

Lecture 7: Knowledge 1

Previously...



Basic ability: search

Path-based search

Iterative-improvement search

Adversarial search

Constraint satisfaction problems

Knowledge

Knowledge

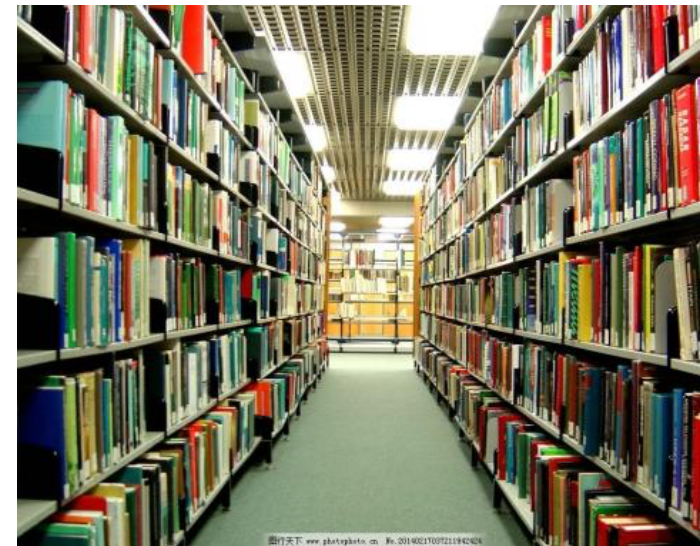


From Wikipedia, the free encyclopedia

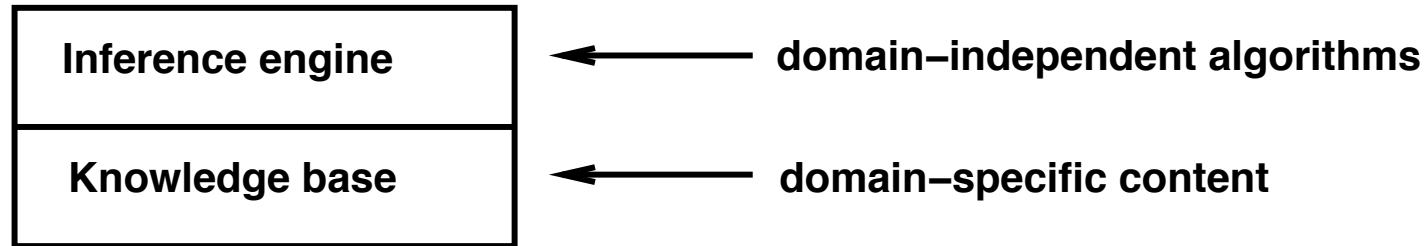
For other uses, see [Knowledge \(disambiguation\)](#).

Knowledge is a familiarity, awareness, or understanding of someone or something, such as [facts](#), [information](#), [descriptions](#), or [skills](#), which is acquired through [experience](#) or [education](#) by [perceiving](#), [discovering](#), or [learning](#).

Knowledge can refer to a [theoretical](#) or [practical](#) understanding of a subject. It can be implicit (as with practical skill or expertise) or explicit (as with the theoretical understanding of a subject); it can be more or less formal or systematic.^[1] In [philosophy](#), the study of knowledge is called [epistemology](#); the philosopher [Plato](#) famously defined knowledge as "[justified true belief](#)", though this definition is now thought by some analytic philosophers^[*citation needed*] to be problematic because of the [Gettier problems](#) while others defend the platonic definition.^[2] However, several definitions of knowledge and theories to explain it exist.



Knowledge bases



Knowledge base = set of sentences in a **formal** language

Declarative approach to building an agent (or other system):

TELL it what it needs to know

Then it can **ASK** itself what to do—answers should follow from the KB

Agents can be viewed at the **knowledge level**

i.e., **what they know**, regardless of how implemented

Or at the **implementation level**

i.e., data structures in KB and algorithms that manipulate them

A simple knowledge-based agent



```
function KB-AGENT(percept) returns an action  
  static: KB, a knowledge base  
           t, a counter, initially 0, indicating time  
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))  
  action ← ASK(KB, MAKE-ACTION-QUERY(t))  
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))  
  t ← t + 1  
  return action
```

The agent must be able to:

- Represent states, actions, etc.

- Incorporate new percepts

- Update internal representations of the world

- Deduce hidden properties of the world

- Deduce appropriate actions

Wumpus World PEAS description

Performance measure

gold +1000, death -1000

-1 per step, -10 for using the arrow

Environment

Squares adjacent to wumpus are smelly

Squares adjacent to pit are breezy

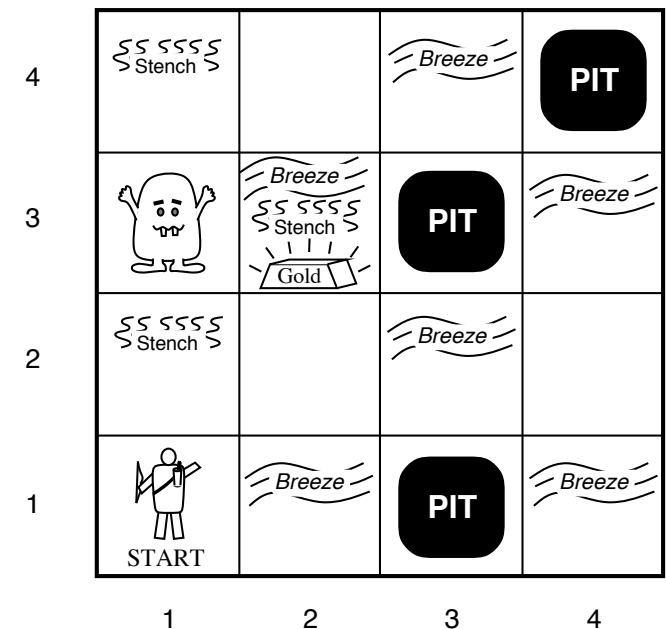
Glitter iff gold is in the same square

Shooting kills wumpus if you are facing it

Shooting uses up the only arrow

Grabbing picks up gold if in same square

Releasing drops the gold in same square



Actuators Left turn, Right turn,

Forward, Grab, Release, Shoot

Sensors Breeze, Glitter, Smell

Wumpus world characterization

Observable?? No—only **local** perception

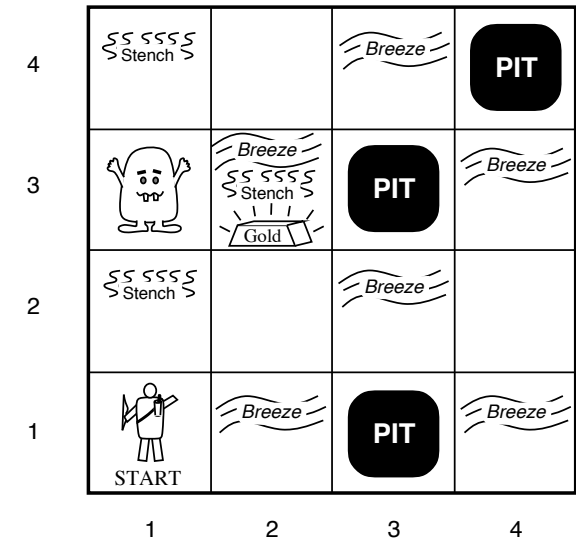
Deterministic?? Yes—outcomes exactly specified

Episodic?? No—sequential at the level of actions

Static?? Yes—Wumpus and Pits do not move

Discrete?? Yes

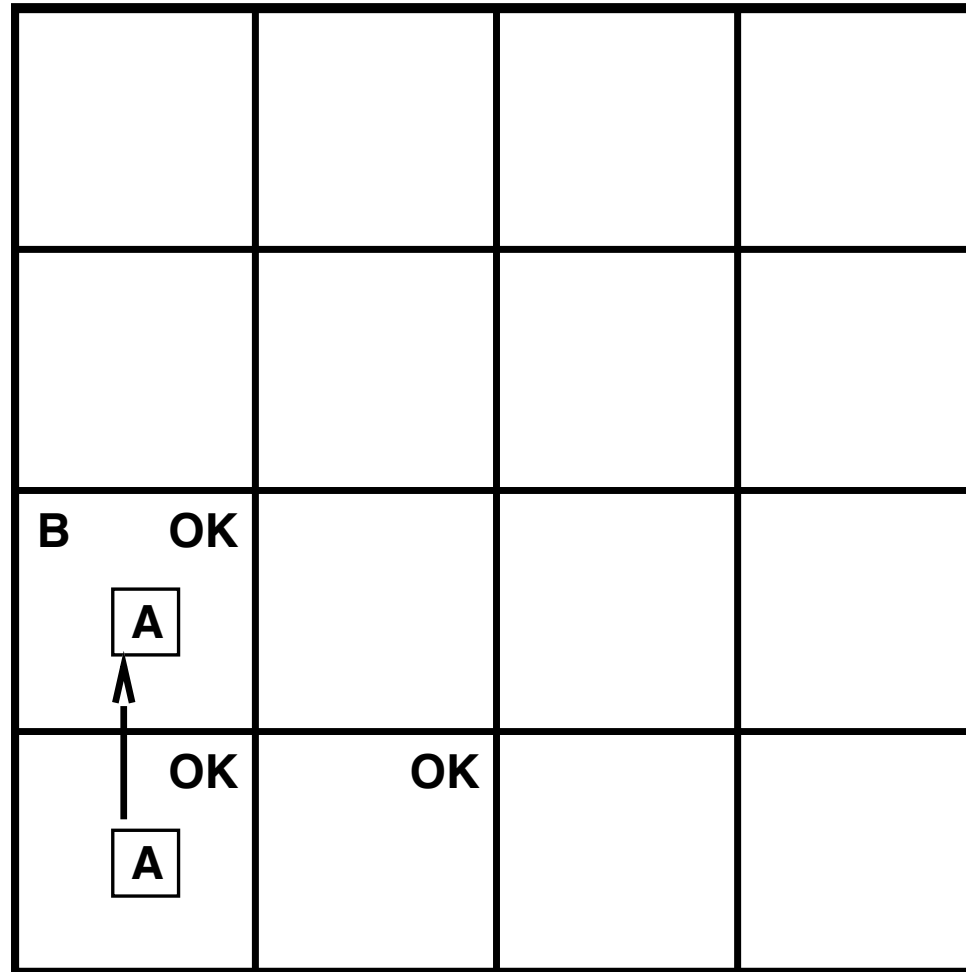
Single-agent?? Yes—Wumpus is essentially a natural feature



