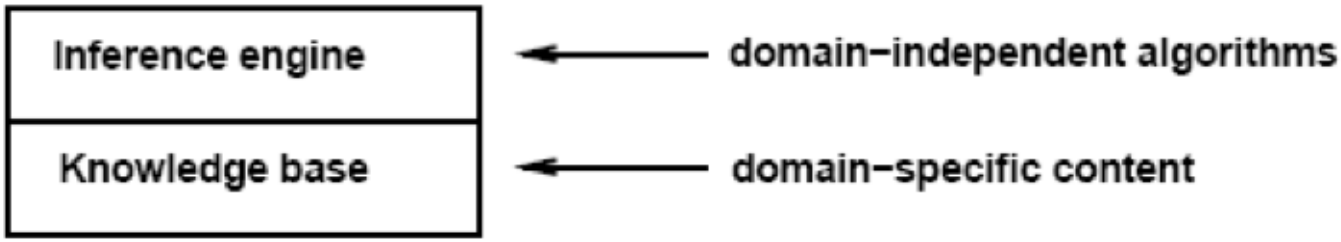
Chapter 4
Knowledge representation, inference and reasoning

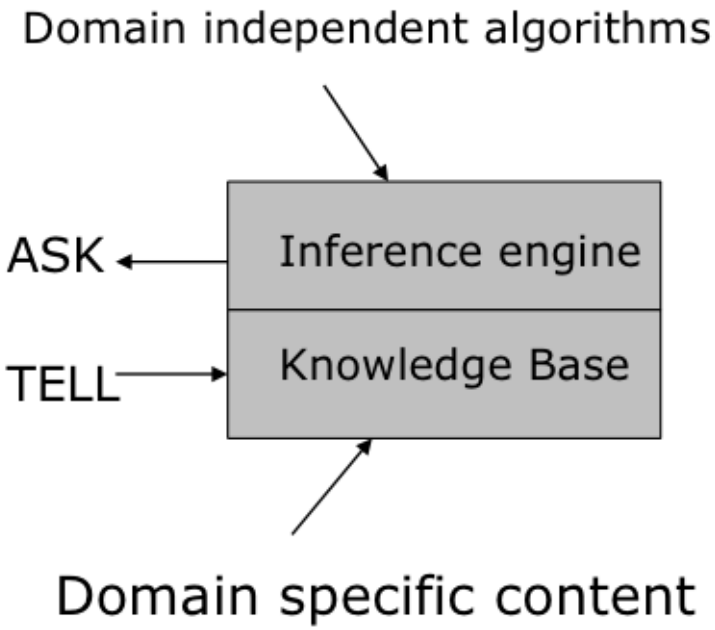
Basanta Joshi, PhD
basanta@ioe.edu.np
Lecture notes can be downloaded from
www.basantajoshi.com.np

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

Knowledge-Based Agent



- Agent that uses **prior** or **acquired** knowledge to achieve its goals
  - Can make more efficient decisions
  - Can make informed decisions
- Knowledge Base (KB): contains a set of representations of facts about the Agent's environment
- Each representation is called a **sentence**
- Use some **knowledge representation language**, to TELL it what to know e.g., (temperature 72F)
- ASK agent to query what to do
- Agent can use inference to deduce new facts from TELLed facts

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

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 $x=0$; $y=6$

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 Commitment
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 logic	facts	degree of belief 0…1
Fuzzy logic	facts, degree of truth	known interval value

