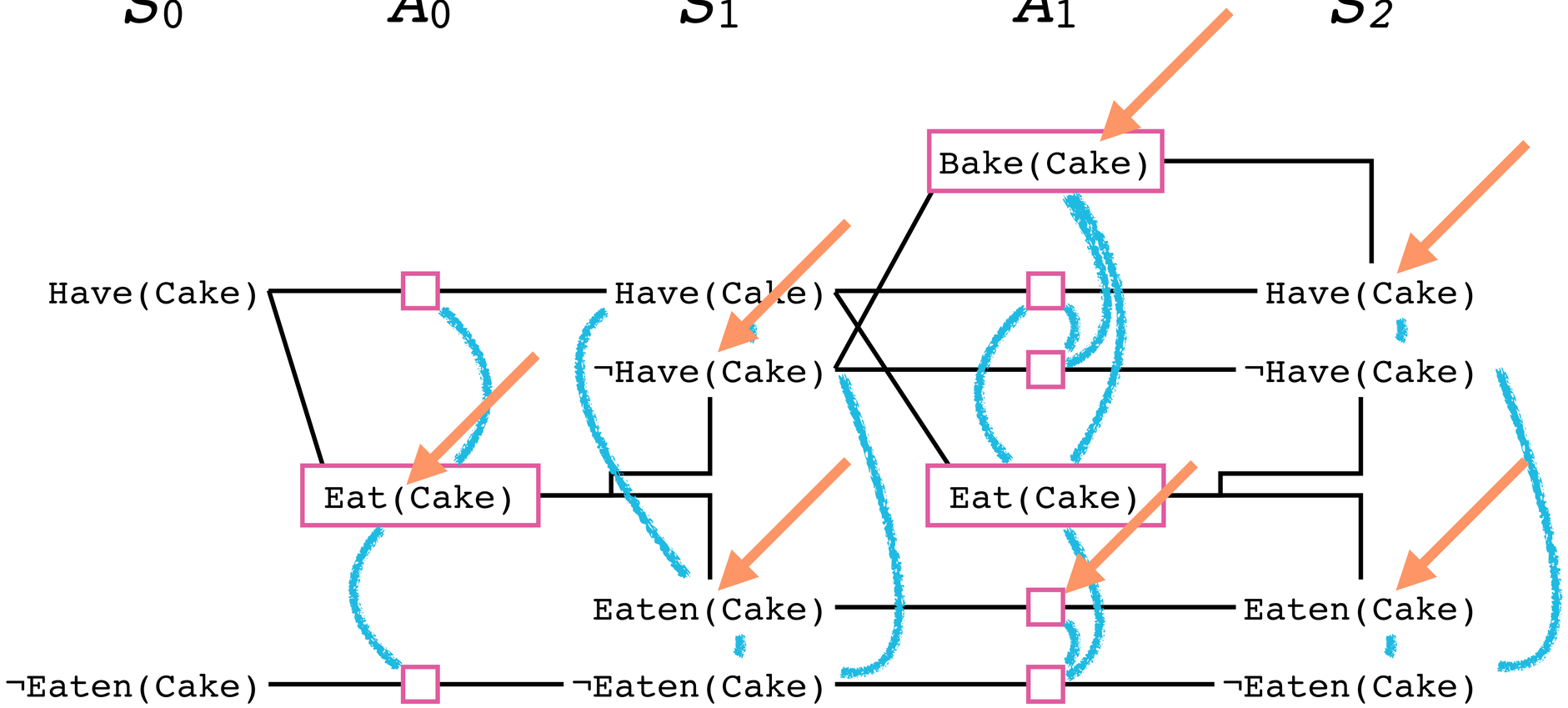
$Goal(\ \text{Have(Cake)} \wedge \text{Eaten(Cake)})$



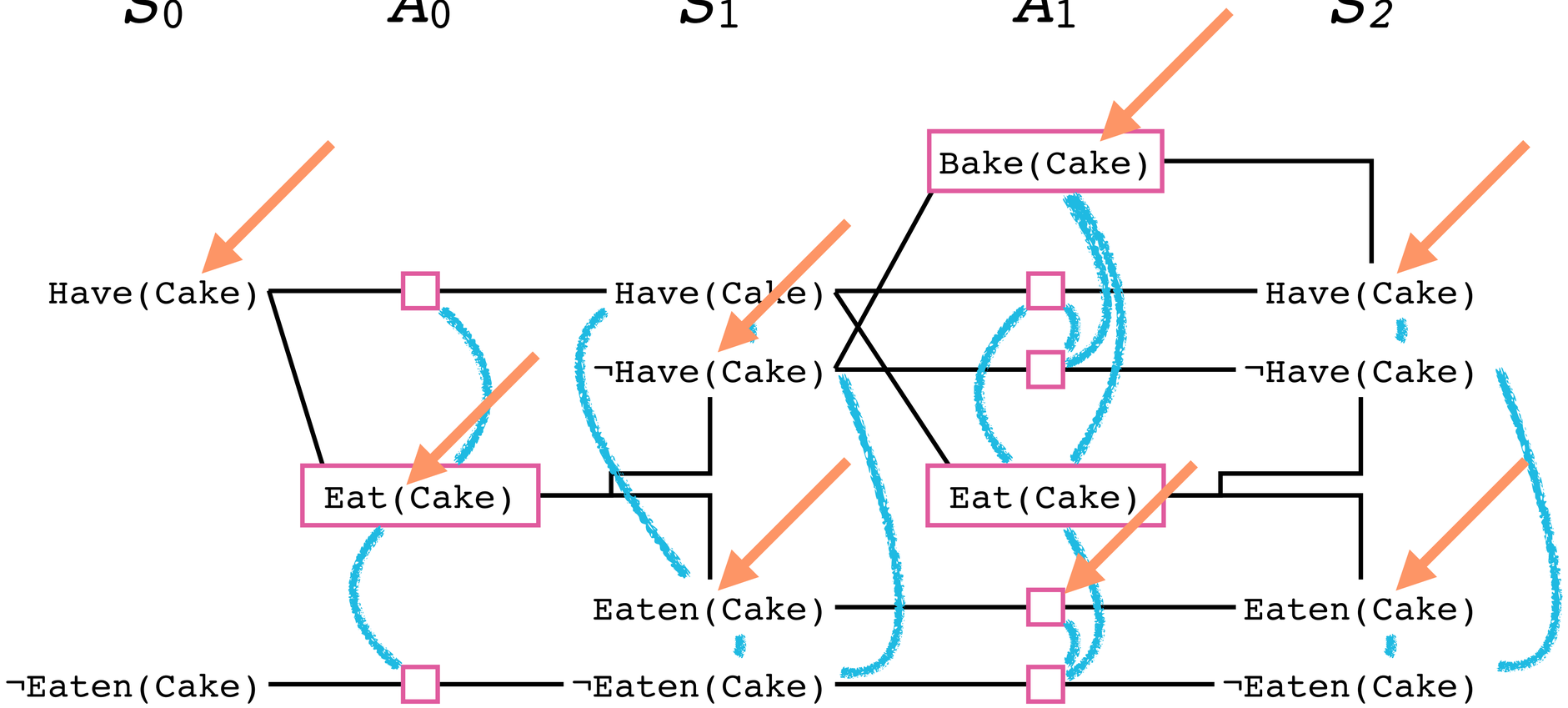
$Goal($ Have(Cake) $\wedge$ Eaten(Cake))



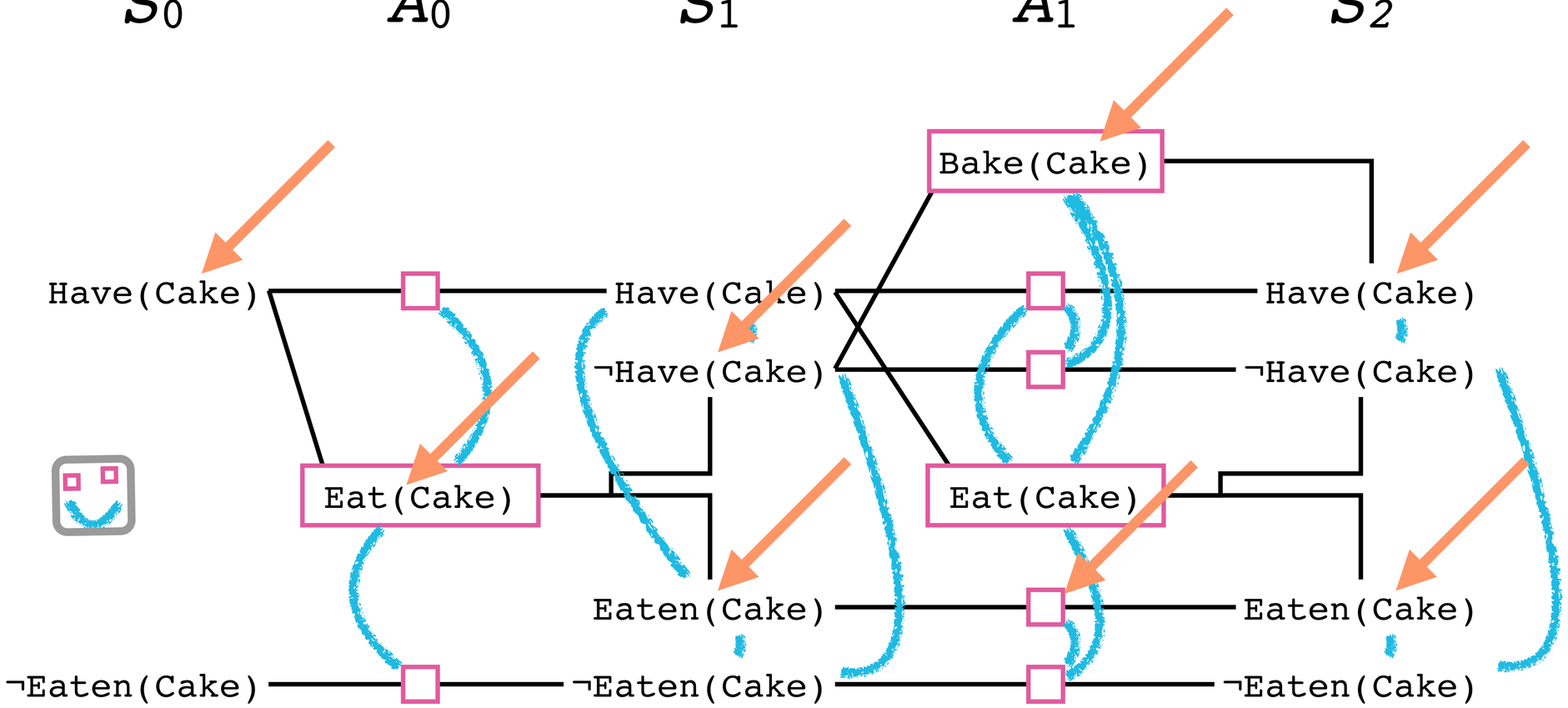
$Goal($ Have(Cake) $\wedge$ Eaten(Cake))



Goal(Have(Cake) ∧ Eaten(Cake))

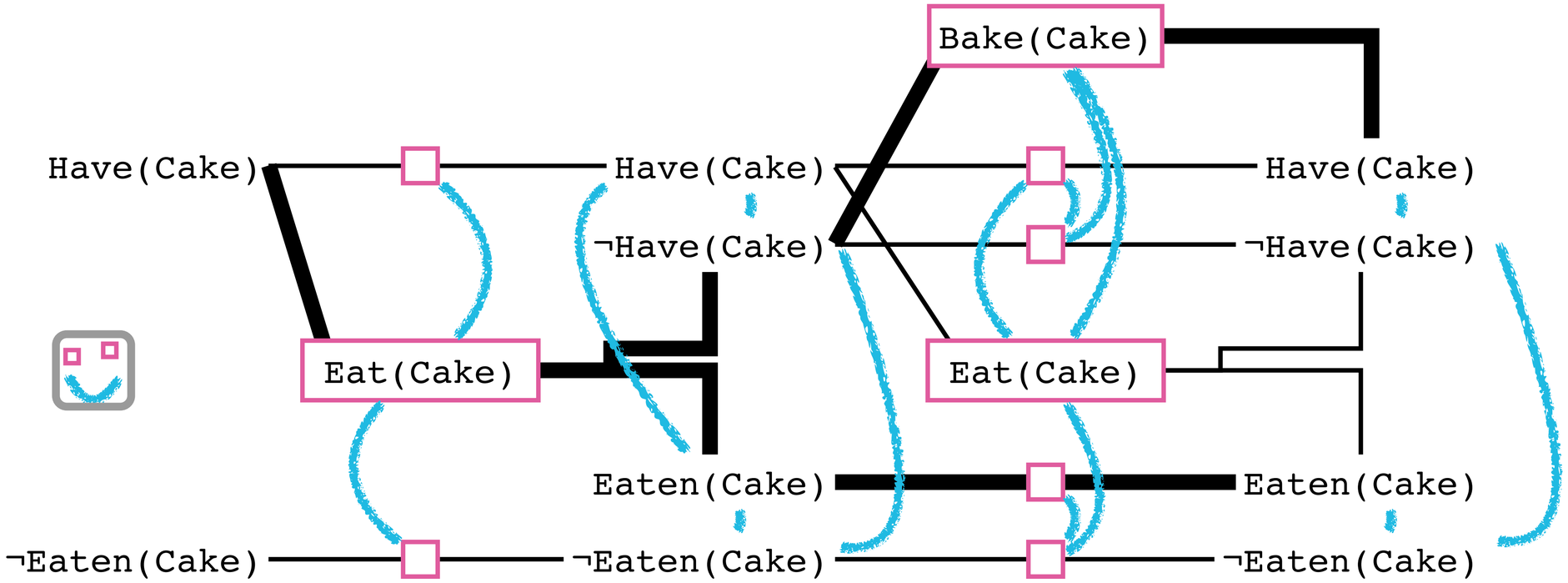


$Goal($ Have(Cake) $\wedge$ Eaten(Cake))



$Goal($ Have(Cake) $\wedge$ Eaten(Cake))

S_0 A_0 S_1 A_1 S_2



The FastForward planner

Hoffmann, J., & Nebel, B. (2001). ***The FF planning system: Fast plan generation through heuristic search***. *Journal of Artificial Intelligence Research*, 14, 253–302.

Ground-breaking forward search state-space planner

Speed comes from:

- plan graph heuristic
- enforced hill-climbing
- helpful actions
- efficient implementation

For problem $\boldsymbol{P}$ to solve goal $\boldsymbol{G}$

For problem P to solve goal G

Apply **ignore delete lists** approach to obtain

relaxed problem P^+

For problem P to solve goal G

Apply **ignore delete lists** approach to obtain
relaxed problem P^+

Heuristic for state s is the length of the
plan to achieve G from s in P^+

For problem P to solve goal G

Apply **ignore delete lists** approach to obtain
relaxed problem P^+

Heuristic for state s is the length of the
plan to achieve G from s in P^+

This is produced using a planning graph

Search is performed using
enforced hill climbing (EHC)

Search is performed using
enforced hill climbing (EHC)

This is a *quick*, greedy approach.

Search is performed using
enforced hill climbing (EHC)

This is a *quick*, greedy approach.
It is sound but **not** complete

Search is performed using
enforced hill climbing (EHC)

This is a *quick*, greedy approach.
It is sound but **not** complete

If EHC fails, try **best-first search**.

Search is performed using
enforced hill climbing (EHC)

This is a *quick*, greedy approach.
It is sound but **not** complete

If EHC fails, try **best-first search**.
This is complete but **not** optimal

Search is performed using
enforced hill climbing (EHC)

Successors are chosen only from the set of
helpful actions

These are actions which appear in
A_0 of the *plan* for P^+

Overview

- Classical Planning: STRIPS / PDDL
- Forward vs Backward Search
- Planning Heuristics
- The Planning Graph
- **Partial Order Planning**

The **PDDL** *representation* does not specify an
algorithm for producing plans

There are many different algorithms we can use
to create plans. Many are based on
domain independent search.