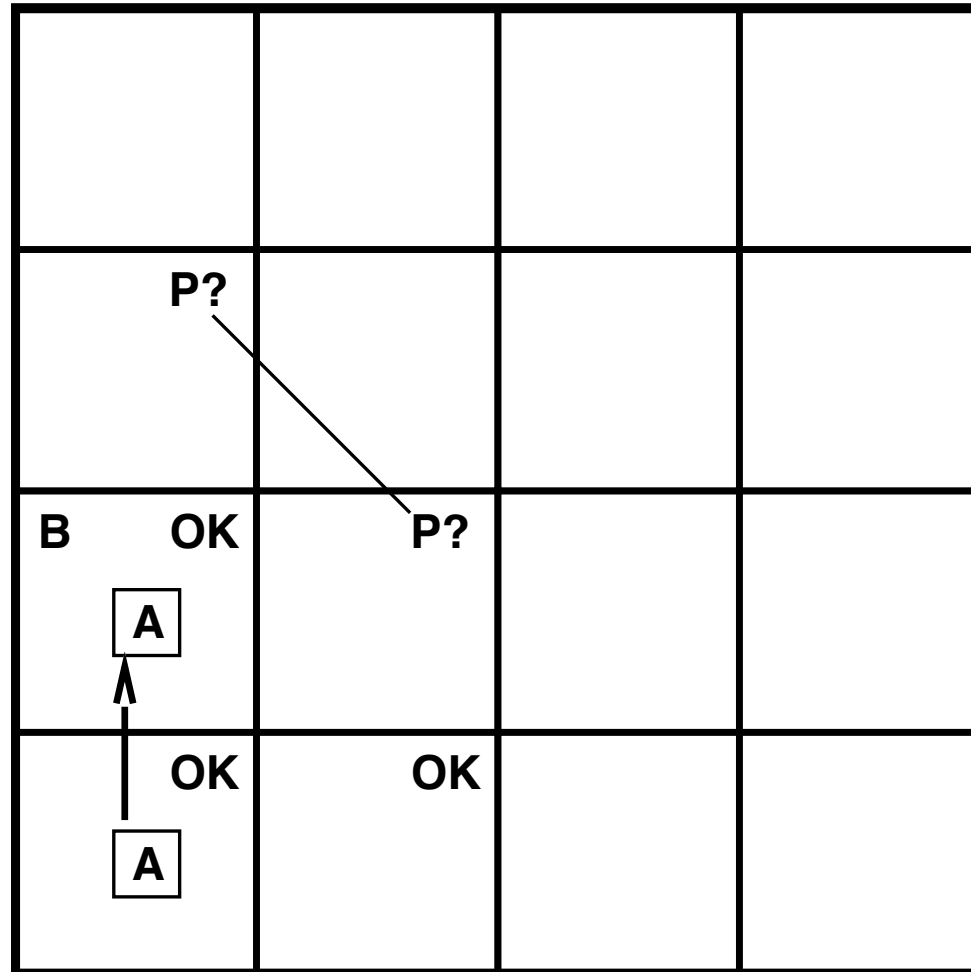
Exploring a wumpus world

OK			
OK A	OK		

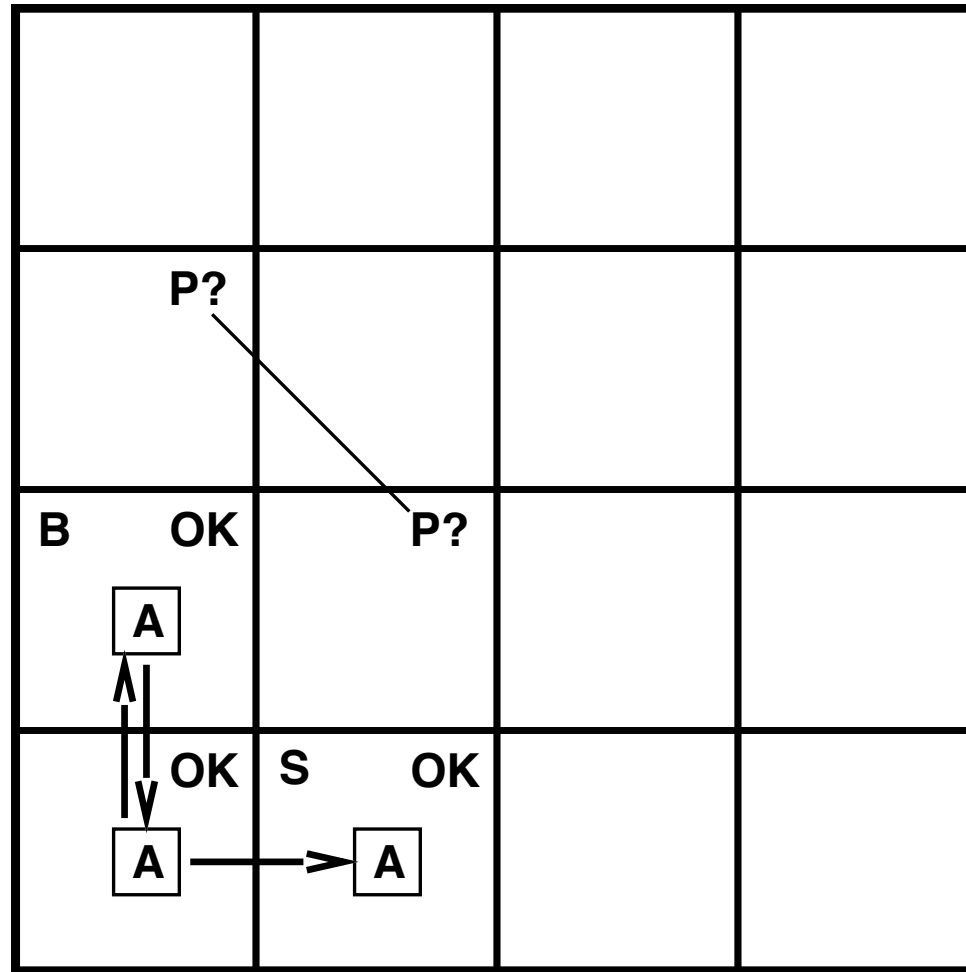
Exploring a wumpus world



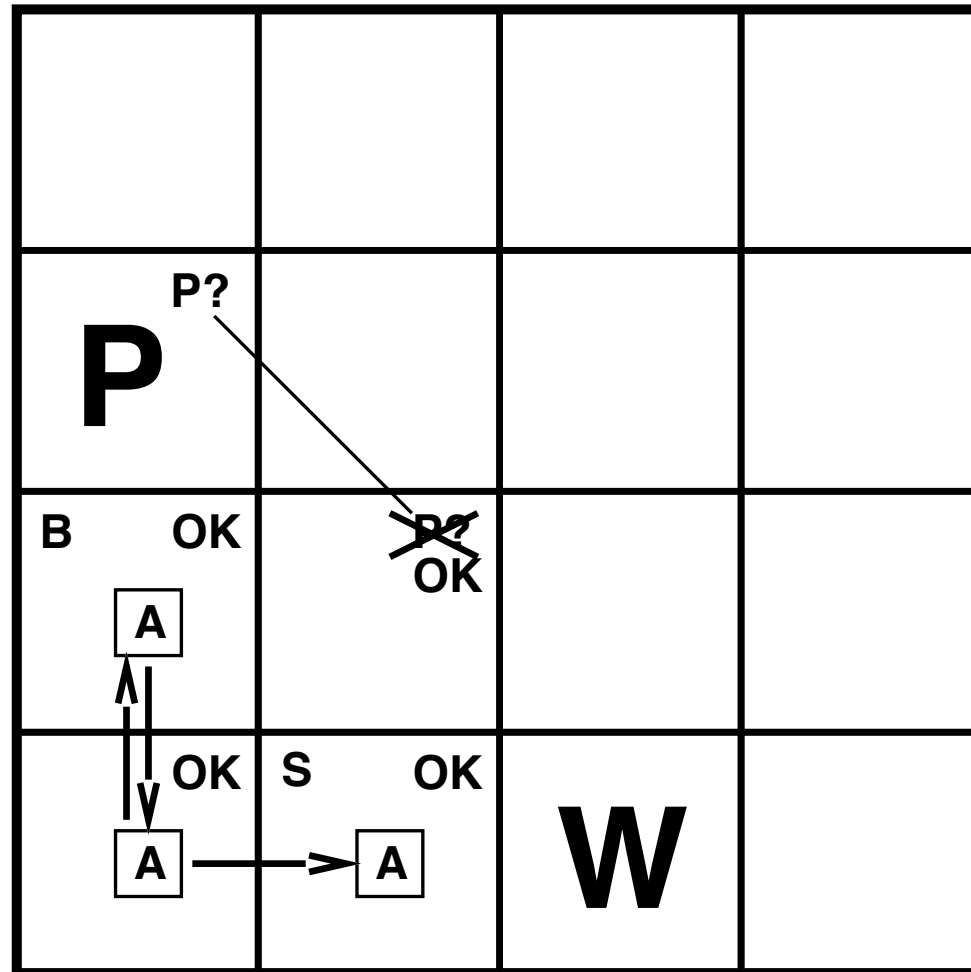
Exploring a wumpus world



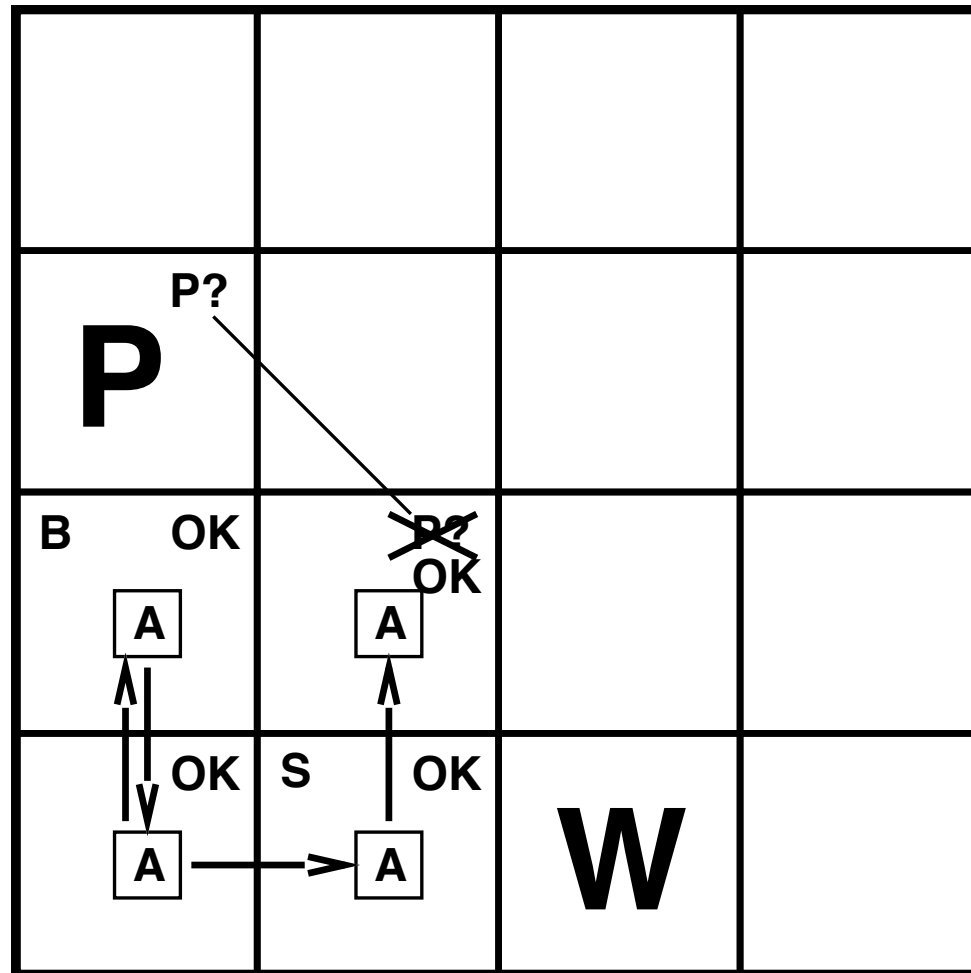
Exploring a wumpus world



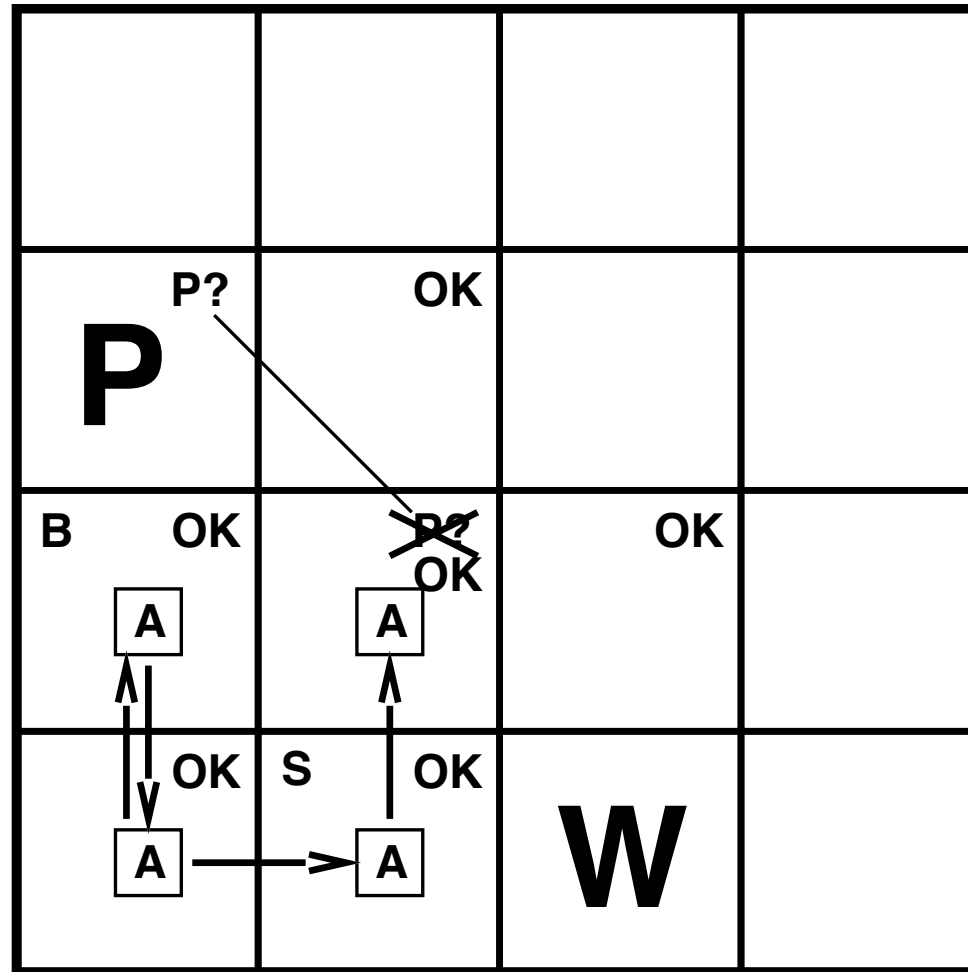
Exploring a wumpus world



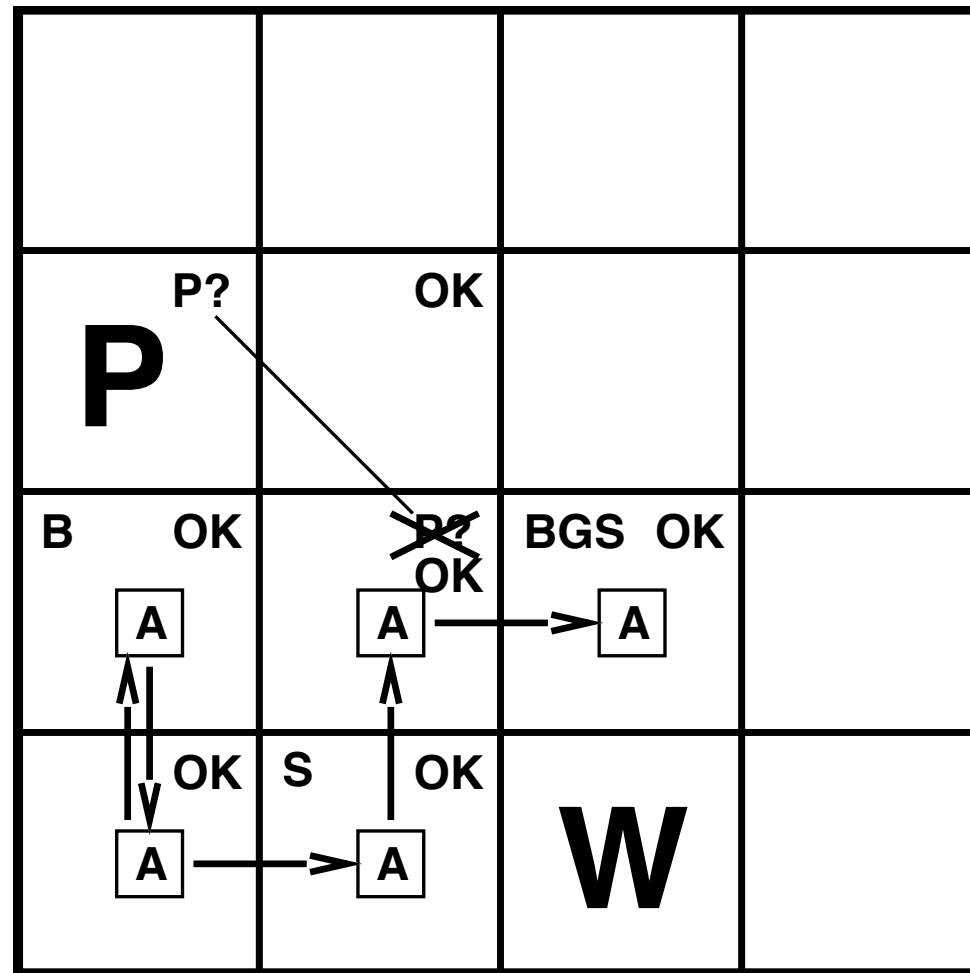
Exploring a wumpus world



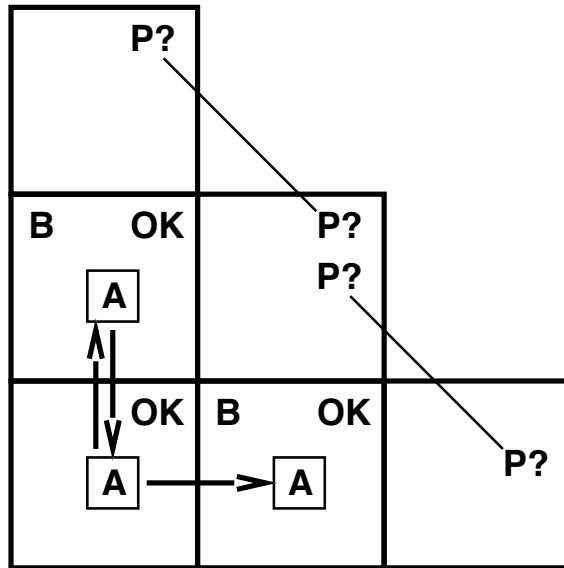
Exploring a wumpus world



Exploring a wumpus world

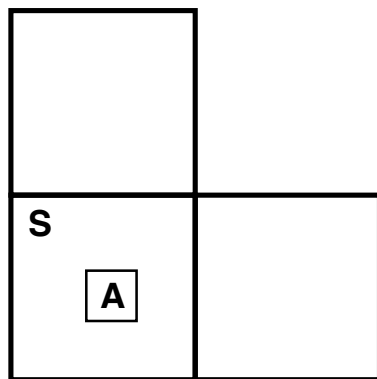


Other tight spots



Breeze in (1,2) and (2,1)
 \Rightarrow no safe actions

Assuming pits uniformly distributed,
 (2,2) has pit w/ prob 0.86, vs. 0.31



Smell in (1,1)

\Rightarrow cannot move

Can use a strategy of **coercion**:

shoot straight ahead

wumpus was there \Rightarrow dead \Rightarrow safe

wumpus wasn't there \Rightarrow safe

Logic in general



Logics are formal languages for representing information such that conclusions can be drawn

Syntax defines the sentences in the language

Semantics define the “meaning” of sentences;
i.e., define truth of a sentence in a world

E.g., the language of arithmetic

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

$x + 2 \geq y$ is true iff 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 $x = 0$, $y = 6$

Entailment (蕴涵 / 蕴含)

Entailment means that one thing **follows from** another:

