Lecture 7: Logic and Planning

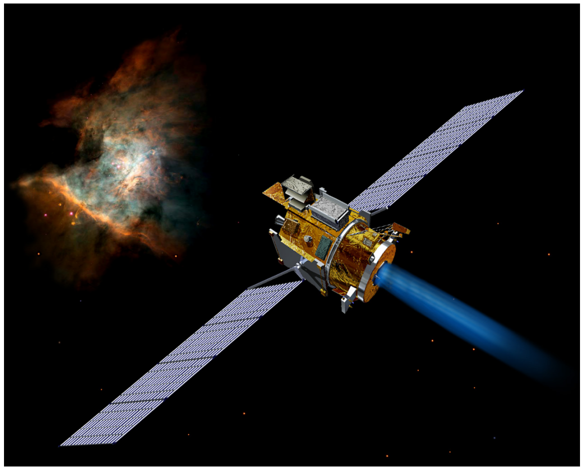
- Planning and representing knowledge
- Logic in general
- Propositional (Boolean) logic and its application to planning

What is planning?

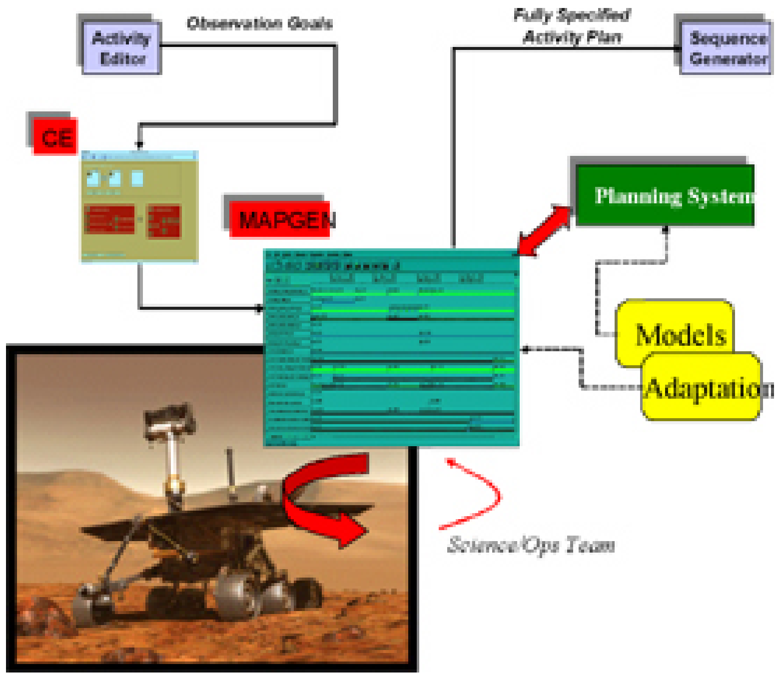
- A *plan* is a collection of actions for performing some task
  - E.g., my daughter's birthday celebration
  - E.g., a conference participation
  - E.g., assembling furniture
- There are many programs that help human planners
- The goal in AI is to *generate plans automatically*

NASA's Deep Space 1

- Launched in Oct. 1998 to test technologies and perform flybys of asteroid Braille and Comet Borrelly.
- Autonav system used for autonomous navigation and finding imaging targets.
- Remote Agent system used to perform automatic fault detection and self-repair.
- First spacecraft to be controlled by AI system without human intervention.



NASA's MAPGEN System



- Planning and scheduling system used daily to generate command sequences for the Mars Exploration Rover mission.
- Uses primarily mixed- initiative planning (i.e. collaborative planning between human and robot)

Search vs. Planning

- In theory, the types of problems above could be tackled by search methods we discussed so far
- Two main difficulties arise in complex search problems:
  - Branching factor is huge!
  - Difficult to find good heuristic functions
- Ideally, we don't want to search over individual states, but over *sets of states* or (as seen last time) *beliefs over states*
- *Key idea in planning:* use a more powerful form of *knowledge representation* to describe sets of states (or similar "abstractions")

Example: Second Life

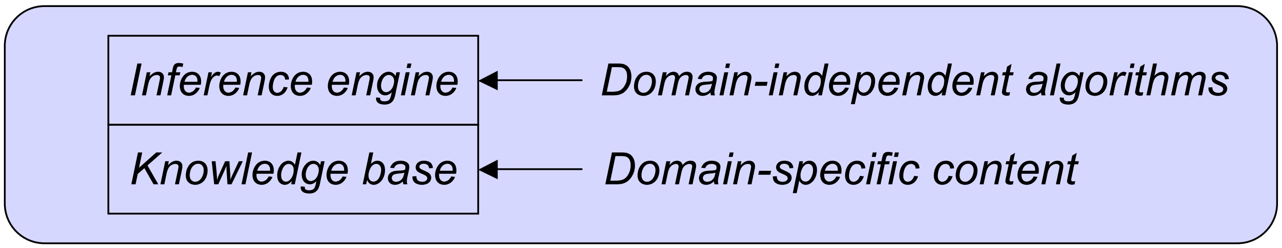
- How do we represent states in this game?
- How do we decide what to do?

Knowledge Representation

- A complete intelligent agent needs to be able to perform several tasks:
  - *Perception*: what is my state?
  - *Cognition/deliberation*: what action should I take?
  - *Action*: how do I execute the action?
- State recognition implies some form of *representation*
- Figuring out the right action implies some form of *inference*
- Two levels to think about:
  - *Knowledge level*: *what* does the agent know?
  - *Implementation level*: *how* is the knowledge represented?

Knowledge Bases

- The golden dream:
  - Tell the agent what it needs to know
  - The agent uses rules of inference to deduce consequences
- This is the *declarative* approach to building agents.