Planning algorithms are often described as either
state-space search
or
plan-space search

The **space of possible actions** is typically smaller than the **space of possible states**, so exploit this.

plan-space search

The **space of possible actions** is typically smaller than the **space of possible states**, so exploit this.

Partial-order planning

plan-space search

Principle of **least commitment**:

- ***partial*** ordering (instead of total)
- not fully instantiated plan (in the first-order case)

Principle of **least commitment**:

- ***partial*** ordering (instead of total)
- not fully instantiated plan (in the first-order case)

Change of problem representation

Principle of **least commitment**:

- ***partial*** ordering (instead of total)
- not fully instantiated plan (in the first-order case)

Change of problem representation

In **state-space planning**
a **node** is a
state of the world

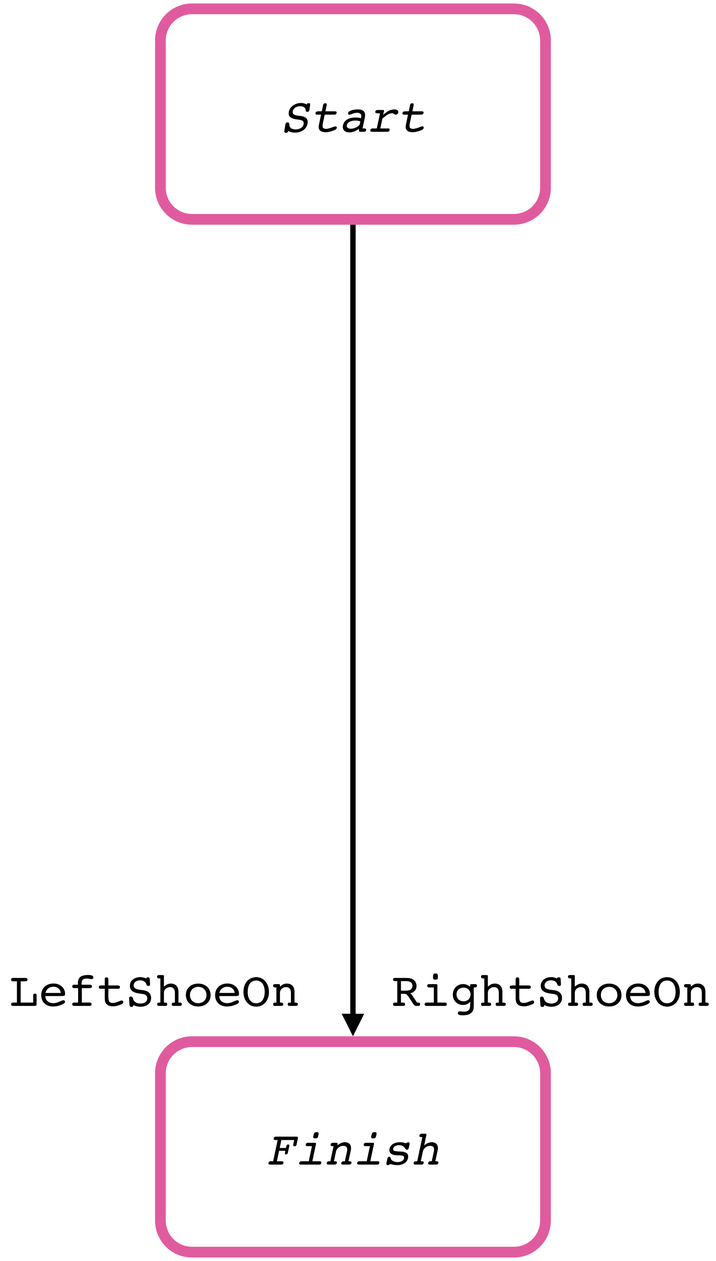
Principle of **least commitment**:

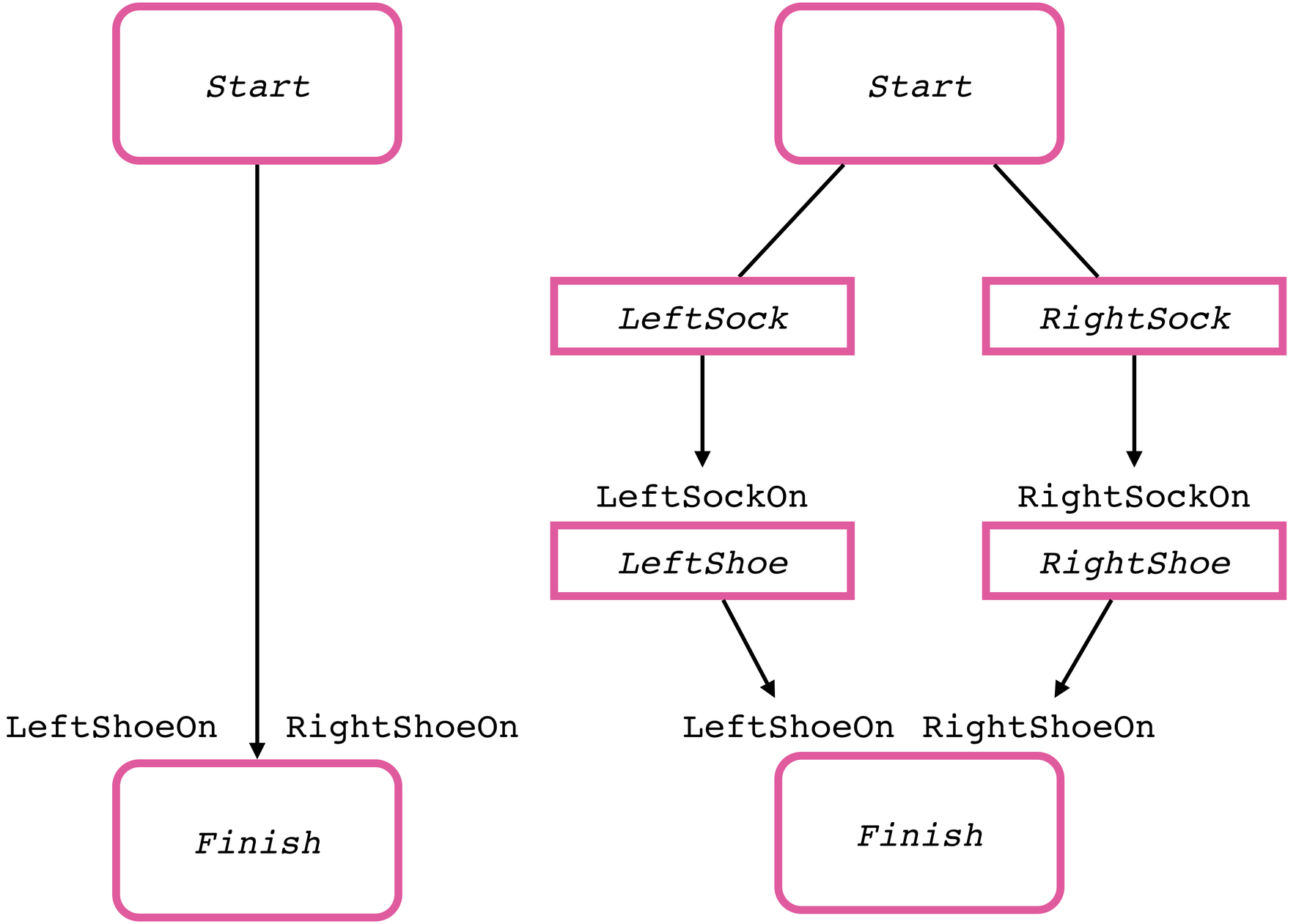
- ***partial*** ordering (instead of total)
- not fully instantiated plan (in the first-order case)

Change of problem representation

In **state-space planning**
a **node** is a
state of the world

In **plan-space planning**
a **node** is a
partial plan


Start
LeftShoeOn
RightShoeOn
Finish


Start
LeftShoeOn
RightShoeOn
Finish
Start
LeftSock
RightSock
LeftSockOn
RightSockOn
LeftShoe
RightShoe
LeftShoeOn
RightShoeOn
Finish

Partially ordered collection of **steps** with preconditions and effects

Partially ordered collection of **steps** with preconditions and effects

Start step has the **initial state** as its *effect*
Finish step has the **goal state** as its *precondition*

Partially ordered collection of **steps** with preconditions and effects

Start step has the **initial state** as its *effect*
Finish step has the **goal state** as its *precondition*

Causal links connect the *effect* of one step to *precondition* `p` of another if it **achieves** it: $\mathtt{A} \xrightarrow{\mathtt{p}} \mathtt{B}$

Partially ordered collection of **steps** with preconditions and effects

Start step has the **initial state** as its *effect*
Finish step has the **goal state** as its *precondition*

Causal links connect the *effect* of one step to *precondition* `p` of another if it **achieves** it: $\mathtt{A} \xrightarrow{\mathtt{p}} \mathtt{B}$

Temporal orderings define the **order** of *pairs* of steps: $\mathtt{A} \prec \mathtt{B}$

Partially ordered collection of **steps** with preconditions and effects

Start step has the **initial state** as its *effect*
Finish step has the **goal state** as its *precondition*

Causal links connect the *effect* of one step to *precondition* $\mathtt{p}$ of another if it **achieves** it: $\mathtt{A} \xrightarrow{\mathtt{p}} \mathtt{B}$

Temporal orderings define the **order** of *pairs* of steps: $\mathtt{A} \prec \mathtt{B}$

Open preconditions are not yet **achieved**

Key idea: **clobbering** and **conflicts**

Key idea: **clobbering** and **conflicts**

A step **clobbers** a causal link if it destroys the precondition the link achieves. E.g., `Fly(P1, BHX, FCO)` clobbers `At(P1, BHX)`

Key idea: **clobbering** and **conflicts**

A step **clobbers** a causal link if it destroys the precondition the link achieves. E.g., `Fly(P1, BHX, FCO)` clobbers `At(P1, BHX)`

If step **C** has the effect ¬**p** then it creates a **conflict** with: $\mathbf{A} \xrightarrow{p} \mathbf{B}$

Key idea: **clobbering** and **conflicts**

A step **clobbers** a causal link if it destroys the precondition the link achieves. E.g., `Fly(P1, BHX, FCO)` clobbers `At(P1, BHX)`

If step **C** has the effect ¬**p** then it creates a **conflict** with: $\mathbf{A} \xrightarrow{p} \mathbf{B}$

Solution:

Key idea: **clobbering** and **conflicts**

A step **clobbers** a causal link if it destroys the precondition the link achieves. E.g., $\texttt{Fly(P}_1\texttt{, BHX, FCO)}$ clobbers $\texttt{At(P}_1\texttt{, BHX)}$

If step **C** has the effect ¬**p** then it creates a **conflict** with: $\texttt{A} \xrightarrow{\texttt{p}} \texttt{B}$

Solution:

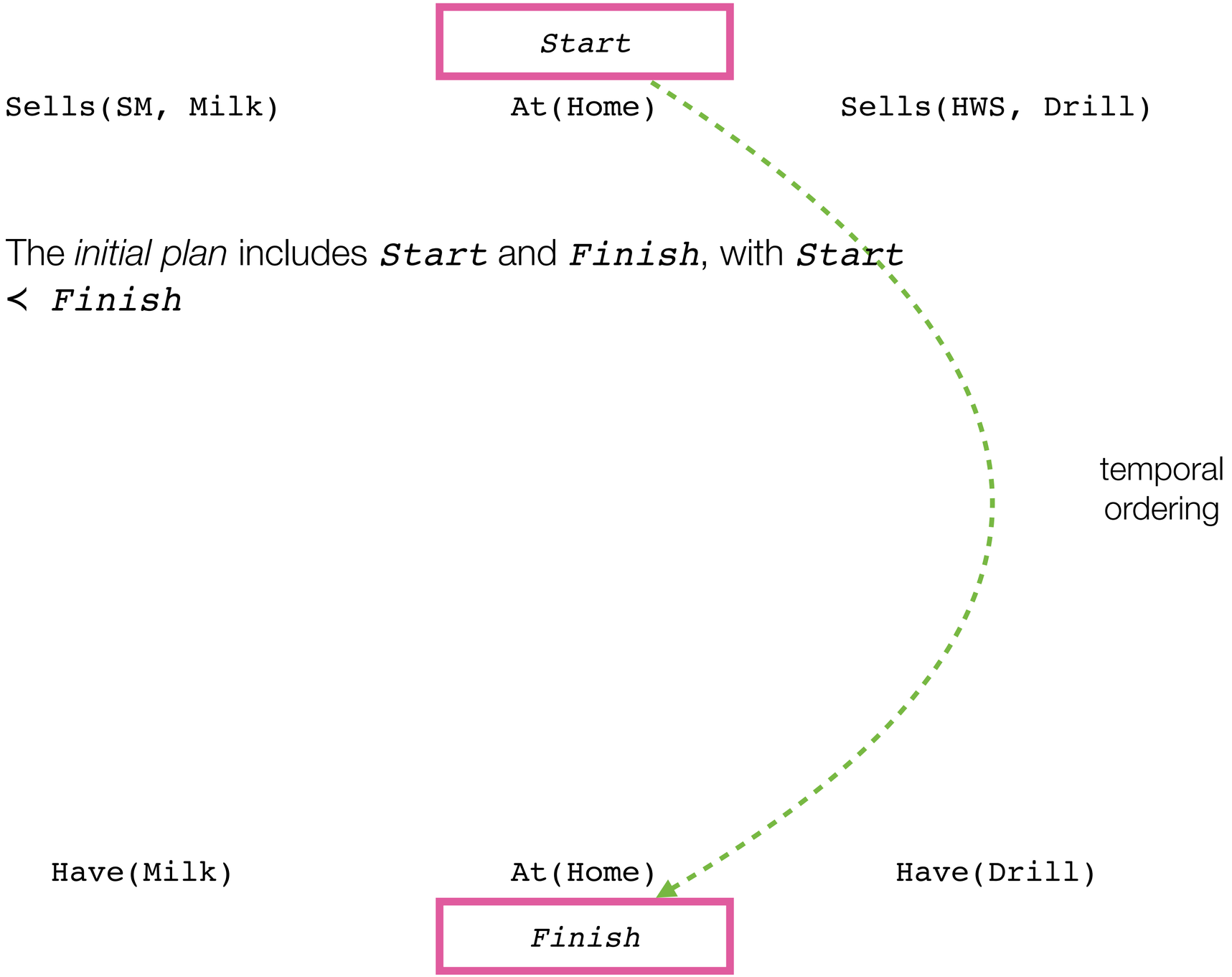
$\texttt{C} \prec \texttt{A}$ or $\texttt{B} \prec \texttt{C}$

