Planning and Theorem Proving

Slides by Svetlana Lazebnik, 9/2016
with modifications by Mark Hasegawa-Johnson, 1/2019

Planning and Theorem Proving

- Examples
- Automatic Theorem Proving: forward-chaining, backward-chaining
- Planning: forward-chaining, backward-chaining
- Admissible Heuristics for Planning and Theorem Proving
    - Number of Steps
    - Planning Graph
- Computational Complexity

Example: River Crossing Problems

https://en.wikipedia.org/wiki/River_crossing_puzzle

- A **farmer** has a **fox**, a **goat**, and a bag of **beans** to get across the river
- His boat will only carry him + one object
- He can't leave the fox with the goat
- He can't leave the goat with the bag of beans

Solution

https://en.wikipedia.org/wiki/River_crossing_puzzle

lower case: on this side of the river
upper case: across the river

fgb -----(farmer, goat)----→ FGb

fGb ←-----(farmer)-----------

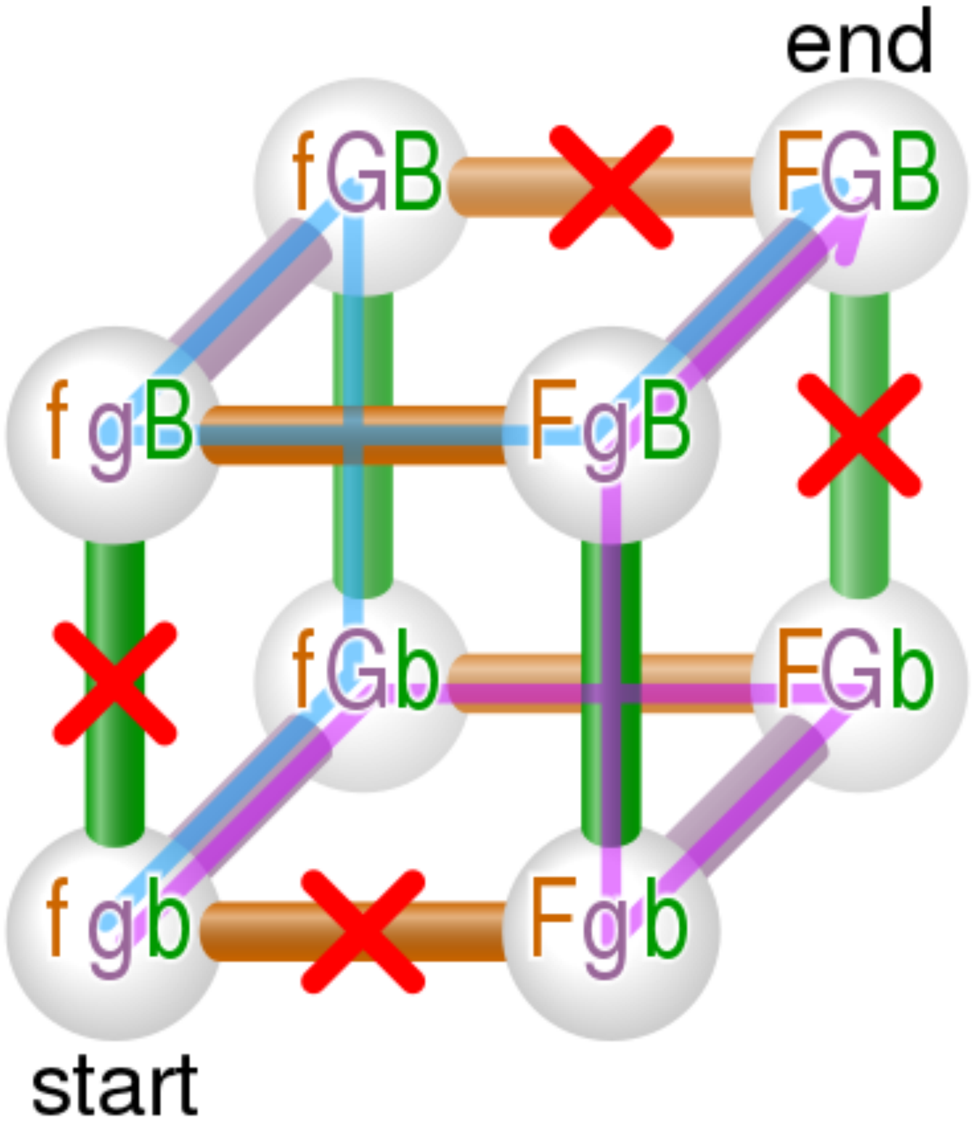
-----(farmer,fox)-----→ FGb

Fgb ←--(farmer,goat)------

-----(farmer,beans)---→ FgB

FgB ←-------(farmer)--------

-----(farmer,goat)----→ FGB



Example: Cargo delivery problem

- You have packages waiting for pickup at Atlanta, Boston, Charlotte, Denver, Edmonton, and Fairbanks
- They must be delivered to Albuquerque, Baltimore, Chicago, Des Moines, El Paso, and Frisco
- You have two trucks. Each truck can hold only two packages at a time.