- Agents have two different parts:
  - A *knowledge base*, which contains a set of facts expressed in some formal, standard language
  - An *inference engine*, with general rules for deducing new facts

An Example: Wumpus World

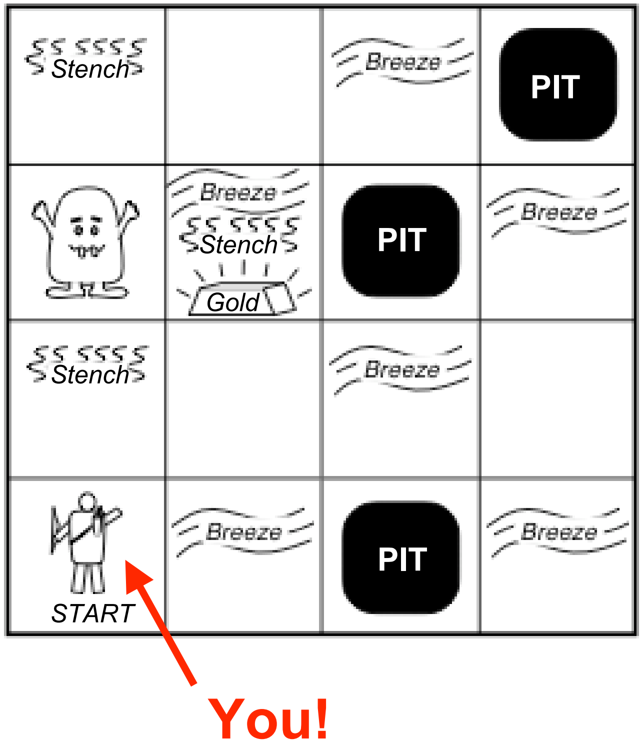
Percepts: Breeze, Glitter, Smell

Actions: Left turn, Right turn, Forward, Grab, Release, Shoot

Goals: Get gold back to start without entering pit or wumpus square

Environment:

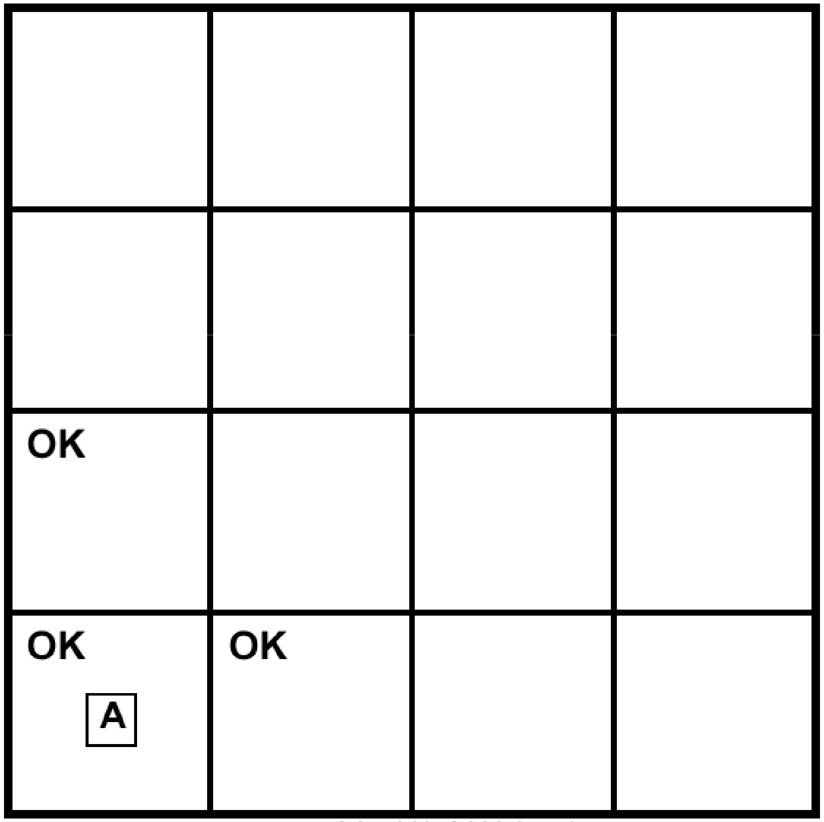
- Squares adjacent to wumpus are smelly
- Squares adjacent to pit are breezy
- Glitter if and only if gold is in the same square
- Shooting kills the wumpus if you are facing it
- Shooting uses up the only arrow
- Grabbing picks up the gold if in the same square



Wumpus World Characteristics

- The world is *static*: the positions of the pits, gold, and monster do not change during the course of a game
- The actions have *deterministic* effects.
- Unlike most search problems we talked about, here the world is *partially observable*!
- The agent does not know the map from the beginning, it has to figure it out based on *local perception*