The Semantic Wall

Physical Symbol System

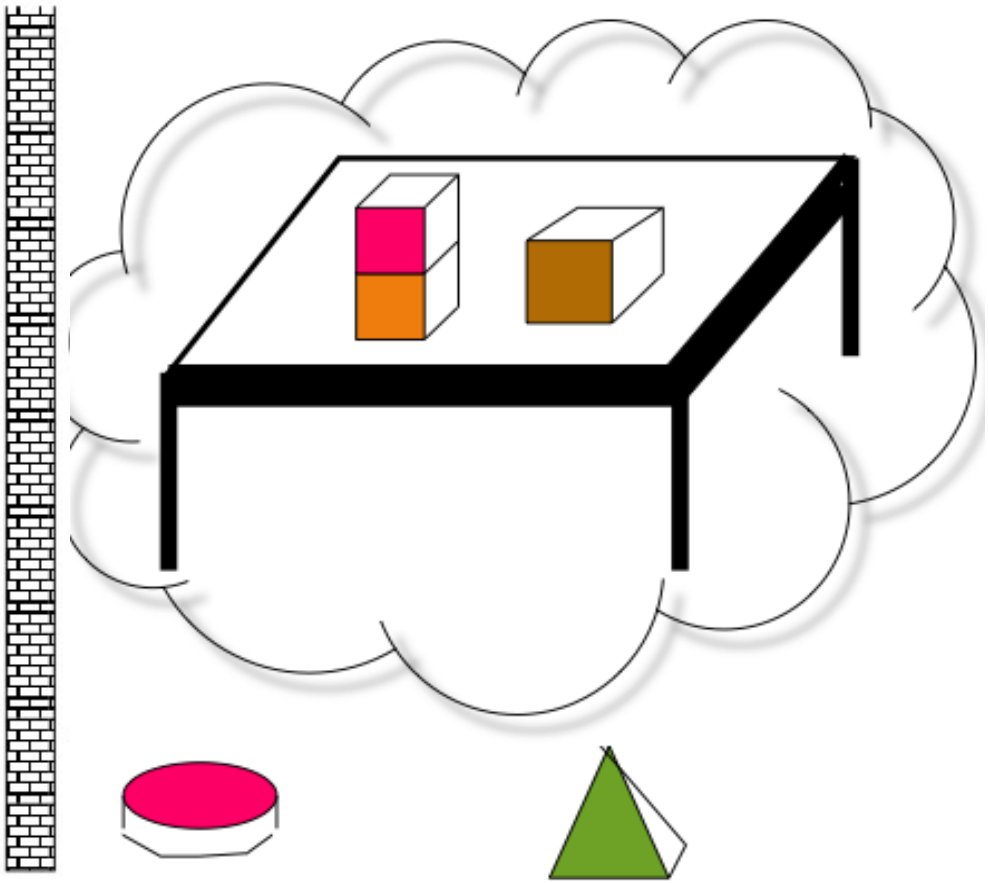
World

+BLOCKA+

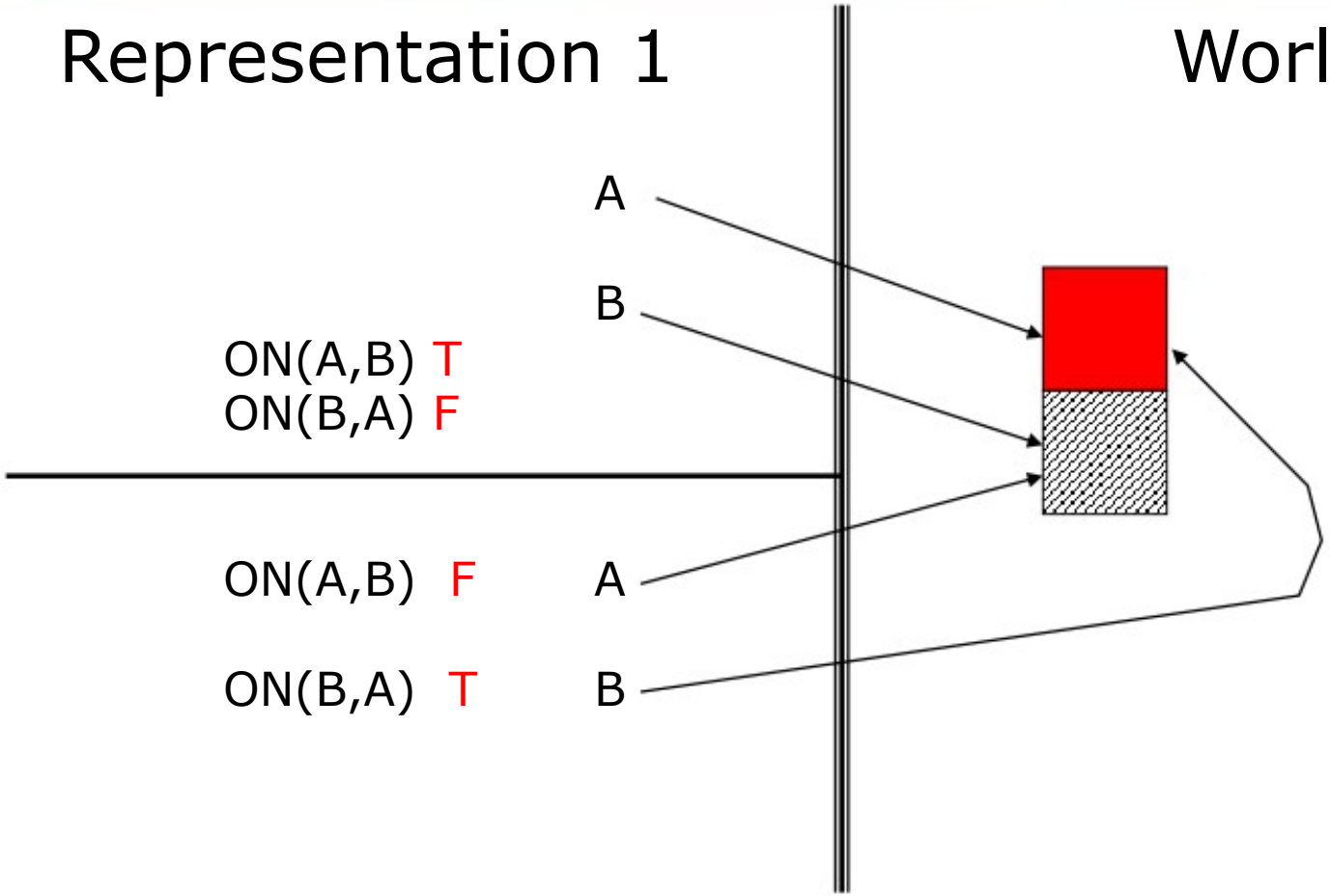
+BLOCKB+

+BLOCKC+

P_1:(IS_ON +BLOCKA+ +BLOCKB+)
P_2:((IS_RED +BLOCKA+)



Truth depends on Interpretation



Entailment

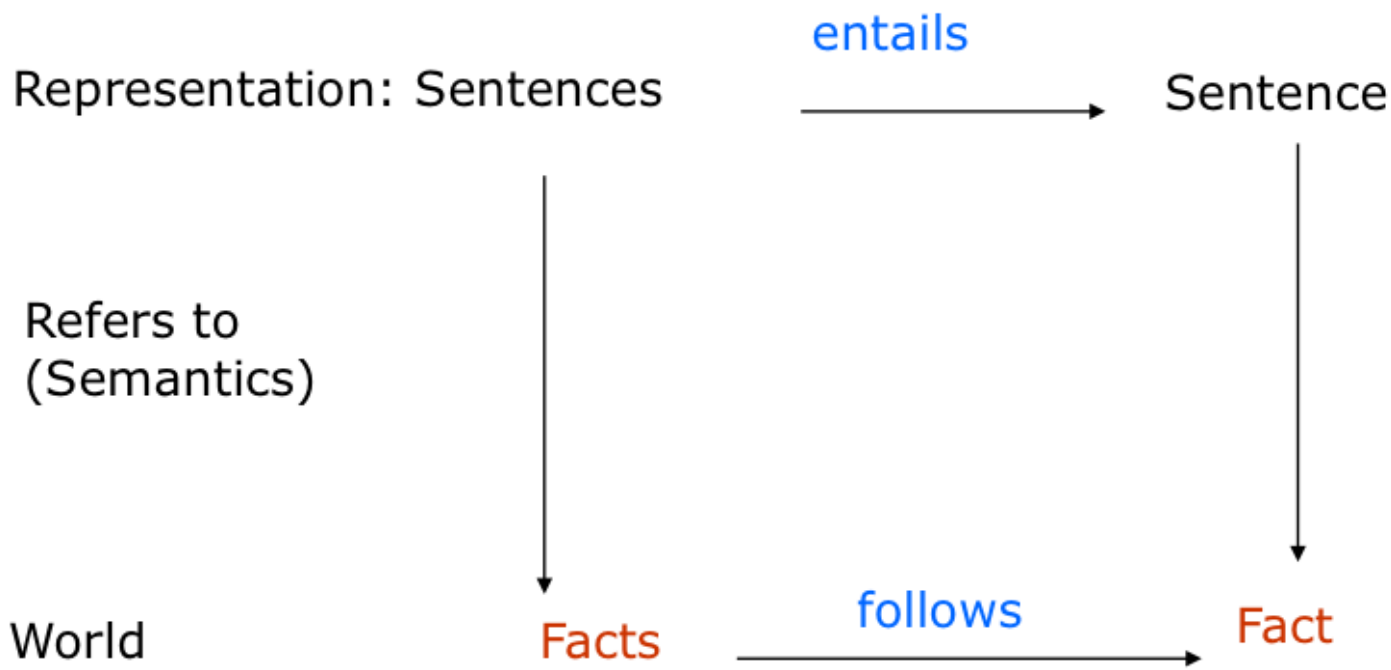
- Entailment means that one thing follows from another:
  $KB \models ®$
- Knowledge base KB entails sentence α
  if and only if (iff)
  ® is true in all worlds where **KB** is true

Eg. The KB containing "The MANCITY won" and " The LIVERPOOL won" entails "Either The MANCITY won or The LIVERPOOL 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)

Entailment is different than inference!