$$KB \models \alpha$$

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 “the Giants won” and “the Reds won”
entails “Either the Giants won or the Reds won”

E.g., $x + y = 4$ entails $4 = x + y$

Entailment is a relationship between sentences (i.e., **syntax**)
that is based on **semantics**

Note: brains process **syntax** (of some sort)

Model (模型)

Logicians typically think in terms of **models**, which are formally structured worlds with respect to which truth can be evaluated

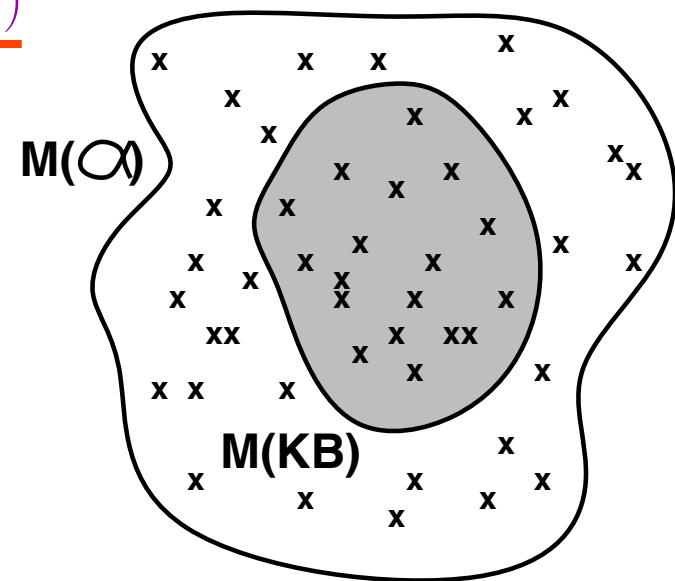
We say m is a **model** of a sentence α if α is true in m

$M(\alpha)$ is the set of all models of α

Then $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

E.g. $KB =$ Giants won and Reds won

$\alpha =$ Giants won

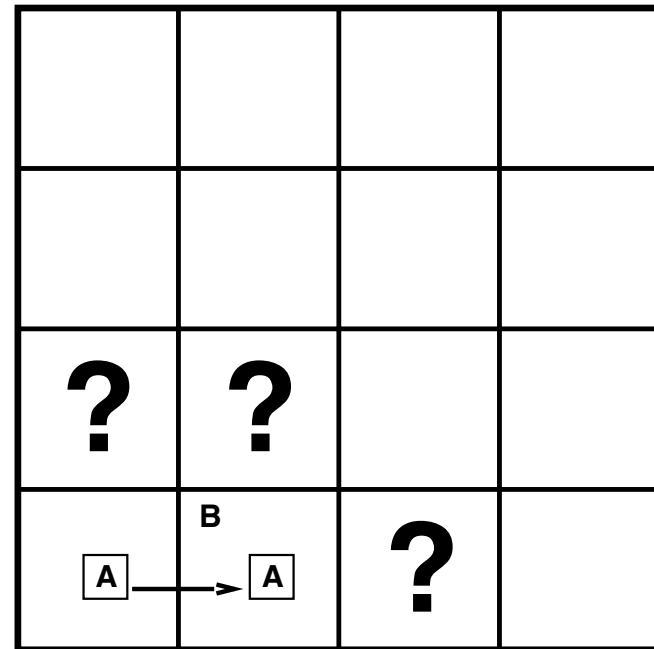


Entailment in the wumpus world

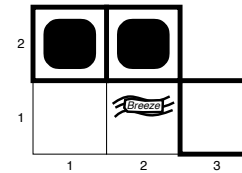
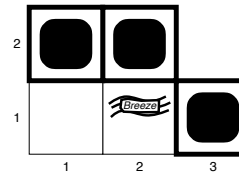
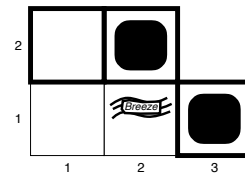
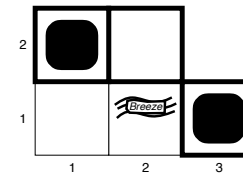
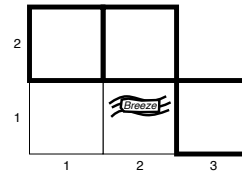
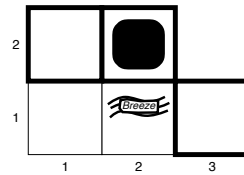
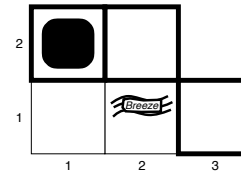
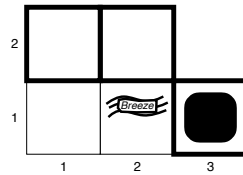
Situation after detecting nothing in [1,1],
moving right, breeze in [2,1]

Consider possible models for ?s
assuming only pits

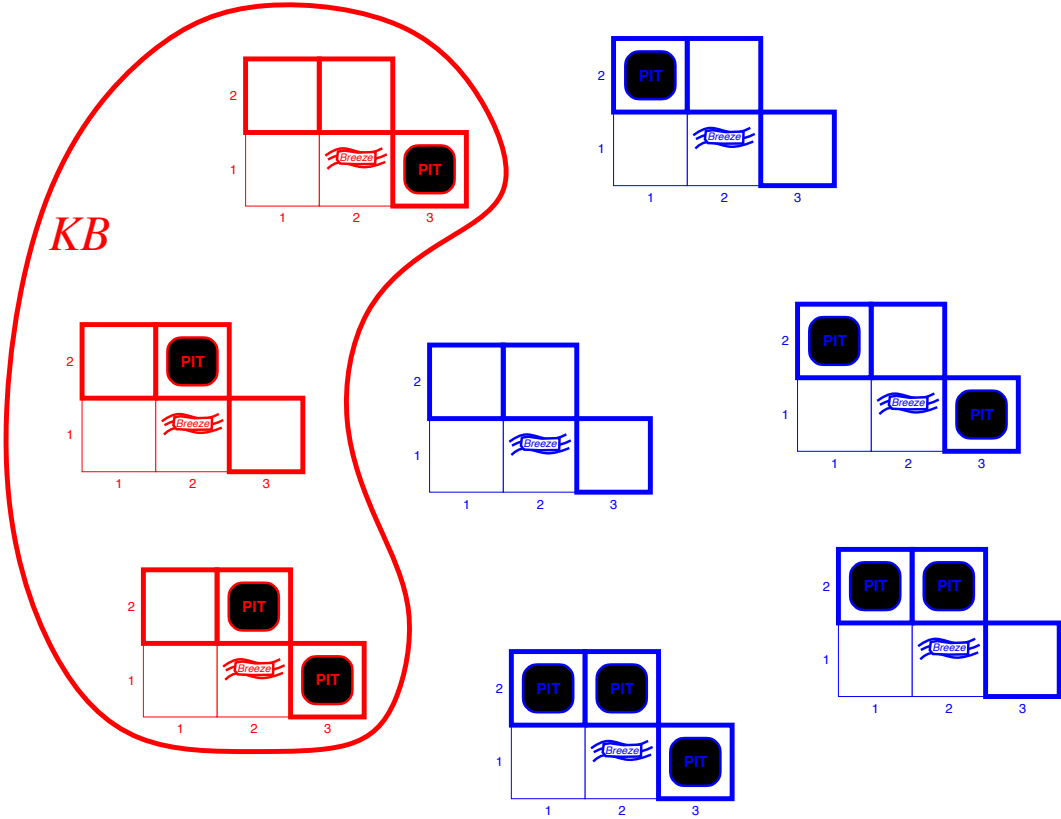
3 Boolean choices \Rightarrow 8 possible models