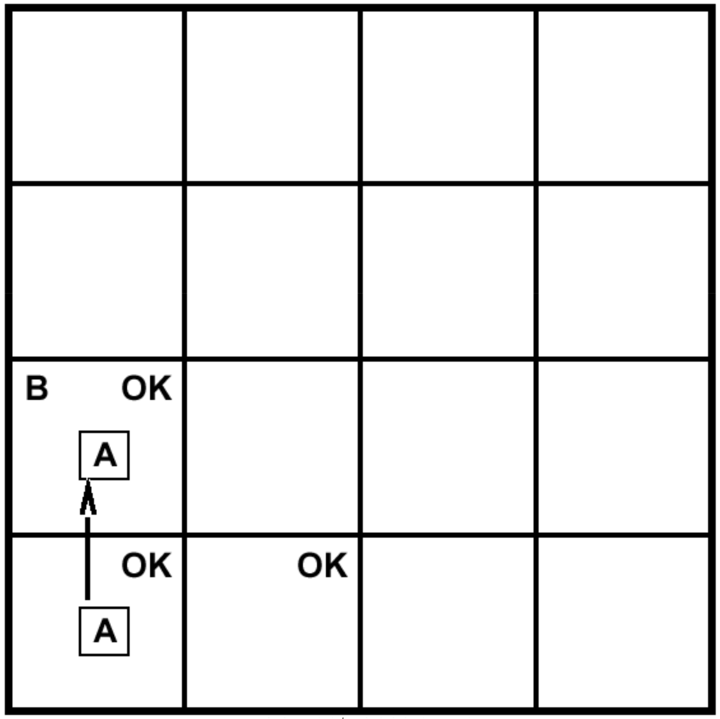
Exploring a Wumpus world



A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

8

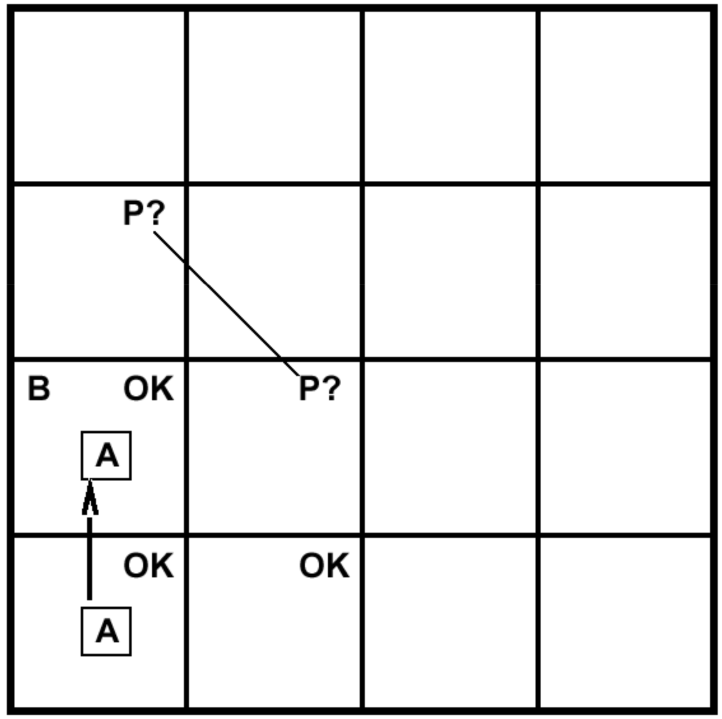
Exploring a Wumpus world



A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

9

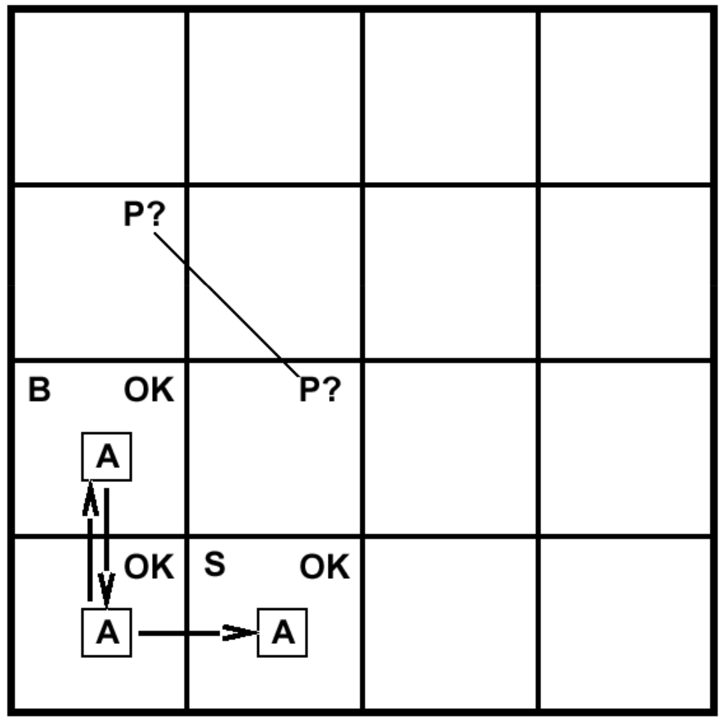
Exploring a Wumpus world



A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

10

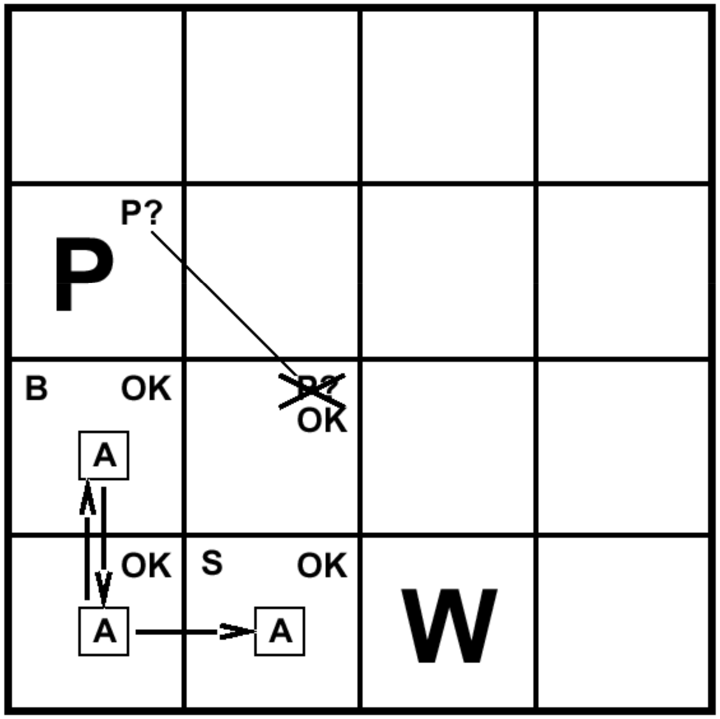
Exploring a Wumpus world



A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

11

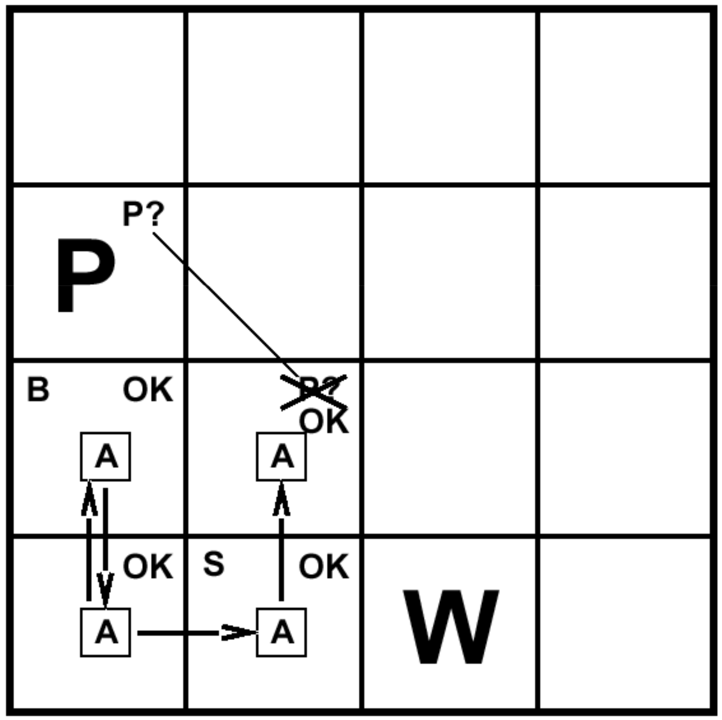
Exploring a Wumpus world



A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

12

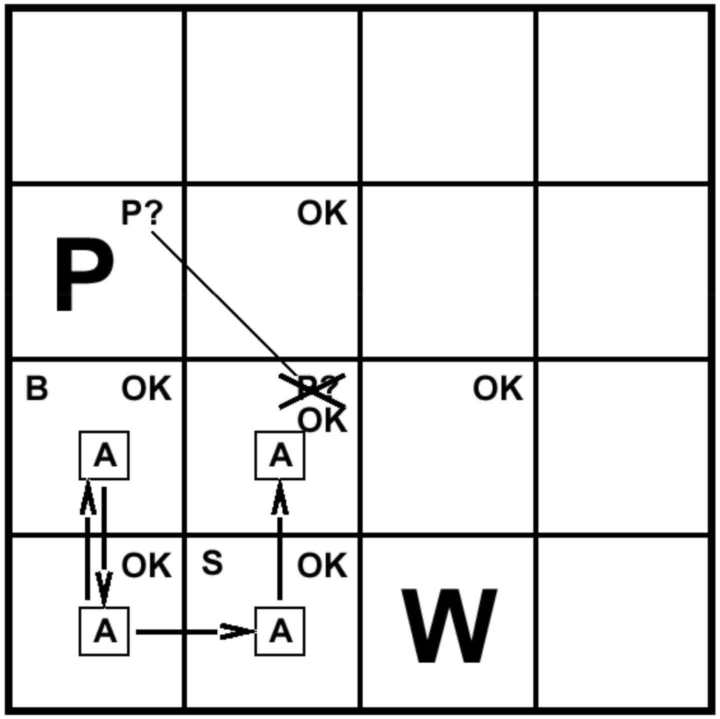
Exploring a Wumpus world



A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

13

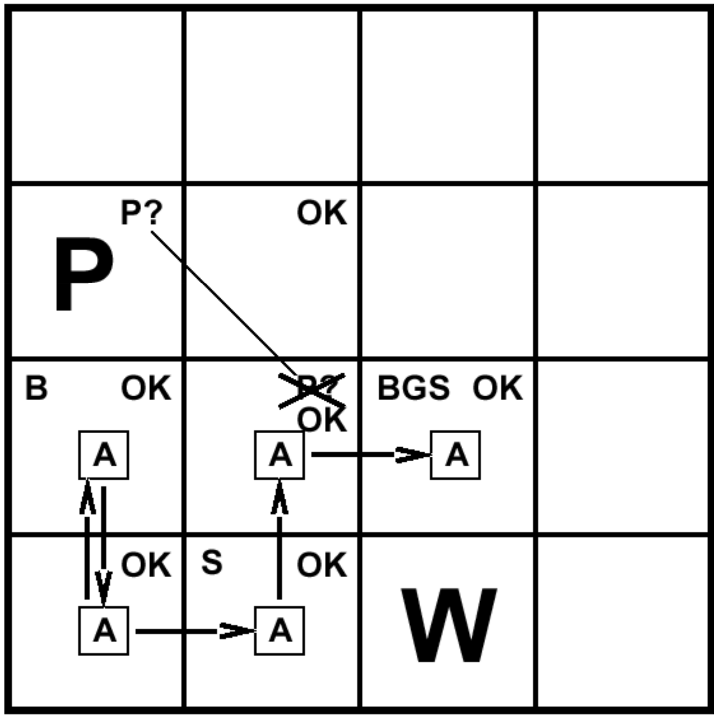
Exploring a Wumpus world



A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

14

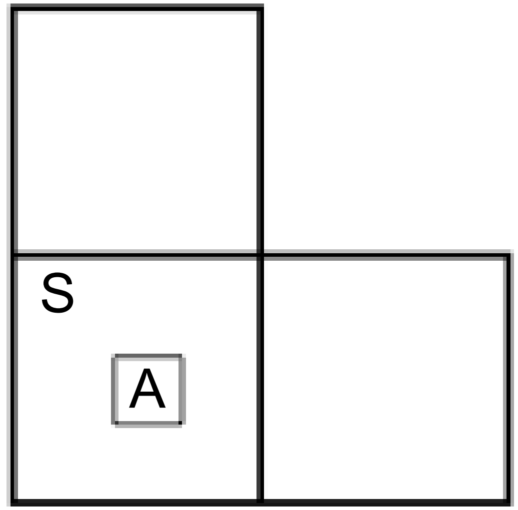
Exploring a Wumpus world



A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

15

Example: Getting Out of Tight Spots



Smell in (1,1) $\Rightarrow$ cannot move
Can use a strategy of coercion:

- Shoot straight ahead
- Wumpus was there $\Rightarrow$ dead $\Rightarrow$ safe
- Wumpus was not there $\Rightarrow$ safe

What knowledge representation supports this reasoning?

Logic

- *Logics* are formal languages for representing information such that conclusions can be drawn
- Logic has two components:
  - *Syntax* defines the sentences in the language
