BLG435E
Artificial Intelligence

Lecture 6: Logical Agents

Outline

- Knowledge-based agents
  - Knowledge representation
  - Inference
- Wumpus world
- Logic in general - models and entailment
- Propositional (Boolean) logic
- Inference rules and theorem proving
  - forward chaining
  - backward chaining
  - Resolution
- First-order logic

Knowledge bases

- Knowledge base = set of sentences in a formal language
- Declarative approach to build 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

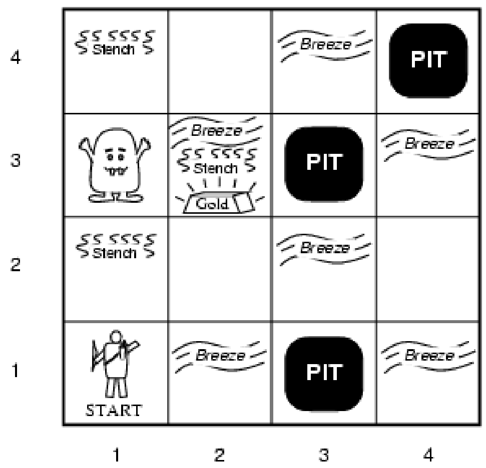
A simple knowledge-based agent

```
function KB-AGENT(percept) returns an action
  persistent: 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: Stench, Breeze, Glitter, Bump, Scream

Wumpus world characterization

- Fully Observable
- Deterministic
- Episodic
- Static
- Discrete
- Single-agent

Wumpus world characterization

- Fully Observable No – only local perception
- Deterministic
- Episodic
- Static
- Discrete
- Single-agent

Wumpus world characterization

- Fully Observable No – only local perception
- Deterministic Yes – outcomes exactly specified
- Episodic
- Static
- Discrete
- Single-agent

Wumpus world characterization

- Fully Observable No – only local perception
- Deterministic Yes – outcomes exactly specified
- Episodic No – sequential at the level of actions
- Static
- Discrete
- Single-agent

Wumpus world characterization

- Fully 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
- Single-agent

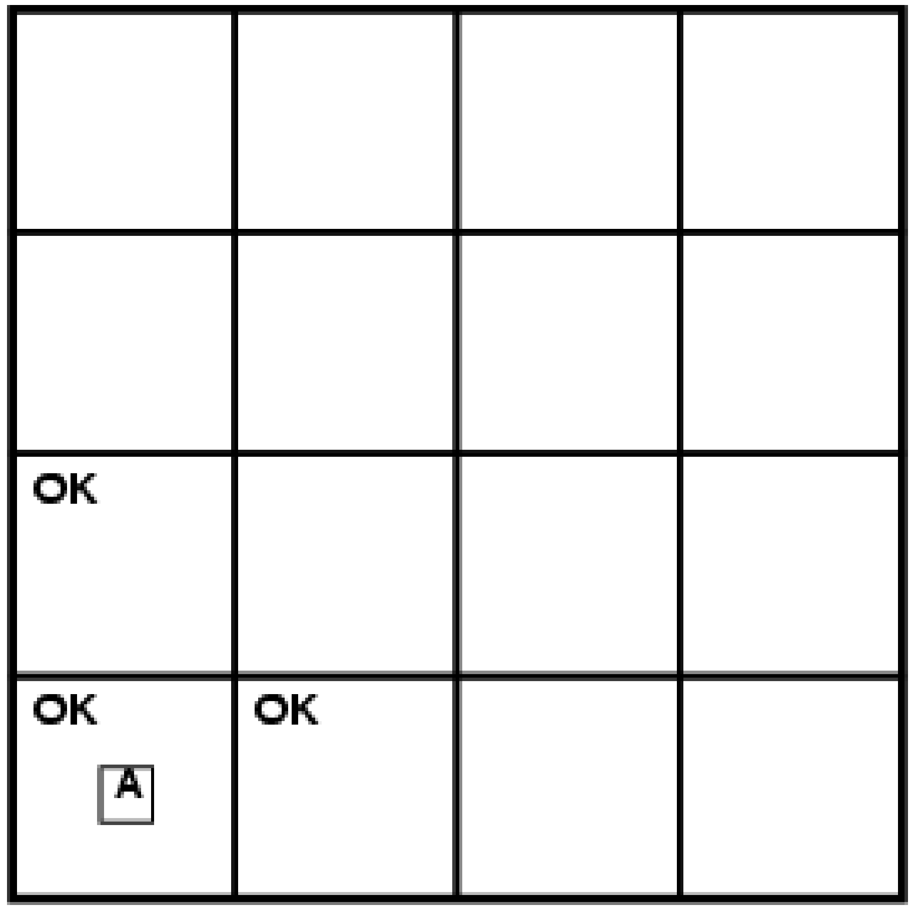
Wumpus world characterization

- Fully 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

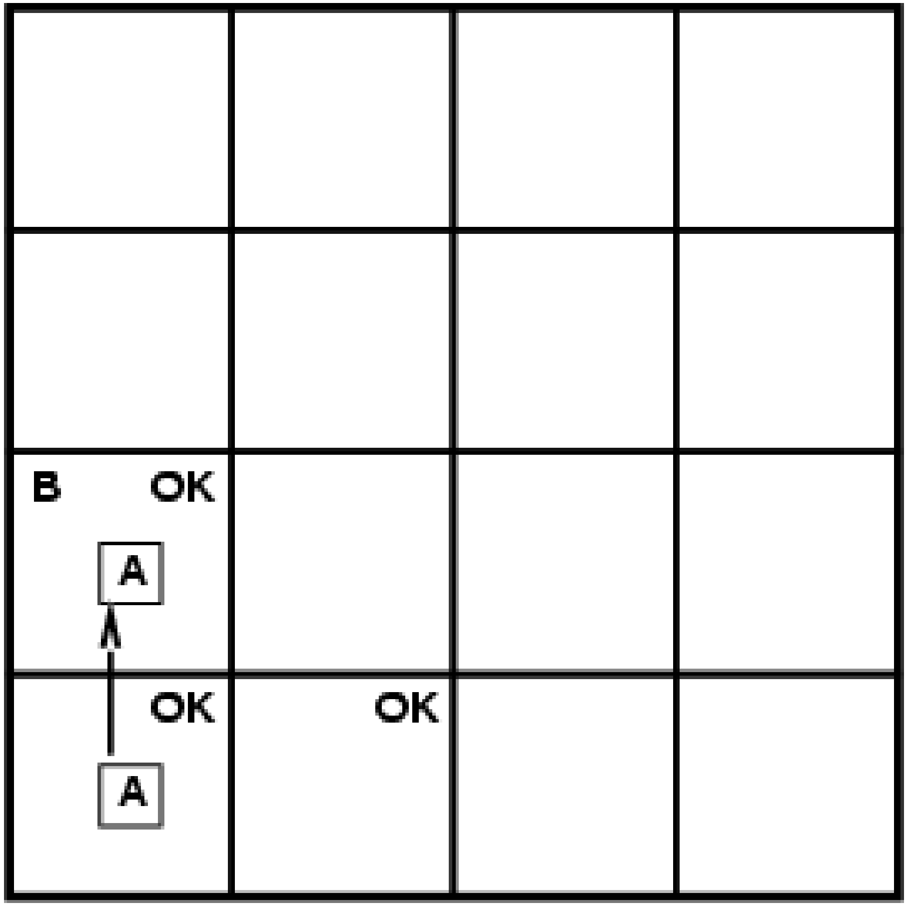
Wumpus world characterization

- Fully 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

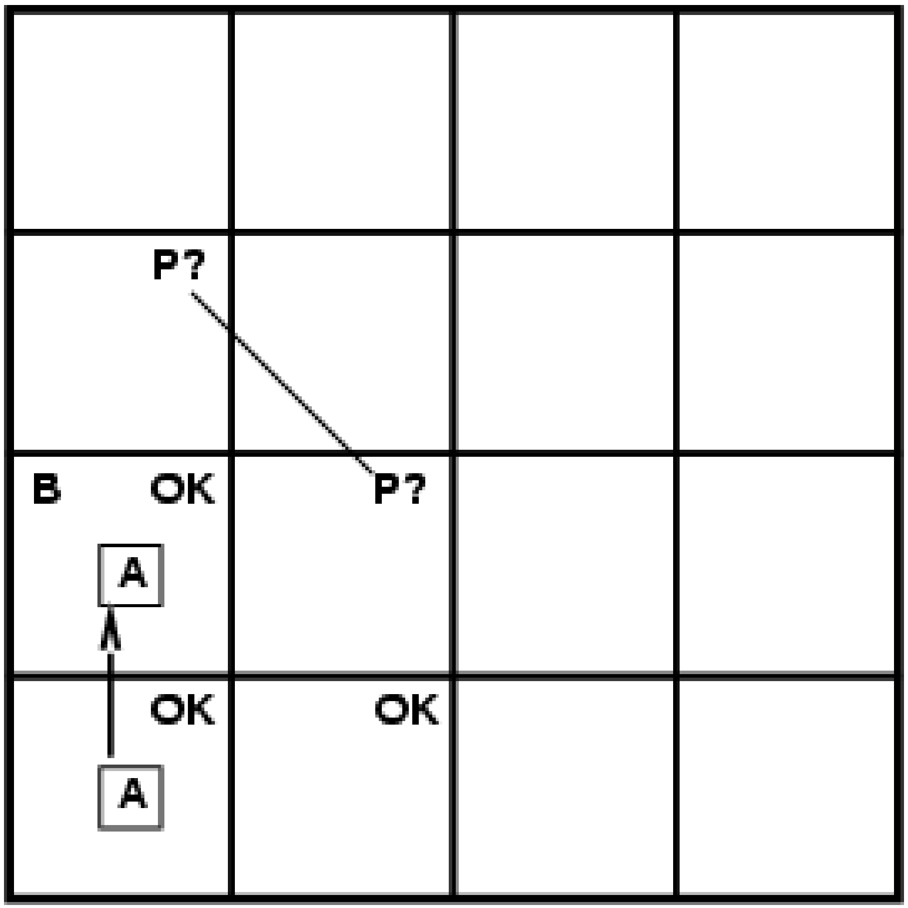
Exploring a wumpus world



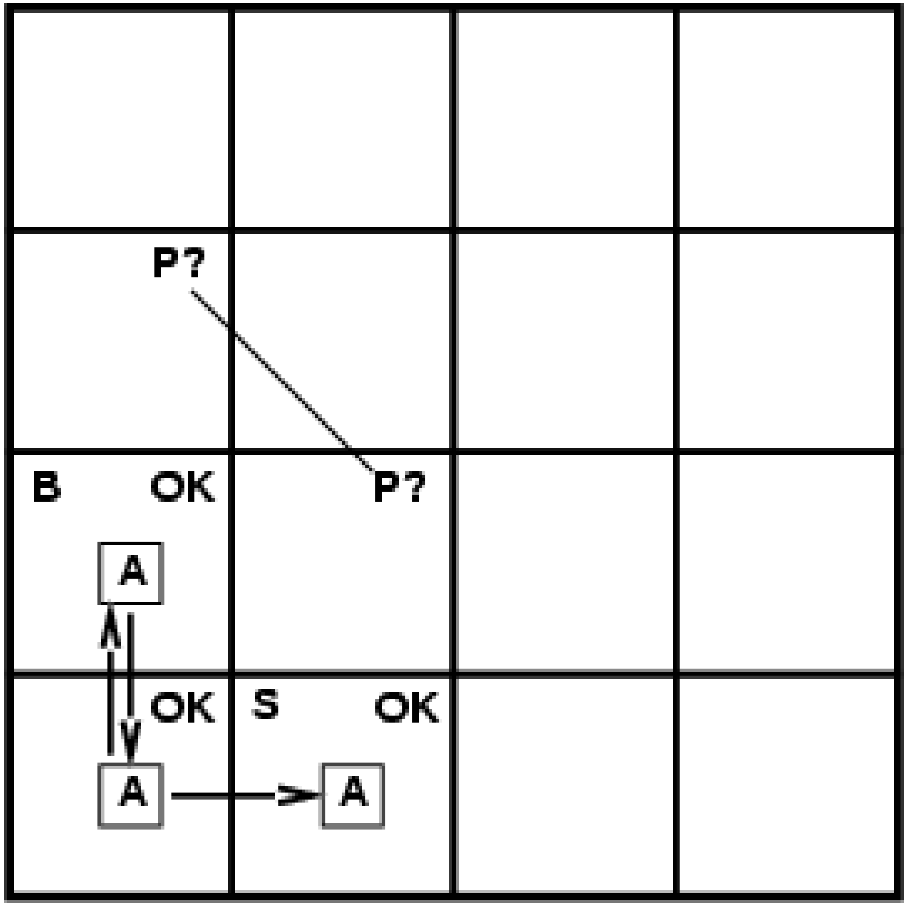
Exploring a wumpus world



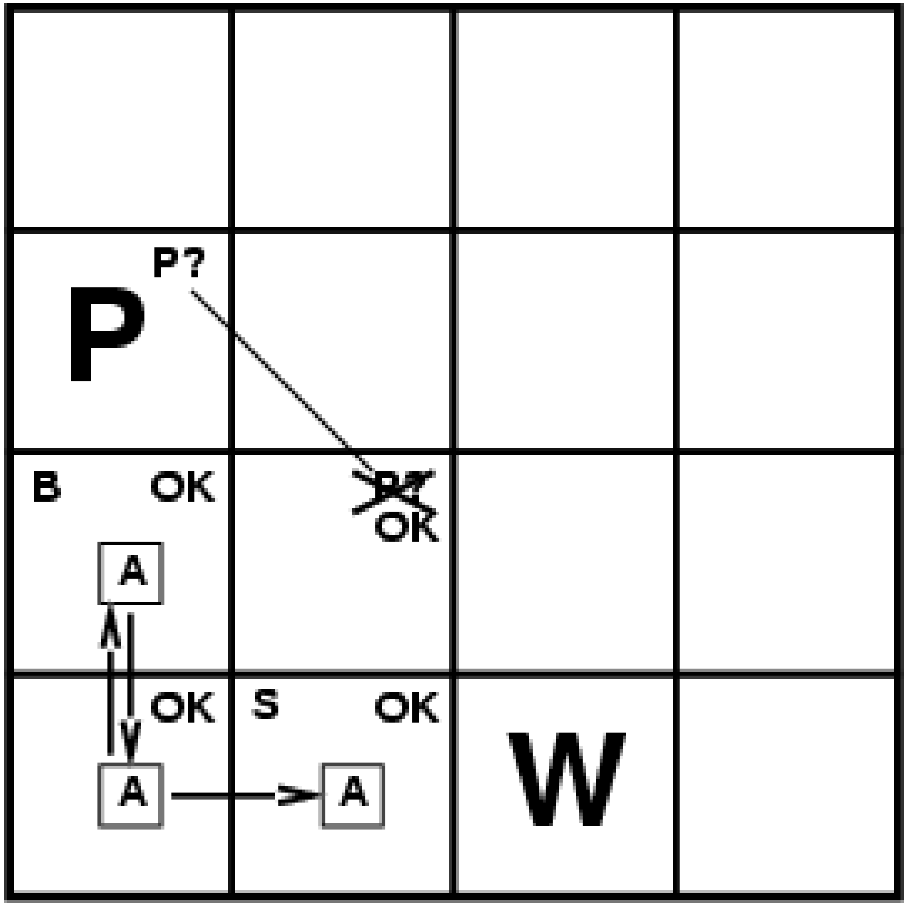
Exploring a wumpus world



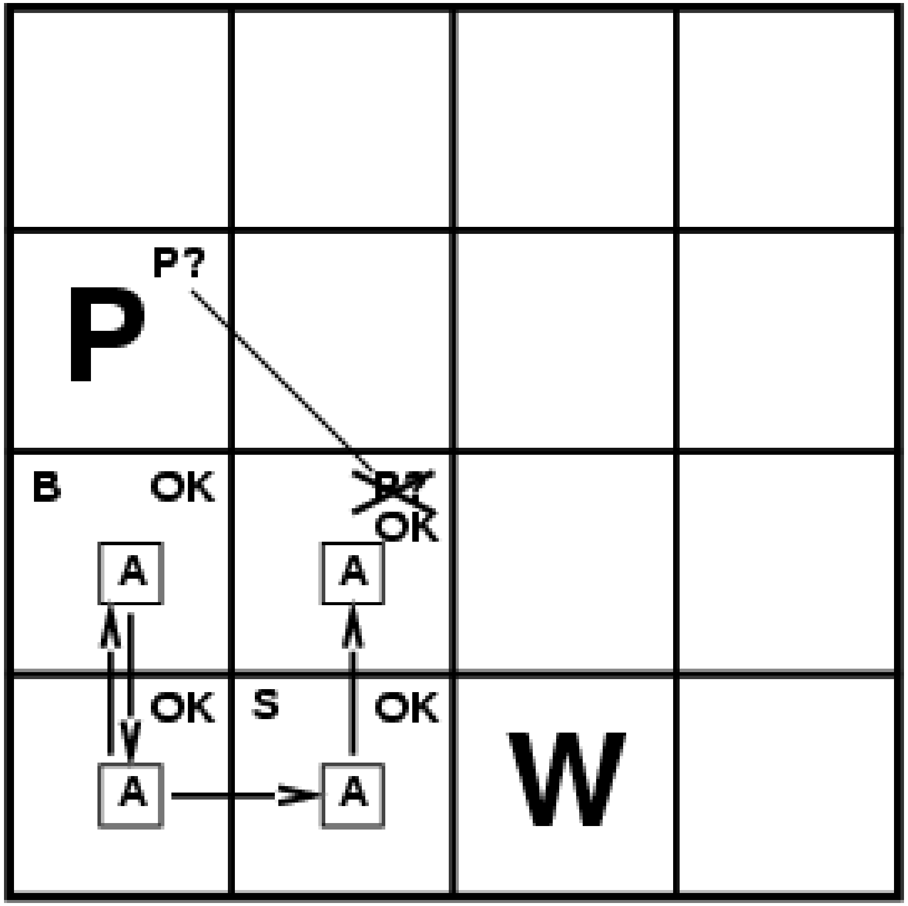
Exploring a wumpus world



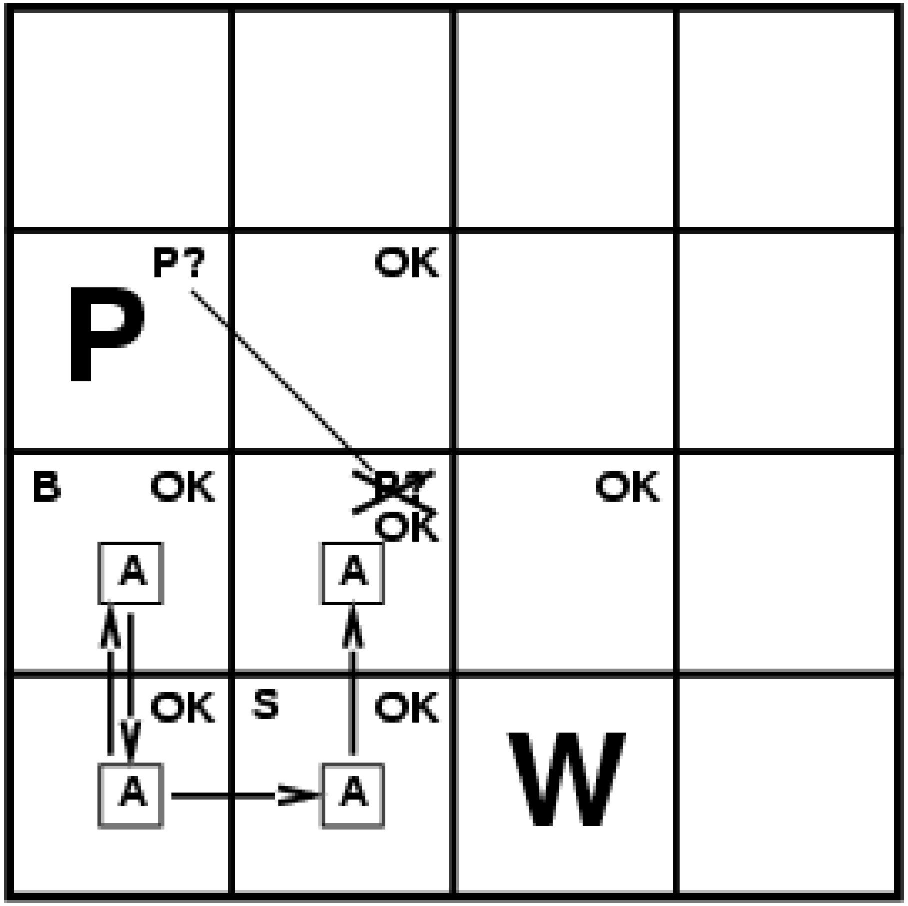
Exploring a wumpus world



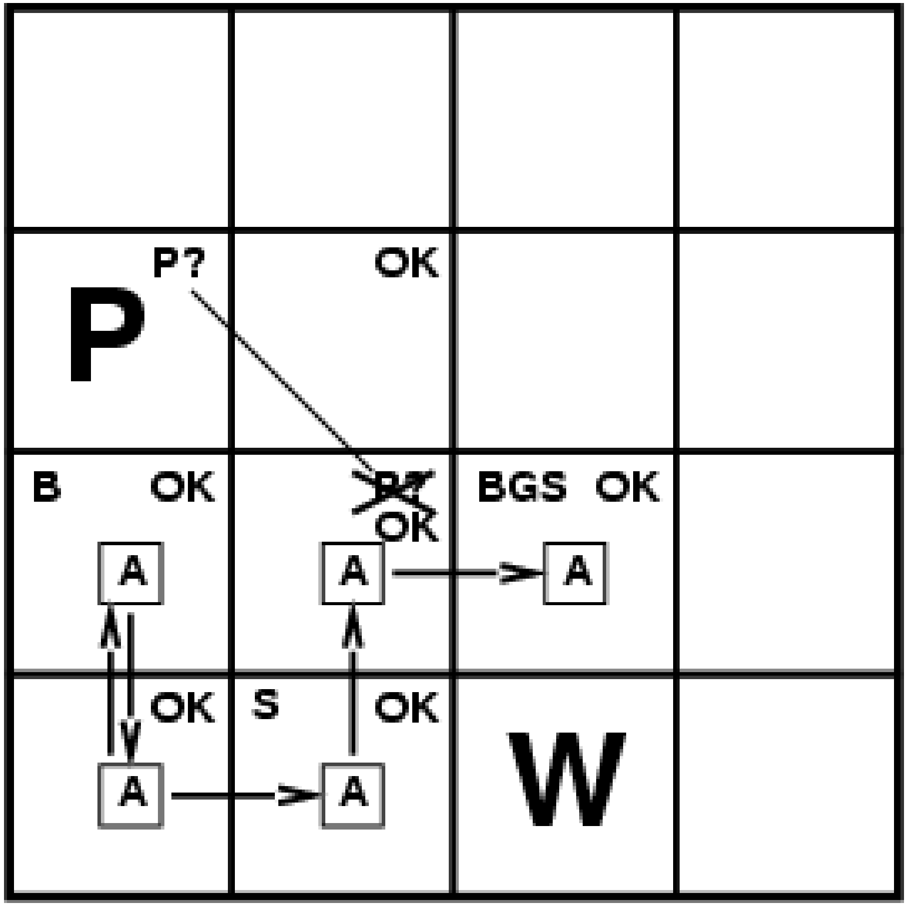
Exploring a wumpus world



Exploring a wumpus world



Exploring a wumpus world



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; $x2+y > \{\}$ 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

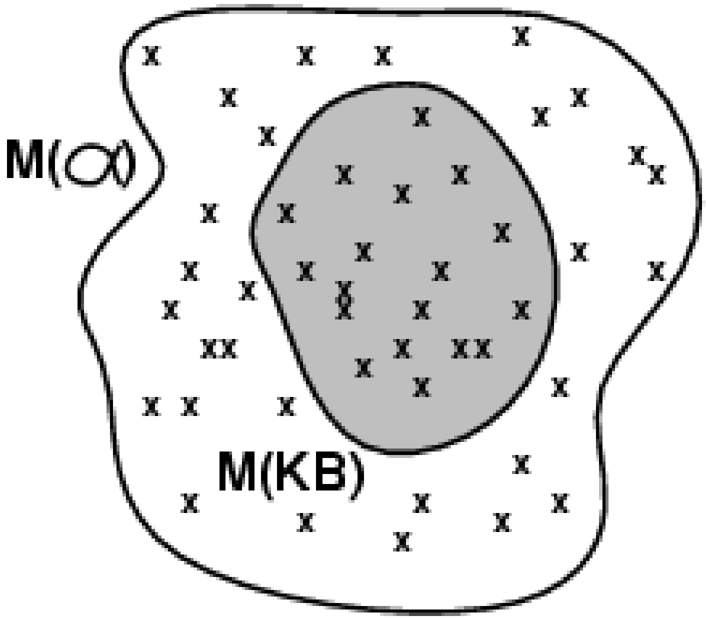
- Entailment means that one thing follows from another:

$$KB \models \alpha$$

- 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 "FB won" and "GS won" entails either "GS won " or " FB won"
  - e.g., x+y = 4 entails 4 = x+y
  - Entailment is a relationship between sentences (i.e., syntax) that is based on semantics

Models

- Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated
  - Possible world
- We say *m* is a model of a sentence α, if α is true in *m*
- *M(α)* is the set of all models of α
- Then KB $\models$ α iff *M(KB)* $\subseteq$ *M*(α)
  - e.g. *KB* = FB won and GS won
  - α = FB won

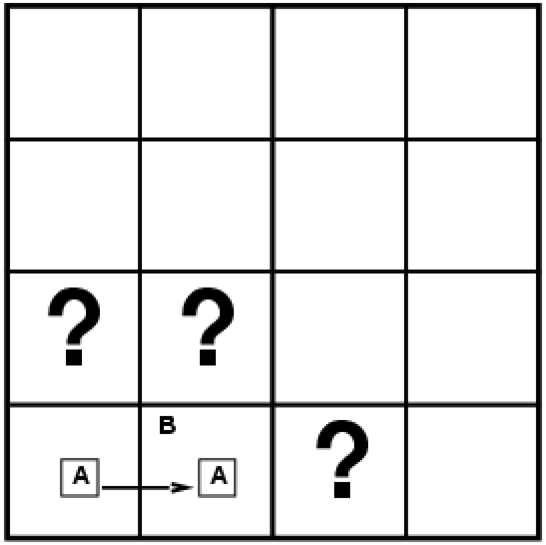


Entailment in the wumpus world

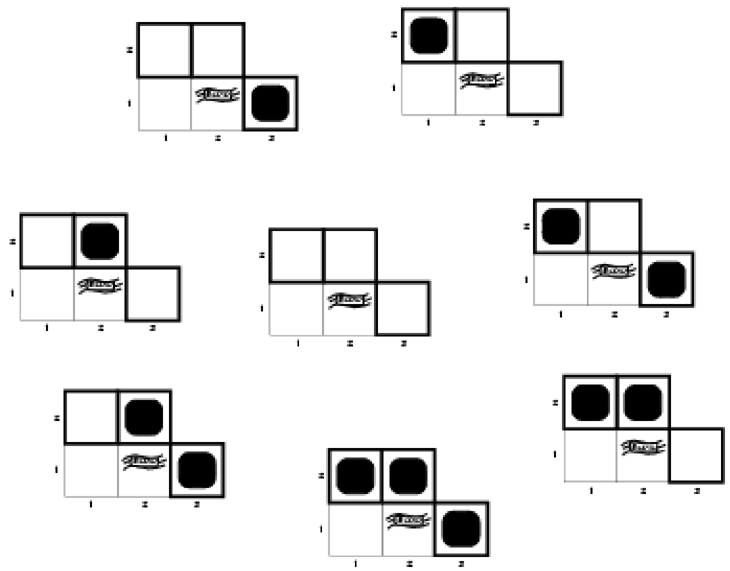
Situation after detecting nothing in [1,1], moving right, breeze in [2,1]

Consider possible models for *KB* assuming only pits

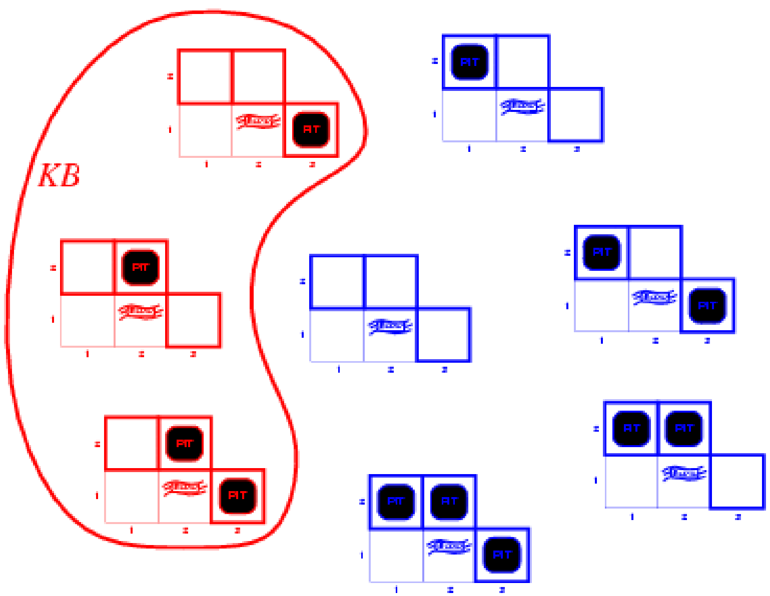