Example: Design for Disassembly

”Simultaneous Selective Disassembly and End-of-Life Decision Making for Multiple Products That Share Disassembly Operations,” Sara Behdad, Minjung Kwak, Harrison Kim and Deborah Thurston. *J. Mech. Des* **132**(4), 2010, doi:10.1115/1.4001207

- Design decisions limit the sequence in which you can disassemble a product at the end of its life
- Problem statement: design the product in order to make disassembly as cheap as possible

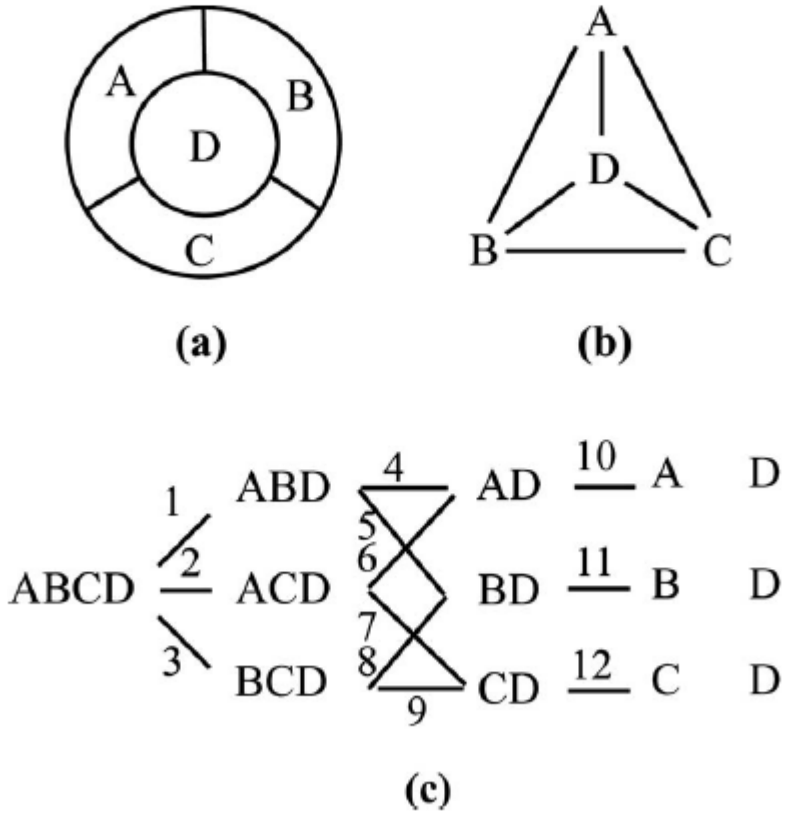


Fig. 1 Simple assembly (*a*), its connection diagram (*b*), and its disassembly graph (*c*) [23]

Application of planning: the Gale-Church alignment algorithm for machine translation

Table 2
Output from alignment program.

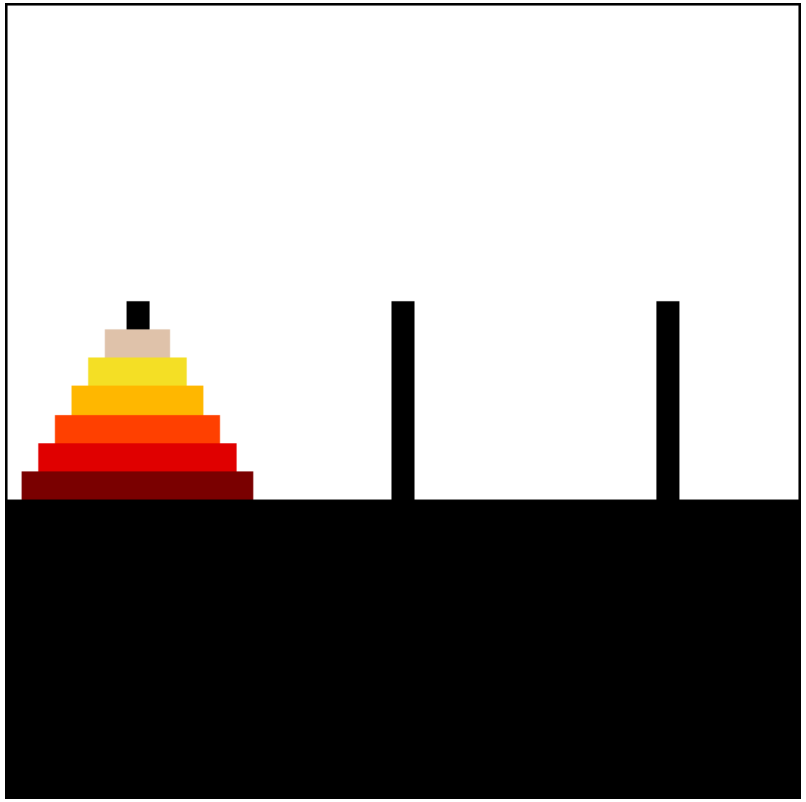
English	French
According to our survey, 1988 sales of mineral water and soft drinks were much higher than in 1987, reflecting the growing popularity of these products. Cola drink manufacturers in particular achieved above-average growth rates.	Quant aux eaux minérales et aux limonades, elles rencontrent toujours plus d'adeptes. En effet, notre sondage fait ressortir des ventes nettement supérieures à celles de 1987, pour les boissons à base de cola notamment.
The higher turnover was largely due to an increase in the sales volume.	La progression des chiffres d'affaires résulte en grande partie de l'accroissement du volume des ventes.
Employment and investment levels also climbed.	L'emploi et les investissements ont également augmenté.