demotion promotion

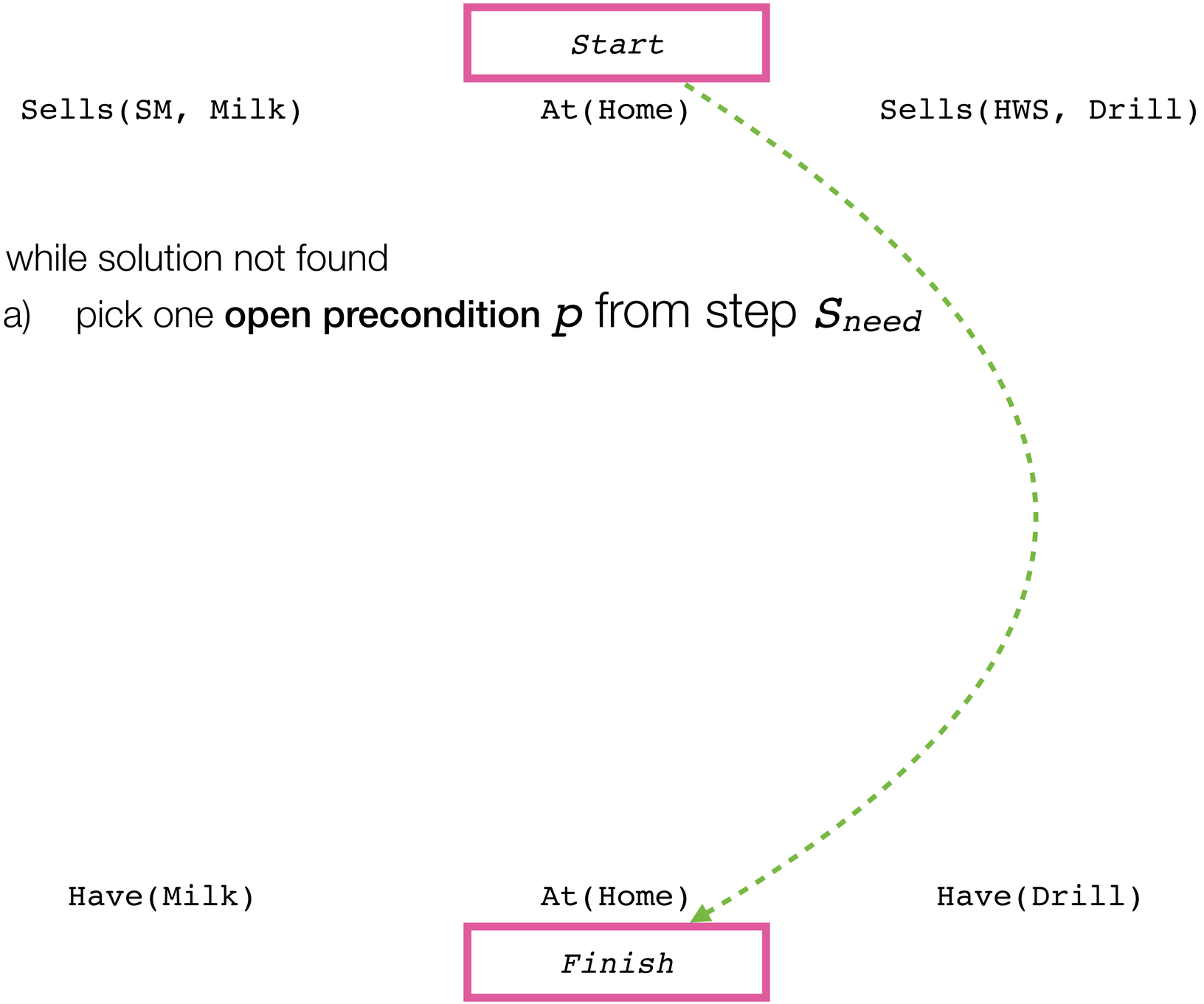
Action: $Buy(x)$
Precondition: $At(p), Sells(p, x)$
Effect: $Have(x)$

Action: $Go(x)$
Precondition: $At(y)$
Effect: $At(x) \wedge \neg At(y)$

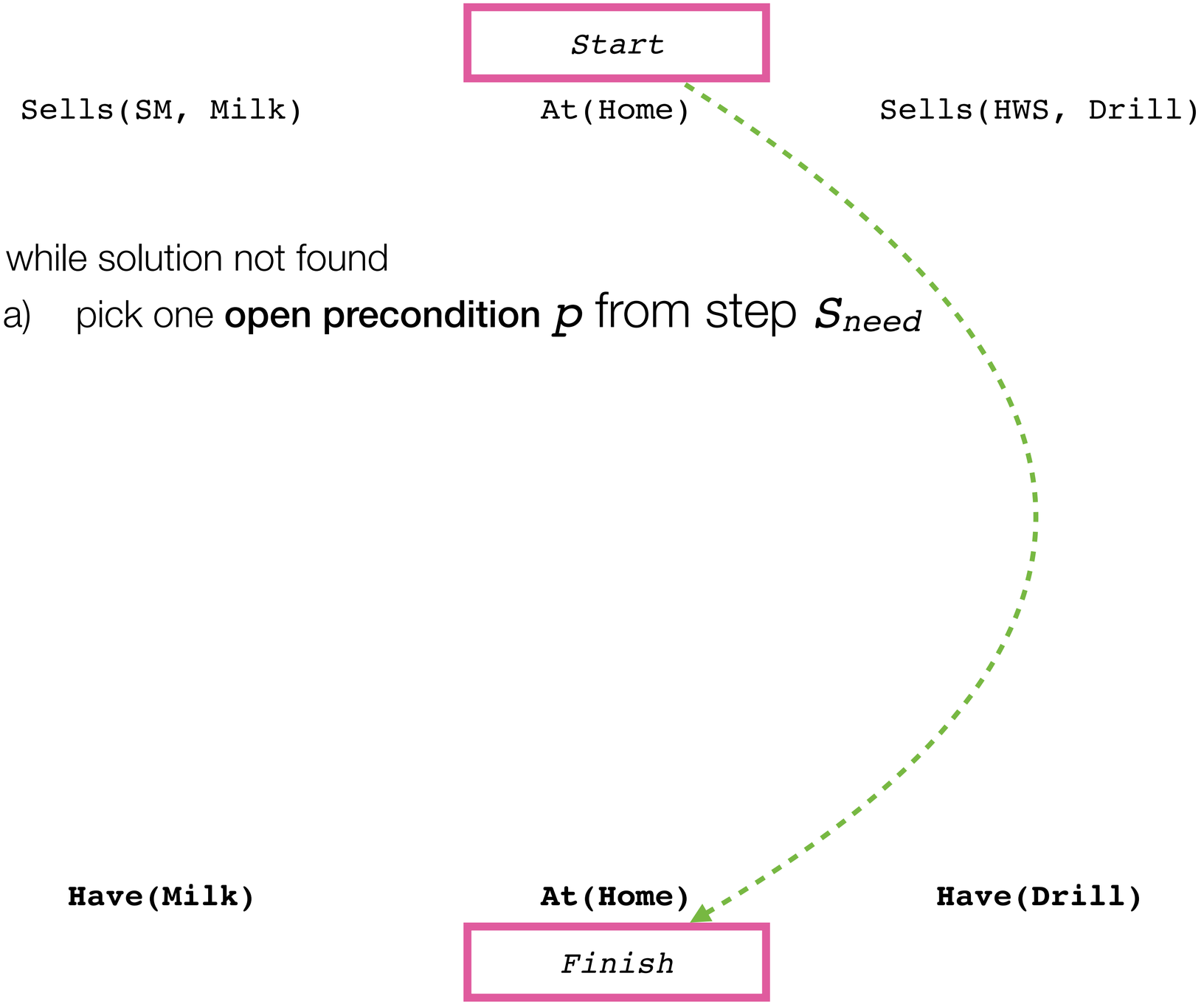
Objects: $Milk, Bananas, Drill, \ldots$
Places: $Home, SM, HWS, \ldots$



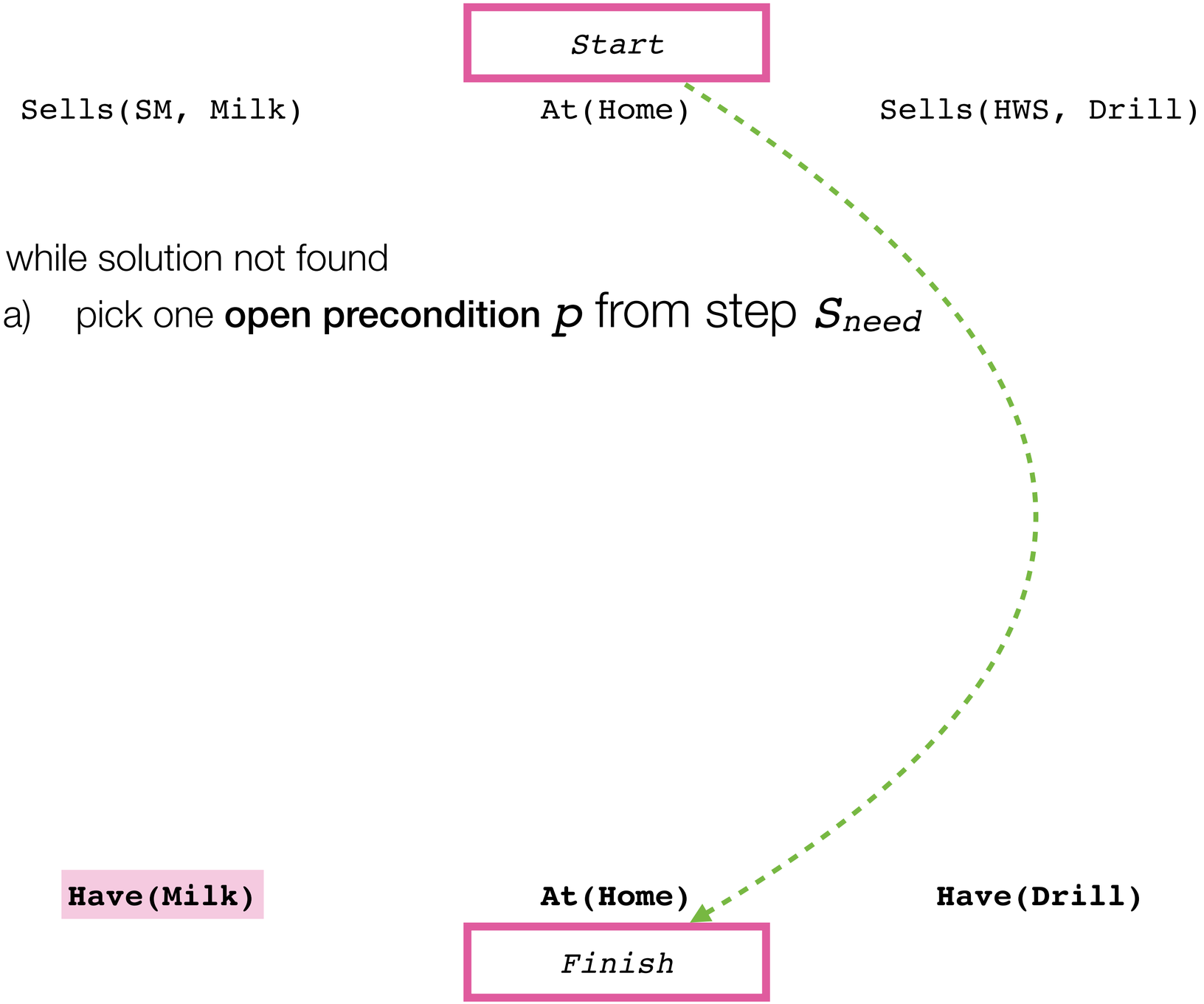
1. The *initial plan* includes ***Start*** and ***Finish***, with ***Start*** < ***Finish***



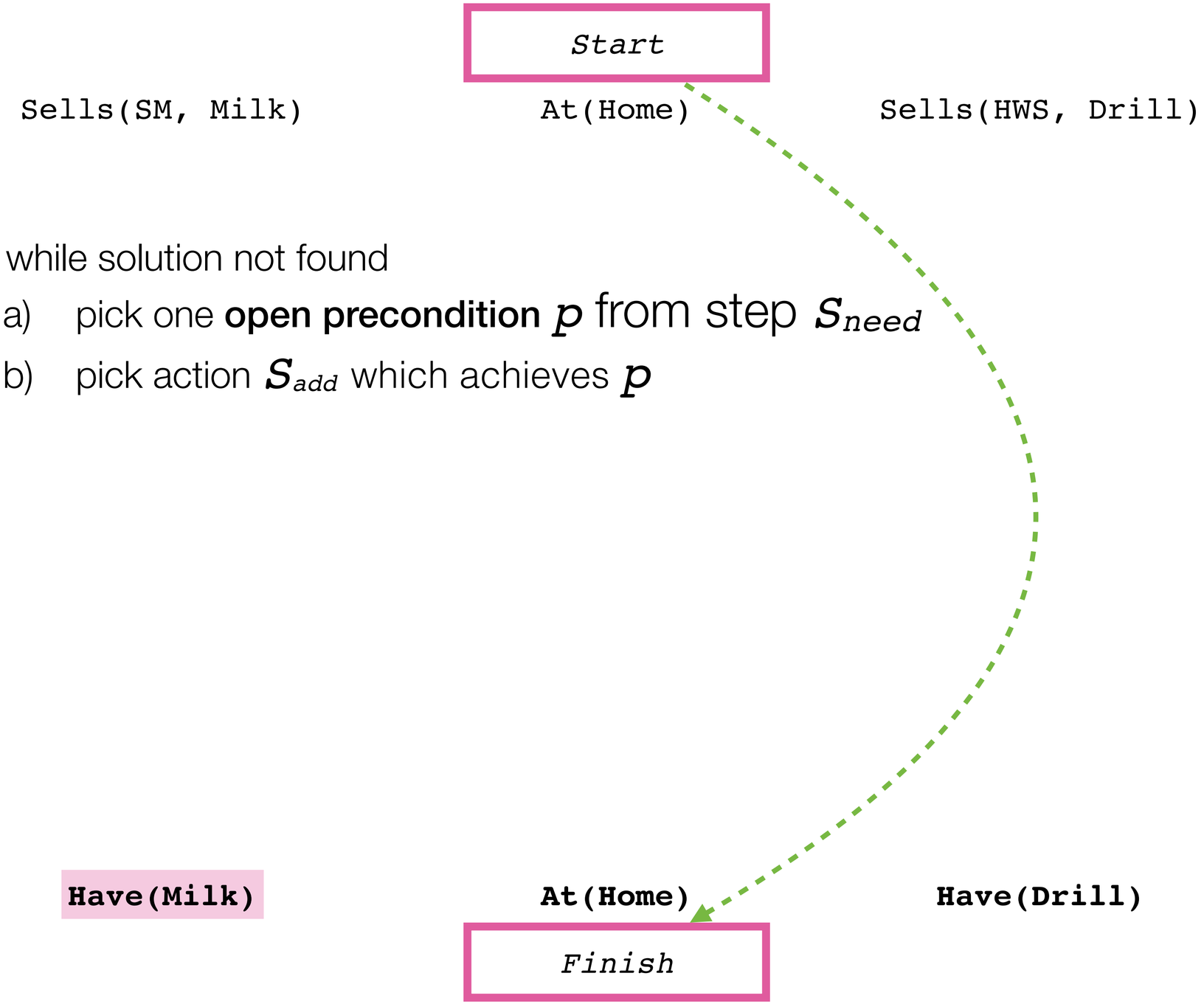
2. while solution not found
   a) pick one **open precondition** p from step S_{need}