3 Boolean choices $\Rightarrow$? possible models



Wumpus models



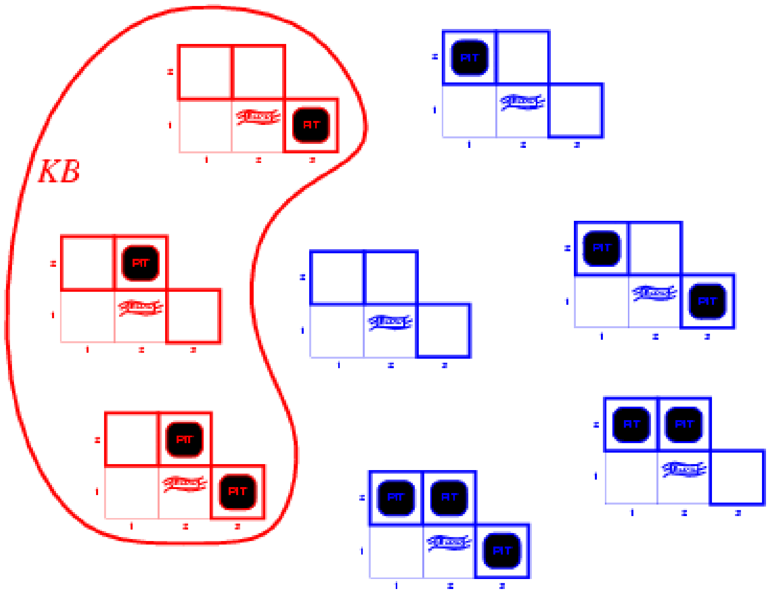
Wumpus models



- *KB* = wumpus-world rules + observations

Wumpus models

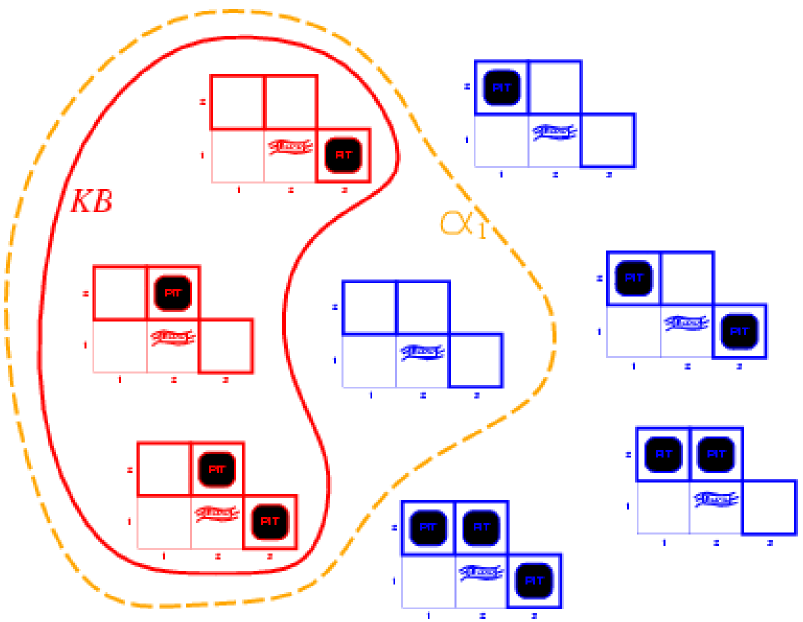
- α_1 = "[1,2] is safe" ??



KB = wumpus-world rules + observations

Wumpus models

- α_1 = "[1,2] is safe“ ??

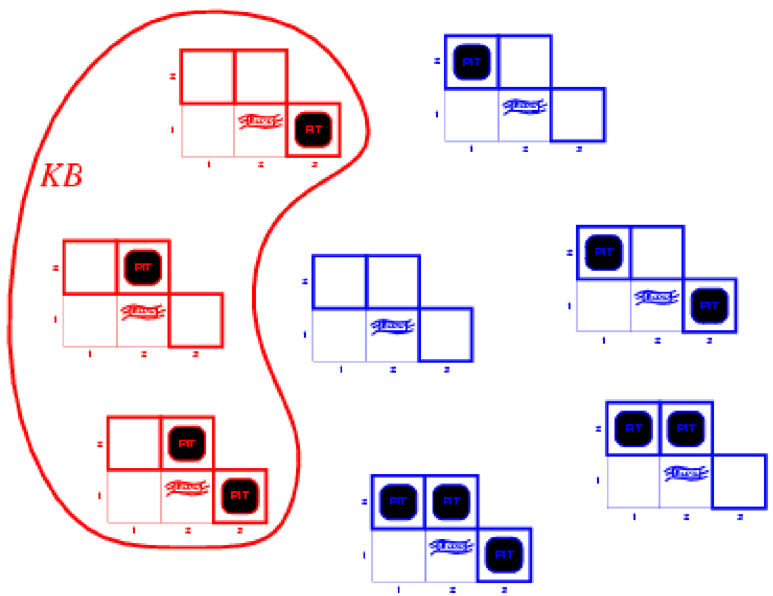


KB = wumpus-world rules + observations

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

Wumpus models

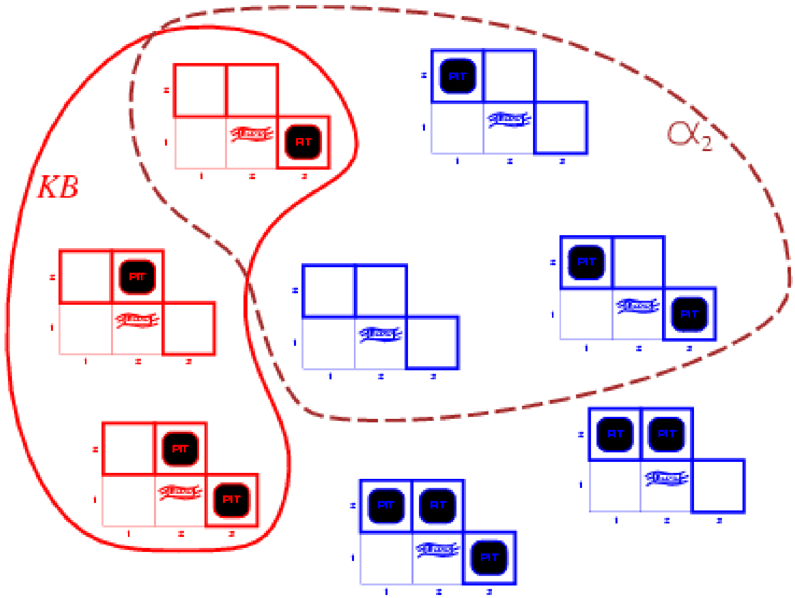
- α_2 = "[2,2] is safe“ ??



KB = wumpus-world rules + observations

Wumpus models

- α_2 = "[2,2] is safe“ ??



KB = wumpus-world rules + observations

α_2 = "[2,2] is safe", $KB \models \alpha_2$

Inference

- $KB \models_i \alpha$ = sentence α can be derived from KB by an inference algorithm "i"
  - *Needle in the haystack*

- Soundness: "i" is sound if whenever $KB \models_i \alpha$, it is also true that $KB \models \alpha$

- Completeness: "i" is complete if whenever $KB \models \alpha$, it is also true that $KB \models_i \alpha$

Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas

- The proposition symbols P_1, P_2 , etc . are sentences
- Complex sentences are constructed from simpler sentences using parentheses and logical connectives
  - 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.	$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
	false	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.,

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

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Wumpus world sentences

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

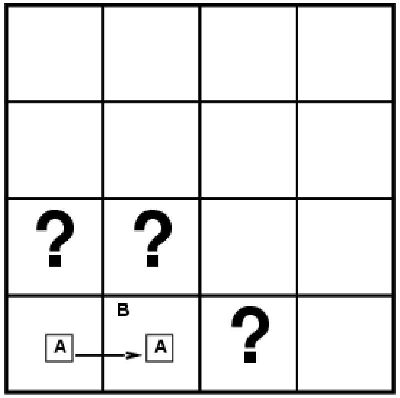
Let $B_{i,j}$ be true if there is a breeze in [i, j].

Knowledge base:

$\neg P_{1,1}$

$\neg B_{1,1}$

$B_{2,1}$



- "A square iz breezy iff there is a pit in a neighboring square"

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

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

Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
$false$	$false$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$false$	$false$	$false$	$false$	$false$	$true$	$false$	$true$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$false$	$true$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$true$	$false$	$false$	$false$	$false$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$false$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$true$	$false$	$false$	$false$	$true$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$true$	$true$	$true$	$true$	$true$	$true$	$true$	$false$	$false$

Inference by enumeration

- Depth-first enumeration of all models, 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 ← a list of the proposition symbols in KB and α
   return TT-CHECK-ALL(KB, α, symbols, { })
--------------------------------------------------------------------------------
function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
   if EMPTY?(symbols) then
      if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
      else return true //  when KB is false, always return true
   else do
      P ← FIRST(symbols)
      rest ← REST(symbols)
      return (TT-CHECK-ALL(KB, α, rest, model ∪ {P = true})
              and
              TT-CHECK-ALL(KB, α, rest, model ∪ {P = false }))
```