  - *Semantics* define the "meaning" of sentences
    I.e. define the truth of a sentence in a world
- E.g., the language of arithmetic

  $x + 2 \geq y$ is a sentence; $x2 + y >$ is not a sentence

  $x + 2 \geq y$ is true if and only if the number $x + 2$ is no less than the number y

  $x + 2 \geq y$ is true in a world where $x = 7,\ y = 1$
  $x + 2 \geq y$ is false in a world where $x = 0,\ y = 6$

Types of logic

- Logics are characterized by what they commit to as "primitives"
  - Ontological commitment: *what exists*—facts? objects? time? beliefs?
  - Epistemological commitment: *what states of knowledge*?

Language	*Ontological Commitment*	*Epistemological Comm.*
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief 0. . . 1
Fuzzy logic	degree of truth	degree of truth 0. . . 1

Interpretations

- We want to have a rule for generating (or testing) new sentences that are always true
- But the truth of a sentence may depend on its interpretation!
- Formally, an *interpretation* is a way of matching objects in the world with symbols in the sentence (or in the knowledge database)
- A sentence may be true in one interpretation and false in another
- Terminology:
  - A sentence is *valid* if it is true in all interpretations
  - A sentence is *satisfiable* if it is true in at least one interpretation
  - A sentence is *unsatisfiable* if it is false in all interpretations

Entailment and Inference

$$KB \models \alpha$$

- Knowledge base *KB* *entails* sentence α if and only if α is true in all worlds where *KB* is true.