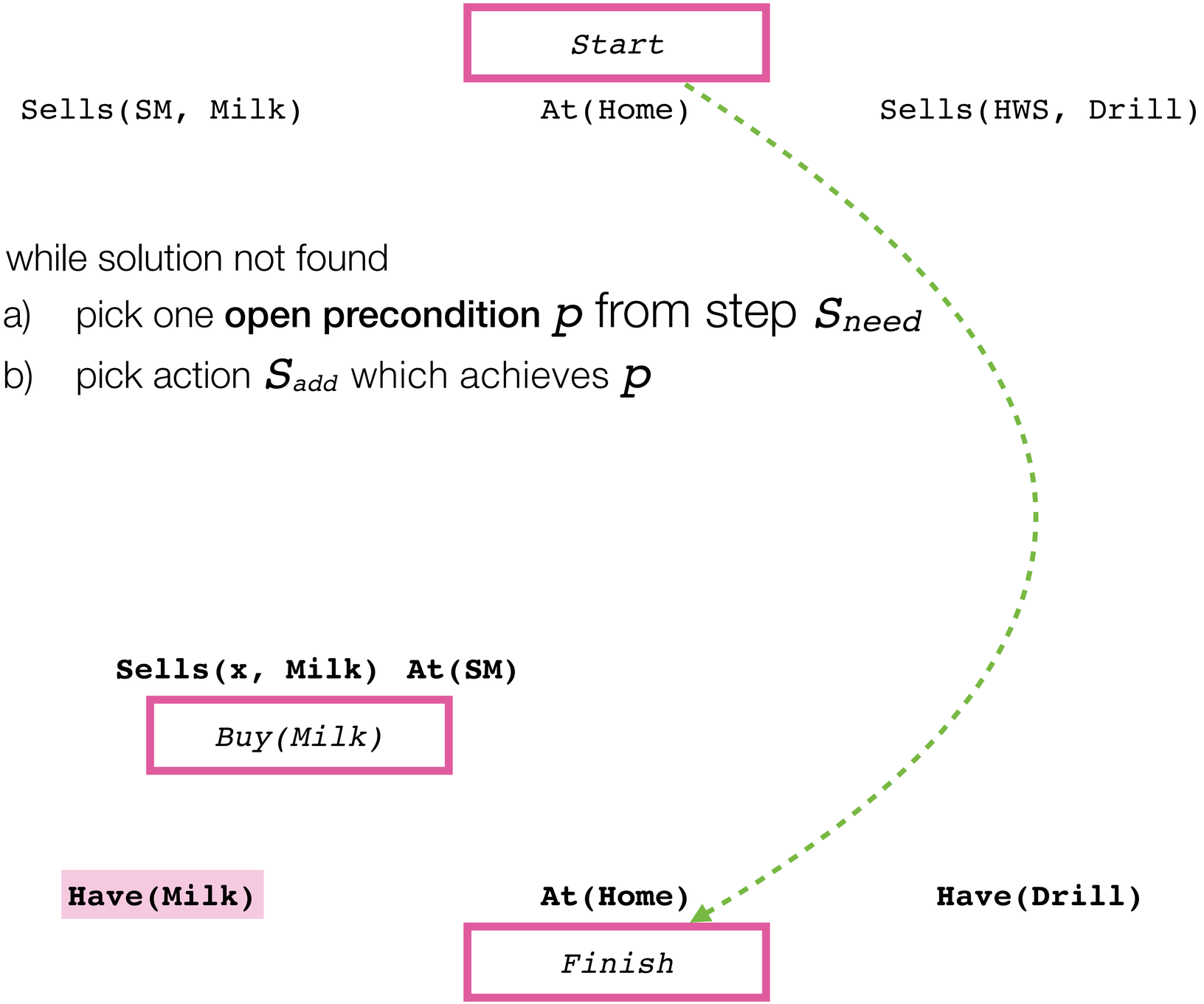
2. while solution not found
   a) pick one **open precondition** p from step S_{need}



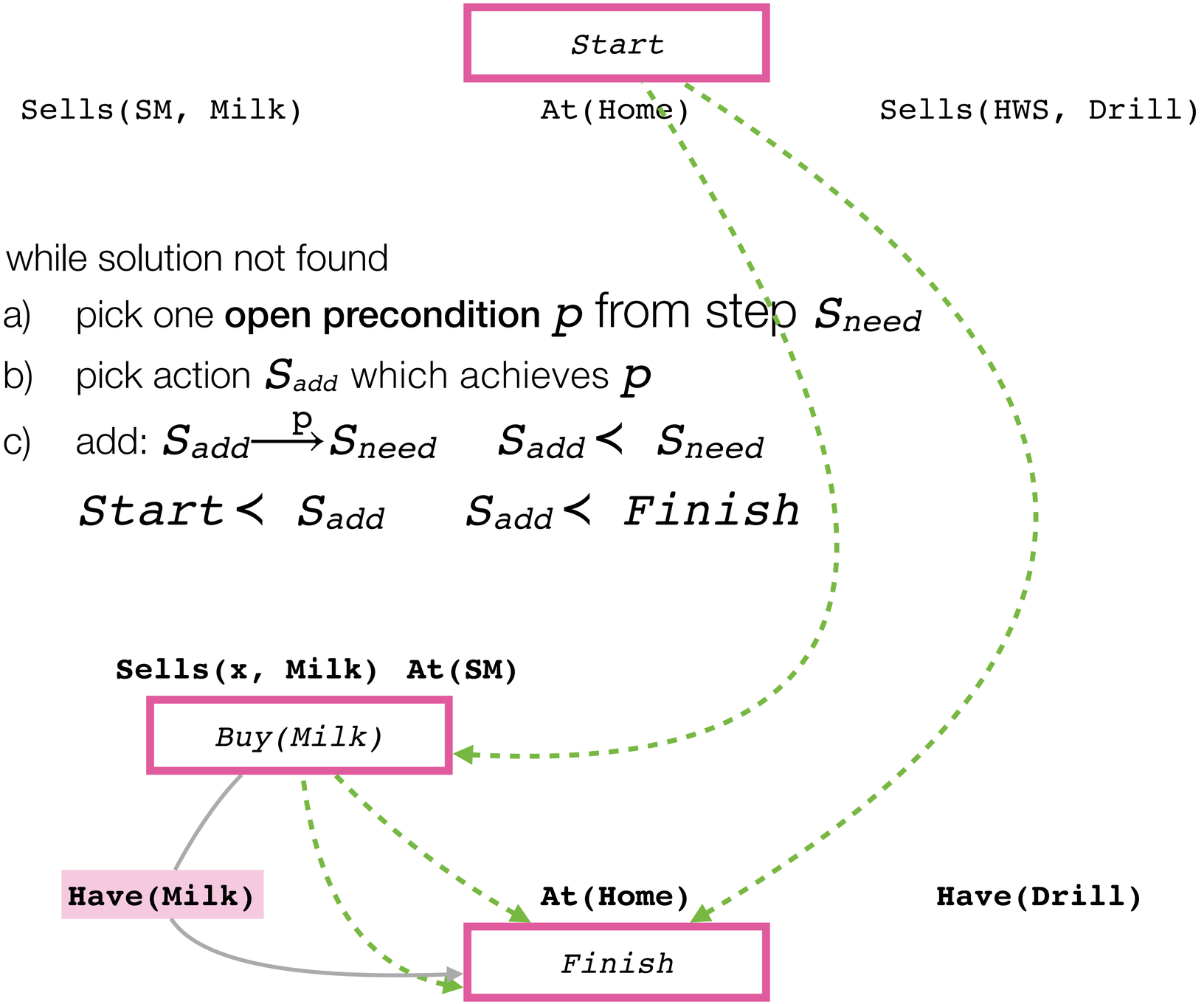
2. while solution not found
   a) pick one **open precondition** p from step S_{need}



2. while solution not found
   a) pick one **open precondition** p from step S_{need}
   b) pick action S_{add} which achieves p

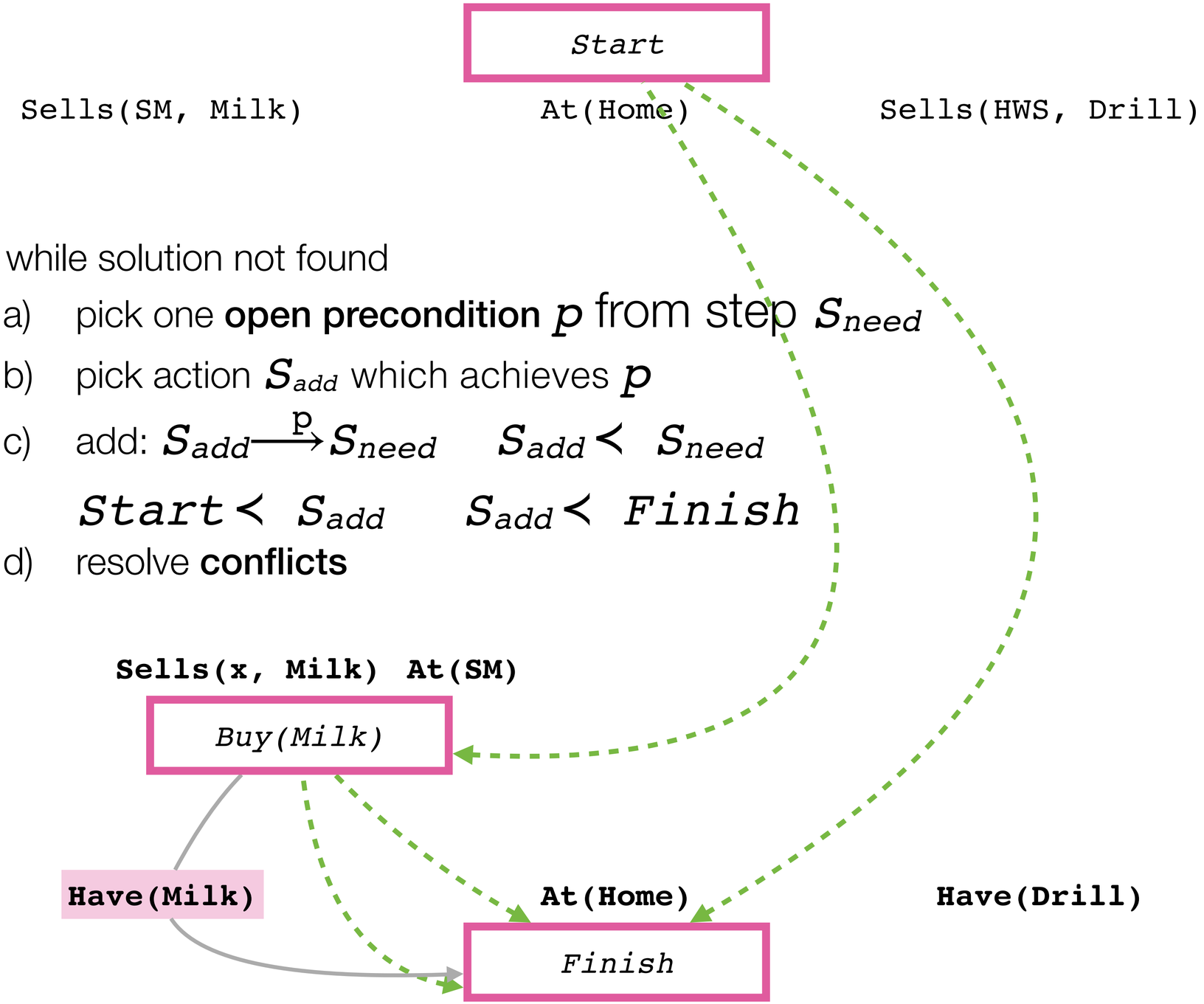


2. while solution not found
   a) pick one **open precondition** p from step S_{need}
   b) pick action S_{add} which achieves p



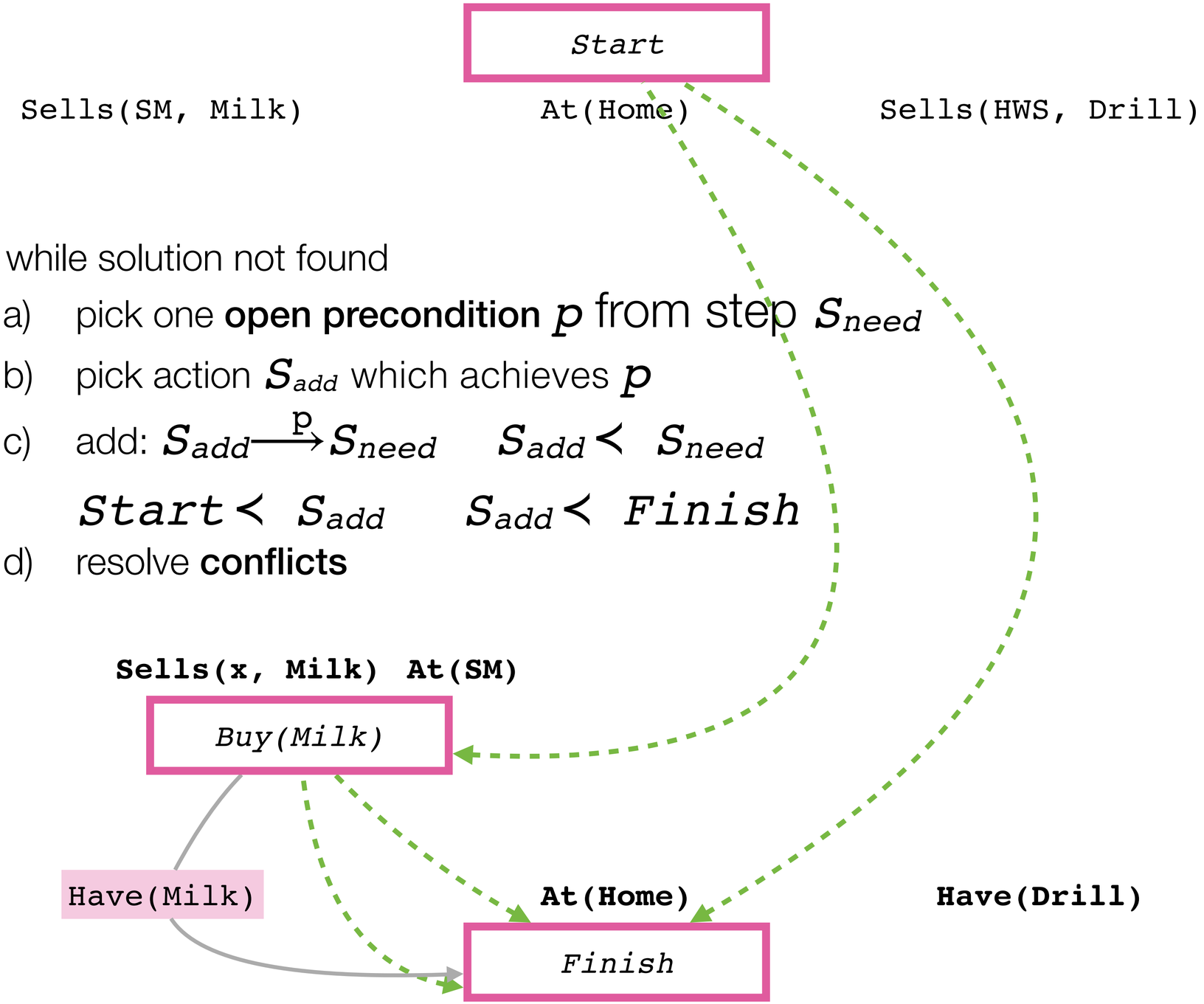
2. while solution not found
   a) pick one **open precondition** p from step S_{need}
   b) pick action S_{add} which achieves p
   c) add: $S_{add} \xrightarrow{p} S_{need}$ $S_{add} \prec S_{need}$