Wumpus models

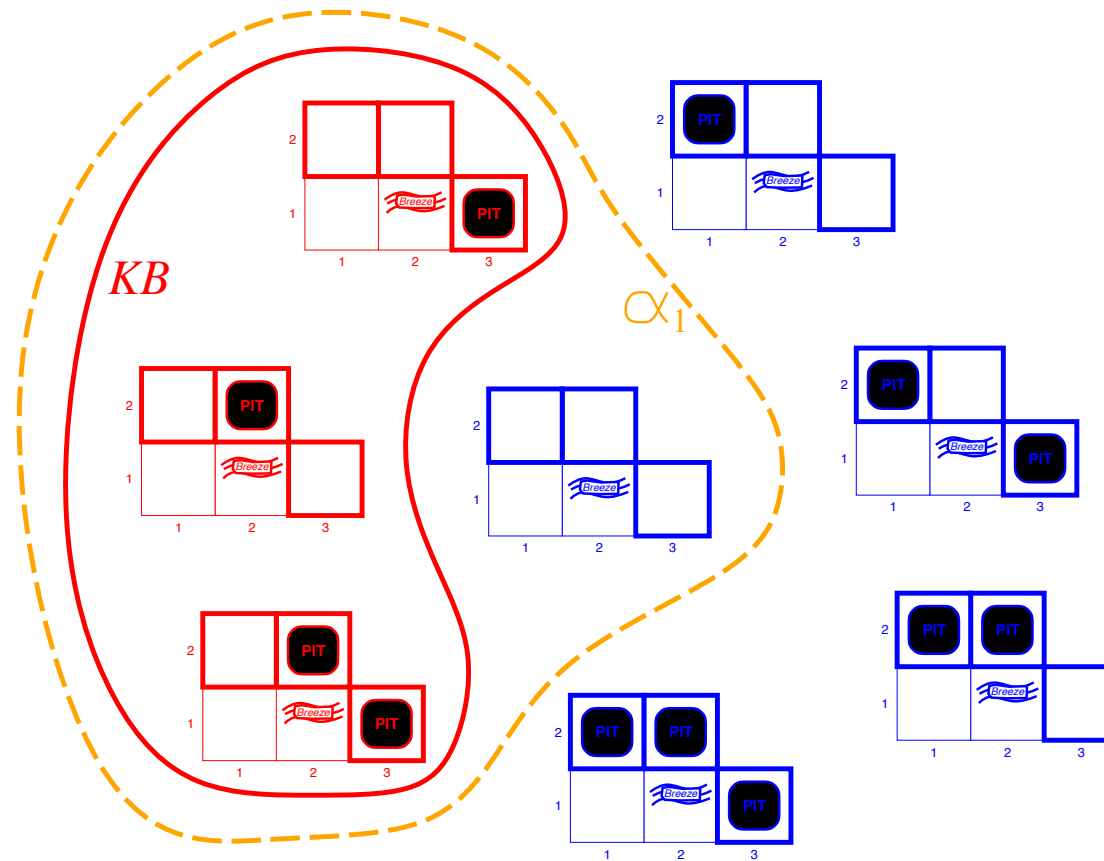


Wumpus models



KB = wumpus-world rules + observations

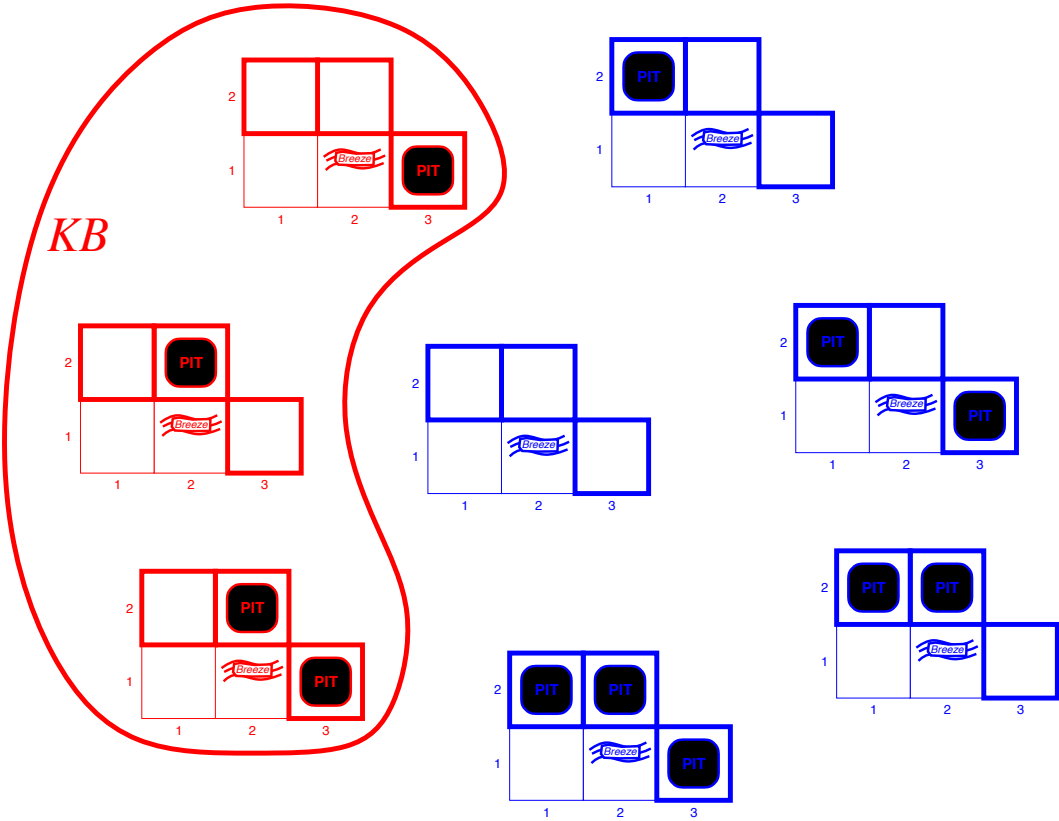
Wumpus models



KB = wumpus-world rules + observations

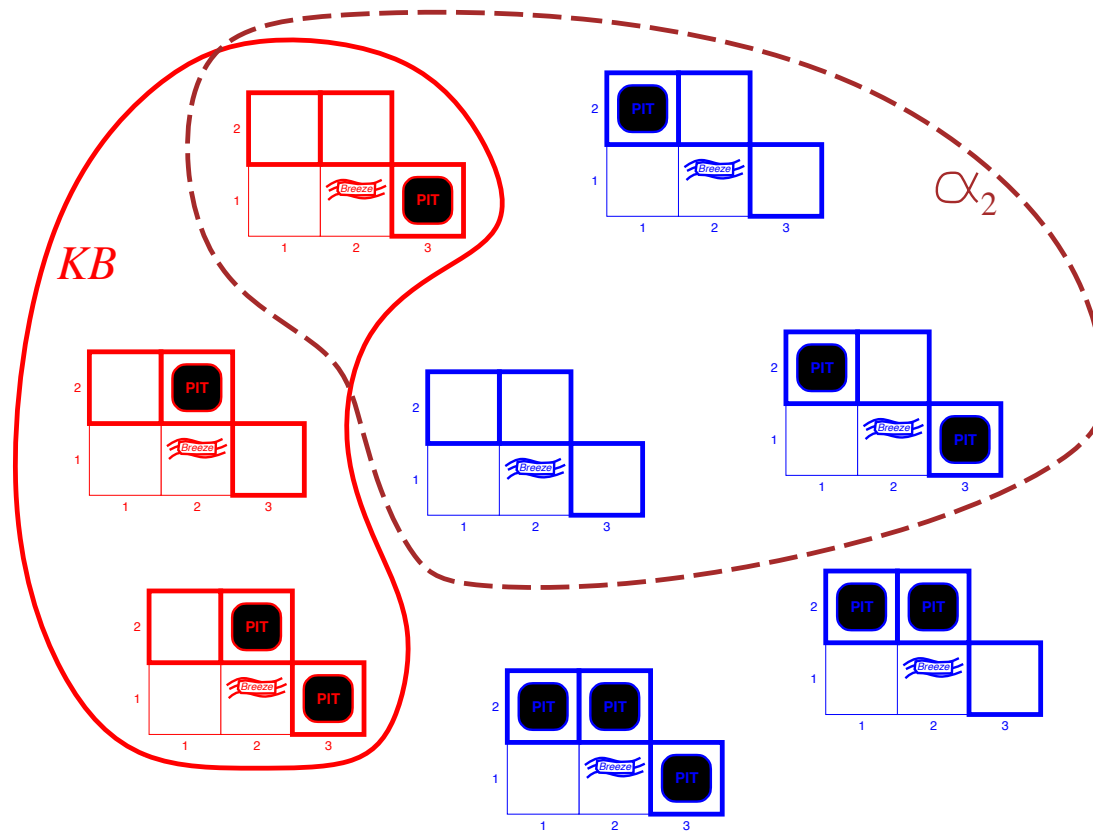
α_1 = “[1,2] is safe”, $KB \models \alpha_1$, proved by model checking

Wumpus models



KB = wumpus-world rules + observations

Wumpus models



KB = wumpus-world rules + observations

α_2 = "[2,2] is safe", $KB \not\models \alpha_2$

Inference



$KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i

Consequences of KB are a haystack; α is a needle.

Entailment = needle in haystack; inference = finding it

Soundness: i is sound if

whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness: i is complete if

whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.

That is, the procedure will answer any question whose answer follows from what is known by the KB .

Propositional logic

Propositional logic (命题逻辑): Syntax



Propositional logic is the simplest logic—illustrates basic ideas

The proposition symbols P_1, P_2 etc are sentences

If S is a sentence, $\neg S$ is a sentence (negation)

If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)

If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)

If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)

If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional logic: Semantics



Each model specifies true/false for each proposition symbol

E.g. $P_{1,2}$ $P_{2,2}$ $P_{3,1}$
true true false

(With these symbols, 8 possible models, can be enumerated automatically.)

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

Simple recursive process evaluates an arbitrary sentence, e.g.,

$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = \textit{true} \wedge (\textit{false} \vee \textit{true}) = \textit{true} \wedge \textit{true} = \textit{true}$

Truth tables for connectives (真值表)

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>
<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>
<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>

Wumpus world sentences



Let $P_{i,j}$ be true if there is a pit in $[i, j]$.

Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

$$\neg P_{1,1}$$

$$\neg B_{1,1}$$

$$B_{2,1}$$

“Pits cause breezes in adjacent squares”

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

“A square is breezy **if and only if** there is an adjacent pit”

Truth tables for inference



$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	false	false	false	false	false	false	true	true	true	true	false	false
false	false	false	false	false	false	true	true	true	false	true	false	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	<u>true</u>
false	true	false	false	false	true	false	true	true	true	true	true	<u>true</u>
false	true	false	false	false	true	true	true	true	true	true	true	<u>true</u>
false	true	false	false	true	false	false	true	false	false	true	true	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	true	true	true	true	true	true	false	true	true	false	true	false

Enumerate rows (different assignments to symbols),
 if **KB** is true in row, check that α is too

Inference by enumeration



Depth-first enumeration of all models is sound and complete

function **TT-ENTAILS?**(KB, α) **returns** *true* or *false*

inputs: KB , the knowledge base, a sentence in propositional logic

α , the query, a sentence in propositional logic

$symbols \leftarrow$ a list of the proposition symbols in KB and α

return **TT-CHECK-ALL**($KB, \alpha, symbols, []$)

function **TT-CHECK-ALL**($KB, \alpha, symbols, model$) **returns** *true* or *false*

if **EMPTY?**($symbols$) **then**

if **PL-TRUE?**($KB, model$) **then return** **PL-TRUE?**($\alpha, model$)

else return *true*

else do

$P \leftarrow$ **FIRST**($symbols$); $rest \leftarrow$ **REST**($symbols$)

return **TT-CHECK-ALL**($KB, \alpha, rest, \text{EXTEND}(P, true, model)$) **and**

TT-CHECK-ALL($KB, \alpha, rest, \text{EXTEND}(P, false, model)$)

$O(2^n)$ for n symbols; problem is **co-NP-complete**

Logical equivalence

Two sentences are **logically equivalent** iff true in same models:

$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$

$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}$$

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{De Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{De Morgan}$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

Validity and satisfiability



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

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

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., $A \vee B$, 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

i.e., prove α by *reductio ad absurdum*

Proof methods divide into (roughly) two kinds:

Application of inference rules

- Legitimate (sound) generation of new sentences from old
- **Proof** = a sequence of inference rule applications
 - Can use inference rules as operators in a standard search alg.
- Typically require translation of sentences into a **normal form**

Model checking

- truth table enumeration (always exponential in n)
- improved backtracking, e.g., Davis–Putnam–Logemann–Loveland
- heuristic search in model space (sound but incomplete)
 - e.g., min-conflicts-like hill-climbing algorithms

Forward and backward chaining



Horn Form (restricted)

KB = **conjunction** of **Horn clauses**

Horn clause =

◇ proposition symbol; or

◇ (conjunction of symbols) \Rightarrow symbol

E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

(肯定式推理) **Modus Ponens** (for Horn Form): complete for Horn KBs

$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

Can be used with **forward chaining** or **backward chaining**.
These algorithms are very natural and run in **linear** time

Forward chaining (前向推理)

Idea: fire any rule whose premises are satisfied in the *KB*,
add its conclusion to the *KB*, until query is found

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

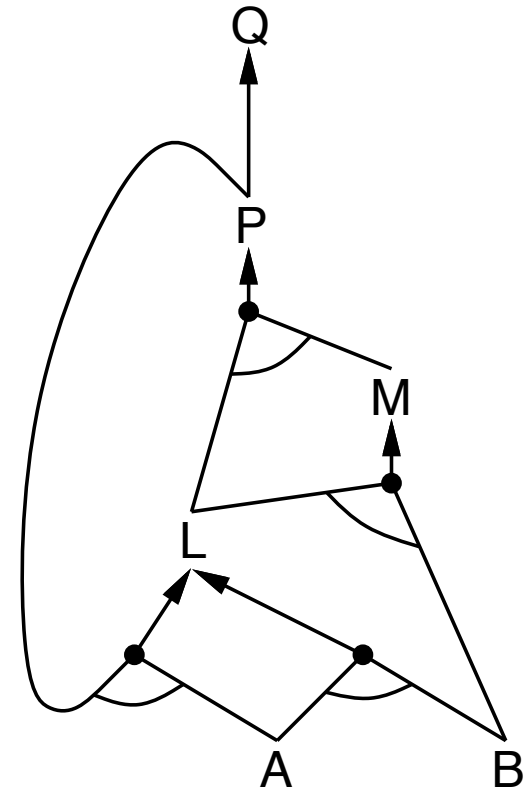
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