- For *n* symbols, time complexity is $O(2^n)$, space complexity is $O(n)$

Propositional Theorem Proving

- A sequence of applications of inference rules

- 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 algorithm
    - Typically require transformation of sentences into a normal form

  - Model checking
    - truth table enumeration (always exponential in *n*)
    - improved backtracking
    - heuristic search in model space (sound but incomplete)
      e.g., min-conflicts-like hill-climbing algorithms

Logical equivalence

- Two sentences are logically equivalent, iff true in same models: $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$

$$
\begin{aligned}
(\alpha \land \beta) &\equiv (\beta \land \alpha) && \text{commutativity of } \land \\
(\alpha \lor \beta) &\equiv (\beta \lor \alpha) && \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) &\equiv (\alpha \land (\beta \land \gamma)) && \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) &\equiv (\alpha \lor (\beta \lor \gamma)) && \text{associativity of } \lor \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \lor \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \land \beta) &\equiv (\neg\alpha \lor \neg\beta) && \text{de Morgan} \\
\neg(\alpha \lor \beta) &\equiv (\neg\alpha \land \neg\beta) && \text{de Morgan} \\
(\alpha \land (\beta \lor \gamma)) &\equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) && \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) &\equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) && \text{distributivity of } \lor \text{ over } \land
\end{aligned}
$$

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

Reductio ad absurdum (reduction to an absurd thing)
Proof by contradiction

Reasoning Patterns in Propositional Logic

- Modus Ponens
  - $\dfrac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$
- And elimination
  - $\dfrac{\alpha \wedge \beta}{\alpha}$

Resolution

- An inference algorithm used with search algorithms

Conjunctive Normal Form (CNF)

conjunction of disjunctions of literals or clauses

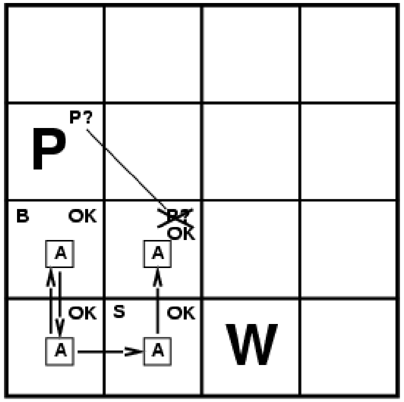
e.g., $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$

- Resolution inference rule (for CNF):

$$\frac{\ell_1 \lor \ldots \lor \ell_k, \qquad m_1 \lor \ldots \lor m_n}{\ell_1 \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \lor m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n}$$

where ℓ_i and m_j are complementary literals

e.g., $$\frac{P_{1,3} \lor P_{2,2}, \qquad \neg P_{2,2}}{P_{1,3}}$$



- 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 ($\wedge$ over $\vee$) and flatten:
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

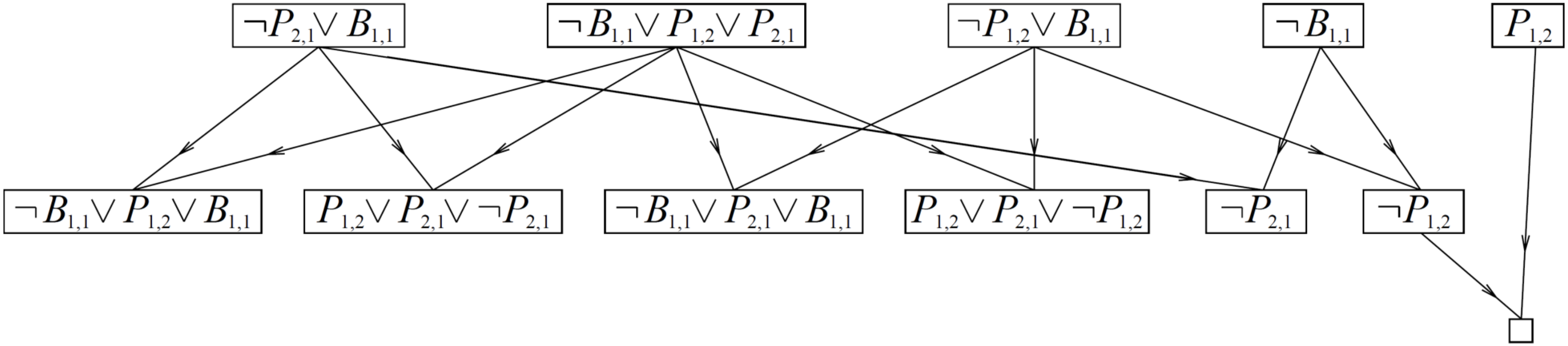
Resolution algorithm

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

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

   clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
   new ← { }
   loop do
      for each pair of clauses Ci, Cj in clauses do
         resolvents ← PL-RESOLVE(Ci, Cj)
         if resolvents contains the empty clause then return true
         new ← new ∪ resolvents
      if new ⊆ clauses then return false
      clauses ← clauses ∪ new
```

Resolution example

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



Forward and backward chaining

- Horn Form (restricted)

  KB = conjunction of Horn clauses

  – Horn clause
    - proposition symbol; or
    - (conjunction of symbols) $\Rightarrow$ symbol

  – e.g., $(C \wedge D \Rightarrow B)$

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

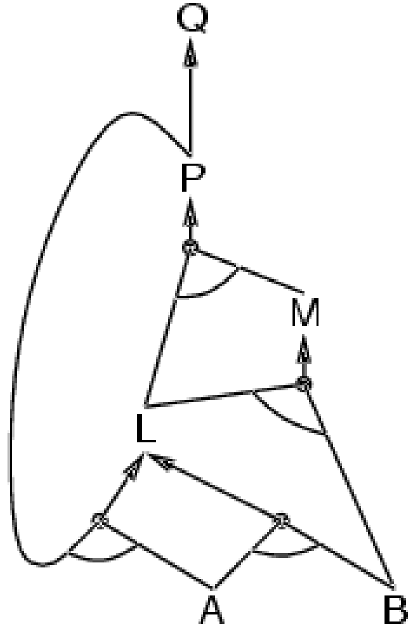
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

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

Forward chaining

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

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

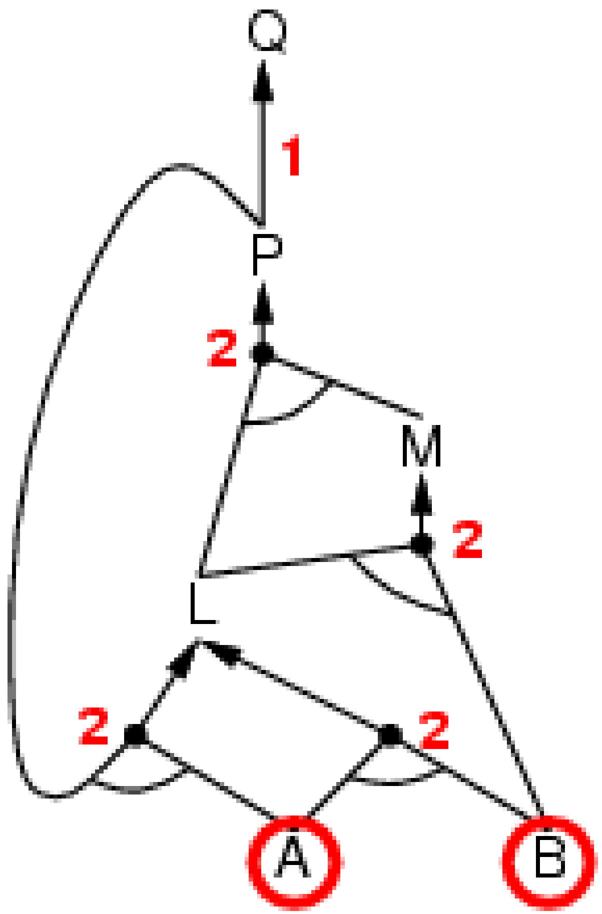


Forward chaining algorithm

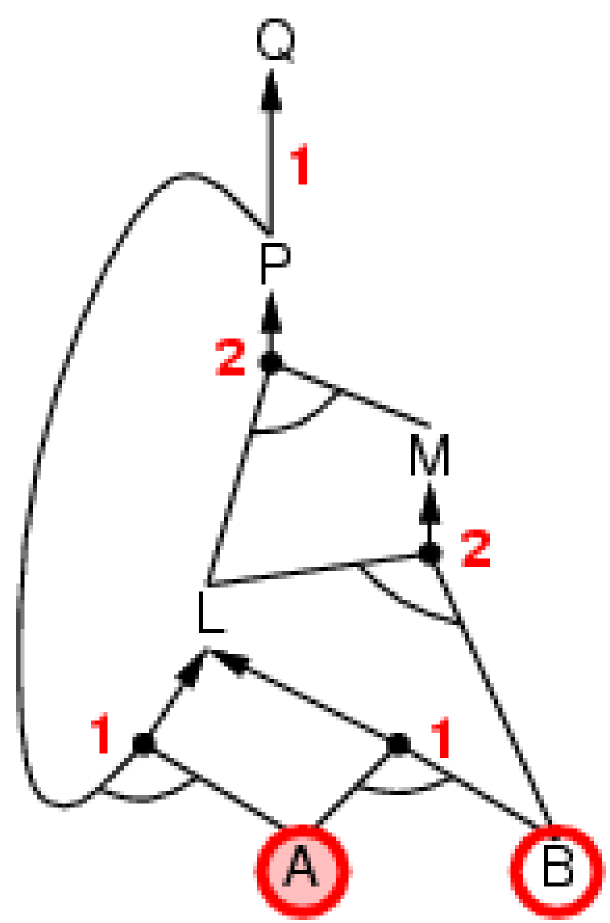
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB

  while agenda is not empty do
     p ← POP(agenda)
     if p = q then return true
     if inferred[p] = false then
         inferred[p] ← true
         for each clause c in KB where p is in c.PREMISE do
             decrement count[c]
             if count[c] = 0 then add c.CONCLUSION to agenda
  return false
```