      $Start \prec S_{add}$ $S_{add} \prec Finish$



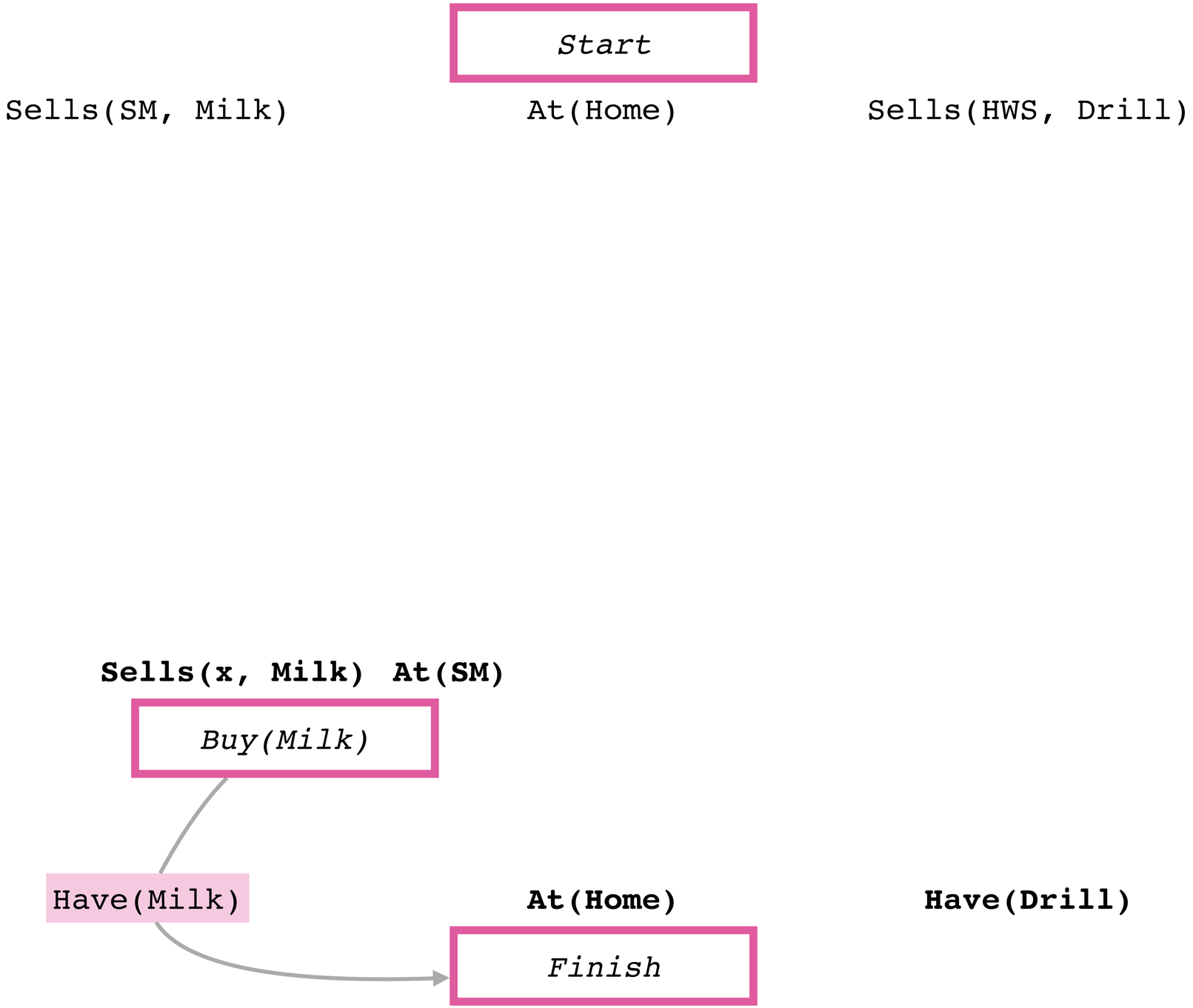
2. while solution not found
   a) pick one **open precondition** p from step S_{need}
   b) pick action S_{add} which achieves p
   c) add: $S_{add} \xrightarrow{p} S_{need}$ $S_{add} \prec S_{need}$

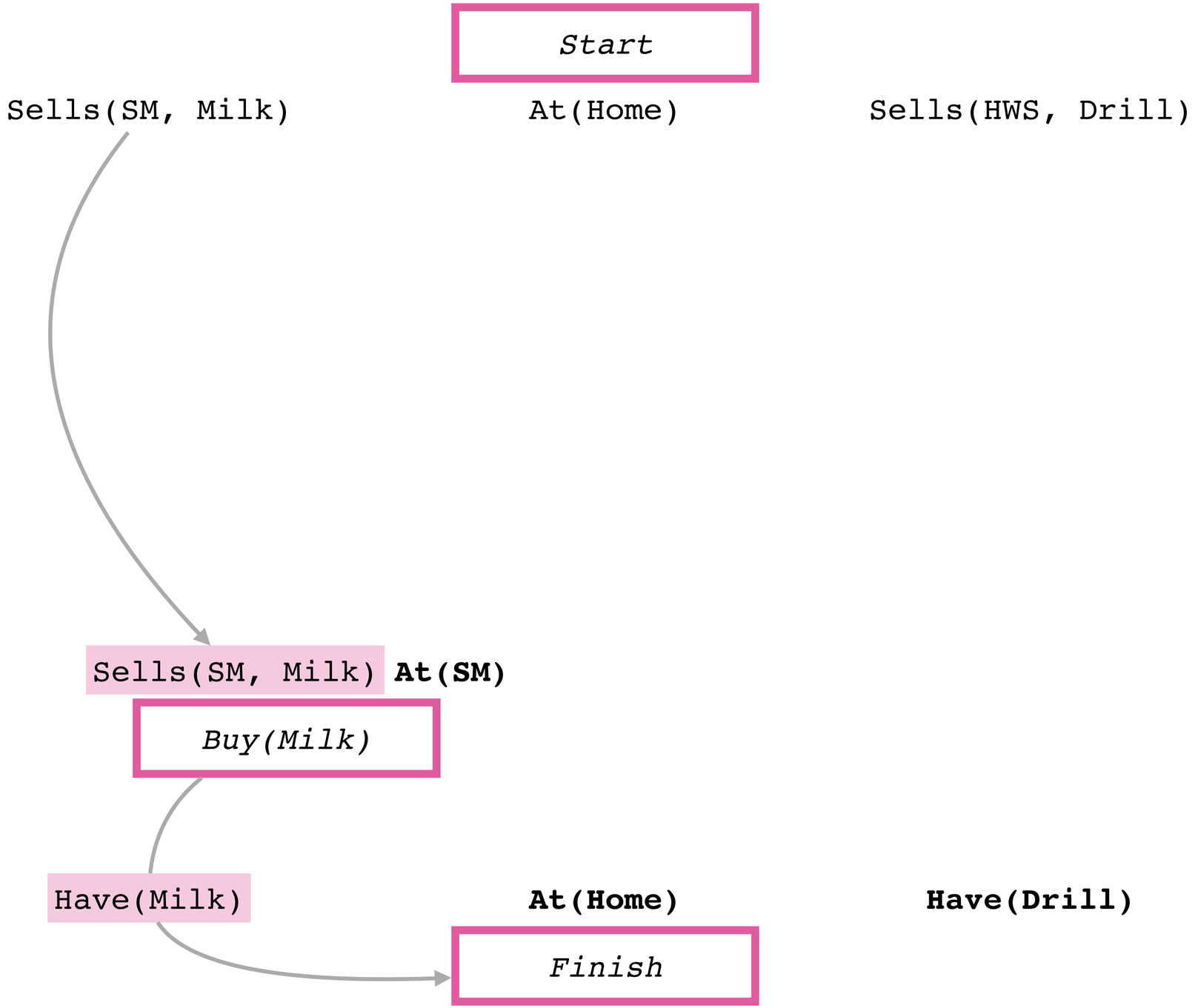
      $Start \prec S_{add}$ $S_{add} \prec Finish$
   d) resolve **conflicts**

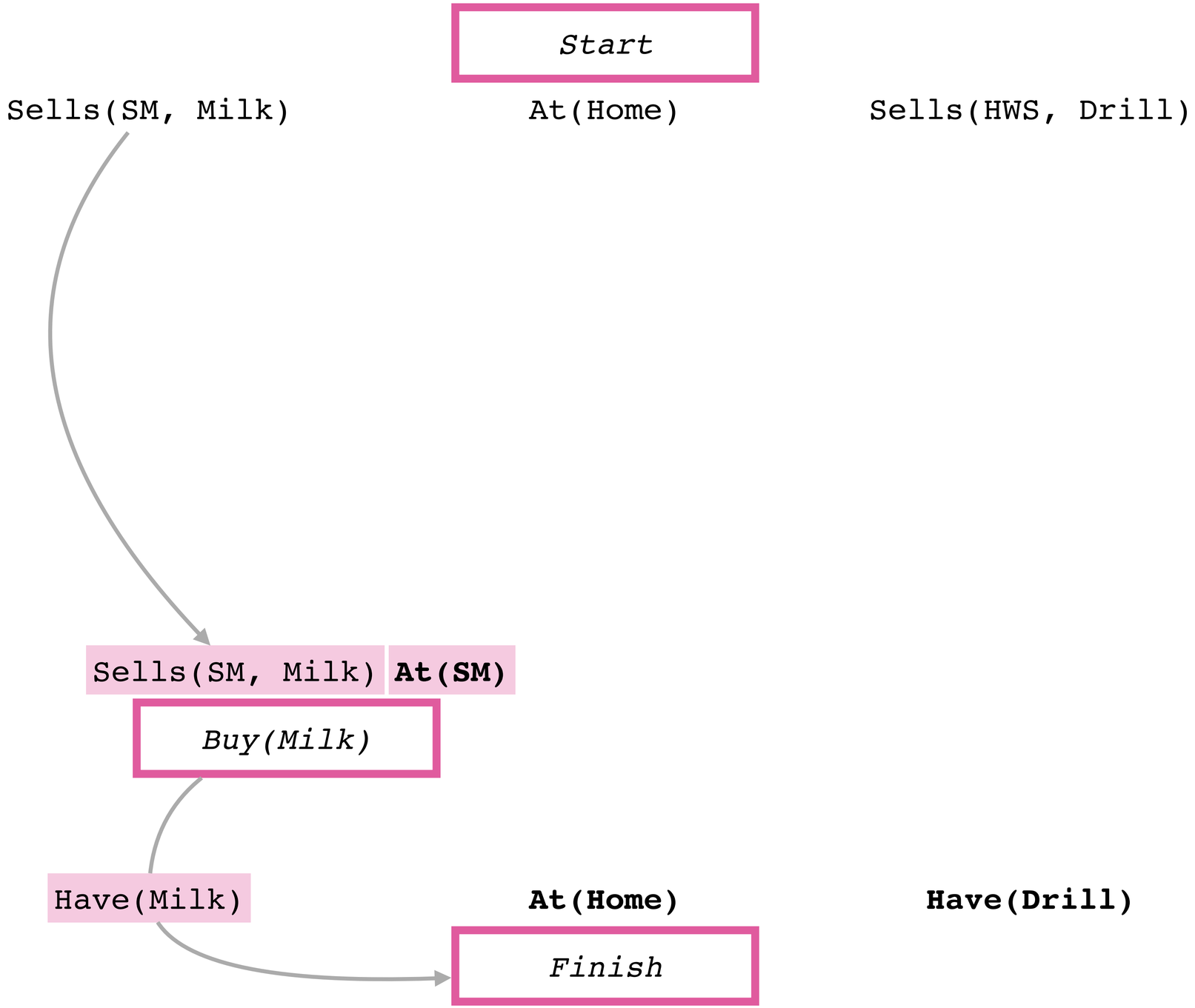


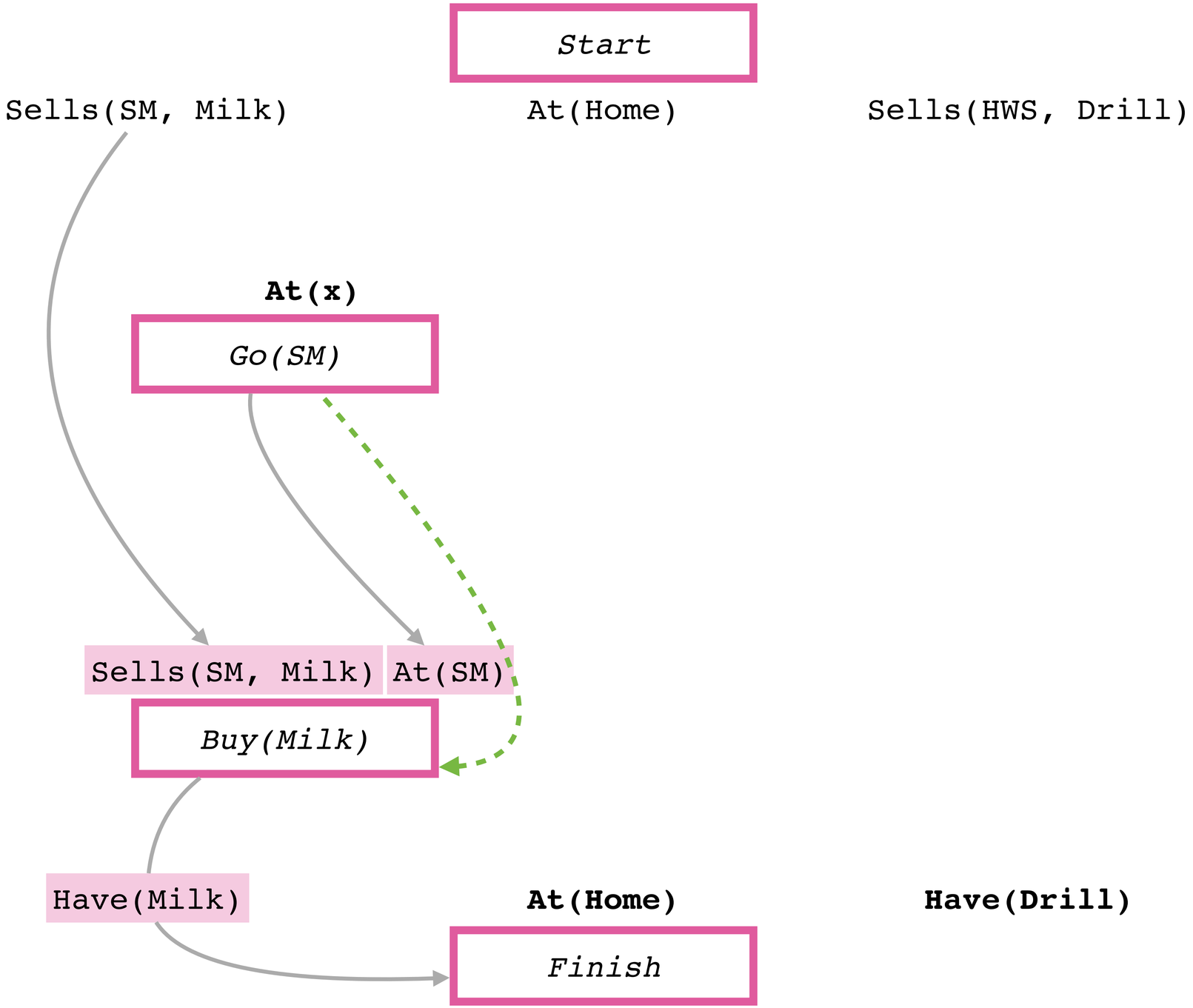
2. while solution not found
   a) pick one **open precondition** p from step S_{need}
   b) pick action S_{add} which achieves p
   c) add: $S_{add} \xrightarrow{p} S_{need}$ $S_{add} \prec S_{need}$