A

B



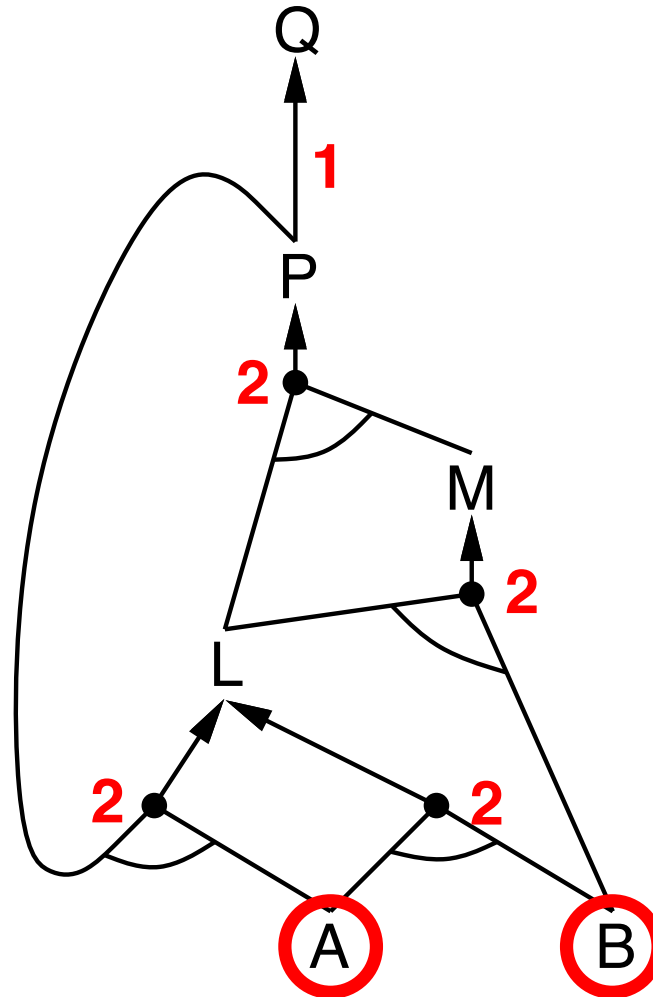
Forward chaining algorithm

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional Horn clauses
           q, the query, a proposition symbol
  local variables: count, a table, indexed by clause, initially the number of premises
                    inferred, a table, indexed by symbol, each entry initially false
                    agenda, a list of symbols, initially the symbols known in KB

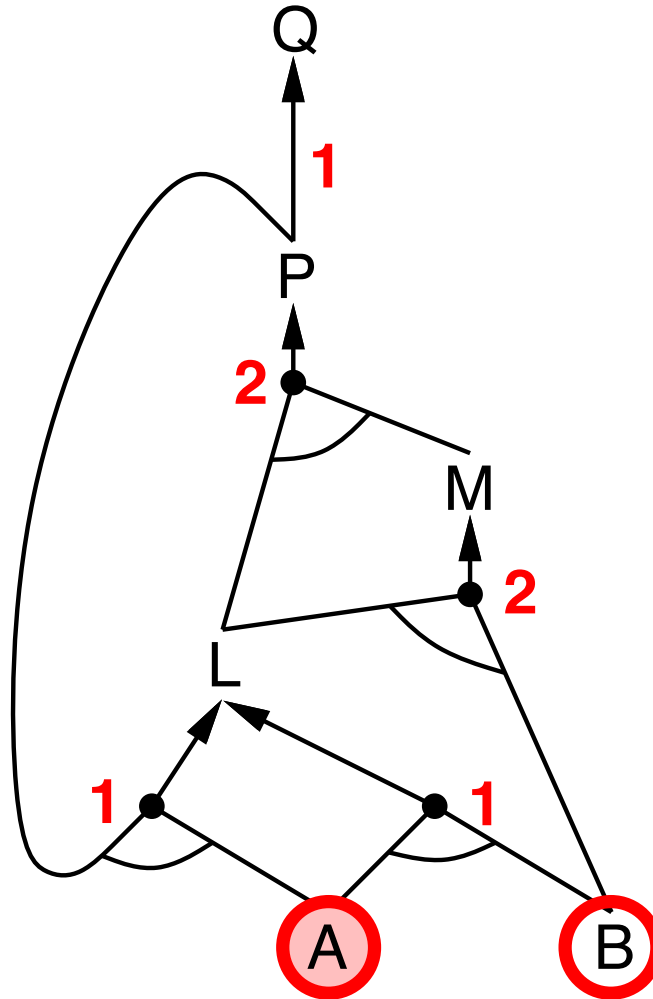
  while agenda is not empty do
    p ← POP(agenda)
    unless inferred[p] do
      inferred[p] ← true
      for each Horn clause c in whose premise p appears do
        decrement count[c]
        if count[c] = 0 then do
          if HEAD[c] = q then return true
          PUSH(HEAD[c], agenda)