- Forward chaining is sound and complete for Horn KB

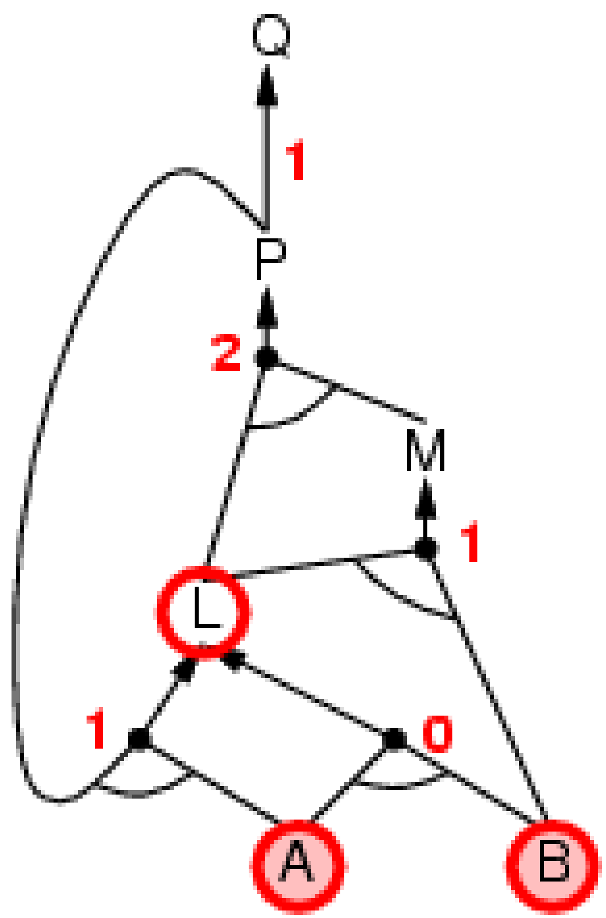
Forward chaining example



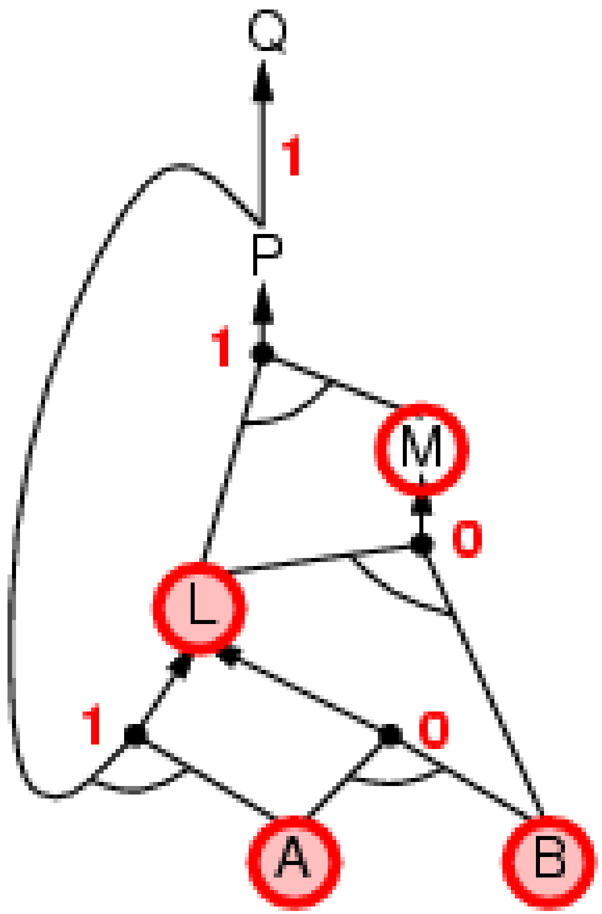
Forward chaining example



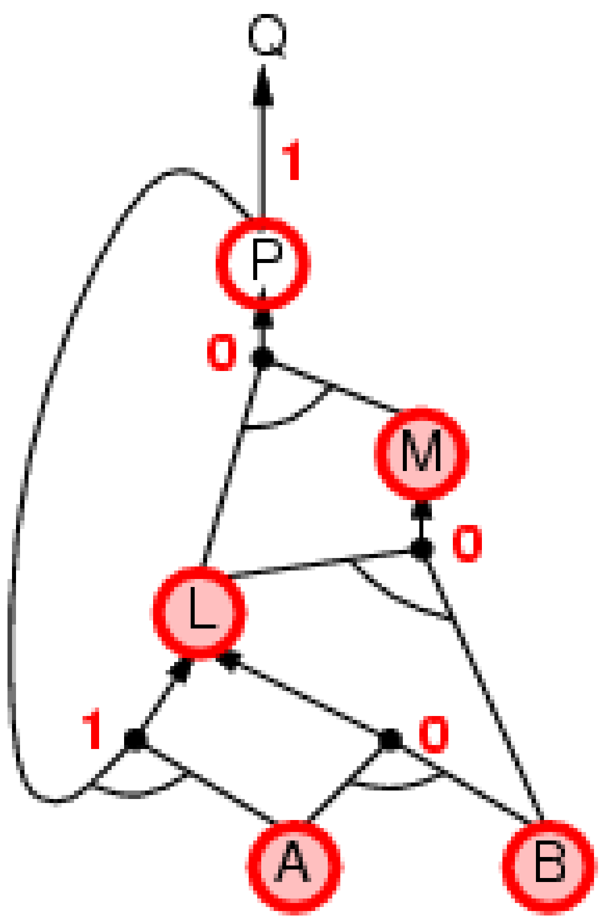
Forward chaining example



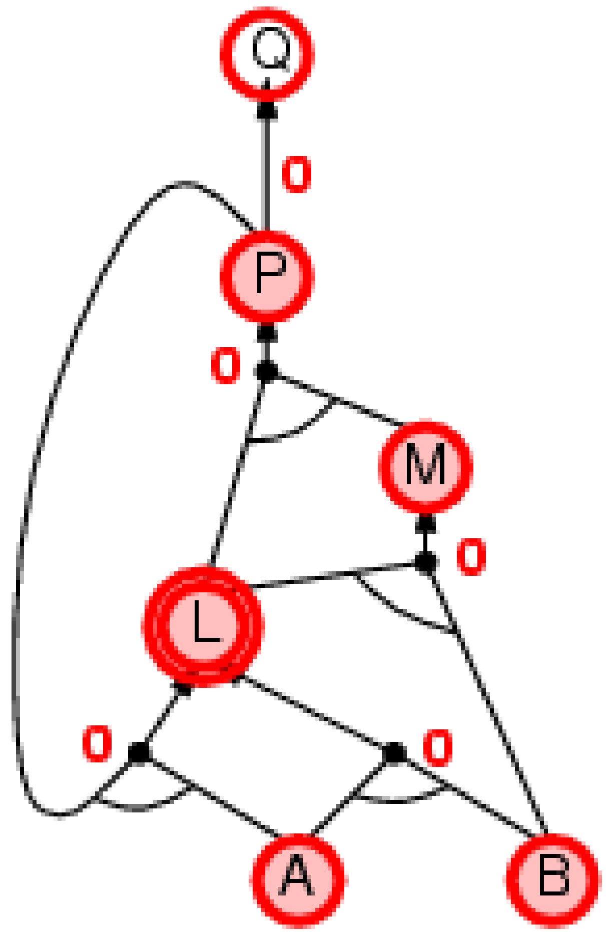
Forward chaining example



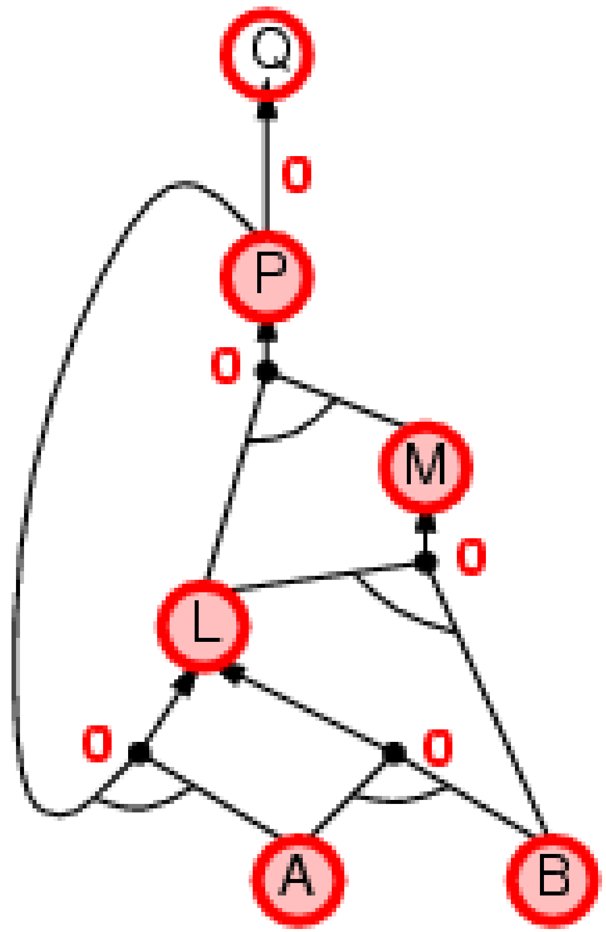
Forward chaining example



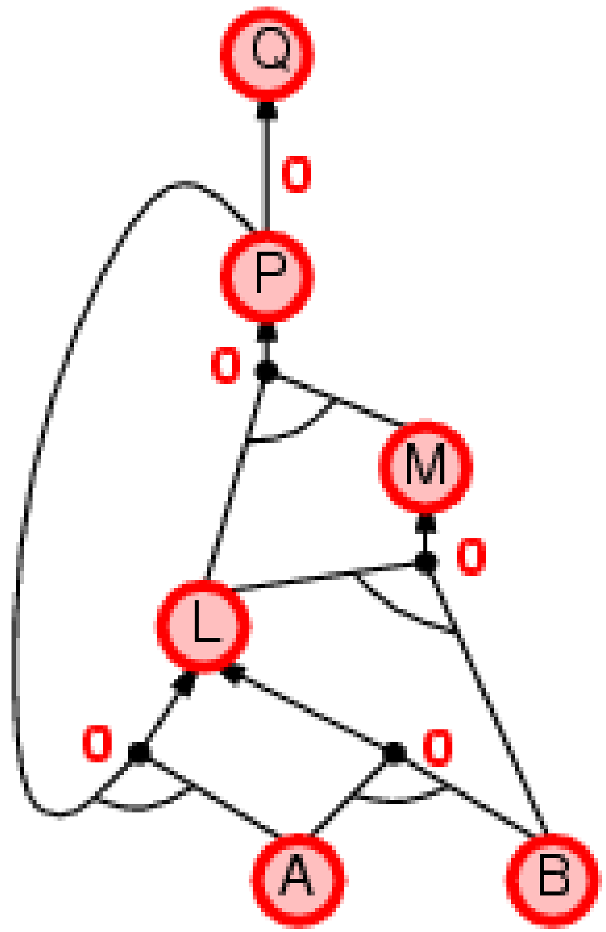
Forward chaining example



Forward chaining example



Forward chaining example



Backward chaining

Idea: work backwards from the query *q*:

to prove *q* by BC,

check if *q* is known already, or

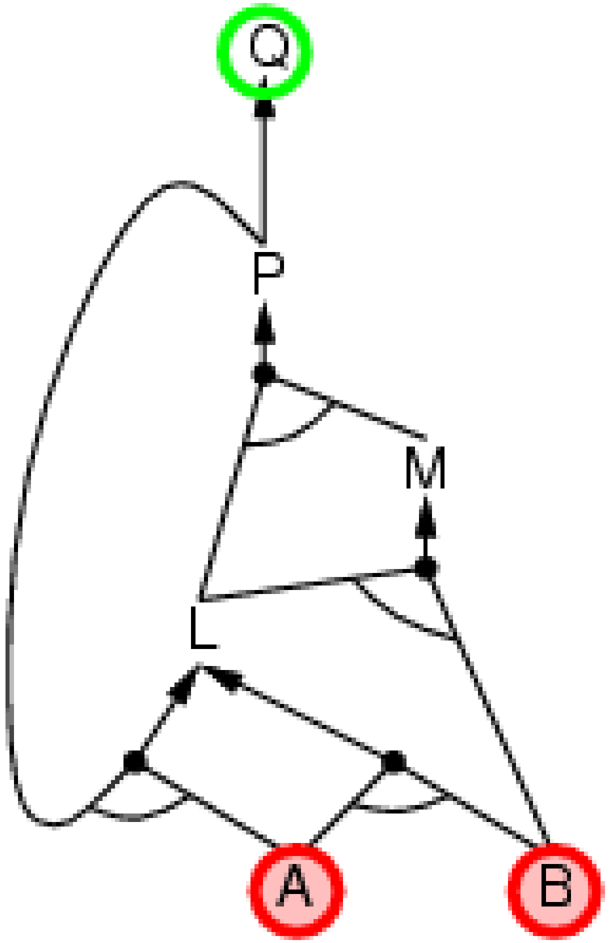
prove by BC all premises of some rule concluding *q*