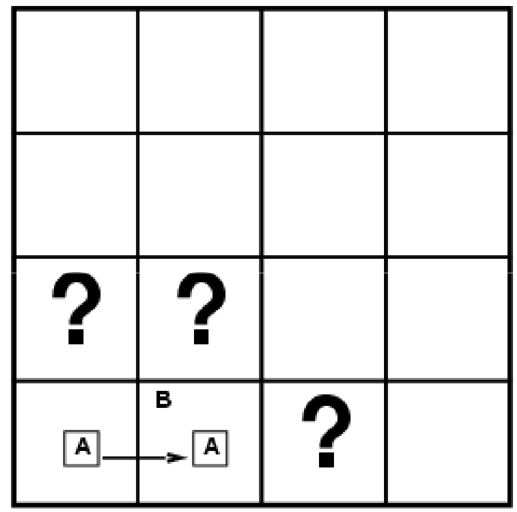
  E.g., the *KB* containing "I finished AI homework" and "I am happy" entails "I finished the AI homework or I am happy"
- *KB* $\vdash_i \alpha$ means sentence α can be derived from *KB* by inference procedure i
- Desired qualities of an inference procedure i:
  - *Soundness*: i is sound if, whenever *KB* $\vdash_i \alpha$, it is also true that *KB* $\models \alpha$. In other words, we infer *only necessary truths*
  - *Completeness*: i is complete if, whenever *KB* $\models \alpha$, it is also true that *KB* $\vdash_i \alpha$. In other words, we can generate *all the necessary truths*

Example: Entailment in wumpus world

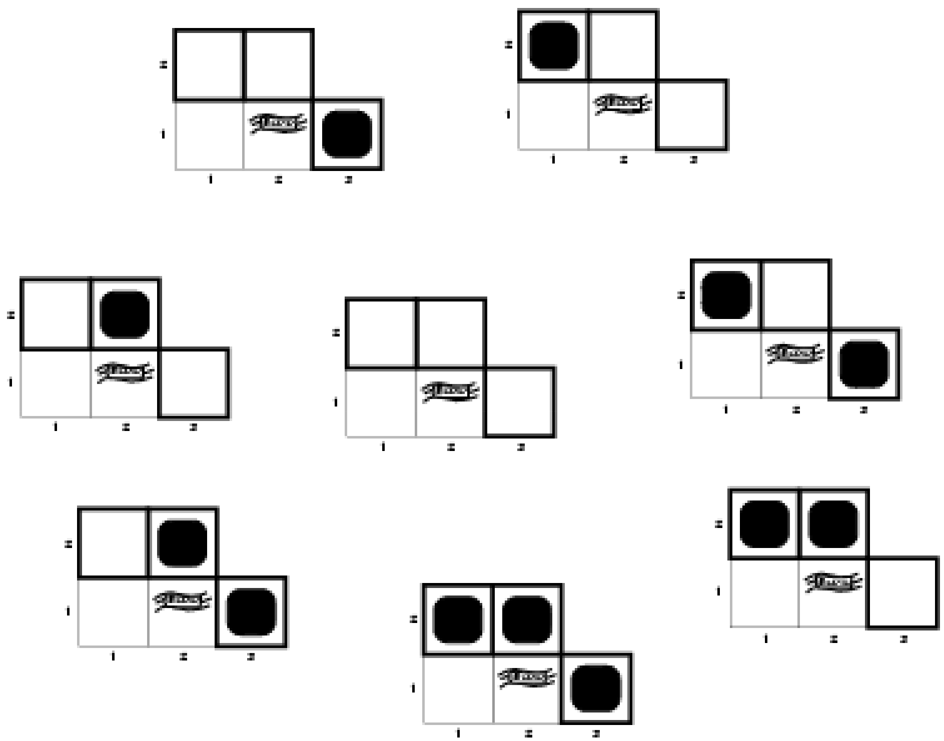
Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

Consider possible models for *KB* assuming only pits

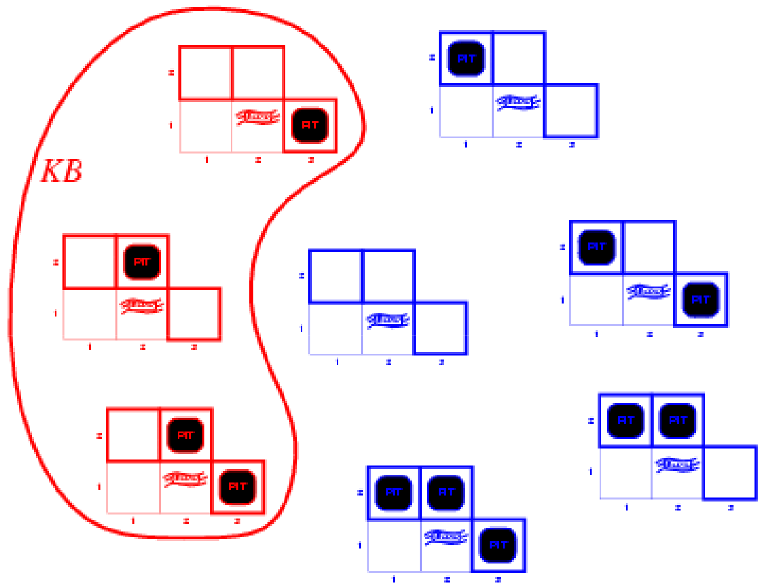
3 Boolean choices $\Rightarrow$ 8 possible models