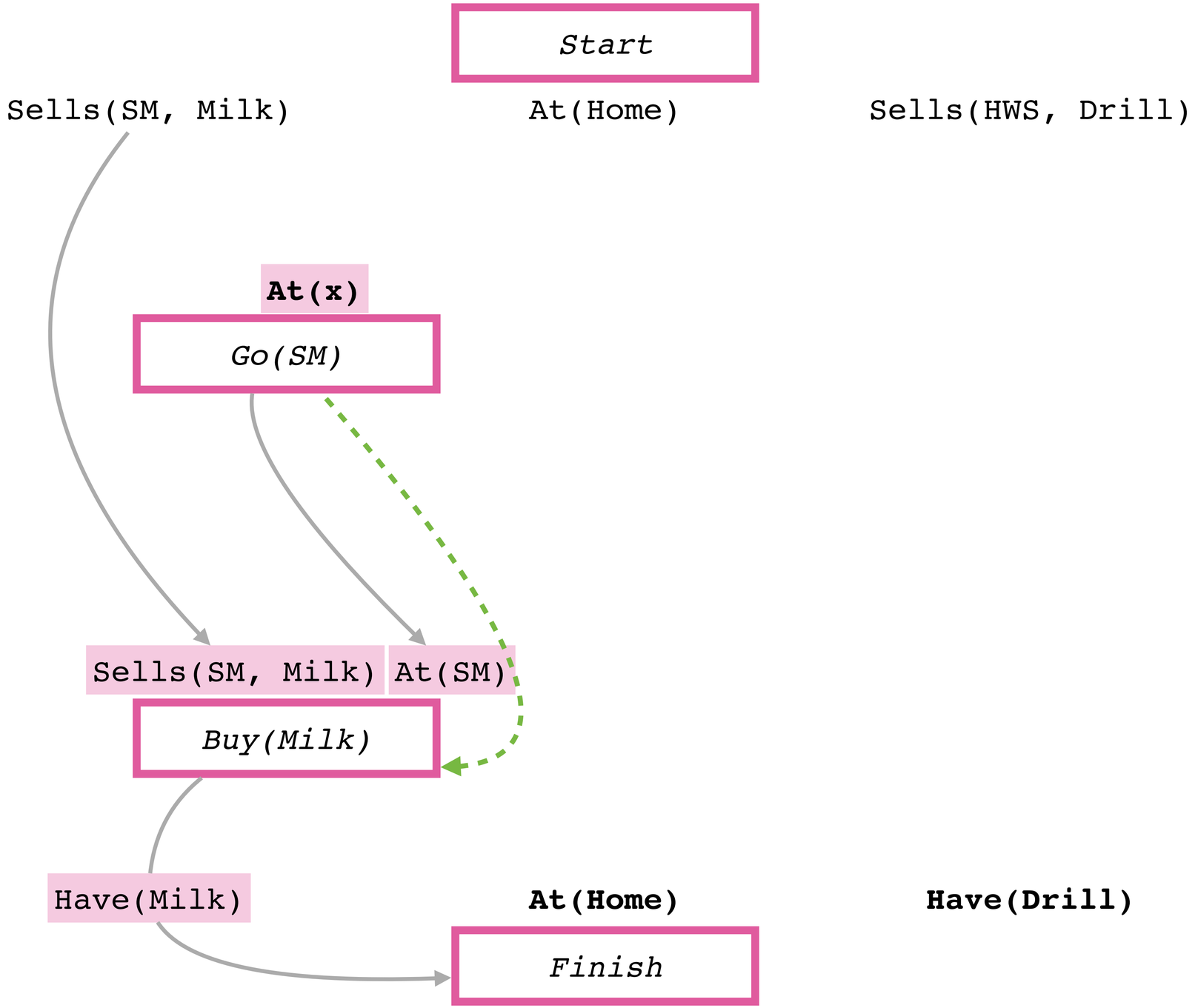
      $Start \prec S_{add}$ $S_{add} \prec Finish$
   d) resolve **conflicts**

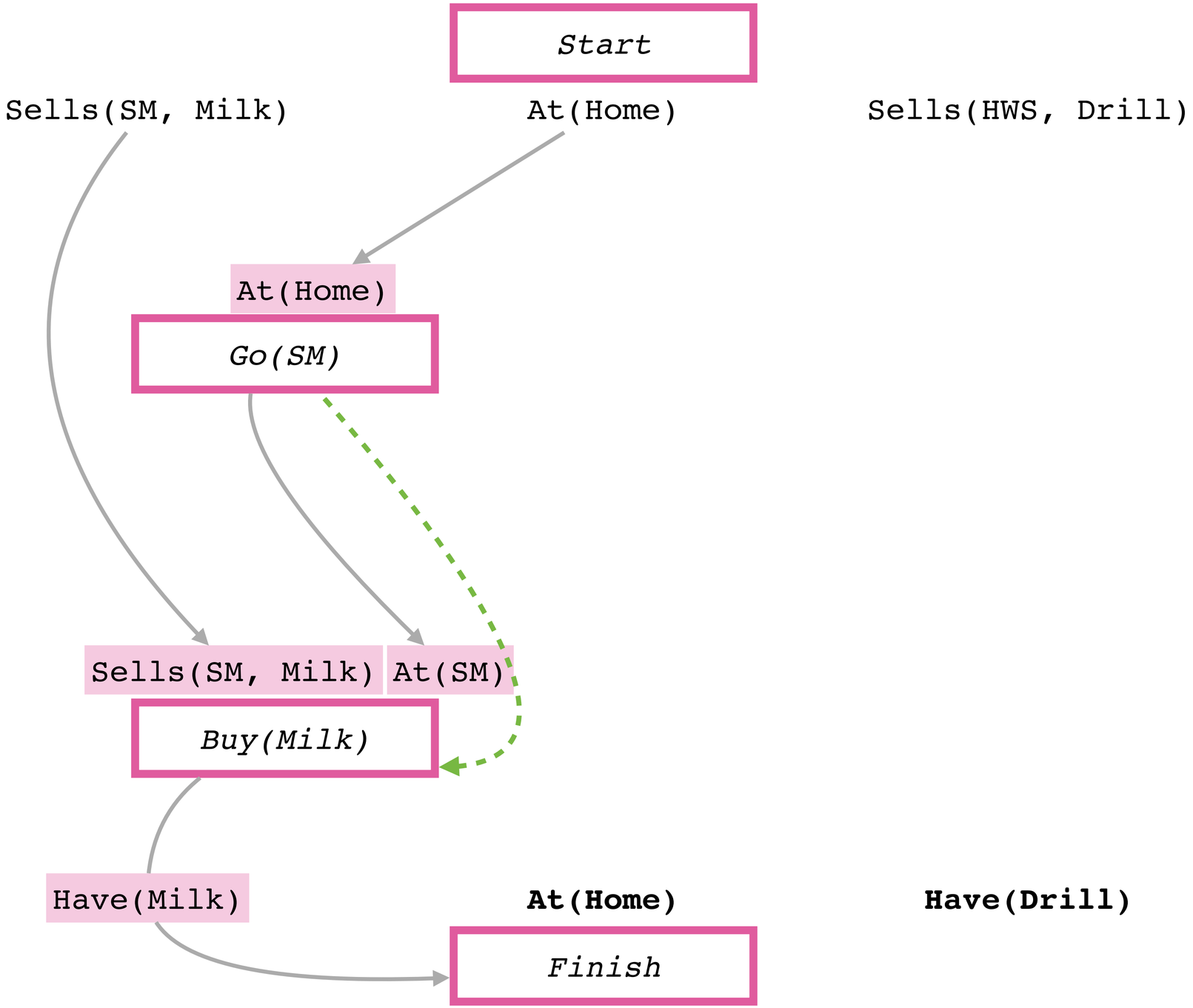

Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
Sells(x, Milk)
At(SM)
Buy(Milk)
Have(Milk)
At(Home)
Have(Drill)
Finish

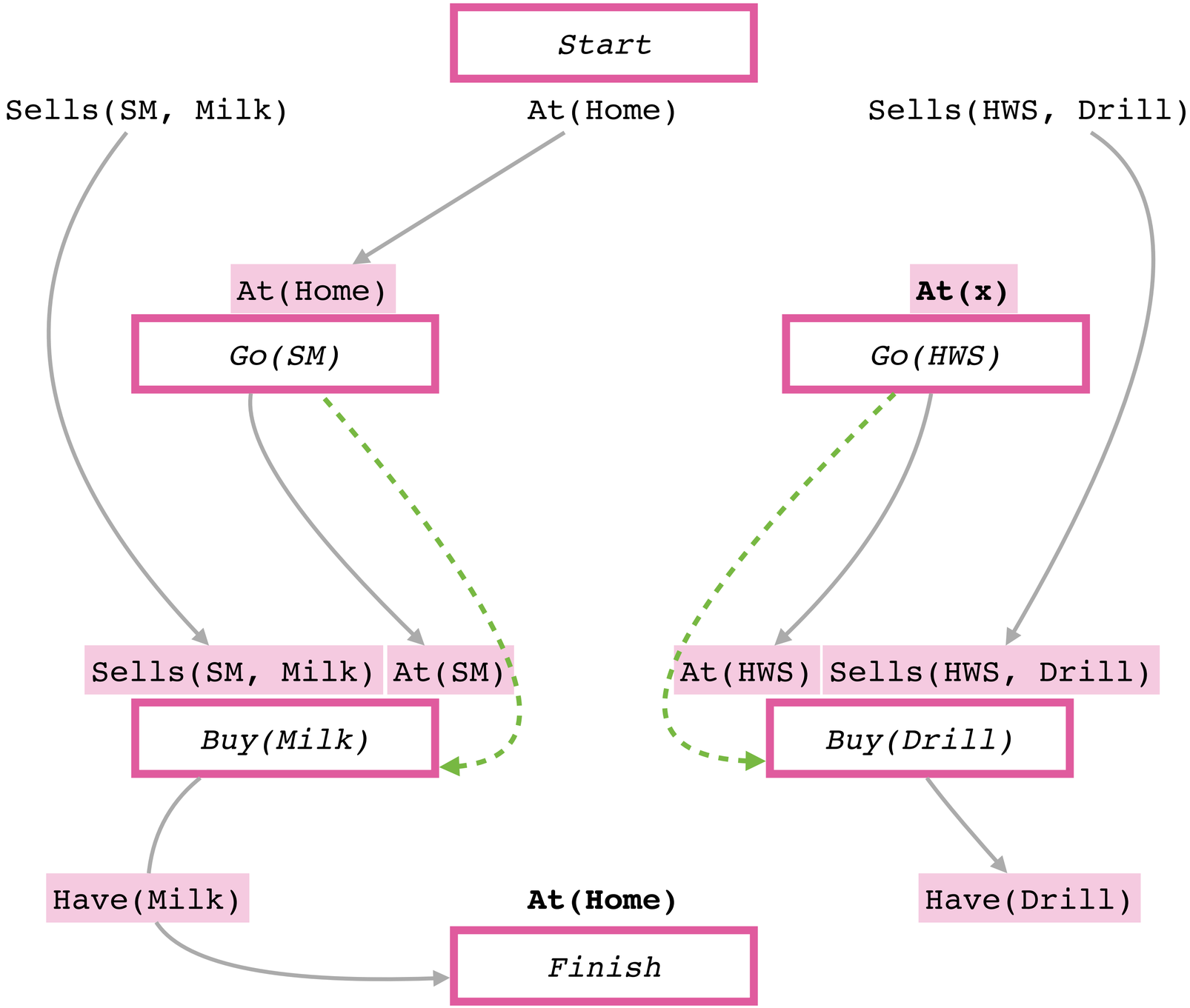

Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
Sells(SM, Milk)
At(SM)
Buy(Milk)
Have(Milk)
At(Home)
Have(Drill)
Finish

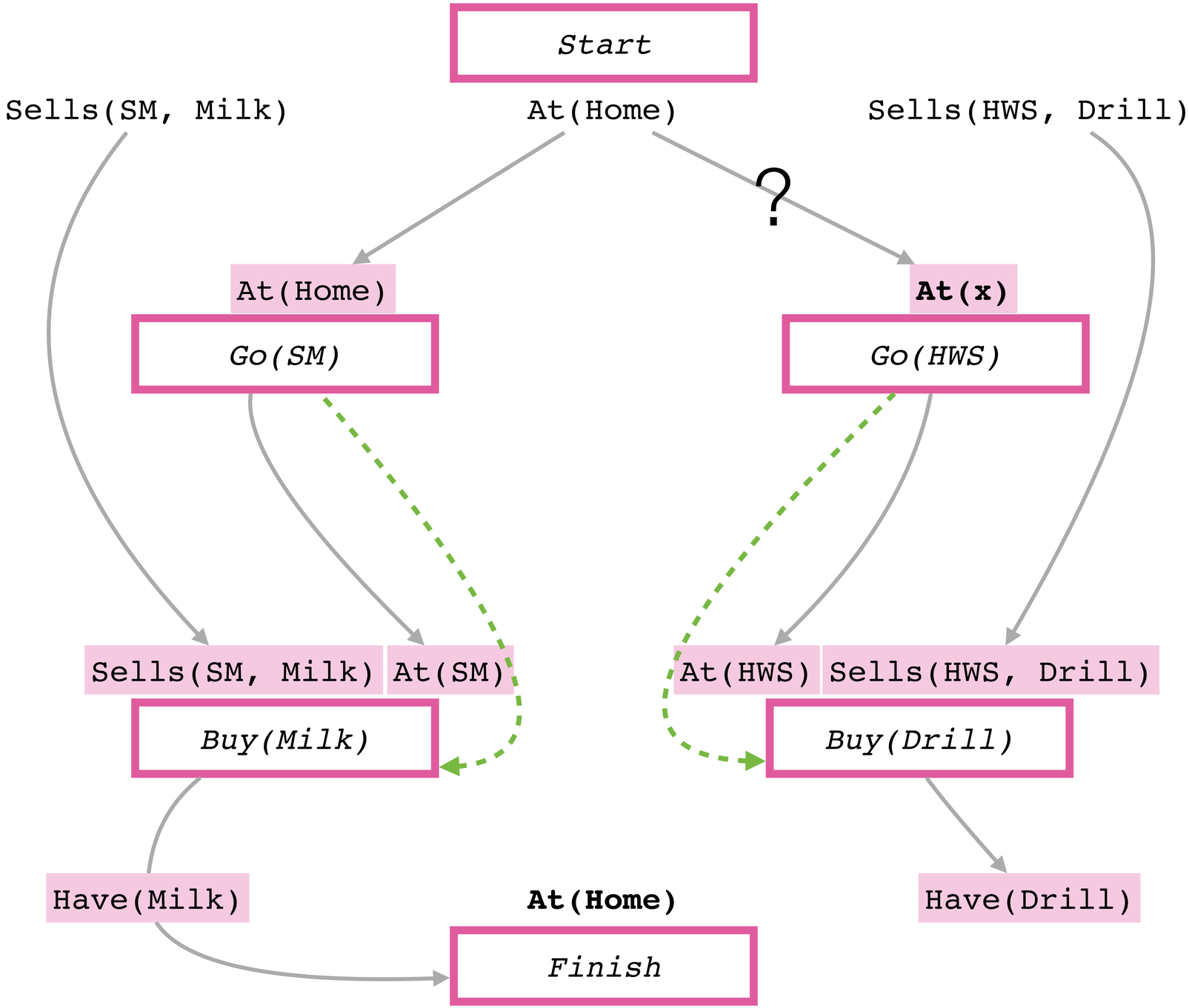

Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
Sells(SM, Milk)
At(SM)
Buy(Milk)
Have(Milk)
At(Home)
Have(Drill)
Finish