  return false
```

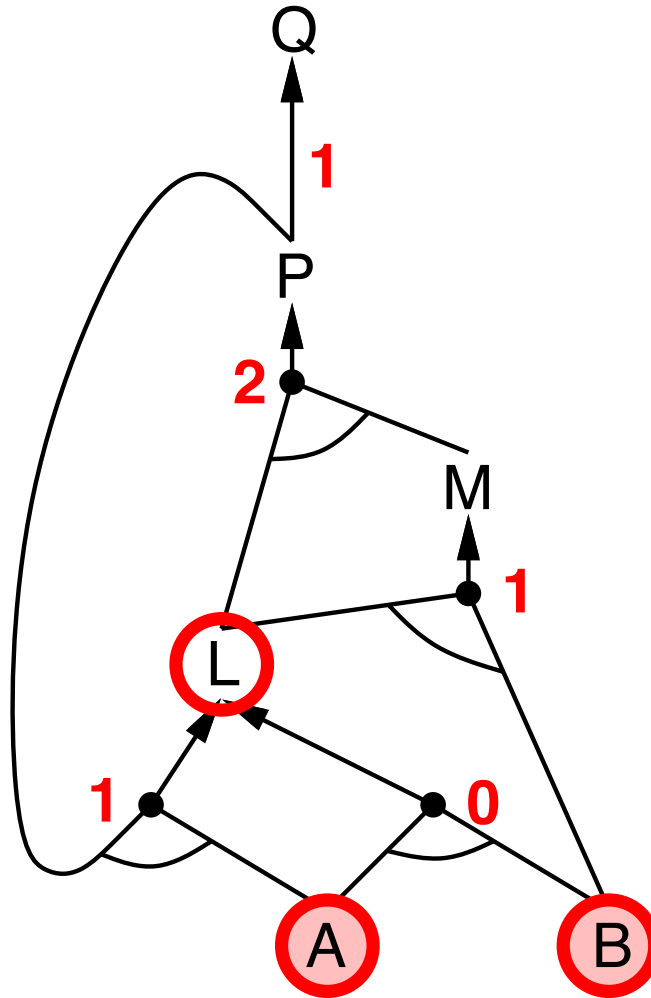
Forward chaining example



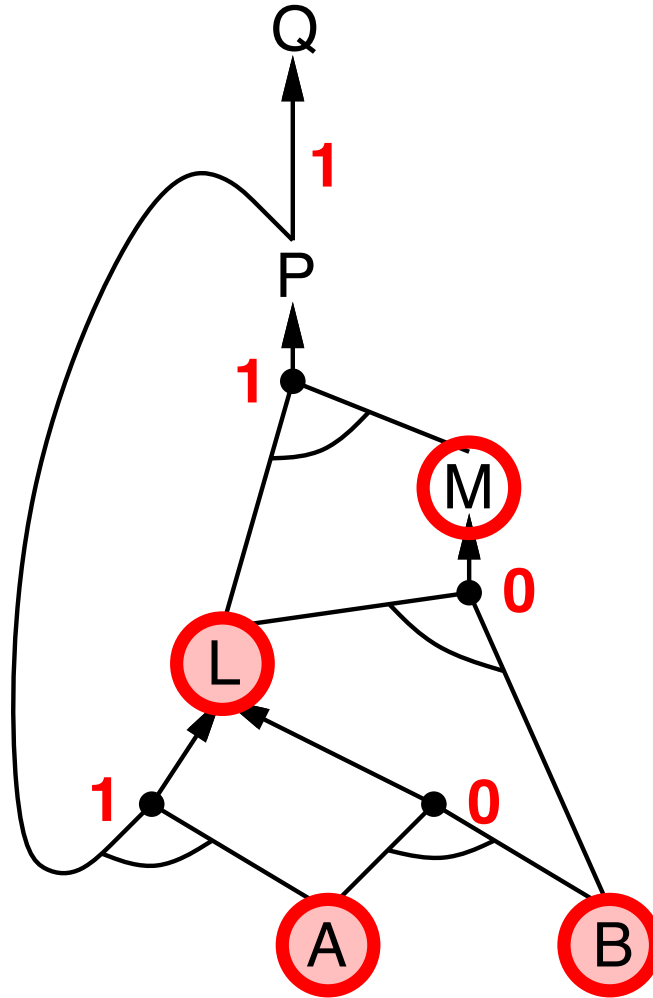
Forward chaining example



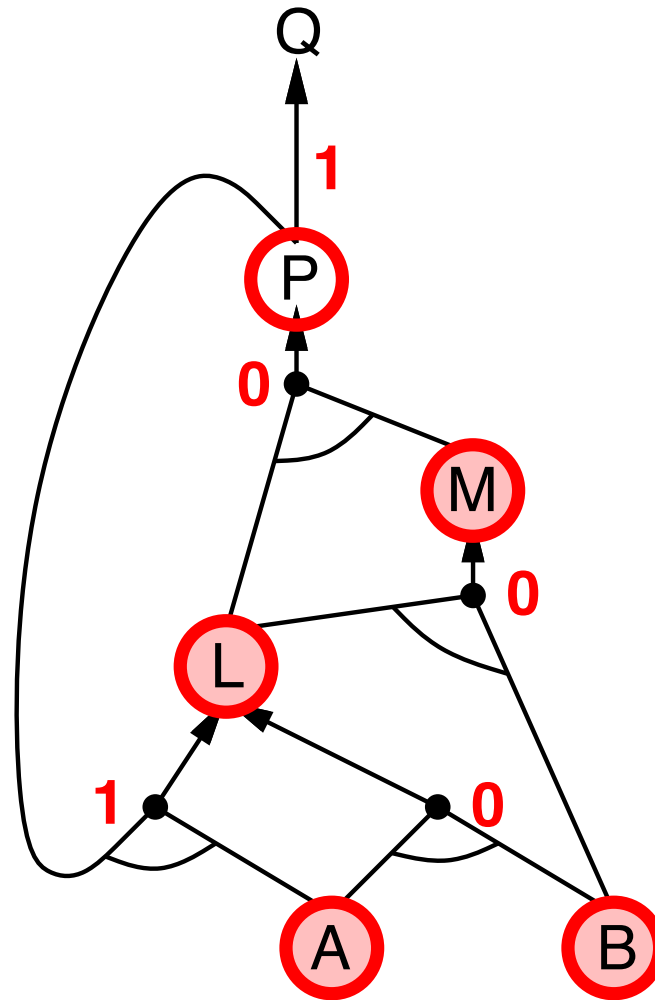
Forward chaining example



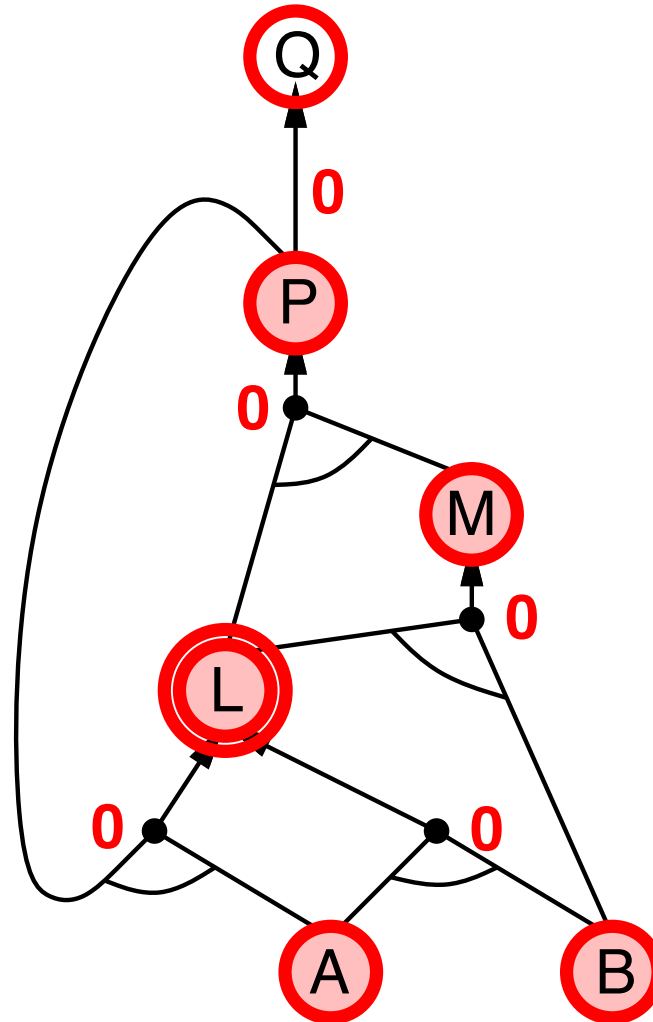
Forward chaining example



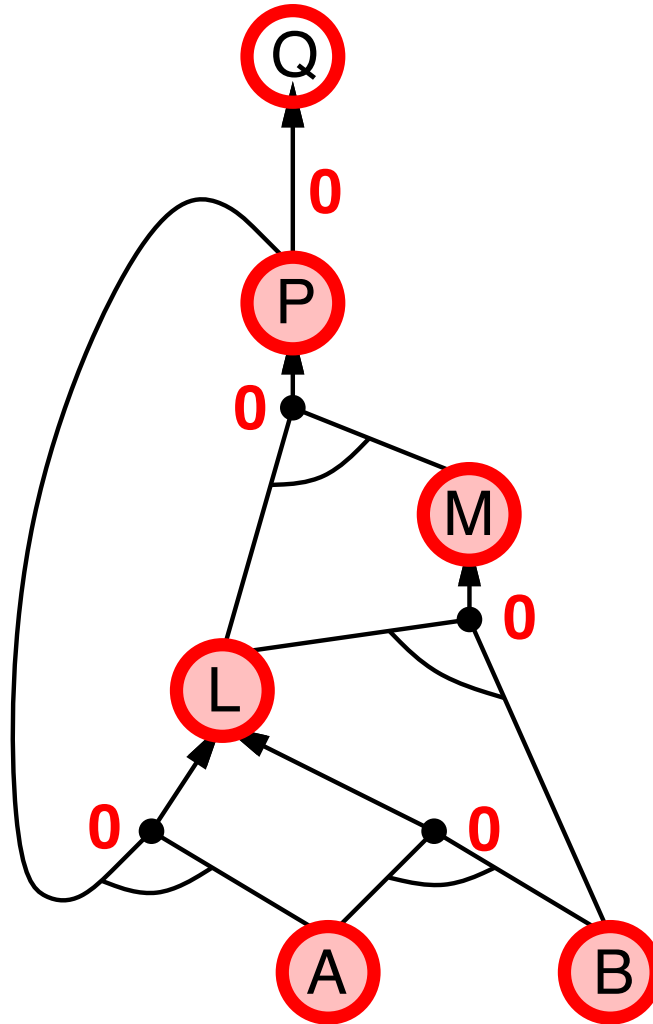
Forward chaining example



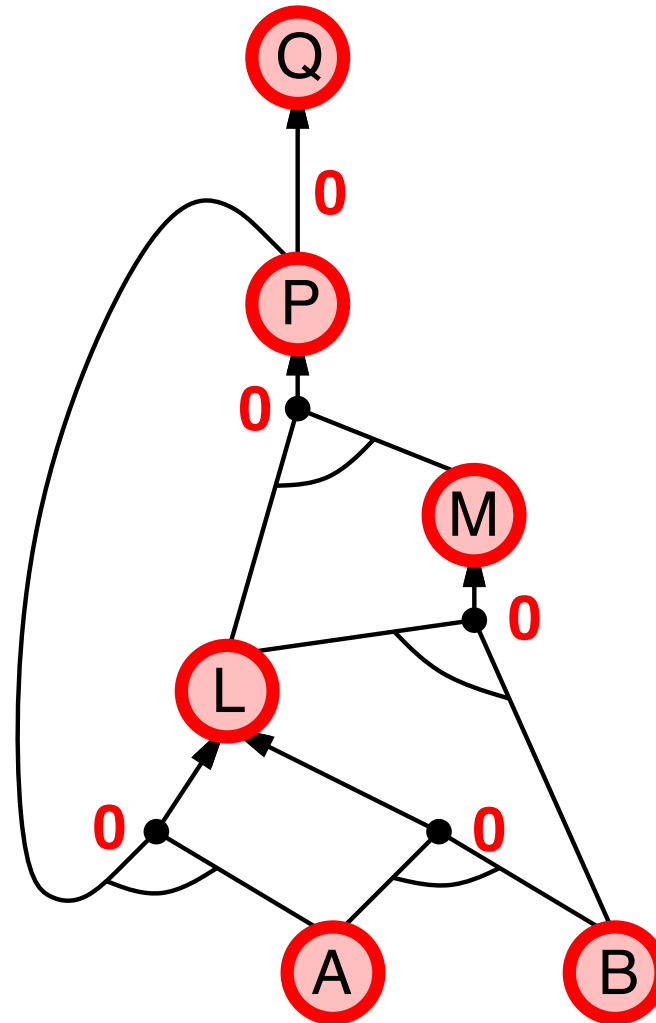
Forward chaining example



Forward chaining example



Forward chaining example



Proof of completeness

FC derives every atomic sentence that is entailed by KB

1. FC reaches a **fixed point** where no new atomic sentences are derived
2. Consider the final state as a model m , assigning true/false to symbols

3. Every clause in the original KB is true in m

Proof: Suppose a clause $a_1 \wedge \dots \wedge a_k \Rightarrow b$ is false in m

Then $a_1 \wedge \dots \wedge a_k$ is true in m and b is false in m

Therefore the algorithm has not reached a fixed point!

4. Hence m is a model of KB

5. If $KB \models q$, q is true in **every** model of KB , including m

General idea: construct any model of KB by sound inference, check α

Backward chaining (后向推理)

Idea: work backwards from the query q :

to prove q by BC,

check if q is known already, or

prove by BC all premises of some rule concluding q

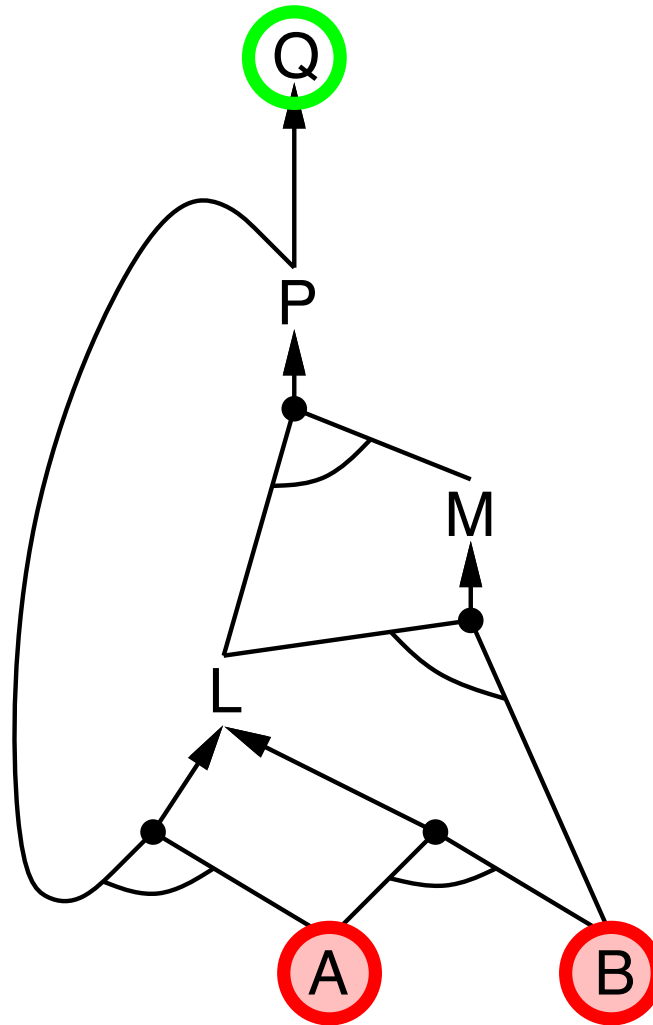
Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal

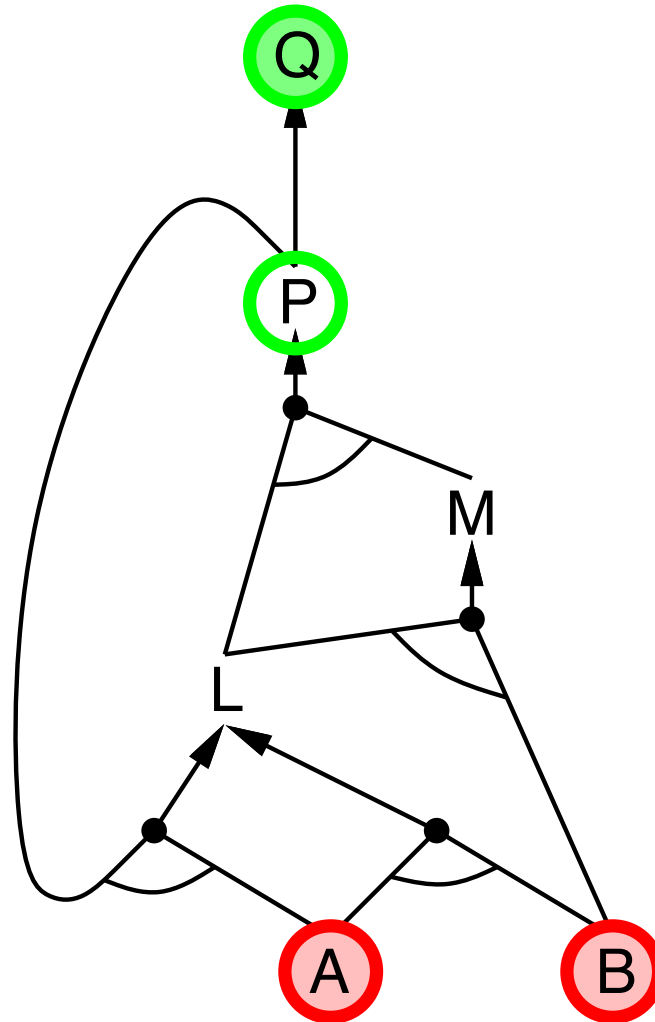
1) has already been proved true, or

2) has already failed

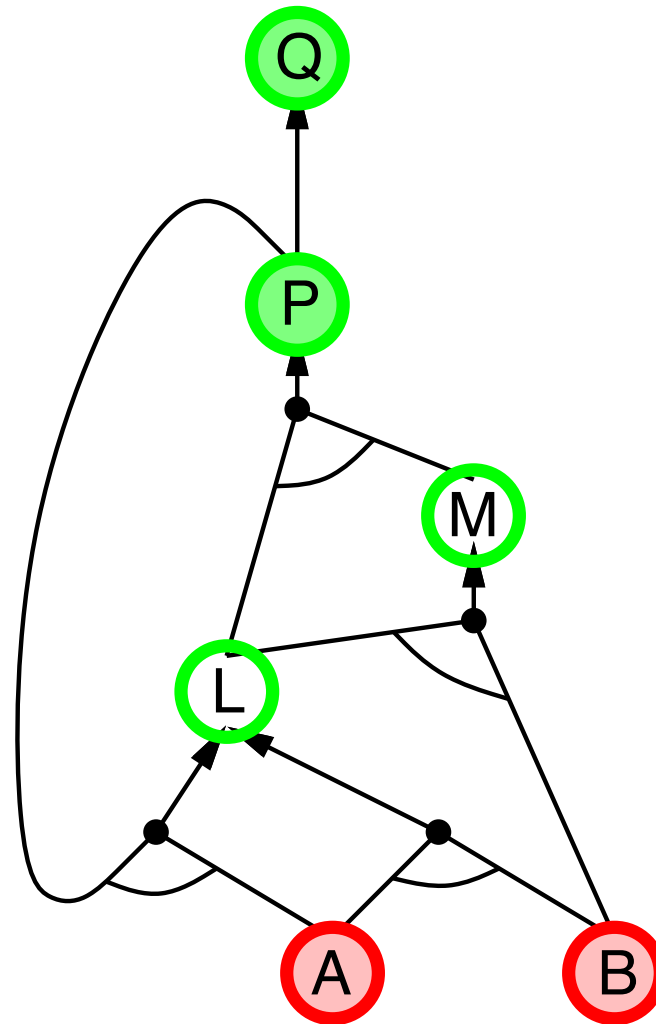
Backward chaining example



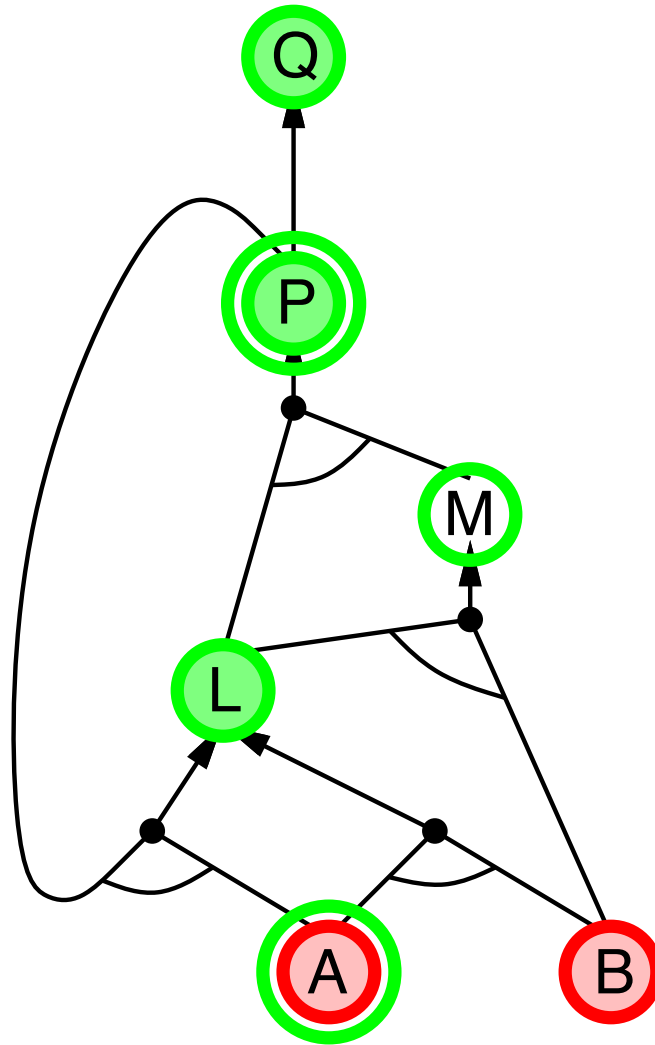
Backward chaining example



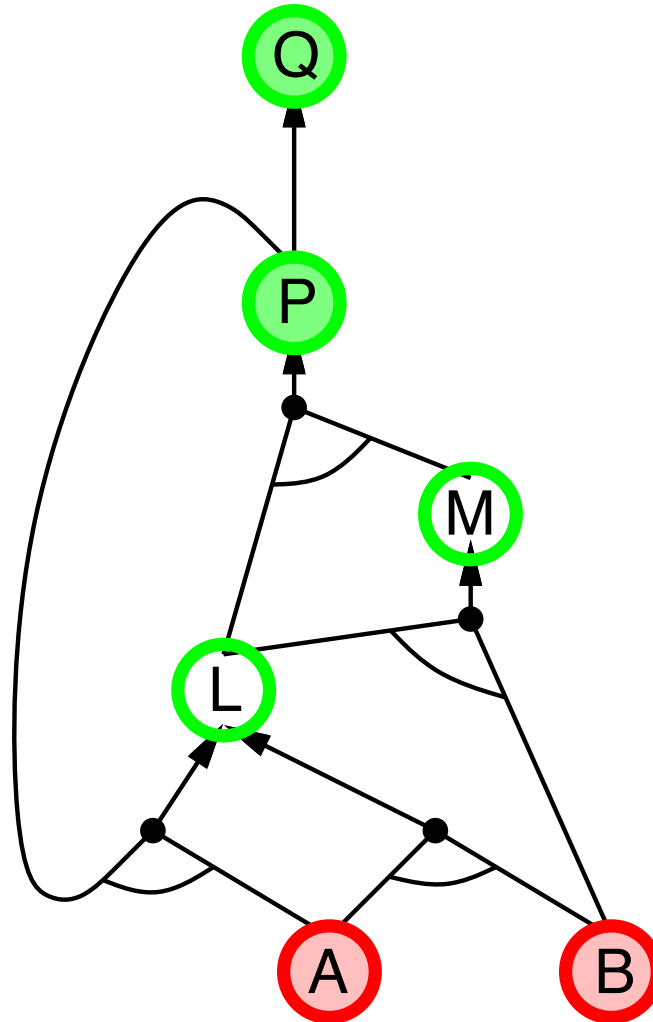
Backward chaining example



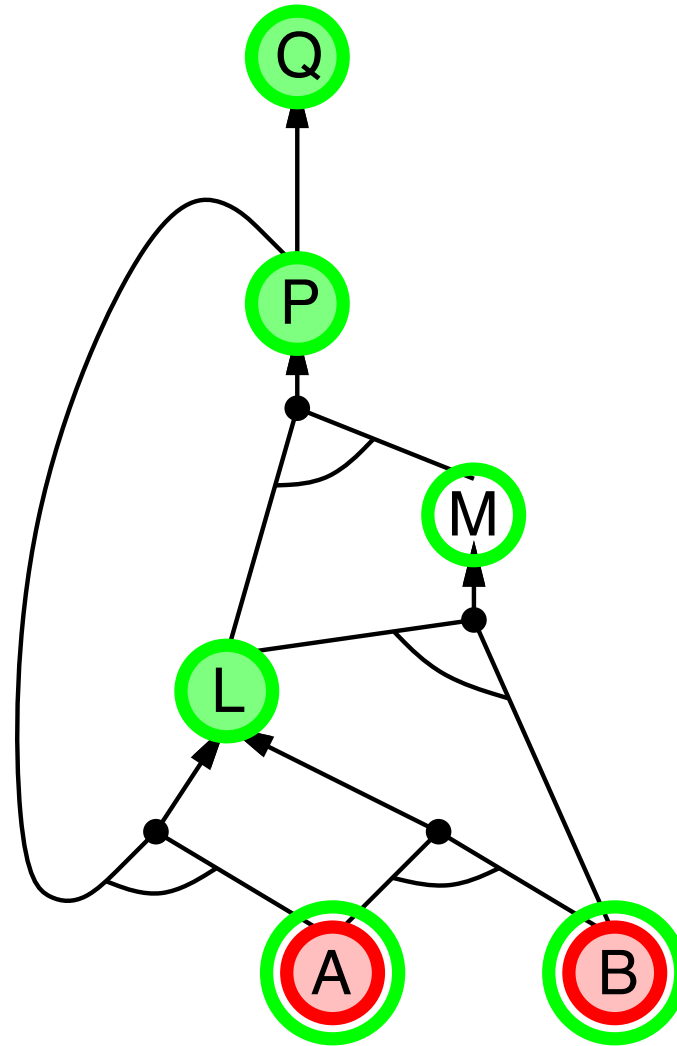
Backward chaining example



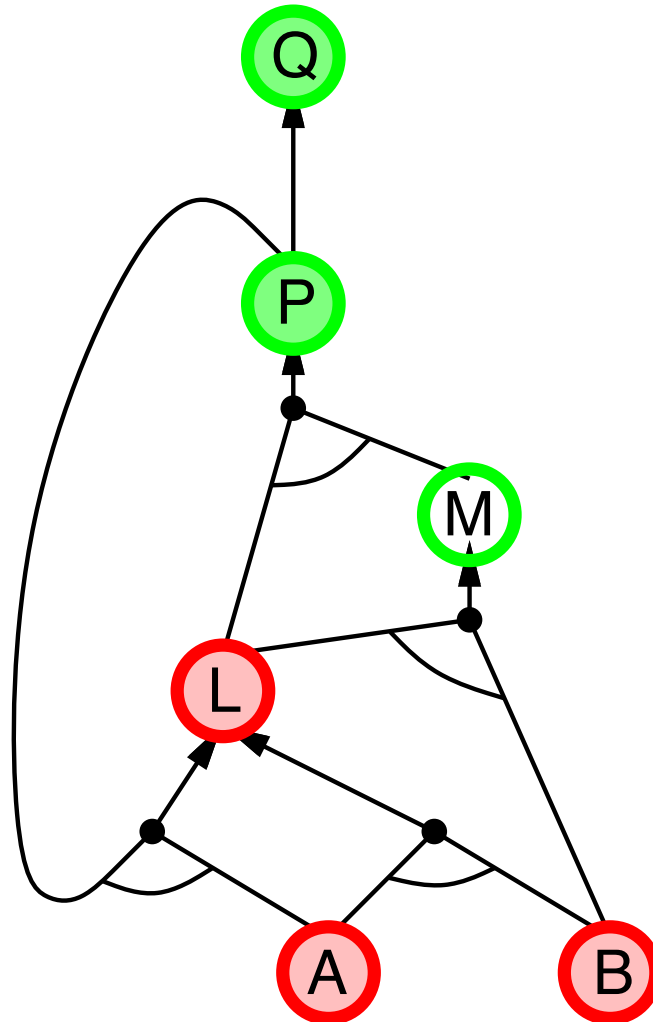