Logic as a representation of the World



Inference

- **KB** ` $_i$ ® = sentence ® can be derived from KB by procedure **i**

- Consequences of **KB** are a haystack; ® is a needle.
  Entailment = needle in haystack; inference = finding it

- Soundness: **i** is sound if
  whenever **KB** ` $_i$ ®, it is also true that **KB** ╞ ®
- Completeness: **i** is complete if
  whenever **KB** ╞ ®, it is also true that **KB** ` $_i$ ®

- 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**.

Basic symbols

- Expressions only evaluate to either "true" or "false."

P	"P is true"	
¬P	"P is false"	negation
P _ Q	"either P is true or Q is true or both"	disjunction
P ^ Q	"both P and Q are true"	conjunction
P) Q	"if P is true, the Q is true"	implication
P , Q	"P and Q are either both true or both false"	equivalence

Propositional logic: 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. $P_{1,2}$ $P_{2,2}$ $P_{3,1}$
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.,
$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (false \vee true) = true \wedge true = true$

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*

Propositional inference: enumeration method

Let ® = **A _ B** and **KB = (A _ C) ^ (B _ ¬ C)**
Is it the case that **KB** ╞ ®?
Check all possible models—® must be true wherever **KB** is true

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

Enumeration: Solution

Let ® = **A _ B** and **KB = (A _ C) ^ (B _ ¬ C)**
Is it the case that **KB** ╞ ®?
Check all possible models—® must be true wherever **KB** is true

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

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 ← 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
   else do
      P ← FIRST(symbols); rest ← REST(symbols)
      return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
                TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

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

Propositional inference: normal forms

Other approaches to inference use syntactic operations on sentences, often expressed in standardized forms

Conjunctive Normal Form (CNF—universal)

conjunction of $\underbrace{\textit{disjunctions of literals}}_{\textit{clauses}}$

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

"product of sums of simple variables or negated simple variables"

Disjunctive Normal Form (DNF—universal)

disjunction of $\underbrace{\textit{conjunctions of literals}}_{\textit{terms}}$

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

"sum of products of simple variables or negated simple variables"

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 $(C \wedge D) \Rightarrow B$

Logical equivalence

Two sentences are logically equivalent iff true in same models:

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

$(\alpha \wedge \beta)$	$\equiv$	$(\beta \wedge \alpha)$	commutativity of $\wedge$
$(\alpha \vee \beta)$	$\equiv$	$(\beta \vee \alpha)$	commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma)$	$\equiv$	$(\alpha \wedge (\beta \wedge \gamma))$	associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma)$	$\equiv$	$(\alpha \vee (\beta \vee \gamma))$	associativity of $\vee$
$\neg(\neg\alpha)$	$\equiv$	α	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv$	$((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta)$	$\equiv$	$(\neg\alpha \vee \neg\beta)$	De Morgan
$\neg(\alpha \vee \beta)$	$\equiv$	$(\neg\alpha \wedge \neg\beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma))$	$\equiv$	$((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma))$	$\equiv$	$((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of $\vee$ over $\wedge$

Validity and satisfiability

- A sentence is valid if it is true in all models,
  e.g., **True, A _ : A, A) A, (A ^ (A) B))) B**
- Validity is connected to inference via the Deduction Theorem:
  KB ╞ ® if and only if (**KB) ®**) is valid

Valid sentence are also known as **tautologies**

- A sentence is satisfiable if it is true in some model
  e.g., **A _ B, C**
- A sentence is unsatisfiable if it is true in no models
  e.g., **A ^ : A**
- Satisfiability is connected to inference via the following:
  KB ╞ ® if and only if (**KB ^ : ®**) is unsatisfiable

Proof methods

- 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

Inference rules

- **Modus Ponens** or **Implication-Elimination**: (From an implication and the premise of the implication, you can infer the conclusion.)

$$\frac{\alpha \Rightarrow \beta, \qquad \alpha}{\beta}$$

- **And-Elimination**: (From a conjunction, you can infer any of the conjuncts.)