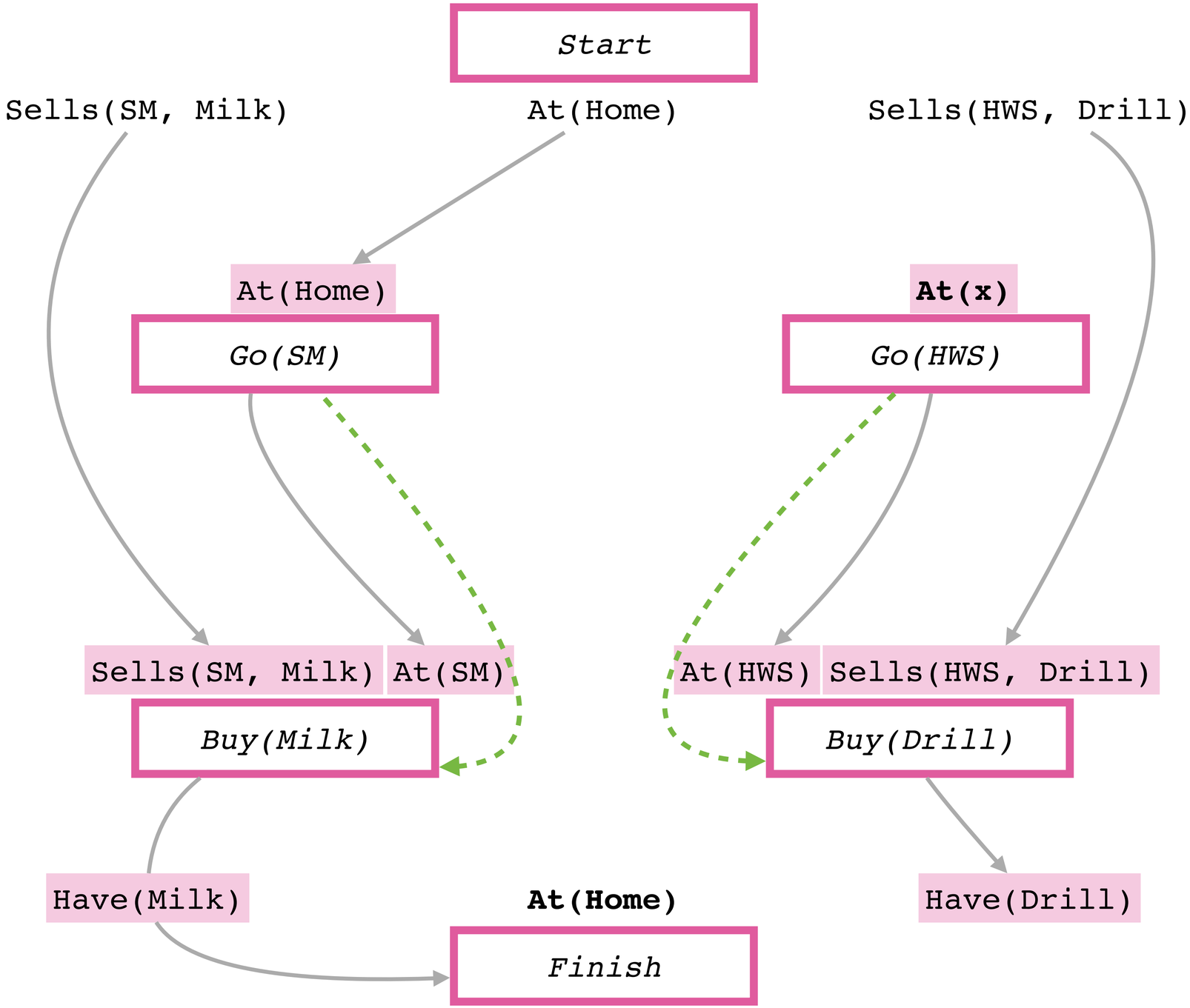

Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
At(x)
Go(SM)
Sells(SM, Milk)
At(SM)
Buy(Milk)
Have(Milk)
At(Home)
Have(Drill)
Finish

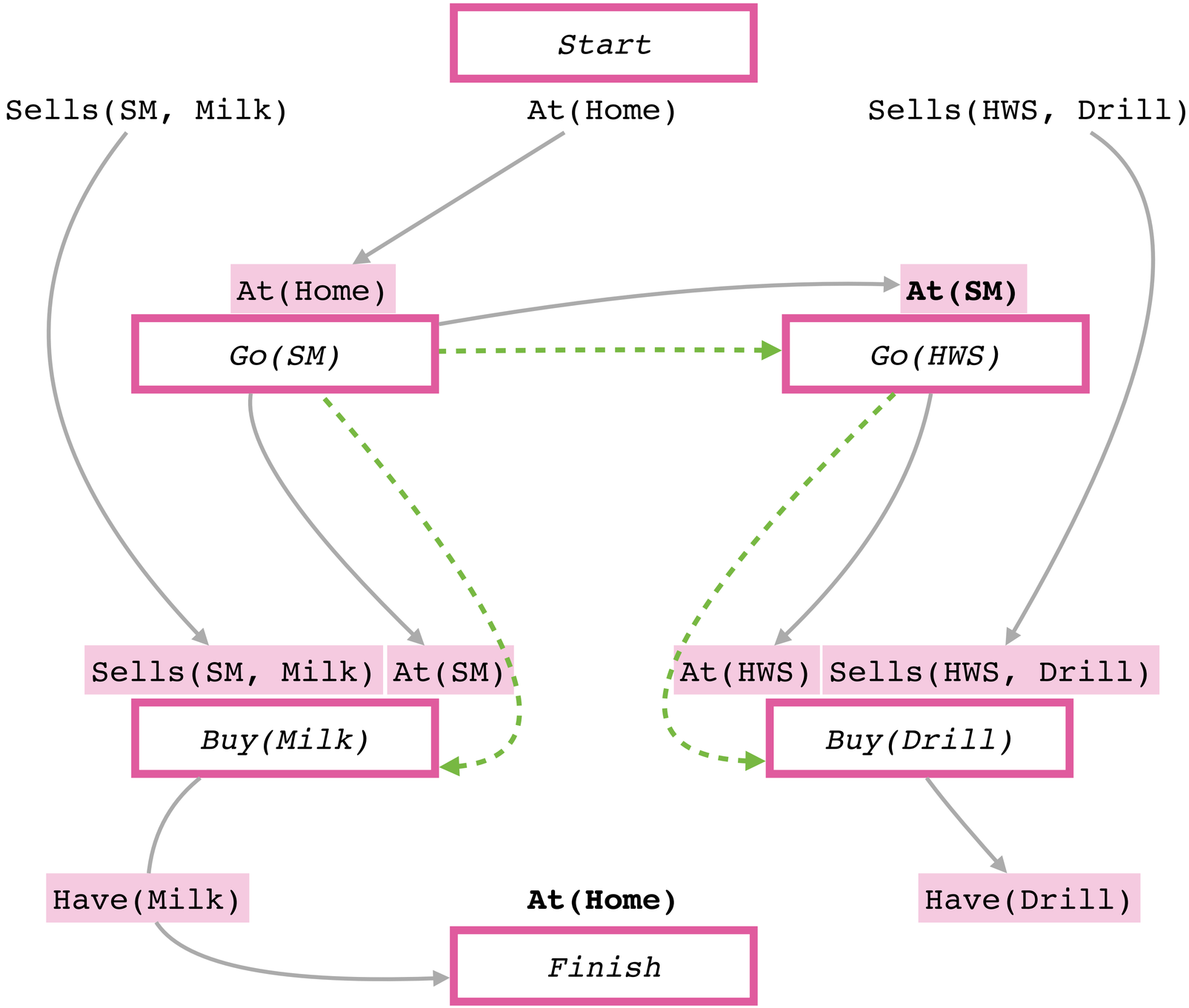

Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
At(x)
Go(SM)
Sells(SM, Milk)
At(SM)
Buy(Milk)
Have(Milk)
At(Home)
Have(Drill)
Finish

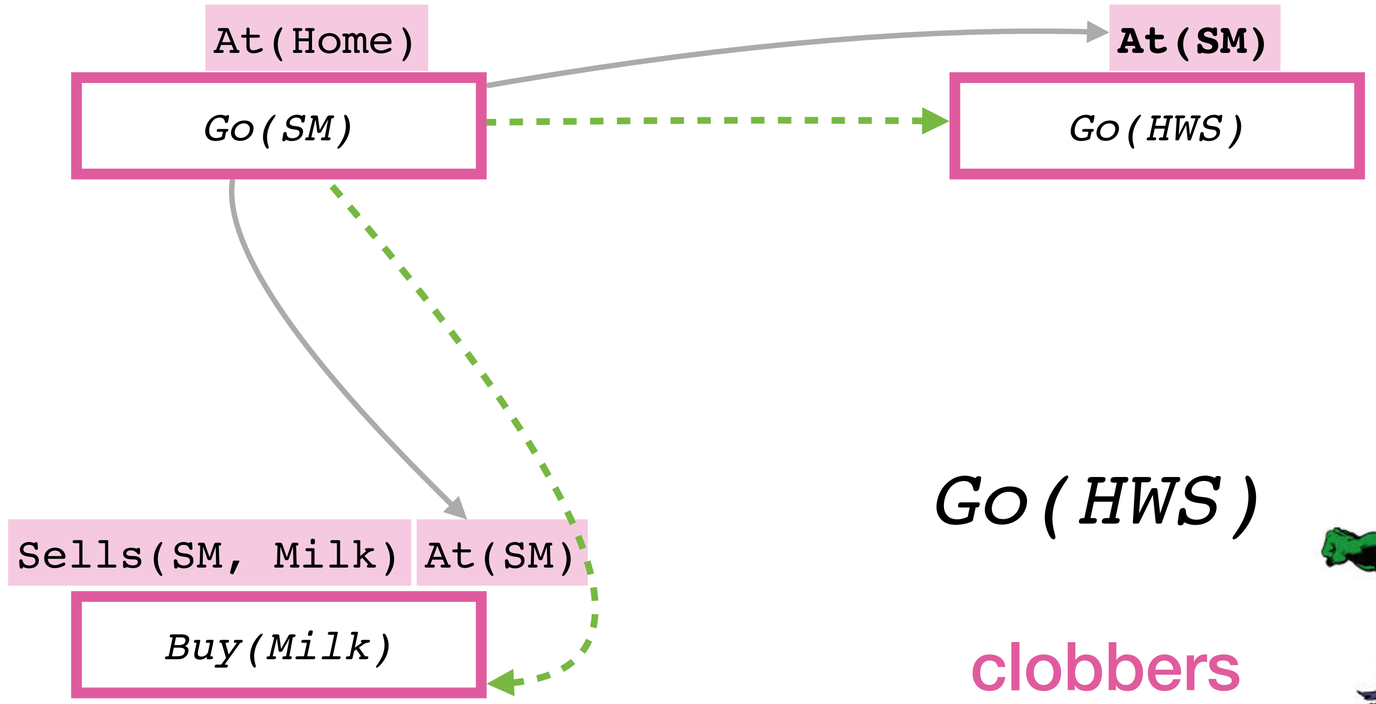

Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
At(Home)
Go(SM)
Sells(SM, Milk)
At(SM)
Buy(Milk)
Have(Milk)
At(Home)
Have(Drill)
Finish


Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
At(Home)
Go(SM)
At(x)
Go(HWS)
Sells(SM, Milk)
At(SM)
At(HWS)
Sells(HWS, Drill)
Buy(Milk)
Buy(Drill)
Have(Milk)
At(Home)
Have(Drill)
Finish


Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
?
At(Home)
Go(SM)
At(x)
Go(HWS)
Sells(SM, Milk)
At(SM)
Buy(Milk)
At(HWS)
Sells(HWS, Drill)
Buy(Drill)
Have(Milk)
At(Home)
Finish
Have(Drill)


Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
At(Home)
Go(SM)
At(x)
Go(HWS)
Sells(SM, Milk)
At(SM)
At(HWS)
Sells(HWS, Drill)
Buy(Milk)
Buy(Drill)
Have(Milk)
At(Home)
Have(Drill)
Finish


Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
At(Home)
Go(SM)
At(SM)
Go(HWS)
Sells(SM, Milk)
At(SM)
Buy(Milk)
At(HWS)
Sells(HWS, Drill)
Buy(Drill)
Have(Milk)
At(Home)
Finish
Have(Drill)