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

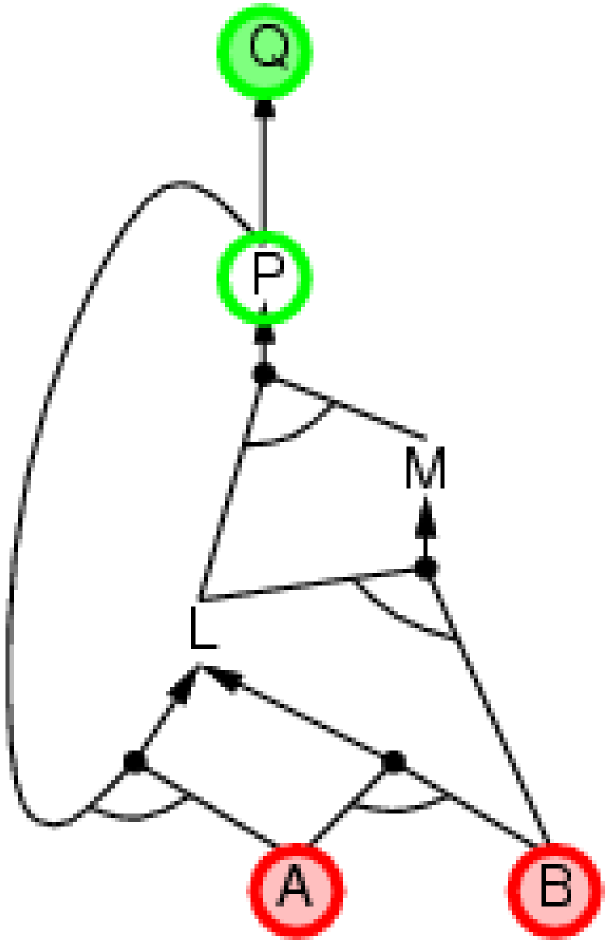
Avoid repeated work: check if new subgoal