Application of planning: the Gale-Church alignment algorithm for machine translation

1. Let $d(x_1, y_1; 0, 0)$ be the cost of substituting x_1 with y_1,
2. $d(x_1, 0; 0, 0)$ be the cost of deleting x_1,
3. $d(0, y_1; 0, 0)$ be the cost of insertion of y_1,
4. $d(x_1, y_1; x_2, 0)$ be the cost of contracting x_1 and x_2 to y_1,
5. $d(x_1, y_1; 0, y_2)$ be the cost of expanding x_1 to y_1 and y_2, and
6. $d(x_1, y_1; x_2, y_2)$ be the cost of merging x_1 and x_2 and matching with y_1 and y_2.

Example: Tower of Hanoi

https://en.wikipedia.org/wiki/Tower_of_Hanoi



Description	**English:** This is a visualization generated with the walnut based on my implementation at [1] of the iterative algorithm described in Tower of Hanoi
Date	30 April 2015
Source	I designed this using http://thewalnut.io/
Author	Trixx

Planning and Theorem Proving

- Examples
- Automatic Theorem Proving: forward-chaining, backward-chaining
- Planning: forward-chaining, backward-chaining
- Admissible Heuristics for Planning and Theorem Proving
  - Number of Steps
  - Planning Graph
- Computational Complexity

The Syntax of First-Order Logic (Textbook p. 293)

$$Sentence \rightarrow Predicate(Term, \ldots)$$

$$| \neg Sentence$$

$$| \; Sentence \wedge Sentence$$

$$| \; Sentence \vee Sentence$$

$$| \; Sentence \implies Sentence$$

$$| Sentence \iff Sentence$$

$$| Quantifier \, Variable, \ldots Sentence$$

A “sentence” is

- an predicate applied to a set of terms, or
- a negated sentence, or
- the conjunction of 2 sentences, or
- the disjunction of 2 sentences, or
- an implication, or
- an equivalence, or
- a sentence with a quantified variable.

$$Term \rightarrow Function(Term)$$

$$| Variable \, | Constant$$

A “term” is an evaluated function, or a variable, or a constant.

$$Quantifier \rightarrow \exists \mid \forall$$

A “quantifier” is “there exists,” or “for all.”

Terms, Sentences, predicates, functions

- **Terms** (variables, constants) refer to entities
- **Sentences** have truth values: they can be true or false
- **Predicates** and **functions** look the same -- both are applied to terms: `American(x), FatherOf(x),`
- When **Predicates** are applied to terms,
  the result is a sentence that can be true or false
- When **Functions** are applied to terms,
  the result is another entity

Examples (Textbook, p. 330)

English	First-Order Logic Notation
It is a crime for Americans to sell weapons to hostile nations.	$American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$
Colonel West sold missiles to Ganymede.	$\exists x\ Missile(x) \land Sells(West, x, Ganymede)$
Colonel West is American.	$American(West)$
Ganymede is an enemy of America.	$Enemy(Ganymede, America)$
Missiles are weapons.	$Missile(x) \Rightarrow Weapon(x)$
An enemy of America is a hostile nation.	$Enemy(x, America) \Rightarrow Hostile(x)$

Automatic Theorem Proving

First-Order Logic Notation
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Longrightarrow Criminal(x)$
$\exists x\ Missile(x) \wedge Sells(West, x, Ganymede)$
$American(West)$
$Enemy(Ganymede, America)$
$Missile(x) \Longrightarrow Weapon(x)$
$Enemy(x, America) \Longrightarrow Hostile(x)$

Can we prove the theorem:

$Criminal(West)$?