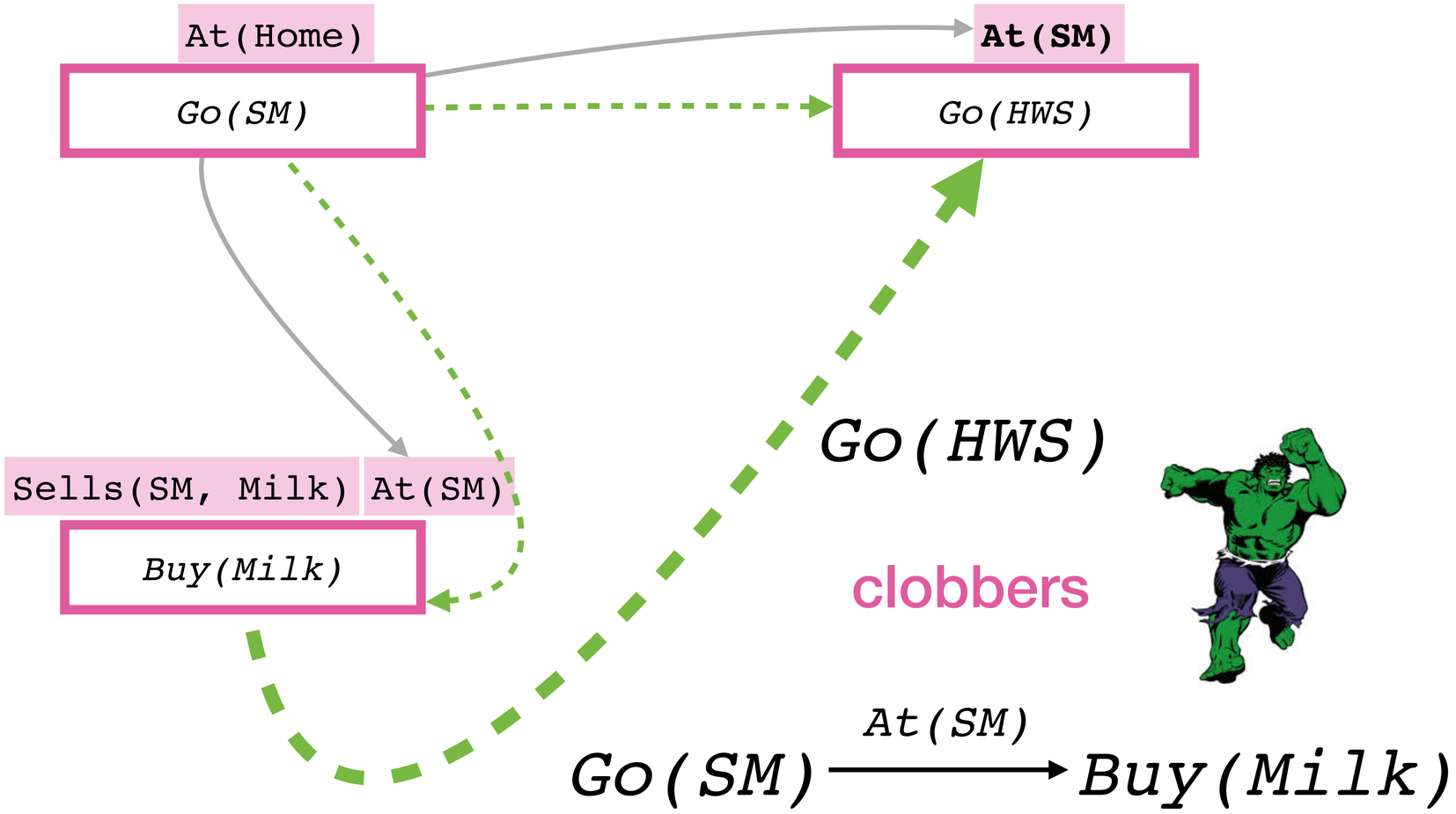


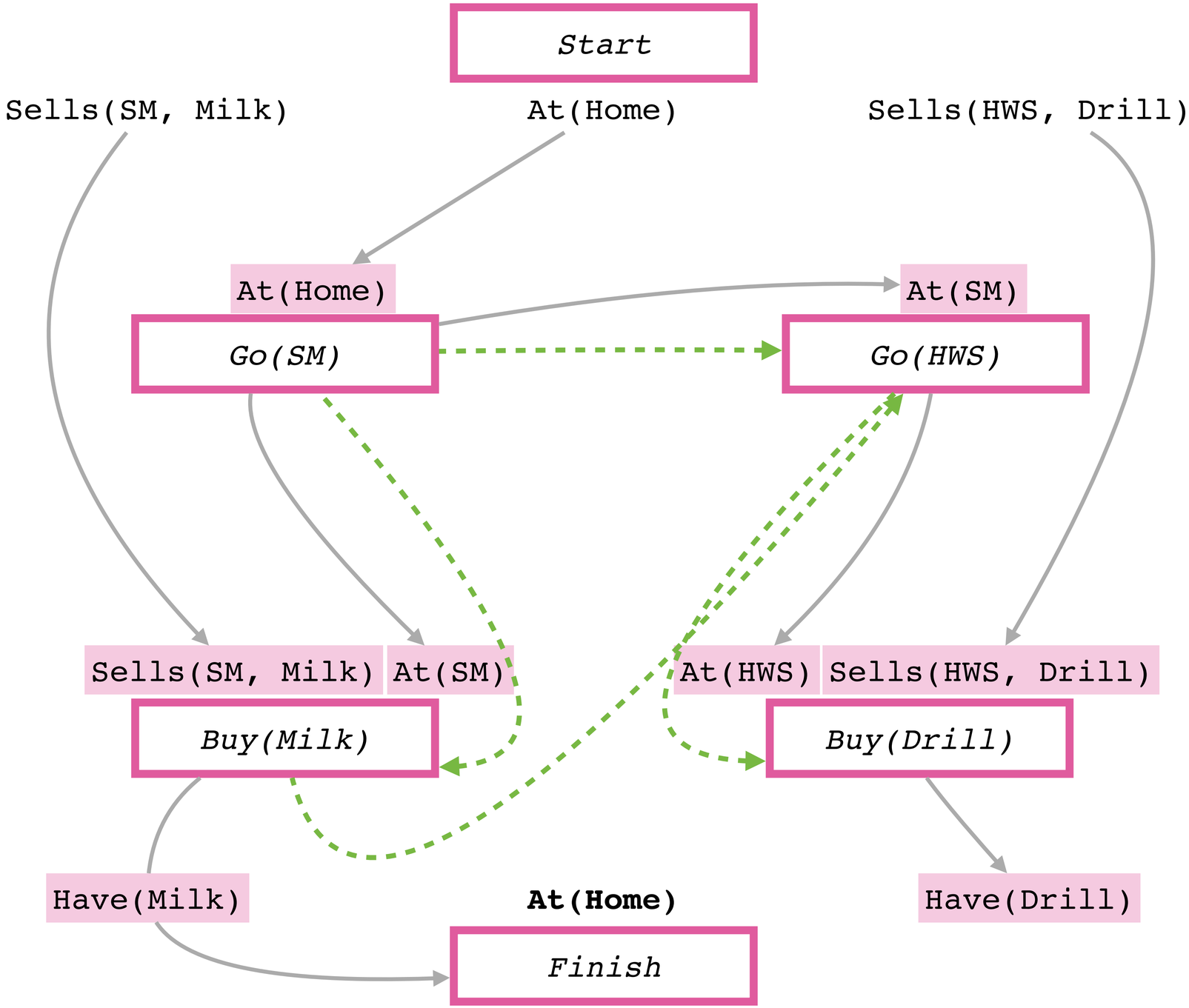
Go(HWS)

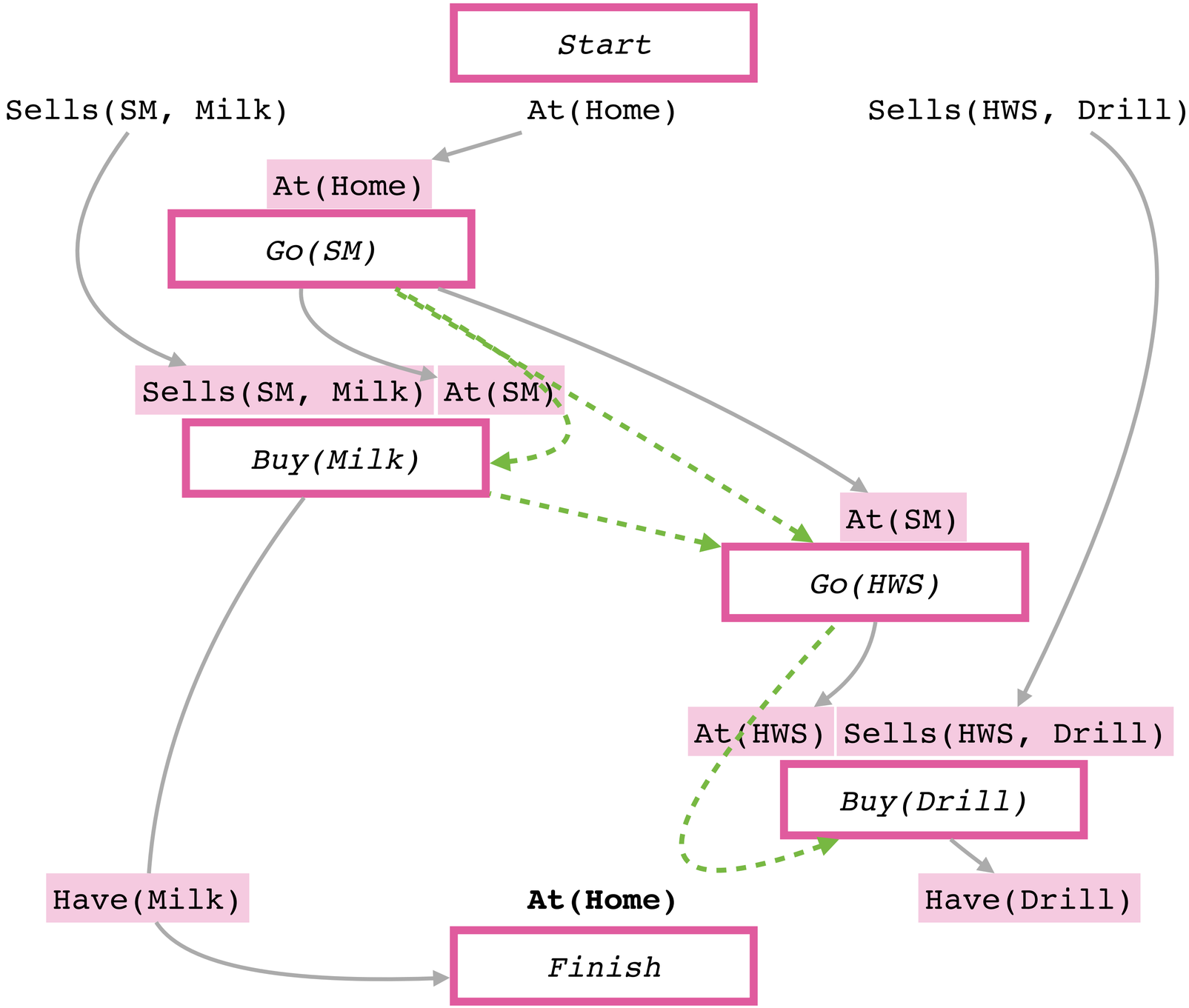
clobbers

Go(SM) —*At(SM)*→ *Buy(Milk)*

Go(HWS) is *promoted* to later in the plan




Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
At(Home)
Go(SM)
At(SM)
Go(HWS)
Sells(SM, Milk)
At(SM)
Buy(Milk)
At(HWS)
Sells(HWS, Drill)
Buy(Drill)
Have(Milk)
At(Home)
Finish
Have(Drill)


Start
Sells(SM, Milk)
At(Home)
Sells(HWS, Drill)
At(Home)
Go(SM)
Sells(SM, Milk)
At(SM)
Buy(Milk)
At(SM)
Go(HWS)
At(HWS)
Sells(HWS, Drill)
Buy(Drill)
Have(Milk)
At(Home)
Have(Drill)
Finish

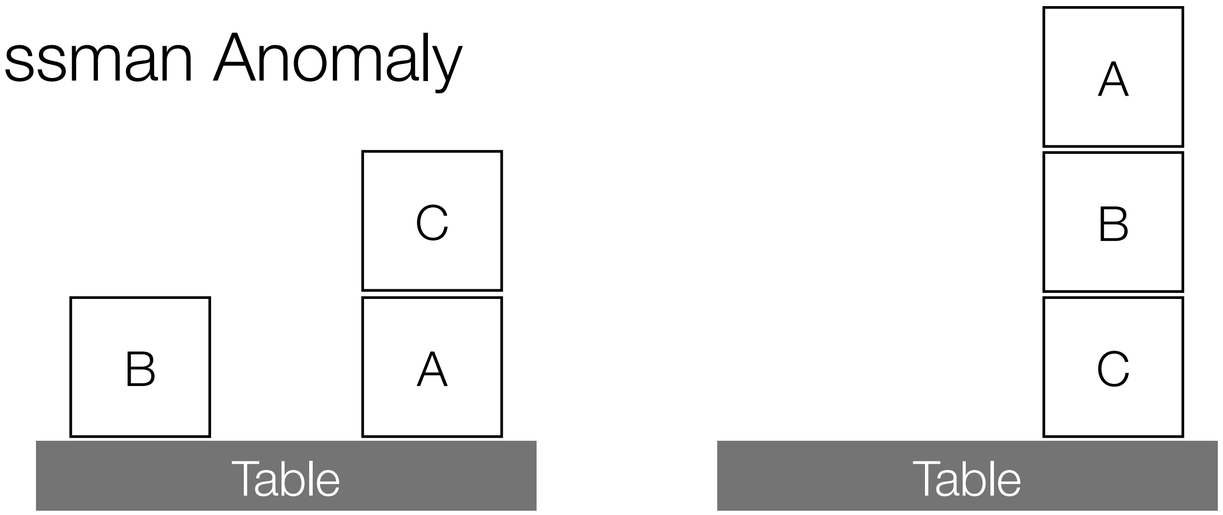
Nondeterministic algorithm: backtracks at choice points on failure:

- choice of step (S_{add}) to achieve open precondition (S_{need})
- choice of demotion or promotion for **conflict**

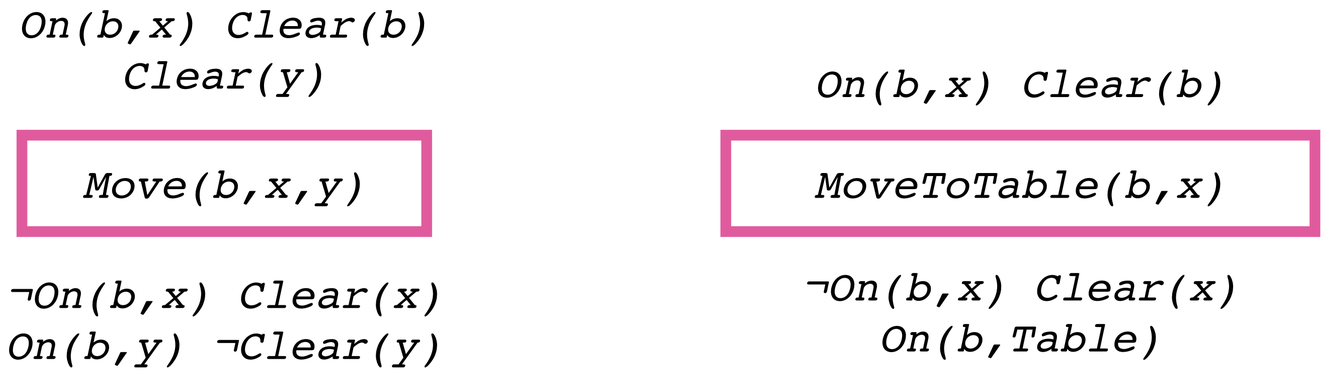
Selection of open precondition (S_{need}) is **irrevocable**: the existence of a plan does not depend on the choice of the open preconditions.

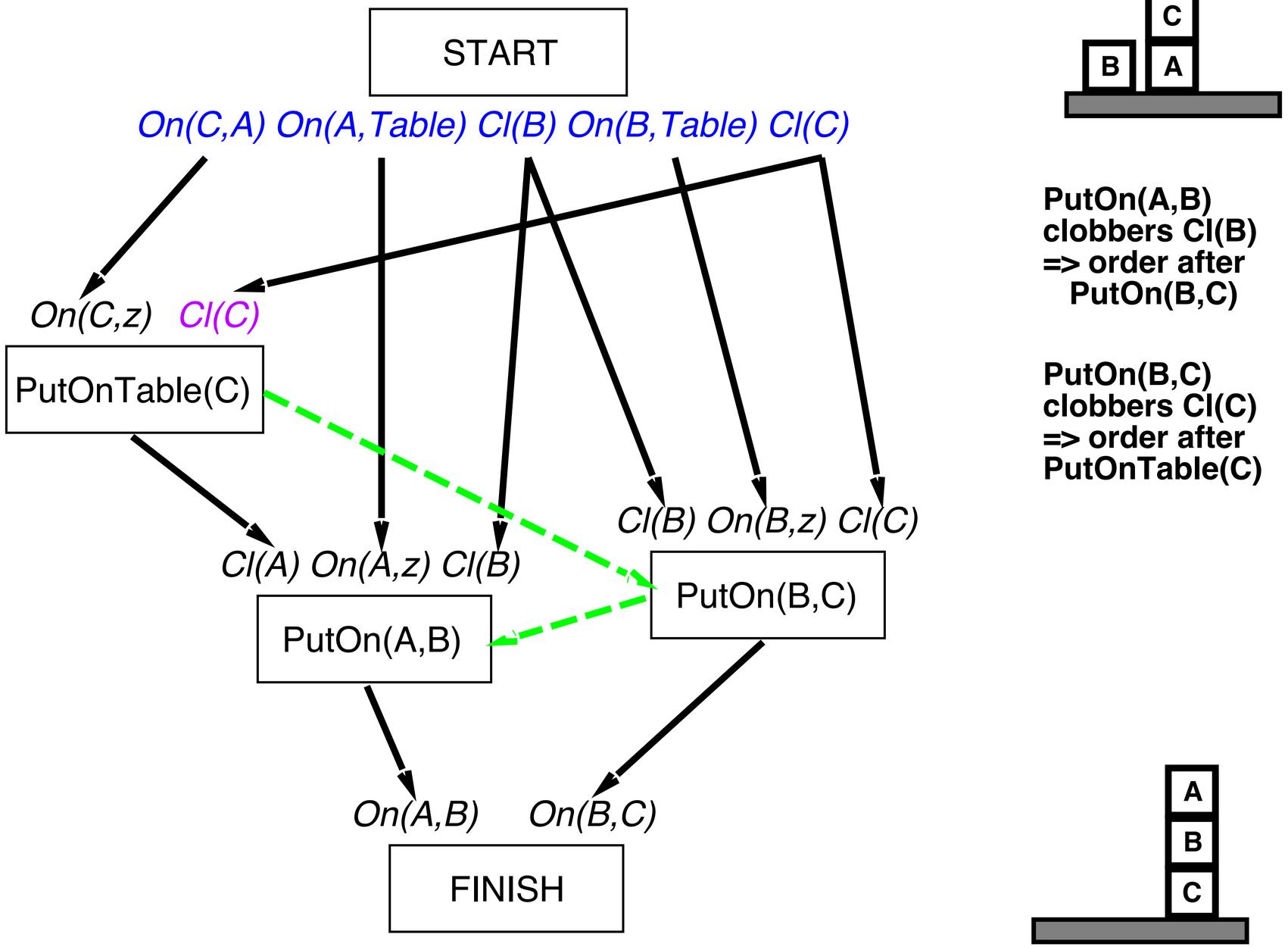
Partial order planning is **sound**, and **complete**.

The Sussman Anomaly



Task: Apply partial order planning to create a plan. Only draw the temporal orderings between non start/finish steps.





PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)

PutOn(B,C)
clobbers Cl(C)
=> order after
PutOnTable(C)

Classical Planning

Classical Planning

fully observable

Classical Planning

fully observable

deterministic

Classical Planning

fully observable

deterministic

static environments

Classical Planning

fully observable

deterministic

static environments

single agent

Real World Planning

Real World Planning

partially observable

Real World Planning

partially observable

non-deterministic

Real World Planning

partially observable

non-deterministic

dynamic environments

Real World Planning

partially observable

non-deterministic

dynamic environments

single or multi-agent

Questions?