Example: Models in wumpus world

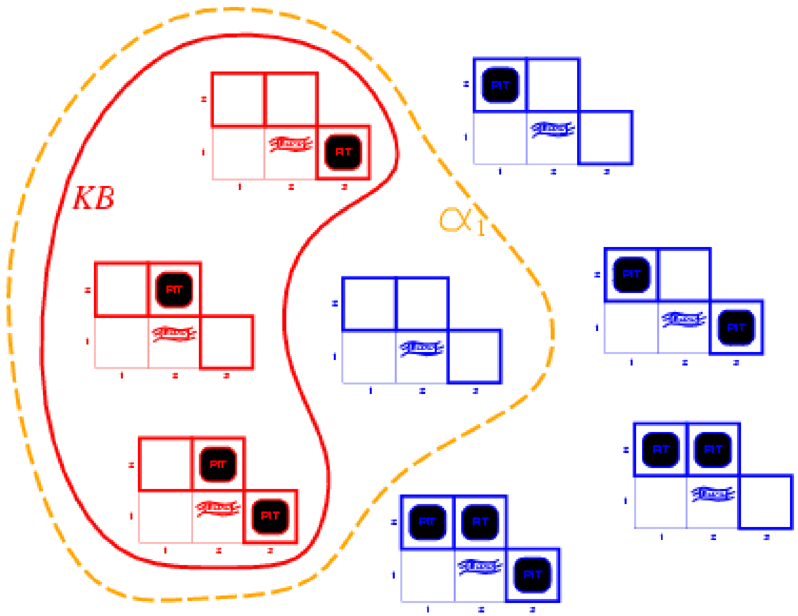


Example: Knowledge base in wumpus world



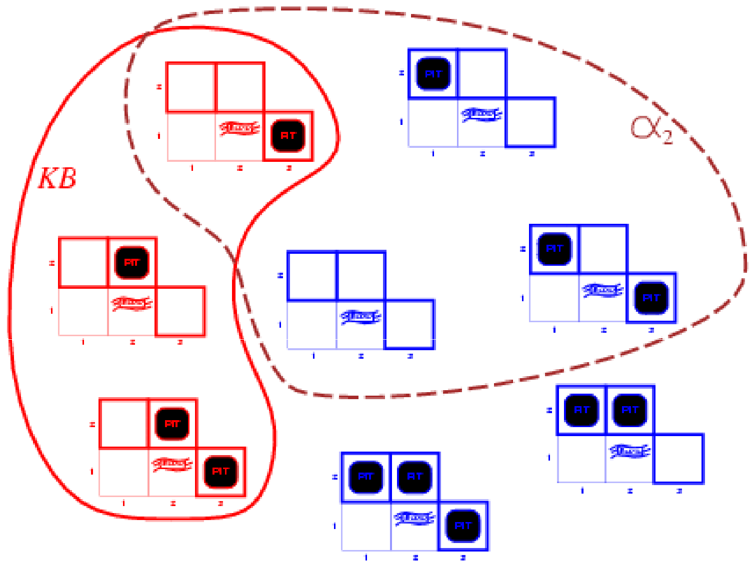
- *KB* = wumpus-world rules + observations

Example: Model checking in wumpus world



- KB = wumpus-world rules + observations
- α_1 = "[1,2] is safe", $KB \models \alpha_1$, proved by **model checking**
- Model Checking works only with FINITE worlds

Example: Model checking in wumpus world



- KB = wumpus-world rules + observations
- α_2 = "[2,2] is safe", KB does not entail α_2
- $KB \not\models \alpha_2$

Propositional Logic: Syntax

- Propositional logic is the simplest logic.
- Syntax rules:
  - Atomic symbols l_1, l_2 etc are sentences
  - If S is a sentence, $\neg S$ is a sentence
  - If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence
  - If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence
  - If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence
  - If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence

Propositional Logic: Semantics

- A *model* specifies true/false for each proposition symbol

E.g.	*A*	*B*	*C*
	True	*True*	*False*

  (Think of a model as a possible world in which the symbols can be evaluated)

- Rules for evaluating truth with respect to a model m:

$\neg S$	is true iff	S	is false		
$S_1 \wedge S_2$	is true iff	S_1	is true and	S_2	is true
$S_1 \vee S_2$	is true iff	S_1	is true or	S_2	is true
$S_1 \Rightarrow S_2$	is true iff	S_1	is false or	S_2	is true
i.e.,	is false iff	S_1	is true and	S_2	is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$	is true and	$S_2 \Rightarrow S_1$	is true

Validity and Satisfiability

- A sentence is *valid* if it is true in all models

  e.g., $A \vee \neg A$, $\quad A \Rightarrow A$, $\quad (A \wedge (A \Rightarrow B)) \Rightarrow B$

  Validity is connected to inference via the *Deduction Theorem*:

  KB $\models \alpha$ if and only if *KB* $\Rightarrow \alpha$ is valid

- A sentence is *satisfiable* if it is true in some model

  e.g., $A \vee B$, $\quad C$

- A sentence is *unsatisfiable* if it is true in no models

  e.g., $A \wedge \neg A$

- Satisfiability is connected to inference via the following:

  KB $\models \alpha$ if and only if *KB* $\wedge \neg\alpha$ is unsatisfiable

  This is proof by contradiction!

Two kinds of inference (proof) methods

- *Model checking*:
  - Truth table enumeration (sound and complete for propositional logic)
  - Heuristic search in model space (sound but incomplete)
- *Application of inference rules*:
  - Legitimate (sound) generation of new sentences from old
  - A *proof* is a sequence of inference rule applications
  - Inference rules can be used as operators in a standard search algorithm!

Propositional Inference: Truth Table Method

- Let $\alpha = A \vee B$ and $KB = (A \vee C) \wedge (B \vee \neg C)$
- Is it the case that $KB \models \alpha$?
- *Check all possible models* — α must be true wherever KB is true

A	B	C	$A \vee C$	$B \vee \neg C$	KB	α
False	*False*	*False*				
False	*False*	*True*				
False	*True*	*False*				
False	*True*	*True*				
True	*False*	*False*				
True	*False*	*True*				
True	*True*	*False*				
True	*True*	*True*				

Properties of truth table method

- The truth table method is a sound and complete inference method, which checks truth in all possible models.
- But the truth table method is very inefficient! 2^n models for n literals
- There must be a more efficient way to prove sentences...

Normal Forms

- Normal forms are standardized forms for writing sentences, which will be useful if we want to apply inference rules in a uniform way
- *Conjunctive Normal Form* (CNF—universal)

  conjunction of $\underbrace{\textit{disjunctions} \text{ of } \textit{literals}}$

  E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- *Disjunctive Normal Form* (DNF—universal)

  disjunction of $\underbrace{\textit{conjunctions} \text{ of } \textit{literals}}$

  E.g., $(A \wedge B) \vee (A \wedge \neg C) \vee (A \wedge \neg D) \vee (\neg B \wedge \neg C) \vee (\neg B \wedge \neg D)$
- *Horn Form* (restricted)

  conjunction of *Horn clauses* (clauses with ≤ 1 positive literal) E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

  Often written as set of implications: $B \Rightarrow A$ and $(C \wedge D) \Rightarrow B$

Inference rules for propositional logic

- *Resolution* (for CNF): complete for propositional logic

$$\frac{\alpha \vee \beta, \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

- *Modus Ponens* (for Horn Form): complete for Horn KBs

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

Can be used with *forward chaining* or *backward chaining*

Forward chaining

- When a new sentence p is added to the KB
  - Look for all sentences that share literals with p
  - Perform resolution
  - Add new sentences to the KB and continue
- Two important properties
  - Forward chaining is *data-driven*
    E.g., inferring properties and categories from new percepts
  - Forward chaining is an *eager* method: new facts are inferred as soon as possible

Backward chaining

- When a query q is asked:
  - If q is in the knowledge base, return true
  - Else use resolution for q with other sentences in KB, and continue from the result
- Two important properties:
  - Backward chaining is *goal-driven*: it centers the reasoning around the query begin asked
  - It is a *lazy* reasoning method: new facts are only inferred as needed, and only to the extent that they help answer the query.

Forward vs backward chaining: Which one is better?