Actions that a Theorem Prover can Take

- **<u>Universal Instantiation</u>**:
  - given the sentence $\forall x, Predicate(x)$,
  - for any known constant C,
  - it is possible to generate the sentence $Predicate(C)$
- **<u>Existential Instantiation</u>**:
  - given the proposition $\exists x, Predicate(x)$,
  - if no known constant A is known to satisfy $Predicate(A)$, then
  - it is possible to define a new, otherwise unspecified constant B, and
  - to generate the sentence $Predicate(B)$.
- **<u>Generalized Modus Ponens:</u>**
  - Given the sentence $p_1(x_1) \wedge p_2(x_2) \wedge \ldots \wedge p_n(x_n) \Longrightarrow q(x_1, \ldots, x_n)$, and
  - given the sentences $p_1(C_1), \ldots, p_n(C_n)$ for any constants $C_1, \ldots, C_n$,
  - it is possible to generate the sentence $q(C_1, \ldots, C_n)$

Automatic Theorem Proving Example

- <u>**Existential Instantiation**</u>:
  - Input: $\exists x, Missile(x) \wedge Sells(West, x, Ganymede)$
  - Output: $Missile(M) \wedge Sells(West, M, Ganymede)$
- <u>**Generalized Modus Ponens:**</u>
  - Input: $Missile(M)$ <u>**and**</u> $Missile(x) \Longrightarrow Weapon(x)$
  - Output: $Weapon(M)$
- <u>**Generalized Modus Ponens:**</u>
  - Input: $Enemy(Ganymede, America)$ <u>**and**</u> $Enemy(x, America) \Longrightarrow Hostile(x)$
  - Output: $Hostile(Ganymede)$
- <u>**Generalized Modus Ponens:**</u>
  - Input: $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Longrightarrow Criminal(x)$
    <u>**and**</u>
    $American(West), Weapon(M), Sells(West, M, Ganymede), Hostile(Ganymede)$
  - Output: $Criminal(West)$

Automatic Theorem Proving as Search

- **State** = the set of all currently known sentences
- **Action** = generate a new sentence
- **Goal State** = a set of sentences that includes the target sentence

(Question to ponder: how do you disprove a target sentence?)

Forward Chaining

- **What's Special About Theorem Proving**:
  - A state, at level n, can be generated by the combination of several states at level n-1.
- **Definition: Forward Chaining** is a search algorithm in which each action
  - generates a new sentence,
  - by combining as many different preceding states as necessary.

Example: Forward Chaining to prove q_3

$$\{p_1, p_2, p_1 \Rightarrow q_1, p_2 \Rightarrow q_2, q_1 \wedge q_2 \Rightarrow q_3\}$$

Initial State

$$\{p_1, p_2, p_1 \Rightarrow q_1, p_2 \Rightarrow q_2, q_1 \wedge q_2 \Rightarrow q_3, q_1\}$$

Search "Tree" Level 1

$$\{p_1, p_2, p_1 \Rightarrow q_1, p_2 \Rightarrow q_2, q_1 \wedge q_2 \Rightarrow q_3, q_2\}$$

$$\{p_1, p_2, p_1 \Rightarrow q_1, p_2 \Rightarrow q_2, q_1 \wedge q_2 \Rightarrow q_3, q_1, q_2, q_3\}$$

Search "Tree" Level 2: Goal Achieved

Backward Chaining

- **<u>What Else is Special About Theorem Proving:</u>**
  - The ”Goal State” is defined to be any set of sentences that includes the target sentence
- **<u>Definition: Backward Chaining</u>** is a search algorithm in which
  - State = {set of known sentences}, {set of desired sentences}
  - Action = apply a known sentence, backward, to a target sentence, in order to generate a new set of desired sentences
  - Goal = all “desired sentences” are part of the set of “known sentences”

Example: Backward Chaining to prove q_3

KNOWN: $\{p_1, p_2, p_1 \Longrightarrow q_1, p_2 \Longrightarrow q_2, q_1 \land q_2 \Longrightarrow q_3\}$

DESIRED: $\{q_3\}$

Initial State

DESIRED: $\{q_1, q_2\}$

Search Tree Level 1

DESIRED: $\{p_1, q_2\}$

DESIRED: $\{q_1, p_2\}$

Search Tree Level 2

DESIRED: $\{p_1, p_2\}$

DESIRED: $\{p_1, p_2\}$

Search Tree Level 3: Goal Achieved

Planning and Theorem Proving

- Examples
- Automatic Theorem Proving: forward-chaining, backward-chaining
- Planning: forward-chaining, backward-chaining
- Admissible Heuristics for Planning and Theorem Proving
  - Number of Steps
  - Planning Graph
- Computational Complexity

Search review

- A **search** problem is defined by:
  - **Initial state**
  - **Goal state**
  - **Actions**
  - **Transition model**
  - **Cost**

A representation for planning

- STRIPS (Stanford Research Institute Problem Solver): classical planning framework from the 1970s
- **States** are specified as **conjunctions of predicates**
  - Start state: At(home) ∧ Sells(SM, Milk) ∧ Sells(SM, Bananas) ∧ Sells(HW, drill)
  - Goal state: At(home) ∧ Have(Milk) ∧ Have(Banana) ∧ Have(drill)
- **Actions** are described in terms of **preconditions** and **effects**:
  - Go(x, y)
    - **Precond:** At(x)
    - **Effect:** ¬At(x) ∧ At(y)
  - Buy(x, store)
    - **Precond:** At(store) ∧ Sells(store, x)
    - **Effect:** Have(x)
- **Planning is “just” a search problem**

Planning as Theorem Proving

- A **planning action** is like a "$p \Longrightarrow q$" statement.
  - In order to be applied, it requires certain input sentences to be true. For example, the action "put the goat in the boat" requires, as its **precondition**, that the boat is empty.
  - The result of the action is the generation of an output sentence. For example: "the goat is now in the boat."
- The **initial state** is a set of sentences that are initially true.
- The **goal state** is a set of sentences that we want to "prove."