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

- **And-Introduction**: (From a list of sentences, you can infer their conjunction.)

$$\frac{\alpha_1, \ \alpha_2, \ \ldots, \ \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$

- **Or-Introduction**: (From a sentence, you can infer its disjunction with anything else at all.)

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

Inference rules

- **Double-Negation Elimination**: (From a doubly negated sentence, you can infer a positive sentence.)

$$\frac{\neg\neg\alpha}{\alpha}$$

- **Unit Resolution**: (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

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

- **Resolution**: (This is the most difficult. Because β cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.)

$$\frac{\alpha \vee \beta, \qquad \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \qquad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

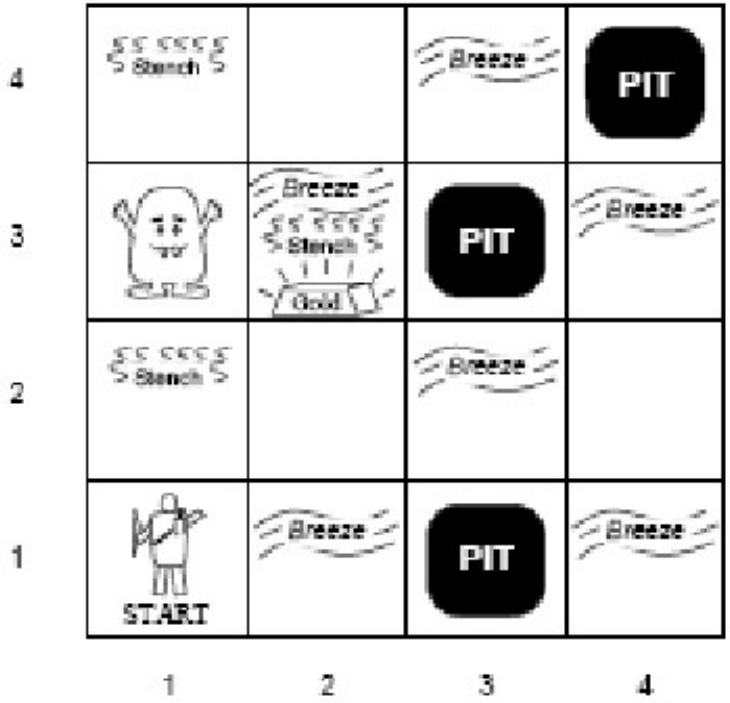
Wumpus World Example

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

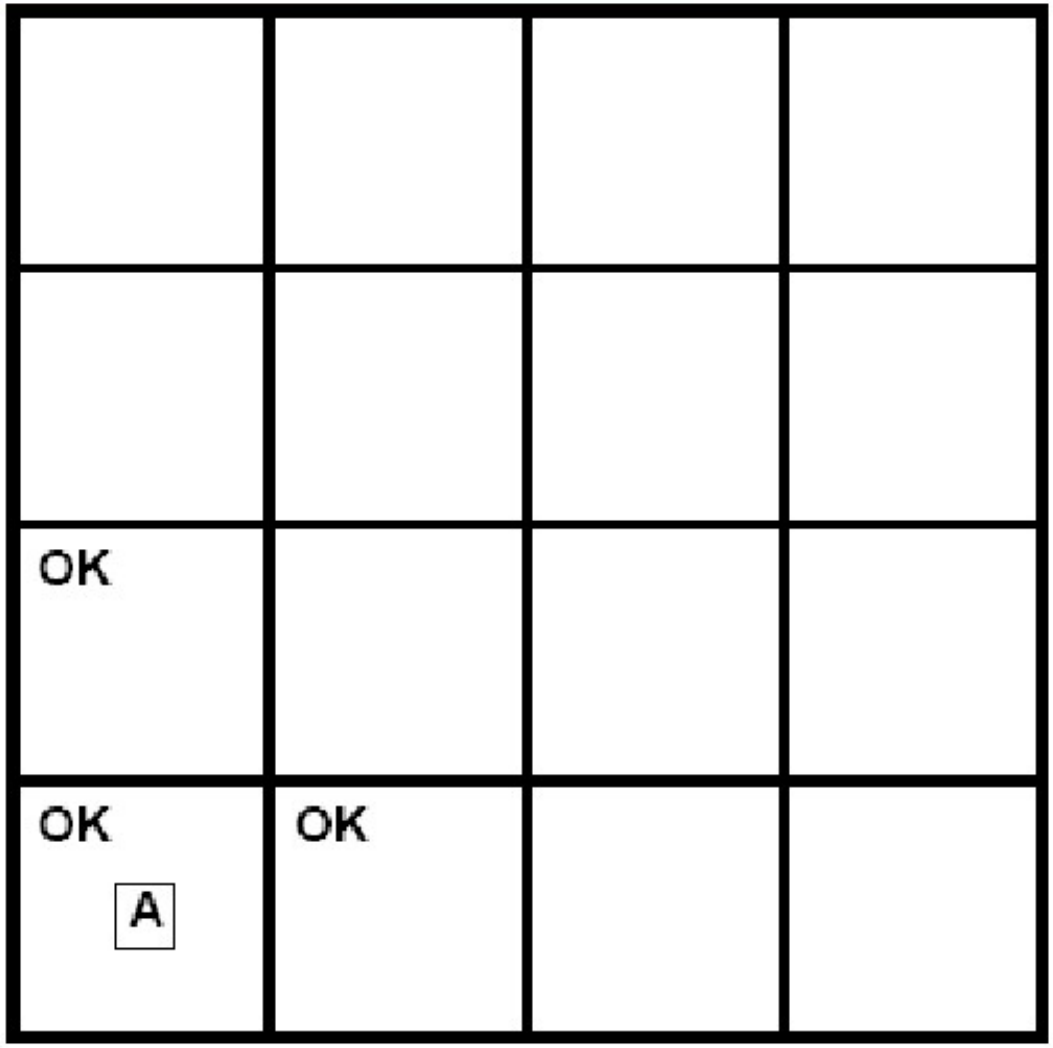
Sensors
- Breeze, Glitter, Smell



Wumpus world characterization

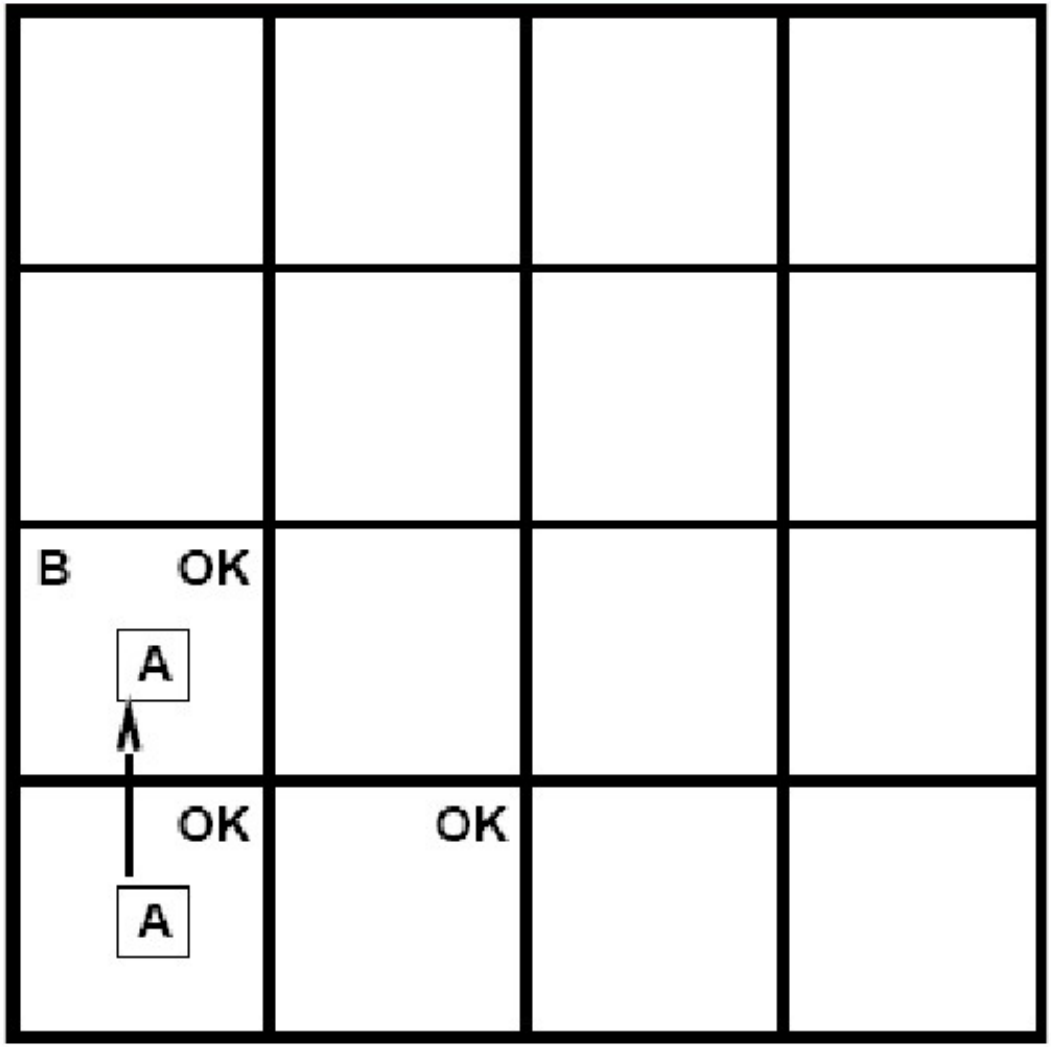
Observable?	No—only local perception
Deterministic?	Yes—outcomes exactly specified
Episodic?	No—sequential at the level of actions
Static?	Yes—Wumpus and pits do not move
Discrete?	Yes
Single-agent?	Yes—Wumpus is a feature