- It depends on the problem at hand!
- Backward chaining is parsimonious in the amount of computation performed, and does not grow the knowledge base as much as forward chaining
- Backward chaining is focused on the proof that needs to be generated, so is generally more efficient
- But it does nothing until questions are asked!
- Backward chaining is usually used in proof by contradiction
- Forward chaining extends the knowledge base, and hence improves the understanding of the world
- Typically, backward chaining is used in proofs by contradiction
- Forward chaining is used in tasks where the focus is not on producing a proof, but on providing a model of the world

Other useful rules

- *And-elimination:*

$$\frac{\alpha_1 \wedge \cdots \wedge \alpha_n}{\alpha_i, \forall i = 1, \ldots n}$$

- *Implication elimination:*

$$\frac{\alpha \Rightarrow \beta}{\neg\alpha \vee \beta}$$

Example

- Knowledge base
  - HaveAILecture $\Rightarrow$ (TodayIsMonday $\vee$ TodayIsWednesday)
  - $\neg$ TodayIsMonday
  - HaveAILecture $\vee$ HaveNoClass
  - HaveNoClass $\Rightarrow$ Sad
  - $\neg$ Sad
- Can you infer what day it is?

Complexity of Inference

- What is the complexity of verifying the validity of a sentence of n literals?
  2^n
- What if our knowledge is expressed only in terms of Horn clauses?

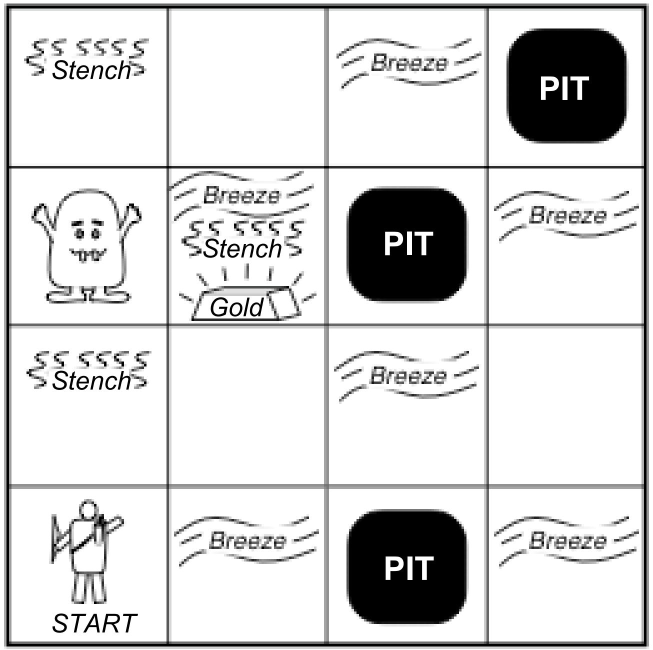
  The inference time becomes polynomial!
  - Every Horn clause establishes exactly one new fact
  - We can add all the new facts implied by the database in n passes

  This is why Horn clauses are often used in expert systems

An Example: Wumpus World

- Knowledge base:

$$\neg S_{1,1} \quad \neg S_{2,1} \quad S_{1,2} \quad \neg B_{1,1} \quad B_{2,1} \quad \neg B_{1,2}$$



- Knowledge about the environment:

$$\neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}$$

$$\neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,2} \wedge \neg W_{2,1} \wedge \neg W_{3,1}$$

$$S_{1,2} \Rightarrow W_{1,1} \vee W_{1,2} \vee W_{2,2} \vee W_{1,3}$$

- Now we can use inference rules to find out where the Wumpus is!

Summary of Propositional Logic

- The good: Propositional logic is very simple!
  Few rules, inference is simple
- The bad: Propositional logic is very simple!
  So we cannot express things in a compact way
- E.g for the wumpus world, we need propositions for ALL positions, AND ALL TIMES!
- We cannot say things like "for all squares" or "the wumpus is in one of the neighboring squares"

Planning with propositional logic

- A planning problem is described just like a search problem (states, actions/operators, goal), but the problem representation is more structured:

	Search	*Planning*
States	Data structures	Logical sentences
Actions	Code	Preconditions/outcomes
Goal	Goal test	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

- Natural idea: represent states using propositions, and use logical inference (forward / backward chaining) to find sequences of actions.

Planning as Satisfiability: SatPlan

- Intorduced by Kautz and Selman, 1990s, very successful method over the years
- Take a description of a planning problem and generate all possible literals, at all time slices
- Generate a humongous SAT problem
- Use a state-of-art SAT solver (eg WalkSAT) to get a plan
- Randomized SAT solvers can be used as well

Complexity of planning

- Clearly NP-hard (as it can be seen as SAT in finite-length plan case)
- But actually worse (PSPACE) if we let plan duration

GraphPlan

- Introduced by Blum and Furst in 1995, currently the state-of-art in planning algorithms
- Main idea:
  - Construct a graph that encodes constraints on possible plans
  - If a valid plan exists it will be part of this planning graph, so search only within this graph
- Planning graph can be built in polynomial time.

Problem description

- Goal is described in conjunctive form
- The preconditions of actions have to be conjunctions
- Usually operators are described in STRIPS-like notation

Planning Graph

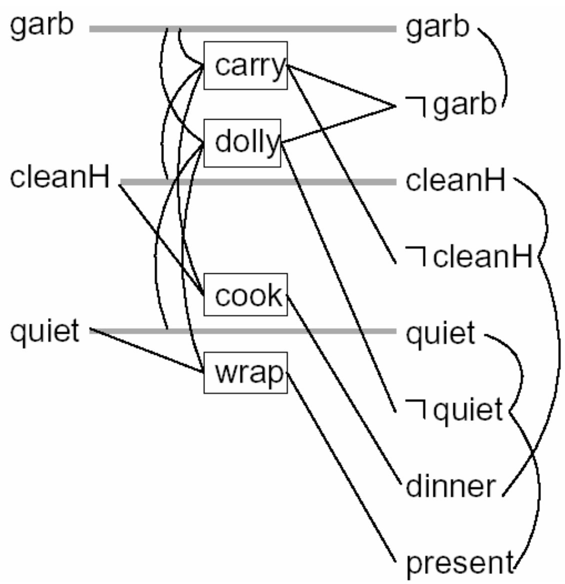
- Two types of nodes:
  - Propositions
  - Actions

  These are arranged in *levels*: propositions and action levels alternate
- Three types of edges between levels:
  - Precondition: edge from P to A if P is a precondition of A
  - Add: edge from A to P if A has P as effect
  - Delete: edge from A to $\neg P$ if A deletes P
- Action level includes actions whose preconditions are satisfied in the previous level, plus "no-op" actions (do nothing)