- has already been proved true, or
- has already failed

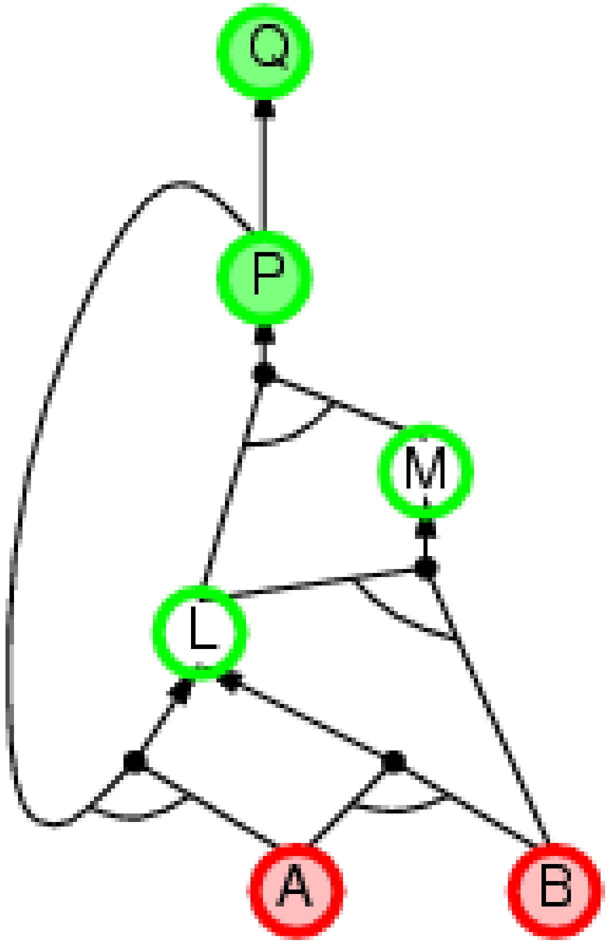
Backward chaining example



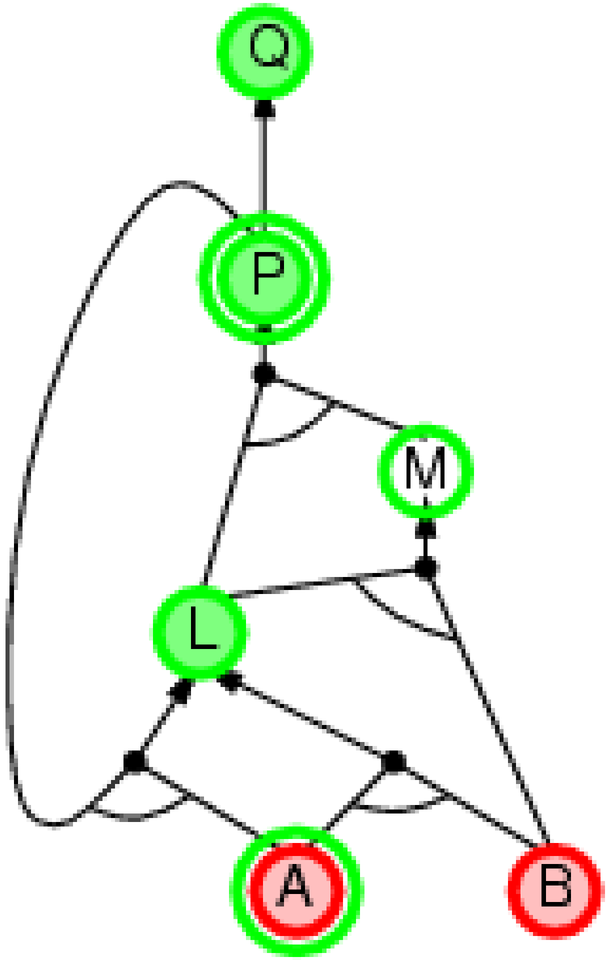
Backward chaining example



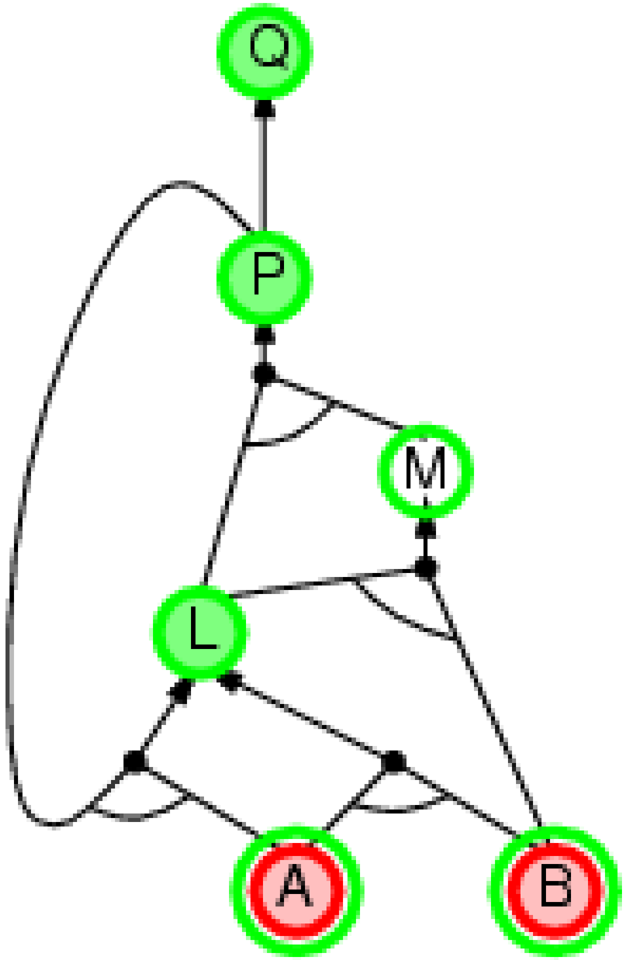
Backward chaining example



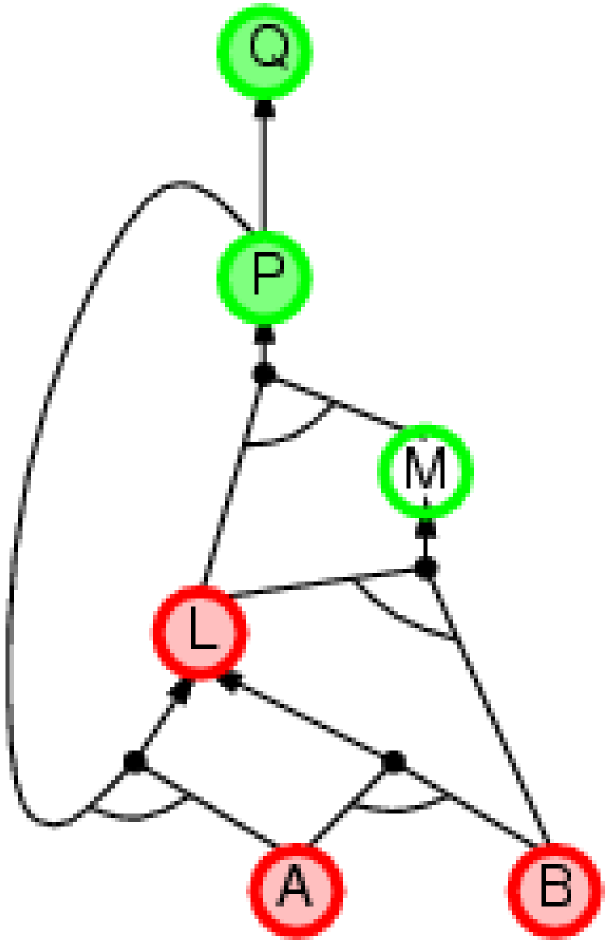
Backward chaining example



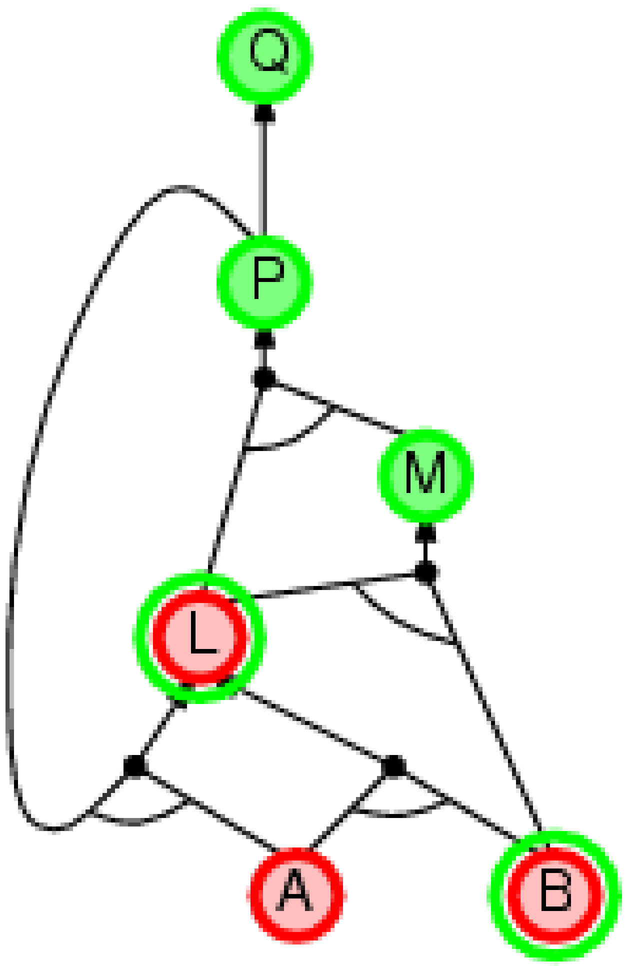
Backward chaining example



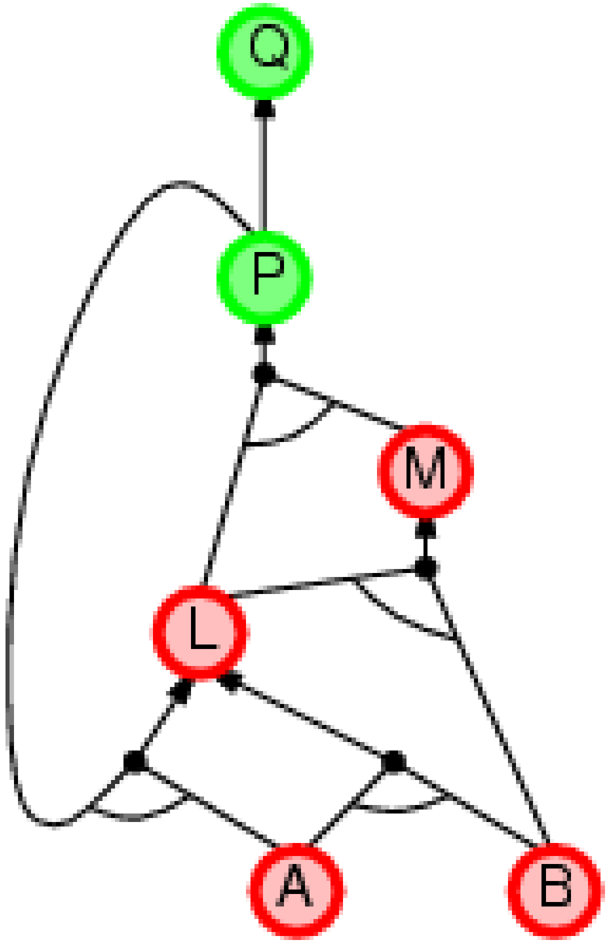
Backward chaining example



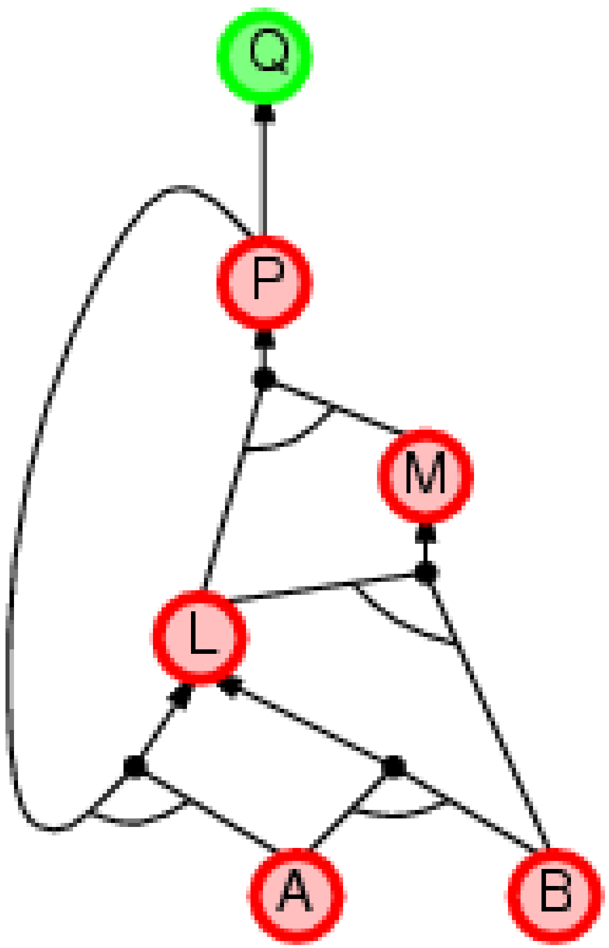
Backward chaining example



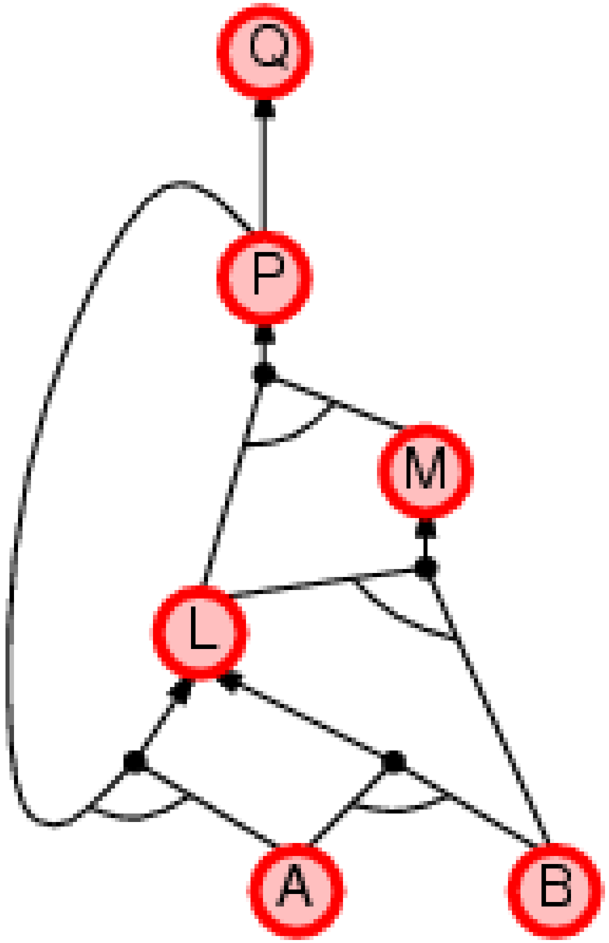
Backward chaining example



Backward chaining example



Backward chaining example



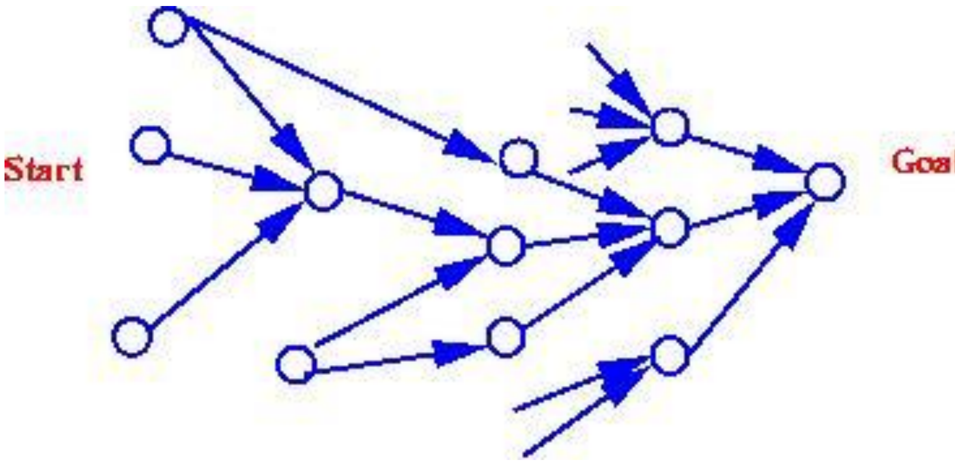
Forward vs. backward chaining

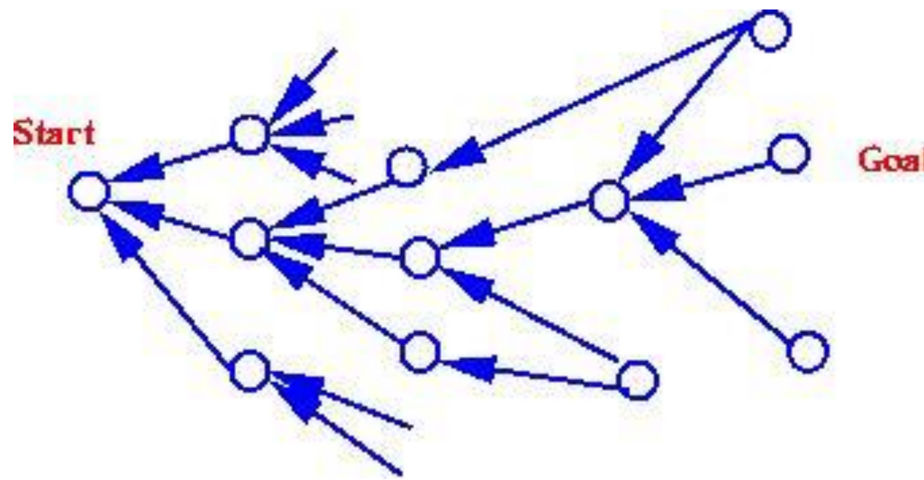
- FC is data-driven, automatic, unconscious processing
  - e.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal

- BC is goal-driven, appropriate for problem-solving
  - Where are my keys?
  - How can I get grade AA from AI?

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

Forward vs. backward chaining





Inference-based agents in the wumpus world

A wumpus-world agent using propositional logic:

$\neg P_{1,1}$

$\neg W_{1,1}$

$B_{x,y} \Leftrightarrow (P_{x,y+1} \lor P_{x,y-1} \lor P_{x+1,y} \lor P_{x-1,y})$

$S_{x,y} \Leftrightarrow (W_{x,y+1} \lor W_{x,y-1} \lor W_{x+1,y} \lor W_{x-1,y})$

$W_{1,1} \lor W_{1,2} \lor \ldots \lor W_{4,4}$

$\neg W_{1,1} \lor \neg W_{1,2}$

$\neg W_{1,1} \lor \neg W_{1,3}$

…

$\Rightarrow$ 64 distinct proposition symbols, 155 sentences

Expressiveness limitation of propositional logic

- KB contains "physics" sentences for every single square

- $L_{1,1} \wedge FacingRight \wedge Forward \Rightarrow L_{2,1}$

- For every time *t* and every location [*x,y*],

$$L_{x,y}{}^{t} \wedge FacingRight^{t} \wedge Forward^{t} \Rightarrow L_{x+1,y}{}^{t+1}$$

- Lacks the expressive power to deal with time, space and universal patterns of relationships among objects

function HYBRID-WUMPUS-AGENT(*percept*) **returns** an *action*
inputs: *percept*, a list, [*stench*,*breeze*,*glitter*,*bump*,*scream*]
persistent: *KB*, a knowledge base, initially the atemporal "wumpus physics"
t, a counter, initially 0, indicating time
plan, an action sequence, initially empty

TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
TELL the *KB* the temporal "physics" sentences for time *t*
$safe \leftarrow \{[x, y] : \text{ASK}(KB, OK^t_{x,y}) = true\}$
if ASK$(KB, Glitter^t) = true$ **then**
$plan \leftarrow [Grab] + \text{PLAN-ROUTE}(current, \{[1,1]\}, safe) + [Climb]$
if *plan* is empty **then**
$unvisited \leftarrow \{[x, y] : \text{ASK}(KB, L^{t'}_{x,y}) = false$ for all $t' \leq t\}$
$plan \leftarrow \text{PLAN-ROUTE}(current, unvisited \cap safe, safe)$
if *plan* is empty and ASK$(KB, HaveArrow^t) = true$ **then**
$possible\_wumpus \leftarrow \{[x, y] : \text{ASK}(KB, \neg W_{x,y}) = false\}$
$plan \leftarrow \text{PLAN-SHOT}(current, possible\_wumpus, safe)$
if *plan* is empty **then** // no choice but to take a risk
$not\_unsafe \leftarrow \{[x, y] : \text{ASK}(KB, \neg OK^t_{x,y}) = false\}$
$plan \leftarrow \text{PLAN-ROUTE}(current, unvisited \cap not\_unsafe, safe)$
if *plan* is empty **then**
$plan \leftarrow \text{PLAN-ROUTE}(current, \{[1, 1]\}, safe) + [Climb]$
$action \leftarrow \text{POP}(plan)$
TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
$t \leftarrow t + 1$
return *action*

---

function PLAN-ROUTE(*current*,*goals*,*allowed*) **returns** an action sequence
inputs: *current*, the agent's current position
goals, a set of squares; try to plan a route to one of them
allowed, a set of squares that can form part of the route

$problem \leftarrow \text{ROUTE-PROBLEM}(current, goals, allowed)$
return A*-GRAPH-SEARCH(*problem*)

Pros and cons of propositional logic

- ☺ Propositional logic is declarative
- ☺ Propositional logic allows partial/disjunctive/negated information
  - (unlike most data structures and databases)
- ☺ Propositional logic is compositional:
  - meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$
- ☺ Meaning in propositional logic is context-independent
  - (unlike natural language, where meaning depends on context)

- ☹ Propositional logic has very limited expressive power
  - (unlike natural language)
  - e.g., cannot say "pits cause breezes in adjacent squares“
    - except by writing one sentence for each square

BLG435E
Artificial Intelligence

Lecture 7: First-Order Logic

First-order logic

- Propositional logic assumes the world contains facts,
- First-order logic (like natural language) assumes the world contains
  - Objects: people, houses, numbers, colors, baseball games, wars, …
  - Relations: red, round, prime, brother of, bigger than, part of, comes between, …
  - Functions: father of, best friend, one more than, plus, …

- “Squares neighboring the wumpus are smelly”

Syntax of FOL: Basic elements

- Constants: KingJohn, 2,...
- Predicates: Brother, >,...
- Functions: Sqrt, LeftLegOf,...
- Variables: x, y, a, b,...
- Connectives: $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$
- Equality: =
- Quantifiers: $\forall$, $\exists$

Atomic sentences

Term is a logical expression that refers to an object.

Term = *function* ($term_1$,...,$term_n$)
or *constant* or *variable*

Atomic sentence = *predicate* ($term_1$,...,$term_n$)
or $term_1$ = $term_2$

<represent facts>

- e.g., *Married (Father(Richard), Mother(Richard))*

Complex sentences

- Complex sentences are made from atomic sentences using connectives

$$\neg S,\ S_1 \wedge S_2,\ S_1 \vee S_2,\ S_1 \Rightarrow S_2,\ S_1 \Leftrightarrow S_2,$$

e.g. $Sibling(KingJohn, Richard) \Rightarrow Sibling(Richard, KingJohn)$

Truth in first-order logic

- Sentences are true with respect to a model and an interpretation
- Model contains objects (domain elements) and relations among them

- Interpretation specifies:

constant symbols	→	objects
predicate symbols	→	relations
function symbols	→	functional relations

- An atomic sentence $predicate(term_1,...,term_n)$ is true iff the objects referred to by $term_1,...,term_n$ are in the relation referred to by *predicate*

Universal quantification

- $\forall$<*variables*> <*sentence*> (For all)

Everyone at ITU is smart:

$\forall$x At(x, ITU) $\Rightarrow$ Smart(x)

- $\forall$x *P* is true in a model *m* iff *P* is true with *x* being each possible object in the model
  - Roughly speaking, equivalent to the conjunction of instantiations of *P*

At(KingJohn, ITU) $\Rightarrow$ Smart(KingJohn) $\wedge$
At(Richard, ITU) $\Rightarrow$ Smart(Richard) $\wedge$
At(ITU, ITU) $\Rightarrow$ Smart(ITU) $\wedge$ ...

A common mistake to avoid

- Typically, $\Rightarrow$ is the main connective with $\forall$
- Common mistake: using $\wedge$ as the main connective with $\forall$:

$\forall x\ At(x,ITU) \wedge Smart(x)$

means "Everyone is at ITU and everyone is smart"

Existential quantification

- $\exists$<*variables*> <*sentence*> (There exists an x such that)

  We can name about some object without naming it

- Someone at ITU is smart:
- $\exists x$ At(x,ITU) $\wedge$ Smart(x)

- $\exists x$ *P* is true in a model *m* iff *P* is true with *x* being some possible object in the model
  - Roughly speaking, equivalent to the disjunction of instantiations of *P*

At(KingJohn,ITU) $\wedge$ Smart(KingJohn) $\vee$
At(Richard,ITU) $\wedge$ Smart(Richard) $\vee$
At(ITU,ITU) $\wedge$ Smart(ITU) $\vee$ ...

Another common mistake to avoid

- Typically, $\wedge$ is the main connective with $\exists$
- Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

$$\exists x \; At(x,ITU) \Rightarrow Smart(x)$$

is true if there is anyone who is not at ITU!

Properties of quantifiers

- $\forall x \, \forall y$ is the same as $\forall y \, \forall x$
- $\exists x \, \exists y$ is the same as $\exists y \, \exists x$

- $\exists x \, \forall y$ is not the same as $\forall y \, \exists x$
- $\exists y \, \forall x$ Loves(x,y)
  - "There is someone who is loved by everyone"
- $\forall x \, \exists y$ Loves(x,y)
  - "Everybody loves somebody"

Quantifier duality: each can be expressed using the other

$\forall x$ Likes(x,IceCream) $\qquad \neg\exists x \, \neg$Likes(x,IceCream)

$\exists x$ Likes(x,Broccoli) $\qquad \neg\forall x \, \neg$Likes(x,Broccoli)

Equality

- $term_1 = term_2$ is true under a given interpretation if and only if $term_1$ and $term_2$ refer to the same object
  - Father(John) = Henry

- e.g., definition of *Sibling* in terms of *Parent*:

$$\forall x,y\ Sibling(x,y) \Leftrightarrow [\neg(x = y) \land \exists m,f\ \neg (m = f) \land Parent(m,x) \land Parent(f,x) \land Parent(m,y) \land Parent(f,y)]$$

Using FOL

The kinship domain:

- Brothers are siblings

  $\forall x,y\ Brother(x,y) \Leftrightarrow Sibling(x,y)$

- One's mother is one's female parent

  $\forall m,c\ Mother(c) = m \Leftrightarrow (Female(m) \wedge Parent(m,c))$

- "Sibling" is symmetric

  $\forall x,y\ Sibling(x,y) \Leftrightarrow Sibling(y,x)$

Using FOL

The set domain:

- $\forall s\ Set(s) \Leftrightarrow (s = \{\}) \lor (\exists x, s_2\ \ Set(s_2) \land s = \{x|s_2\})$
- $\neg\exists x,s\ \ \{x|s\} = \{\}$
- $\forall x,s\ \ x \in s \Leftrightarrow s = \{x|s\}$
- $\forall x,s\ \ x \in s \Leftrightarrow [\ \exists y, s_2\ \ (s = \{y|s_2\} \land (x = y \lor x \in s_2))]$
- $\forall s_1,s_2\ \ s_1 \subseteq s_2 \Leftrightarrow (\forall x\ \ x \in s_1 \Rightarrow x \in s_2)$
- $\forall s_1,s_2\ \ (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \land s_2 \subseteq s_1)$
- $\forall x,s_1,s_2\ \ x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \land x \in s_2)$
- $\forall x,s_1,s_2\ \ x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \lor x \in s_2)$

Knowledge base for the wumpus world

- Suppose a wumpus-world agent is using a FOL KB and perceives a smell and a breeze (but no glitter) at *t=5*:

  `Tell`(KB,Percept([Stench,Breeze,None],5))
  `Ask`(KB,∃a BestAction(a,5))

- i.e., does the KB entail some best action at *t=5*?