Exploring a wumpus world



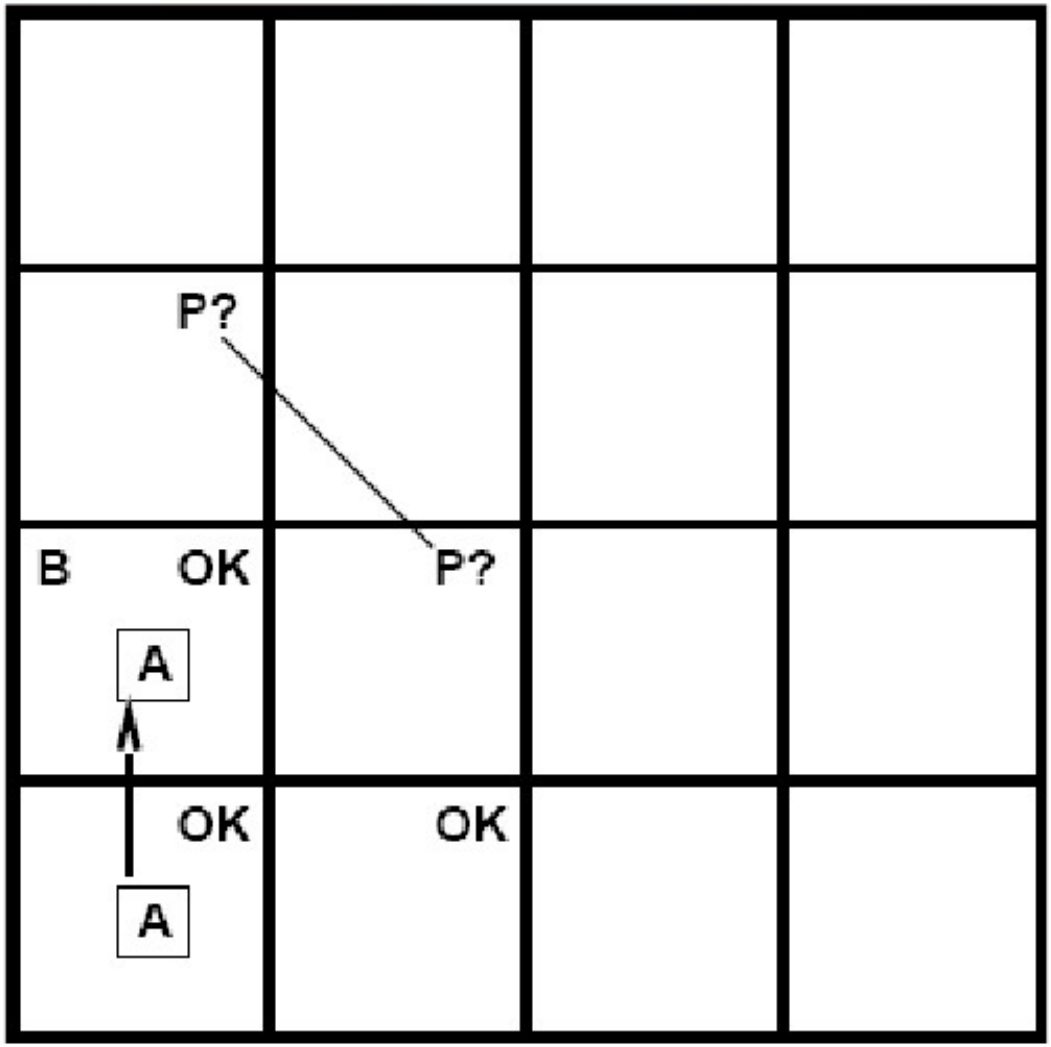
A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