Example: Dinner Date

- Initial state: garbage $\wedge$ cleanHands $\wedge$ quiet
- Goal state: dinner $\wedge$ present $\wedge \neg$garbage
- Actions:
  - Cook: precondition: cleanHands; effect: dinner
  - Wrap: precondition: quiet; postcondition: present
  - Carry: effect: $\neg$garbage $\wedge$ $\neg$cleanHands
  - Dolly: effect: $\neg$garbage $\wedge$ $\neg$quiet

Example: First Level of Planning Graph

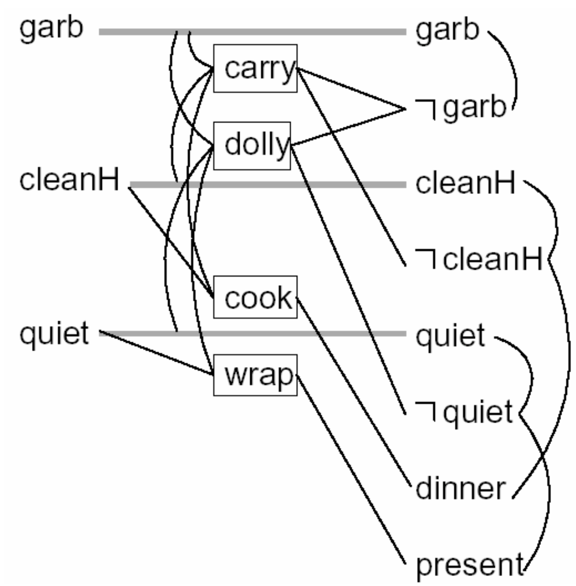


- Thicker lines correspond to doing nothing
- Action level contains *all* actions whose preconditions are satisfied
- Edges between nodes on same level indicate *mutual exclusion*

Mutual exclusion

- Two actions are *mutually exclusive (mutex)* at some stage if no valid plan could contain both at that stage
- Two actions at the same level can be mutex because of:
  - *Inconsistent effects*: an effect of one negates the effect of the other
  - *Interference*: one negates a precondition of the other
  - *Competing needs*: the actions have mutex preconditions
- Two propositions at the same level are mutex if:
  - One is a *negation* of the other
  - *Inconsistent support*: All ways of achieving the propositions are pairwise mutex

Example: Mutual Exclusions

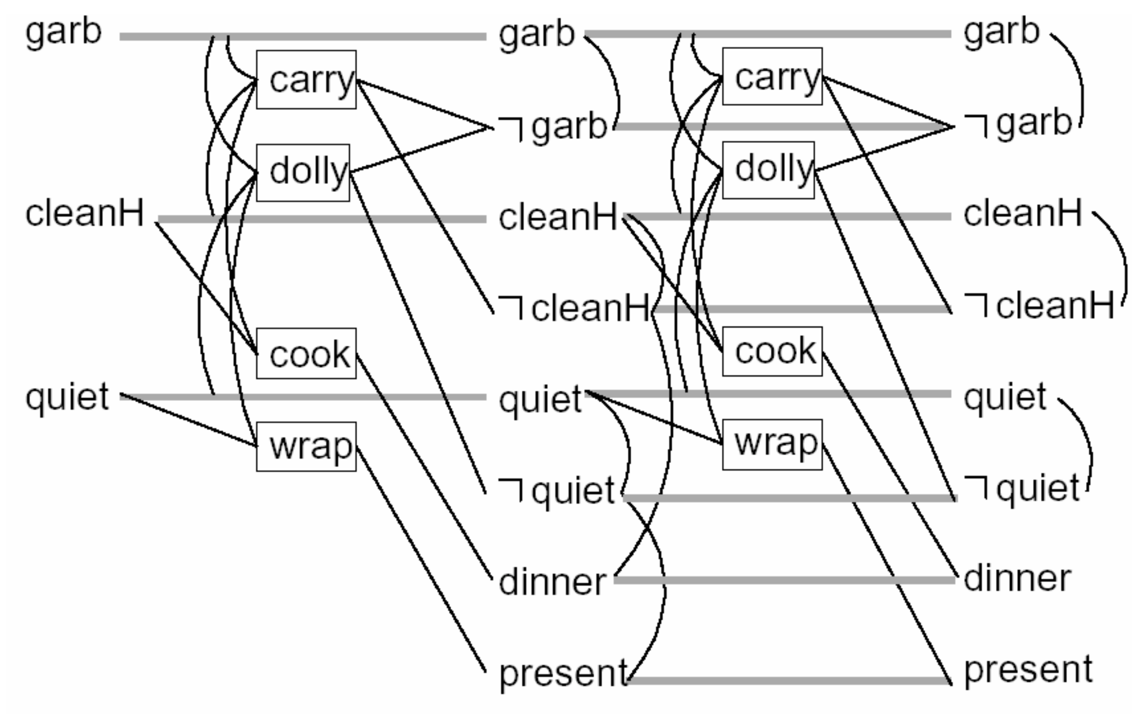


- wrap and dolly are mutex because dolly negates the precondition of wrap
- carry and the no-op are mutex because one negates the effect of the other
- present and ¬quiet are mutex because the actions achieving them, wrap and dolly, are mutex

Constructing the Planning Graph

- Level P_1 is initialized with all the literals from the initial state
- Add an action at level A_i if all its preconditions are present in level P_i
- Add a proposition in level P_{i+1} if it is the effect of some action in level A_i (including no-ops)
- Maintain a set of exclusion relations to eliminate incompatible propositions and actions

Example: Two-level Planning Graph



Observations

- Number of propositions always increases (because all the ones from the previous level are carried forward)
- Number of actions always increases (because the number of preconditions that are satisfied increases)
- Number of propositions that are mutex decreases (because there are more ways to achieve same propositions, and not all will be mutex)
- Number of actions that are mutex decreases (because of the decrease in the mutexes between actions)
- After some time, all levels become identical: graph "levels off"
- Because there is a finite number of propositions and actions, mutexes will not reappear

Valid Plan

- A valid plan is a subgraph of the planning graph such that:
  - All goal propositions are satisfied in the last level
  - No goal propositions are mutex
  - Actions at the same level are not mutex
  - Each action's preconditions are made true by the plan
- Algorithm:
  1. Grow the planning graph until all goal propositions are reachable and not mutex
  2. If the graph levels off first, return failure (no valid plan exists)
  3. Search the graph for a planning graph
  4. If no valid plan is found add a level and try again

Example: Plan extraction