Important differences between Planning and Theorem Proving, #1: Negating your preconditions

- **A planning action may NEGATE some of its preconditions**.
  - Example: the action “put the goat in the boat” requires, as its precondition, the sentence ¬Boat(goat).
  - It generates, as its output, the sentence: Boat(goat).
- **No action can combine two world states that contain contradictory sentences.**
  For example, you can’t combine the states {p,q} and {p,¬q} to get the state {p,q,¬q}.

Algorithms for planning: Forward Chaining

Starting with the **start state**,

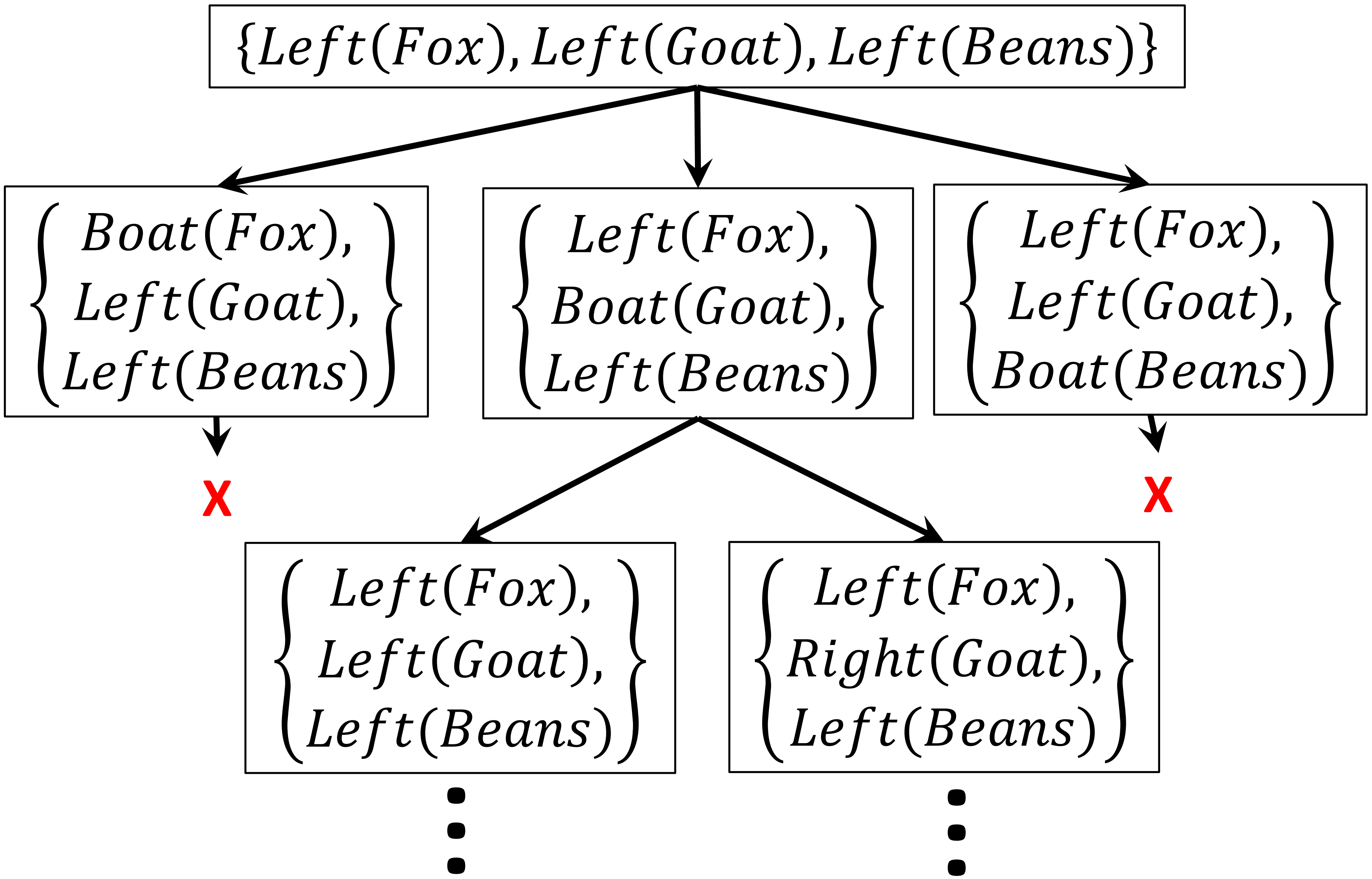
find **all applicable actions**
(actions for which preconditions are satisfied),