Backward chaining example



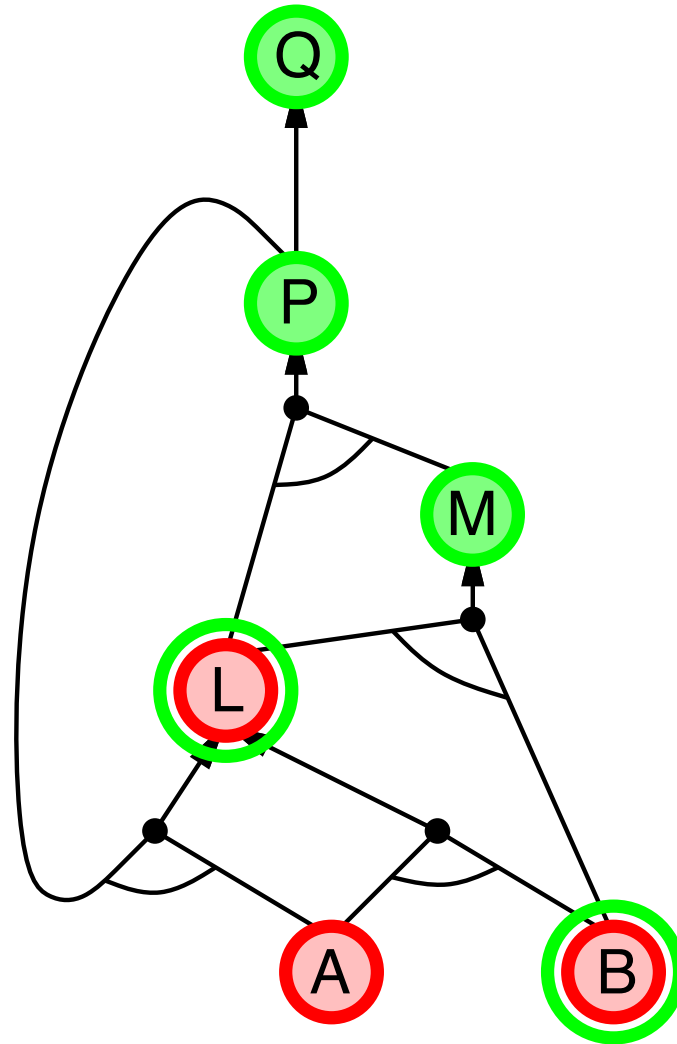
Backward chaining example



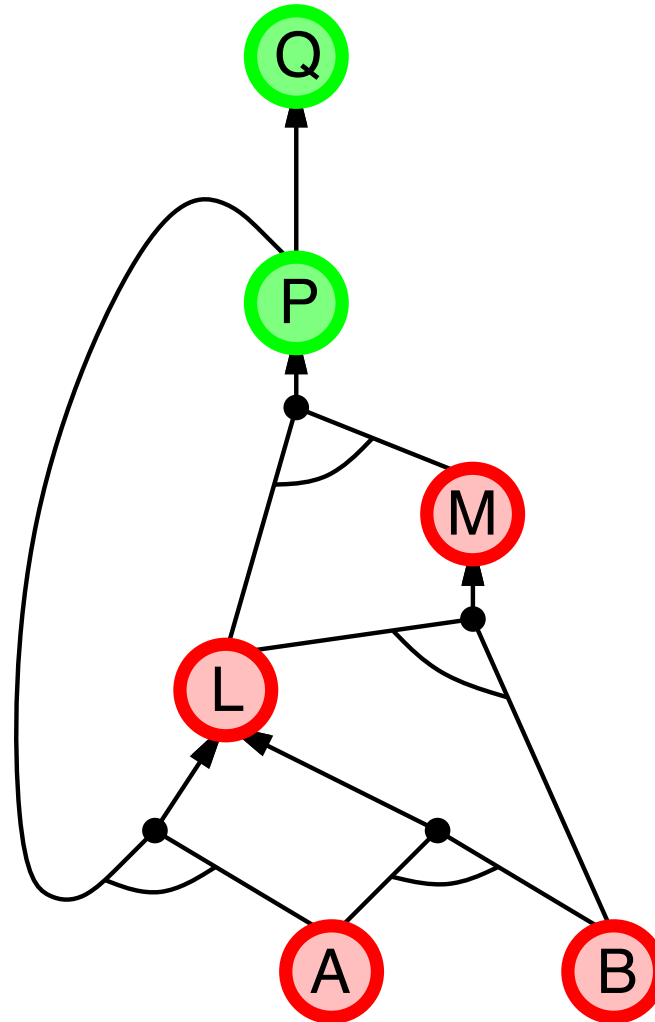
Backward chaining example



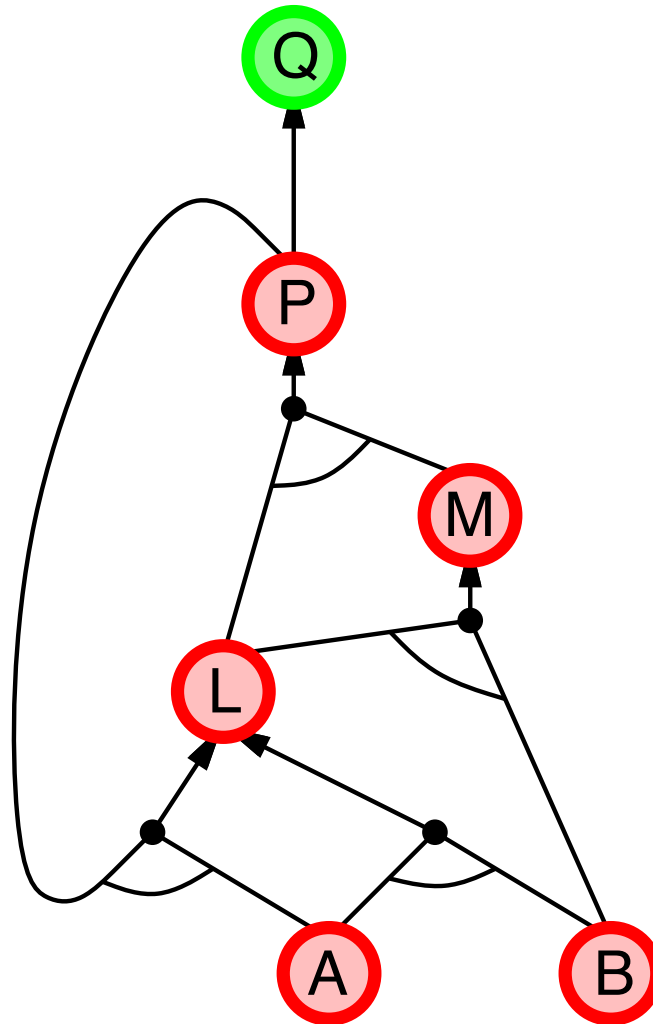
Backward chaining example



Backward chaining example



Backward chaining example



Comparison



FC is **data-driven**, cf. automatic, unconscious processing,
e.g., object recognition, routine decisions

May do lots of work that is irrelevant to the goal

BC is **goal-driven**, appropriate for problem-solving,
e.g., Where are my keys? How do I get into a PhD program?

Complexity of BC can be **much less** than linear in size of KB

Resolution (消解)

Conjunctive Normal Form (CNF—universal)
conjunction of **disjunctions** of **literals**
clauses

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

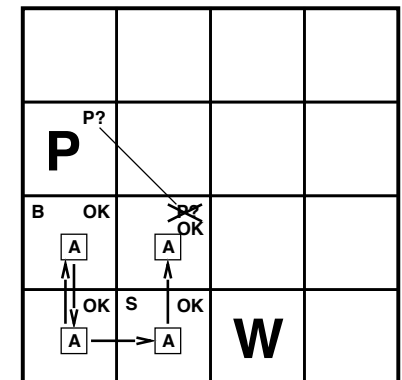
Resolution inference rule (for CNF): complete for propositional logic

$$\frac{l_1 \vee \dots \vee l_k, \quad m_1 \vee \dots \vee m_n}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

where l_i and m_j are complementary literals. E.g.,

$$\frac{P_{1,3} \vee P_{2,2}, \quad \neg P_{2,2}}{P_{1,3}}$$

Resolution is sound and complete for propositional logic



Conversion to CNF



$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

1. Eliminate \Leftrightarrow , replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

2. Eliminate \Rightarrow , replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

3. Move \neg inwards using de Morgan's rules and double-negation:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

4. Apply distributivity law (\vee over \wedge) and flatten:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

Resolution algorithm

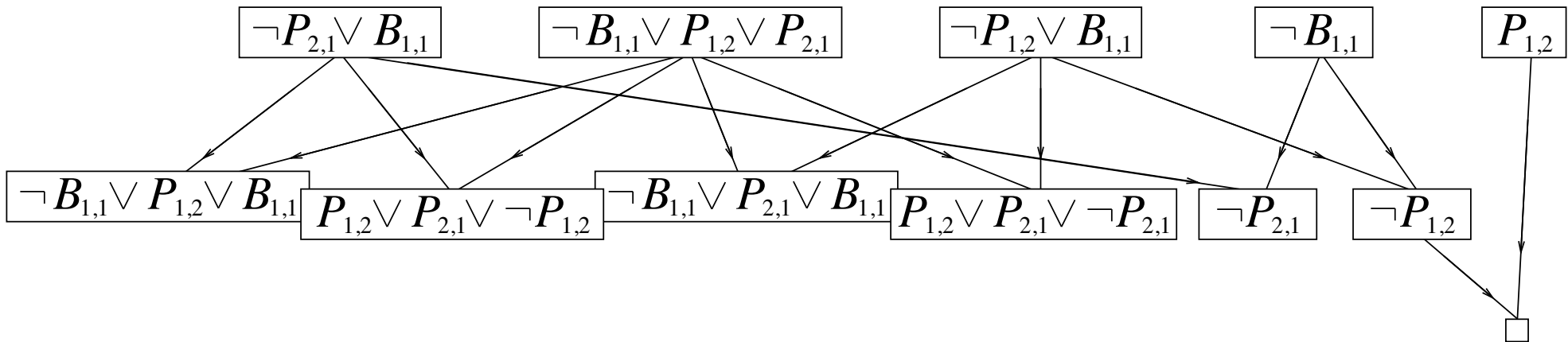
Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