- Answer: *Yes*, {*a/Shoot*} ← substitution (binding list)

Knowledge base for the wumpus world

- Given a sentence S and a substitution σ,
- σ denotes the result of plugging σ into S; e.g.,

  S = Smarter(x,y)

  σ = {x/Hillary, y/Bill}

  Sσ = Smarter(Hillary,Bill)

- `Ask`(KB,S) returns some/all σ such that KB $\models$ σ
- Perception
  - $\forall$t,s,b Percept([s,b,Glitter],t) $\Rightarrow$ Glitter(t)
- Reflex
  - $\forall$t Glitter(t) $\Rightarrow$ BestAction(Grab,t)

Deducing hidden properties

- $\forall x,y,a,b$ *Adjacent*$([x,y],[a,b]) \Leftrightarrow$
  $[a,b] \in \{[x+1,y], [x-1,y],[x,y+1],[x,y-1]\}$

- Home(Wumpus), unary predicate for Wumpus location

Properties of squares:

- $\forall s,t$ *At*$(Agent,s,t) \wedge Breeze(t) \Rightarrow Breezy(s)$

Deducing hidden properties

Squares are breezy near to a pit:

- Diagnostic rule
  - infer cause from effect

$$\forall s \; Breezy(s) \Rightarrow \exists r \;\; Adjacent(r,s) \wedge Pit(r)$$

- Causal rule
  - infer effect from cause

$$\forall r \; Pit(r) \Rightarrow [\forall s \; Adjacent(r,s) \Rightarrow Breezy(s)]$$

- Systems that use casual rules are called model-based reasoning systems

Knowledge engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base

BLG435E
Artificial Intelligence

Lecture 8: Inference in First-Order Logic

Universal instantiation (UI)

- Every instantiation of a universally quantified sentence is entailed by it:

$$\frac{\forall v\ \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$

for any variable *v* and ground term *g*

e.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:

$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$

$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$

$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$

Existential instantiation (EI)

- For any sentence α, variable *v*, and constant symbol *k* that does not appear elsewhere in the knowledge base:

$$\frac{\exists v\ \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

- e.g., $\exists x\ Crown(x) \wedge OnHead(x,John)$ yields:

$$Crown(C_1) \wedge OnHead(C_1,John)$$

provided C_1 is a new constant symbol, called a Skolem constant (Thoralf Skolem)

Reduction to propositional inference

Suppose the KB contains just the following:

$\forall$x King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
King(John)
Greedy(John)
Brother(Richard,J ohn)

- Instantiating the universal sentence in all possible ways, we have:
  King(John) $\wedge$ Greedy(John) $\Rightarrow$ Evil(John)
  King(Richard) $\wedge$ Greedy(Richard) $\Rightarrow$ Evil(Richard)
  King(John)
  Greedy(John)
  Brother(Richard,John)

- The new KB is propositionalized: proposition symbols are:
  - King(John), Greedy(John), Evil(John), King(Richard), etc.

Reduction contd.

- Every FOL KB can be propositionalized so as to preserve entailment
- A ground sentence is entailed by new KB iff entailed by the original KB
- Idea: propositionalize KB and query, apply resolution, return result
- Problem: with function symbols, there are infinitely many ground terms,
  - e.g., *Father*(*Father*(*Father*(*John*)))

Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences

- e.g., from:

  $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

  $King(John)$

  $\forall y\ Greedy(y)$

  $Brother(Richard, John)$

- it seems obvious that *Evil*(*John*), but propositionalization produces lots of facts such as *Greedy*(*Richard*) that are irrelevant

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and ∀y *Greedy(y)*
- $\theta = \{x/John, y/John\}$ works
- $Unify(\alpha,\beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
Knows(John,x)	Knows(John,Jane)	
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and ∀y *Greedy(y)*
- θ = {x/John, y/John} works
- Unify(α,β) = θ if αθ = βθ
- Returns a unifier

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and ∀y *Greedy(y)*
- θ = {x/John, y/John} works
- Unify(α,β) = θ if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and ∀y *Greedy(y)*
- θ = {x/John, y/John} works
- Unify(α,β) = θ if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}
Knows(John,x)	Knows(x,OJ)	

Unification

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}
Knows(John,x)	Knows(x,OJ)	{fail}

Unification

- To unify *Knows(John,x)* and *Knows(y,z)*,
  θ = {y/John, x/z } or θ = {y/John, x/John, z/John}

- The first unifier is more general than the second.

- There is a single most general unifier (MGU) that is unique up to renaming of variables.
- MGU = { y/John, x/z }

The unification algorithm

```
function UNIFY(x, y, θ) returns a substitution to make x and y identical
   inputs: x, a variable, constant, list, or compound
           y, a variable, constant, list, or compound
           θ, the substitution built up so far

   if θ = failure then return failure
   else if x = y then return θ
   else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
   else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
   else if COMPOUND?(x) and COMPOUND?(y) then
       return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
   else if LIST?(x) and LIST?(y) then
       return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
   else return failure
```

The unification algorithm

function UNIFY-VAR(var, x, θ) **returns** a substitution
  inputs: var, a variable
    x, any expression
    θ, the substitution built up so far

  if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) **then return** failure
  else return add $\{var/x\}$ to θ

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

p_1' is *King*(*John*) p_1 is *King*(*x*)

p_2' is *Greedy*(*y*) p_2 is *Greedy*(*x*)

θ is {x/John,y/John} q is *Evil*(*x*)

SUBST(θ, q) is *Evil*(*John*)

Example knowledge base

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

- Prove that Colonel West is a criminal

Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:

American(x) $\wedge$ *Weapon(y)* $\wedge$ *Sells(x,y,z)* $\wedge$ *Hostile(z)* $\Rightarrow$ *Criminal(x)*

Nono … has some missiles:

$\exists$x Owns(Nono,x) $\wedge$ Missile(x):

Owns(Nono,M_1*) and Missile(*M_1*)*

... all of its missiles were sold to it by Colonel West

Missile(x) $\wedge$ *Owns(Nono,x)* $\Rightarrow$ *Sells(West,x,Nono)*

Example knowledge base contd.

Missiles are weapons:

$Missile(x) \Rightarrow Weapon(x)$

An enemy of America counts as "hostile“:

$Enemy(x,America) \Rightarrow Hostile(x)$

West, who is American …

$American(West)$

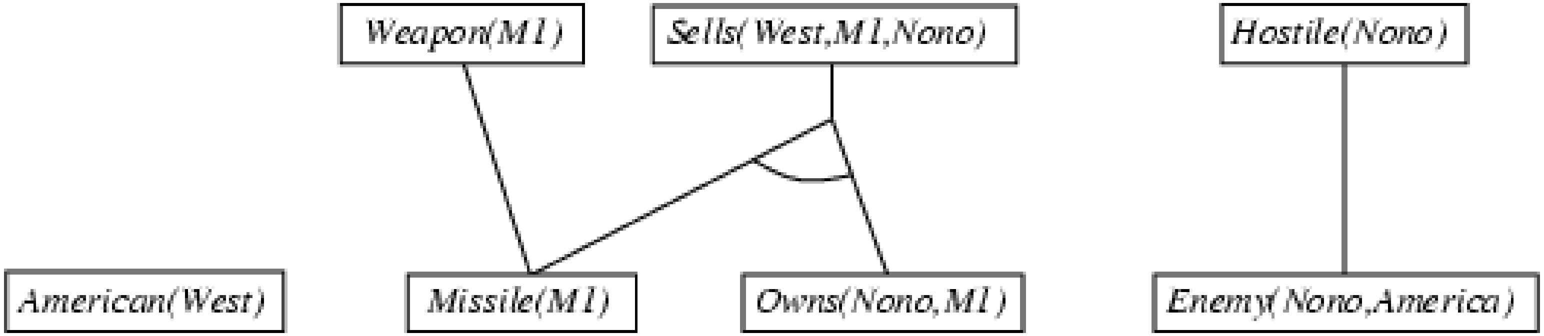
The country Nono, an enemy of America …

$Enemy(Nono,America)$

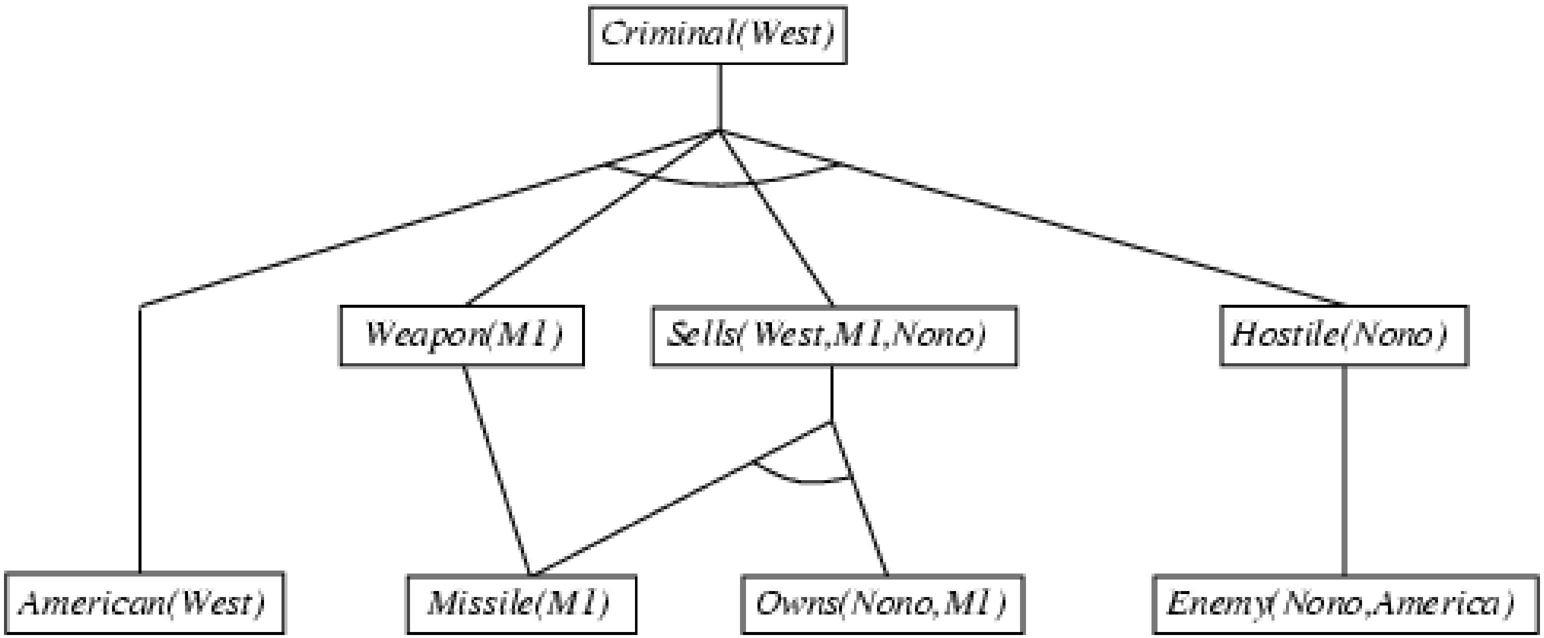
Forward chaining proof

$\boxed{American(West)}$ $\boxed{Missile(M1)}$ $\boxed{Owns(Nono, M1)}$ $\boxed{Enemy(Nono, America)}$

Forward chaining proof



Forward chaining proof



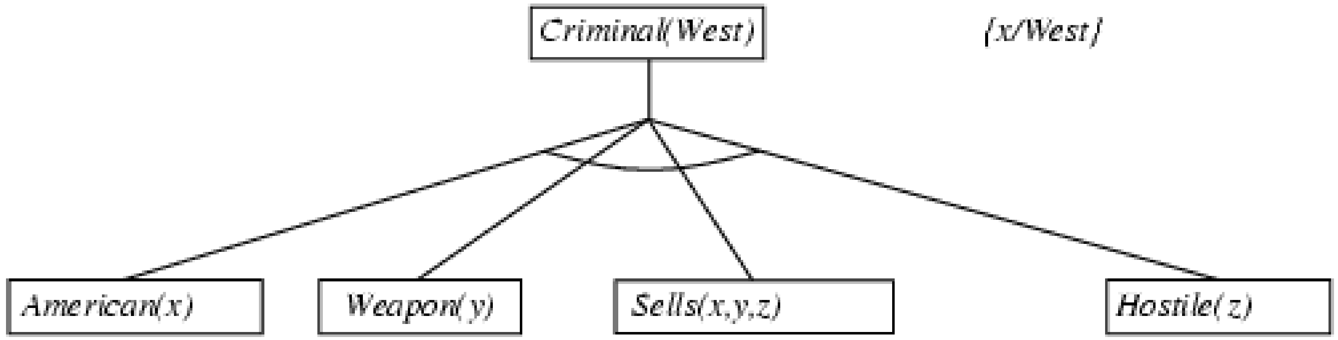
Properties of forward chaining

- Sound and complete for first-order definite clauses
- May not terminate in general if α is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable

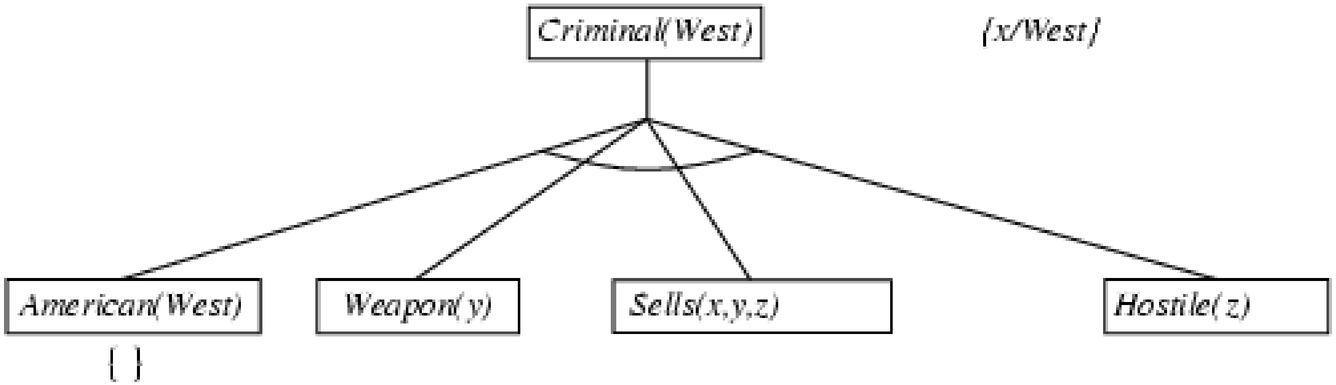
Backward chaining example

Criminal(West)