compute **the successor state** based on the effects,

keep searching **until goals are met**

- Can work well with good heuristics

Forward-Chaining Example: Fox, Goat & Beans

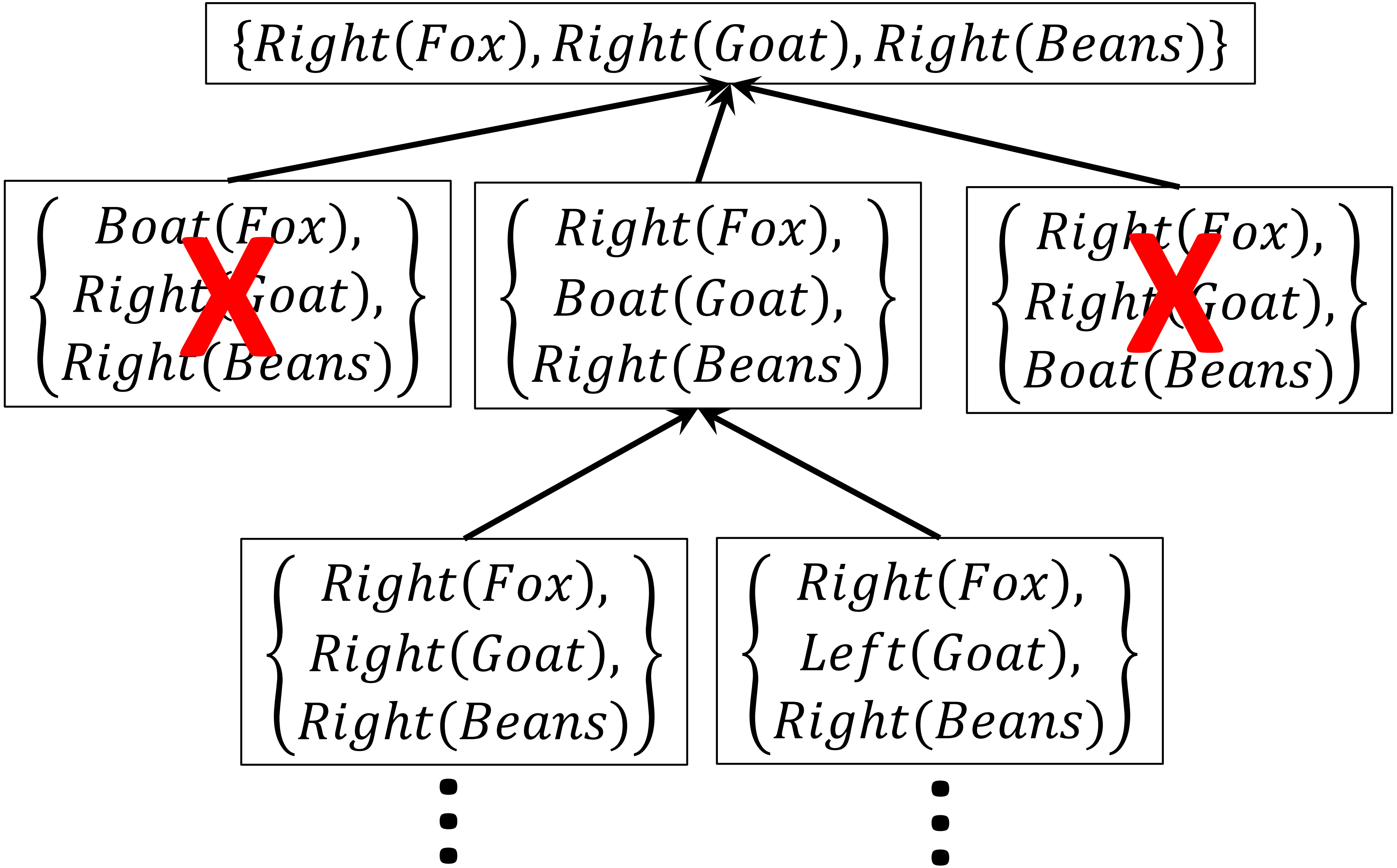


Algorithms for planning: Backward Chaining

Starting with the **goal state**
(a set of **target sentences**),

- find **all applicable actions**
  (actions that would **generate a sentence in the goal state**).
- For each applicable action, generate the **predecessor state**
  as a new set of target sentences.
- Keep searching until **all target sentences**
  are in the initial state.

Backward-Chaining Example: Fox, Goat & Beans



Planning and Theorem Proving

- Examples
- Automatic Theorem Proving: forward-chaining, backward-chaining
- Planning: forward-chaining, backward-chaining
- Admissible Heuristics for Planning and Theorem Proving
  - Number of Steps
  - Planning Graph
- Computational Complexity

A* Heuristics by Constraint Relaxation

- **Heuristics from Constraint Relaxation**:
  The heuristic h(n) is the number of steps it would take to get from n to G, if problem constraints were relaxed --- this guarantees that h(n) is admissible
- $h_1(n)$ dominates $h_2(n)$ ($h_1(n) \geq h_2(n)$)
  if $h_1(n)$ is computed by relaxing fewer constraints.