```
function PL-RESOLUTION( $KB, \alpha$ ) returns true or false
  inputs:  $KB$ , the knowledge base, a sentence in propositional logic
            $\alpha$ , the query, a sentence in propositional logic

  clauses  $\leftarrow$  the set of clauses in the CNF representation of  $KB \wedge \neg\alpha$ 
  new  $\leftarrow$  { }
  loop do
    for each  $C_i, C_j$  in clauses do
      resolvents  $\leftarrow$  PL-RESOLVE( $C_i, C_j$ )
      if resolvents contains the empty clause then return true
      new  $\leftarrow$  new  $\cup$  resolvents
  if new  $\subseteq$  clauses then return false
  clauses  $\leftarrow$  clauses  $\cup$  new
```

Resolution example

$$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1} \quad \alpha = \neg P_{1,2}$$



Summary

Logical agents apply **inference** to a **knowledge base** to derive new information and make decisions

Basic concepts of logic:

- **syntax**: formal structure of **sentences**
- **semantics**: **truth** of sentences wrt **models**
- **entailment**: necessary truth of one sentence given another
- **inference**: deriving sentences from other sentences
- **soundness**: derivations produce only entailed sentences
- **completeness**: derivations can produce all entailed sentences

Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Forward, backward chaining are linear-time, complete for Horn clauses
Resolution is complete for propositional logic

Propositional logic lacks expressive power

SAT problems



Propositional logic, CNF

literals: x_1, x_2, \dots, x_n

clauses: $(x_1 \vee x_2 \vee x_5) \quad (\neg x_2 \vee x_3 \vee \neg x_7) \quad \dots$

problem: find an assignment to literals so that the conjunction of the clauses is true, or prove unsatisfiable

$$(x_1 \vee x_2 \vee x_5) \wedge (\neg x_2 \vee x_3 \vee \neg x_7) \wedge \dots$$

2SAT: every clause has at most 2 literals

P-solvable

3SAT: every clause has at most 3 literals

NP-hard

SAT solvers



SAT problems have many important applications

many SAT solvers are ready for use

DPLL

WalkSAT

Davis–Putnam–Logemann–Loveland algorithm

function DPLL-SATISFIABLE?(*s*) **returns** *true* or *false*

inputs: *s*, a sentence in propositional logic

clauses \leftarrow the set of clauses in the CNF representation of *s*

symbols \leftarrow a list of the proposition symbols in *s*

return DPLL(*clauses*, *symbols*, { })

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

if every clause in *clauses* is true in *model* **then return** *true*

if some clause in *clauses* is false in *model* **then return** *false*

P, *value* \leftarrow FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)

if *P* is non-null **then return** DPLL(*clauses*, *symbols* – *P*, *model* \cup { *P*=*value* })

P, *value* \leftarrow FIND-UNIT-CLAUSE(*clauses*, *model*)

if *P* is non-null **then return** DPLL(*clauses*, *symbols* – *P*, *model* \cup { *P*=*value* })

P \leftarrow FIRST(*symbols*); *rest* \leftarrow REST(*symbols*)

return DPLL(*clauses*, *rest*, *model* \cup { *P*=*true* }) **or**

DPLL(*clauses*, *rest*, *model* \cup { *P*=*false* })

a deep-first search with heuristics

DPLL heuristics



Pure symbol heuristic: A **pure symbol** is a symbol that always appears with the same “sign” in all clauses.

$$(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee A)$$

A and B is pure, but not C

Unit clause heuristic: A **unit clause** is a clause with just one literal.

$$(A \vee \neg B) \text{ with } A = \text{true}$$

is a unit clause

Other tricks



Component analysis : find disjoint subsets

Variable and value ordering : assign most frequent variable at first

Intelligent backtracking : remember conflicts

Random restart

Clever indexing

a local search hill-climbing or others.

```
function WALKSAT(clauses, p, max_flips) returns a satisfying model or failure  
inputs: clauses, a set of clauses in propositional logic  
         p, the probability of choosing to do a “random walk” move, typically around 0.5  
         max_flips, number of flips allowed before giving up  
  
model ← a random assignment of true/false to the symbols in clauses  
for i = 1 to max_flips do  
    if model satisfies clauses then return model  
    clause ← a randomly selected clause from clauses that is false in model  
    with probability p flip the value in model of a randomly selected symbol from clause  
    else flip whichever symbol in clause maximizes the number of satisfied clauses  
return failure
```

failure ≠ unsatisfiable

The landscape of random SAT problems

Not all SAT instances are hard

under-constraint: a few clauses => easy to enumerate

over-constraint: too many clauses => unsatisfiable

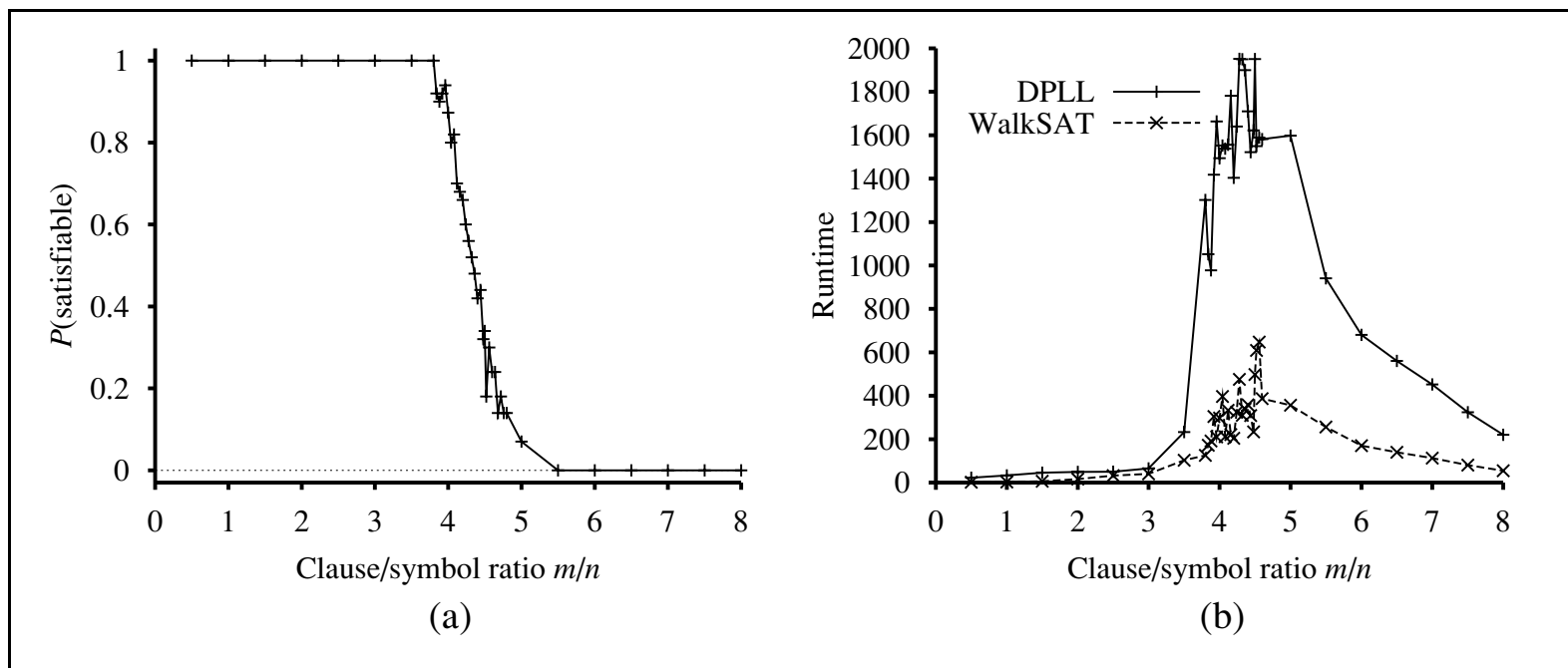


Figure 7.19 (a) Graph showing the probability that a random 3-CNF sentence with $n = 50$ symbols is satisfiable, as a function of the clause/symbol ratio m/n . (b) Graph of the median run time (measured in number of recursive calls to DPLL, a good proxy) on random 3-CNF sentences. The most difficult problems have a clause/symbol ratio of about 4.3.