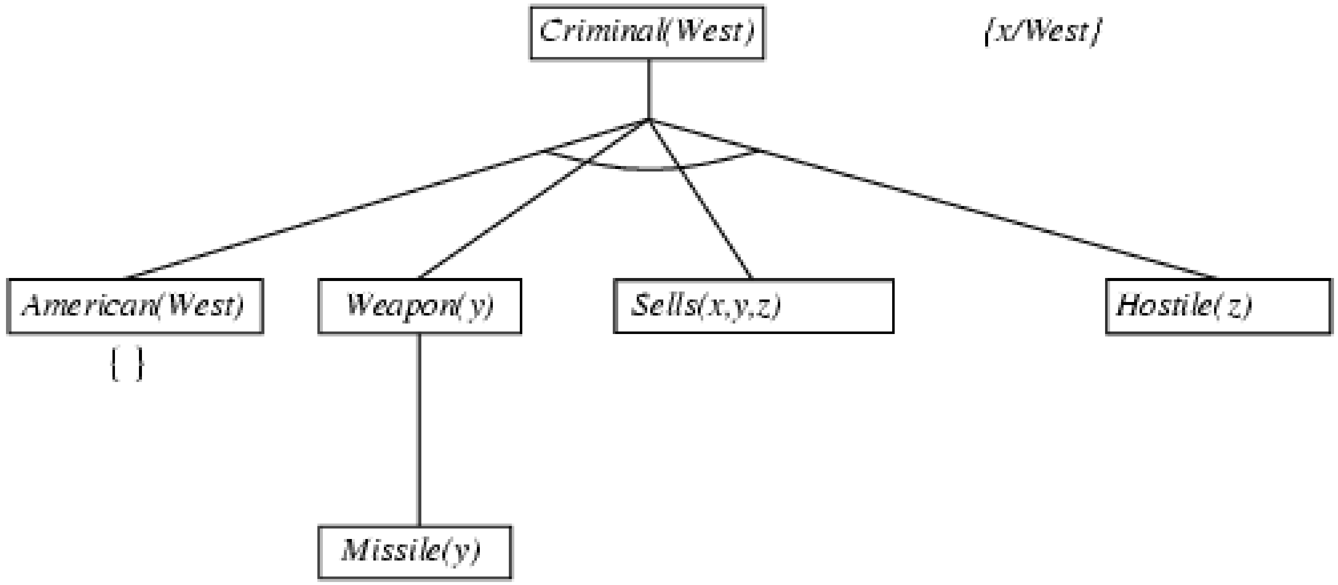
Backward chaining example



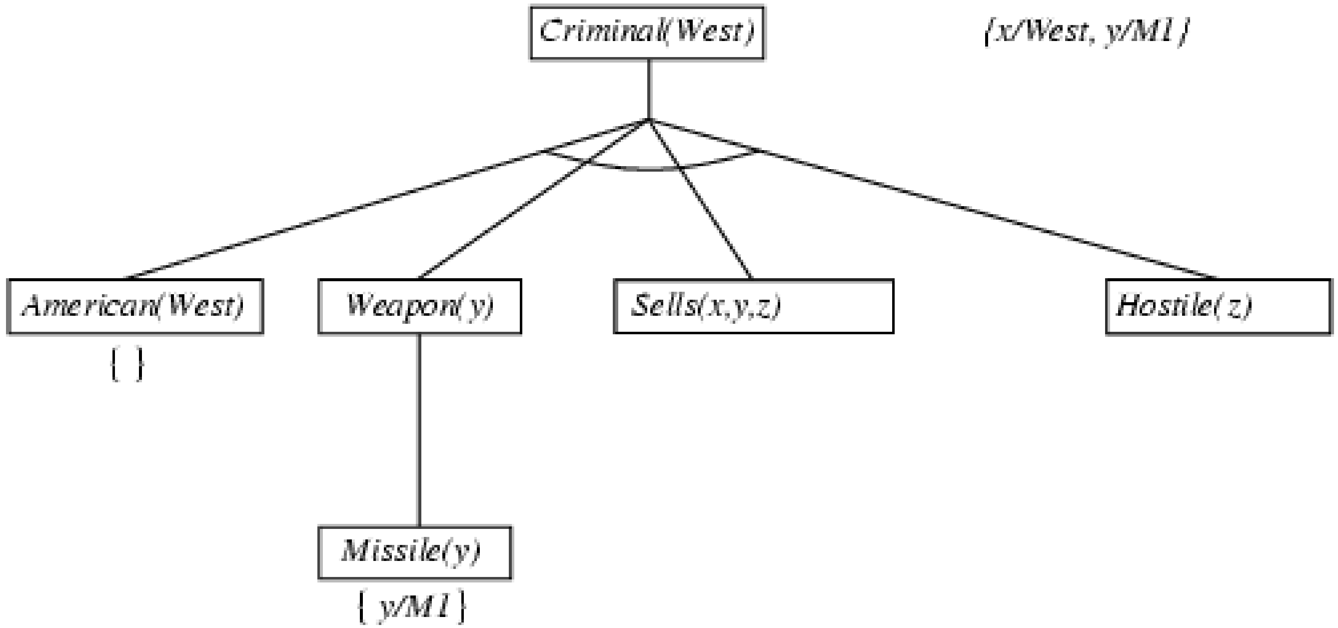
Backward chaining example



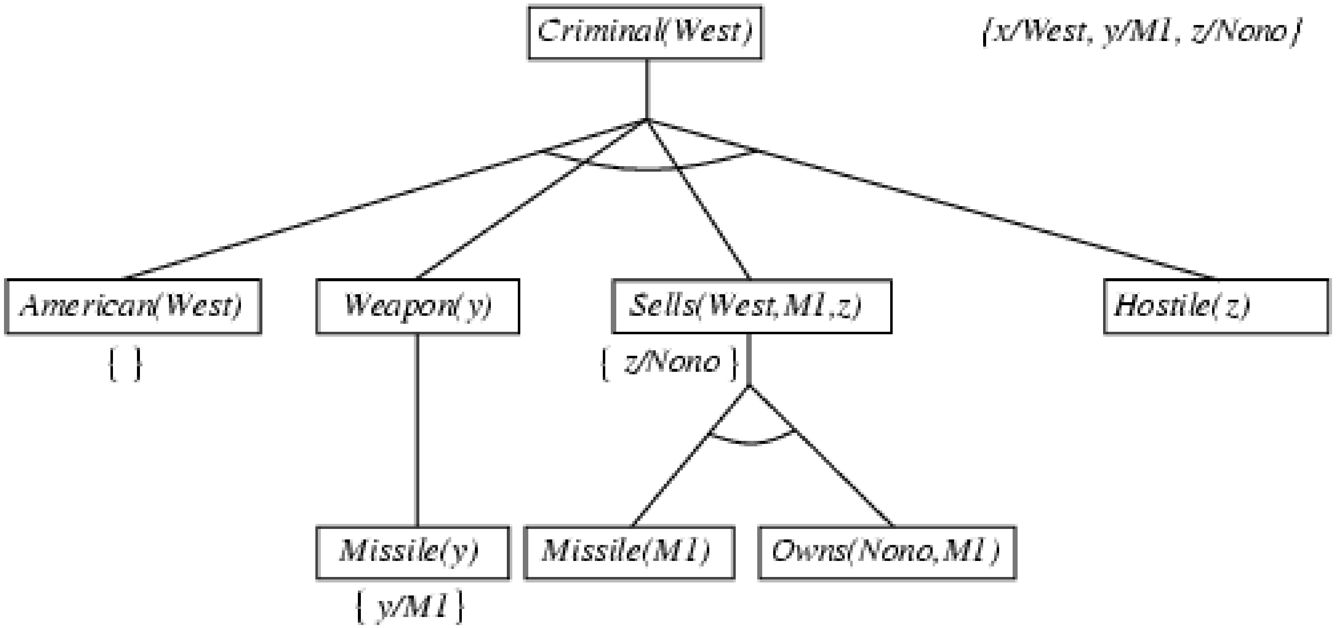
Backward chaining example



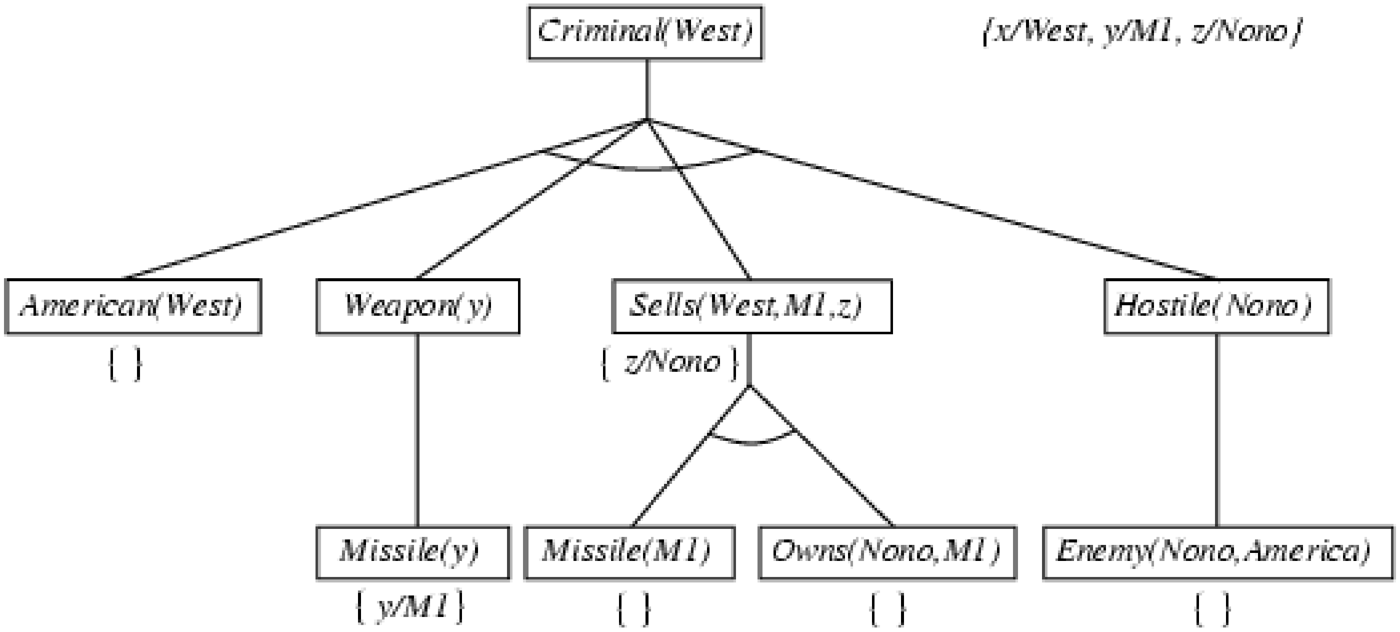
Backward chaining example



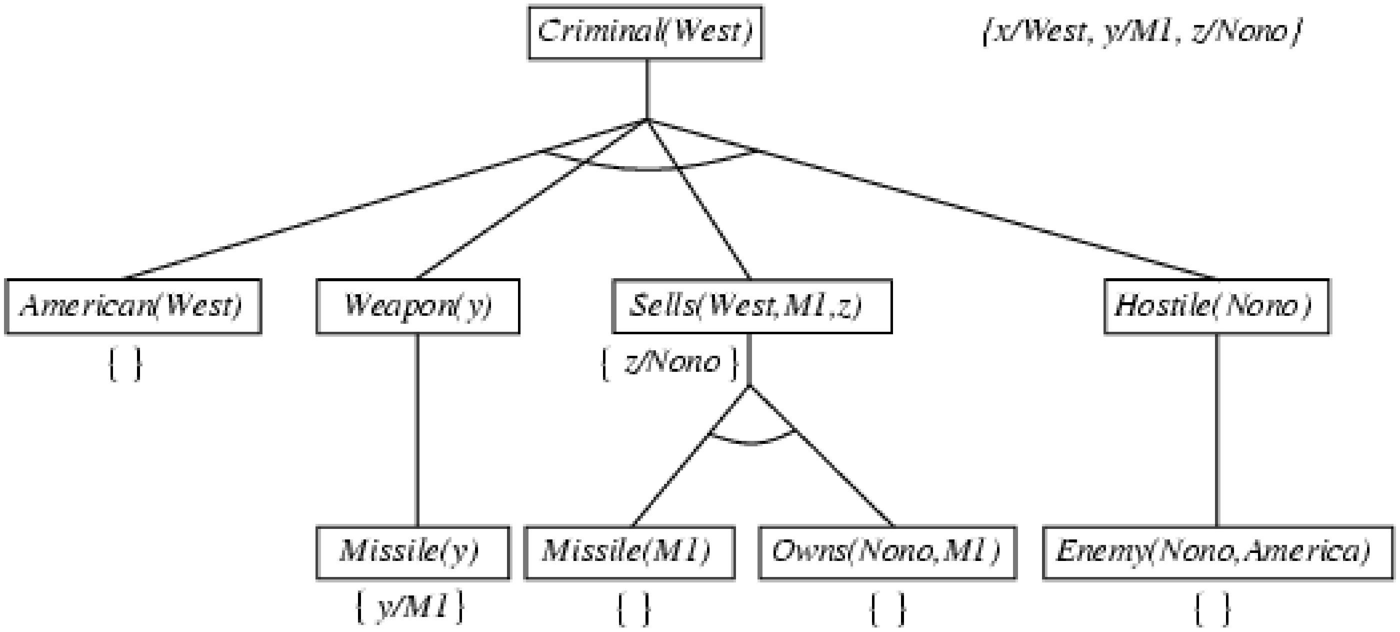
Backward chaining example



Backward chaining example



Backward chaining example



Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - $\Rightarrow$ fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - $\Rightarrow$ fix using caching of previous results (extra space)
- Widely used for logic programming

Resolution: brief summary

- Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\mathtt{Unify}(\ell_i, \neg m_j) = \theta$.

- The two clauses are assumed to be standardized apart so that they share no variables:

$$\frac{\neg Rich(x) \vee Unhappy(x) \qquad Rich(Ken)}{Unhappy(Ken)}$$

with $\theta = \{x/Ken\}$

Conversion to CNF

- Apply resolution steps to CNF(KB $\wedge \neg\alpha$); complete for FOL
- Everyone who loves all animals is loved by someone:

  $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

1. Eliminate biconditionals and implications

$\forall x\ [\neg\forall y\ \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\ Loves(y,x)]$

2. Move $\neg$ inwards:

$\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ Loves(y,x)]$

$\forall x\ [\exists y\ \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]$

$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]$

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different one

$\forall x\ [\exists y\ \mathit{Animal}(y) \wedge \neg \mathit{Loves}(x,y)] \vee [\exists z\ \mathit{Loves}(z,x)]$

4. Skolemize: a more general form of existential instantiation.

Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

$\forall x\ [\mathit{Animal}(F(x)) \wedge \neg \mathit{Loves}(x,F(x))] \vee \mathit{Loves}(G(x),x)$

5. Drop universal quantifiers:

$[\mathit{Animal}(F(x)) \wedge \neg \mathit{Loves}(x,F(x))] \vee \mathit{Loves}(G(x),x)$

6. Distribute $\vee$ over $\wedge$:

$[\mathit{Animal}(F(x)) \vee \mathit{Loves}(G(x),x)] \wedge [\neg \mathit{Loves}(x,F(x)) \vee \mathit{Loves}(G(x),x)]$

Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono … has some missiles:

$\exists x\ Owns(Nono,x) \wedge Missile(x)$:

$Owns(Nono,M_1)$ and $Missile(M_1)$

… all of its missiles were sold to it by Colonel West

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

Example knowledge base contd.

Missiles are weapons:

$Missile(x) \Rightarrow Weapon(x)$

An enemy of America counts as "hostile“:

$Enemy(x,America) \Rightarrow Hostile(x)$

West, who is American …

$American(West)$

The country Nono, an enemy of America …

$Enemy(Nono,America)$

Resolution proof: definite clauses