First heuristic: number of goal sentences left to achieve

Heuristic #1: Count the number of actions necessary to generate all of the sentences in the goal state that aren't already true.

- What got relaxed: we ignore action pre-requisites.

Example: 6 people on left side of the river, we want 6 people on the right side, we have a 2-person boat.

Minimum # actions: $h(n) = 3$.

Second heuristic: <u>planning graph</u>

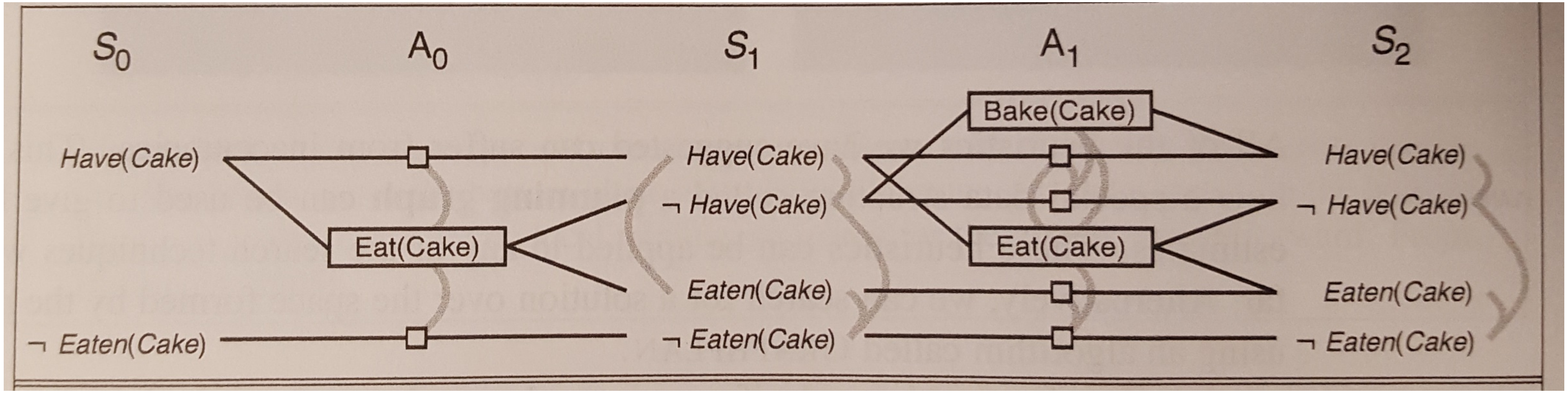
A **planning graph** is a trellis whose stages are:

- **Action stages** (A_n): list all of the actions whose pre-requisites are available in "Sentences stage" S_n
- **Sentence stages** (S_{n+1}): list all of the sentences that were available in S_n, plus any new sentences that could have been generated by any action in A_n

And within each stage, we have:

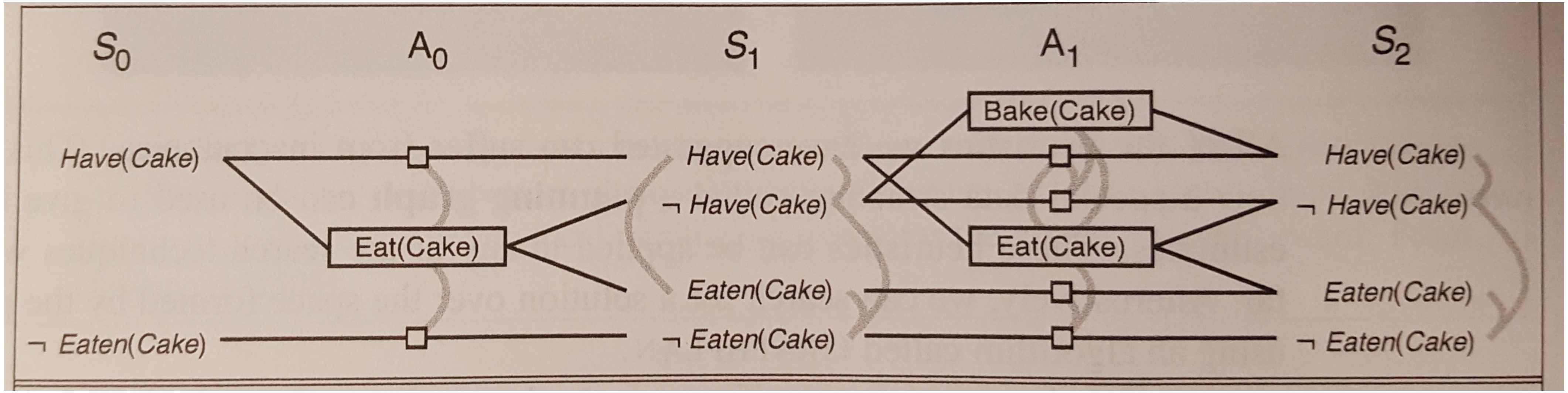
- **Mutex links**: If ALL actions that generate output sentence p also generate $\neg q$, then the sentences p and q become **mutex** (mutually exclusive).

Example planning graph



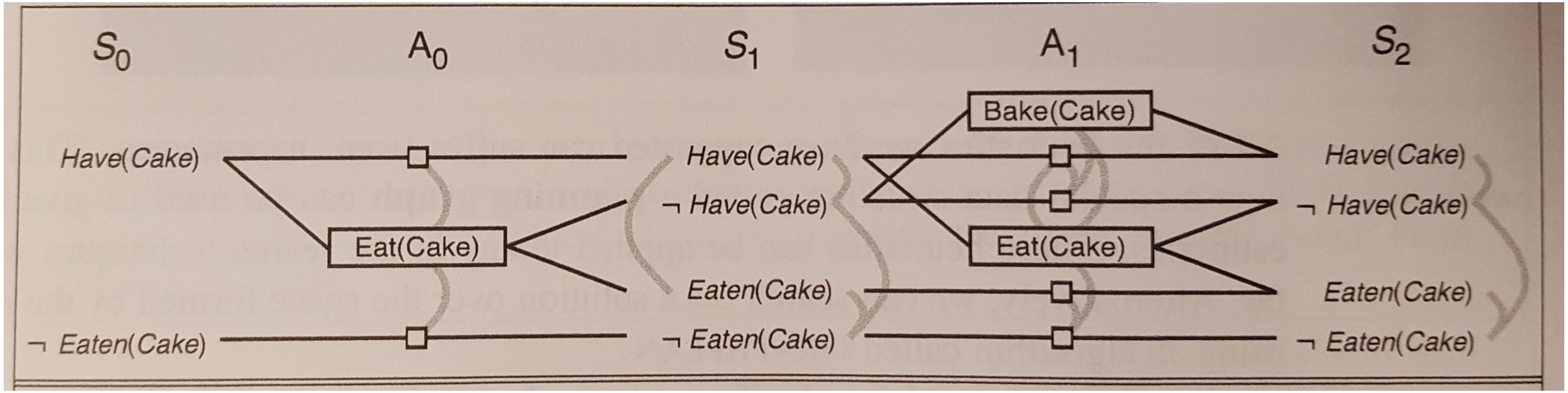
- A_0 has only two possible actions:
  - Do nothing: reproduces the initial state, {Have(Cake), ¬Eaten(Cake)}
  - Eat(Cake): generates {¬Have(Cake), Eaten(Cake)}
- Therefore, at S_1, Have(Cake) is mutex with Eaten(Cake)
- A_1: Bake(Cake) → Have(Cake), without generating ¬Eaten(Cake), so…
- S_1: Have(Cake) and Eaten(Cake) are no longer mutex.

Convergence of the Planning Graph



- **<u># of mutex links is monotonically non-increasing</u>**:
  If a pair of sentences are not mutex at stage S_n, then they are also not mutex at S_{n+1}
- **<u># possible actions is monotonically non-decreasing:</u>**
  If an action is possible at stage A_n, then it is also possible at A_{n+1}

Heuristic #2: Number of stages until target sentences are non-mutex



Heuristic: # stages between the current stage and the first stage at which all of the goal-state sentences are no longer mutex

Planning and Theorem Proving

- Examples
- Automatic Theorem Proving: forward-chaining, backward-chaining
- Planning: forward-chaining, backward-chaining
- Admissible Heuristics for Planning and Theorem Proving
  - Number of Steps
  - Planning Graph
- Computational Complexity

Complexity

- Planning is *PSPACE-complete* > *NP-complete*
  - The computational complexity of finding a plan is exponential
  - The length of the plan is exponential
    - Space necessary to represent it
    - Time necessary to implement it
  - The only thing that's polynomial: the amount of space necessary to represent the world state while finding or implementing a plan
- Example: towers of Hanoi

Complexity of planning

- Planning is *PSPACE-complete*
  - The length of a plan can be exponential in the number of "objects" in the problem!
  - So is game search
- Archetypal PSPACE-complete problem: *quantified boolean formula* (QBF)
  - Example: is this formula true?

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4\ (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$$

- Compare to SAT:

$$\exists x_1 \exists x_2 \exists x_3 \exists x_4\ (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$$

- Relationship between SAT and QBF is akin to the relationship between puzzles and games

Real-world planning

- **Resource constraints**
  - Instead of “**static**,” the world is “**semidynamic**:” we can’t think forever
- Actions at different levels of granularity: **hierarchical planning**
  - In order to make the depth of the search smaller, we might convert the world from “fully observable” to “partially observable”
- **Contingencies**: actions failing
  - Instead of being “deterministic,” maybe the world is “**stochastic**”
- Incorporating **sensing and feedback**
  - Possibly necessary to address **stochastic** or **multi-agent** environments