Exploring a wumpus world



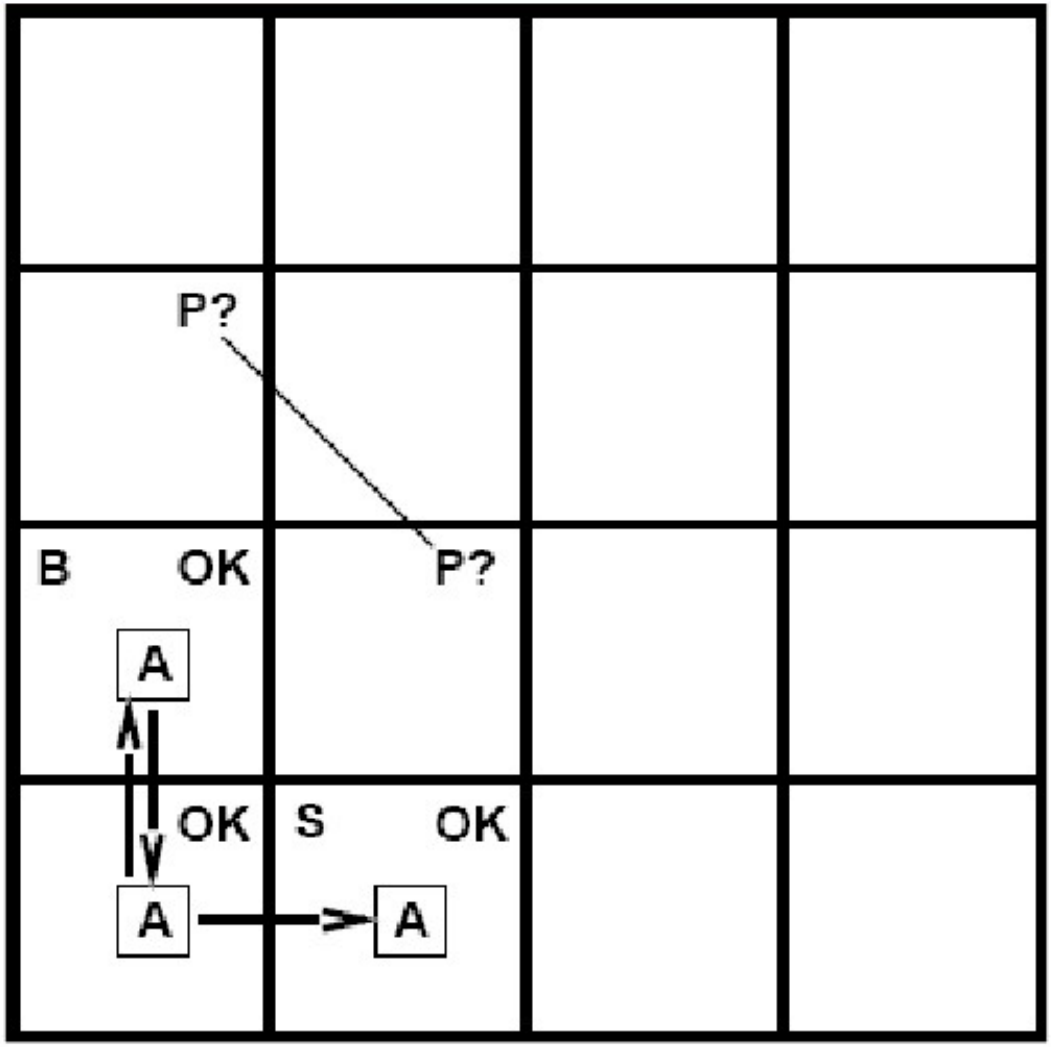
A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

Exploring a wumpus world



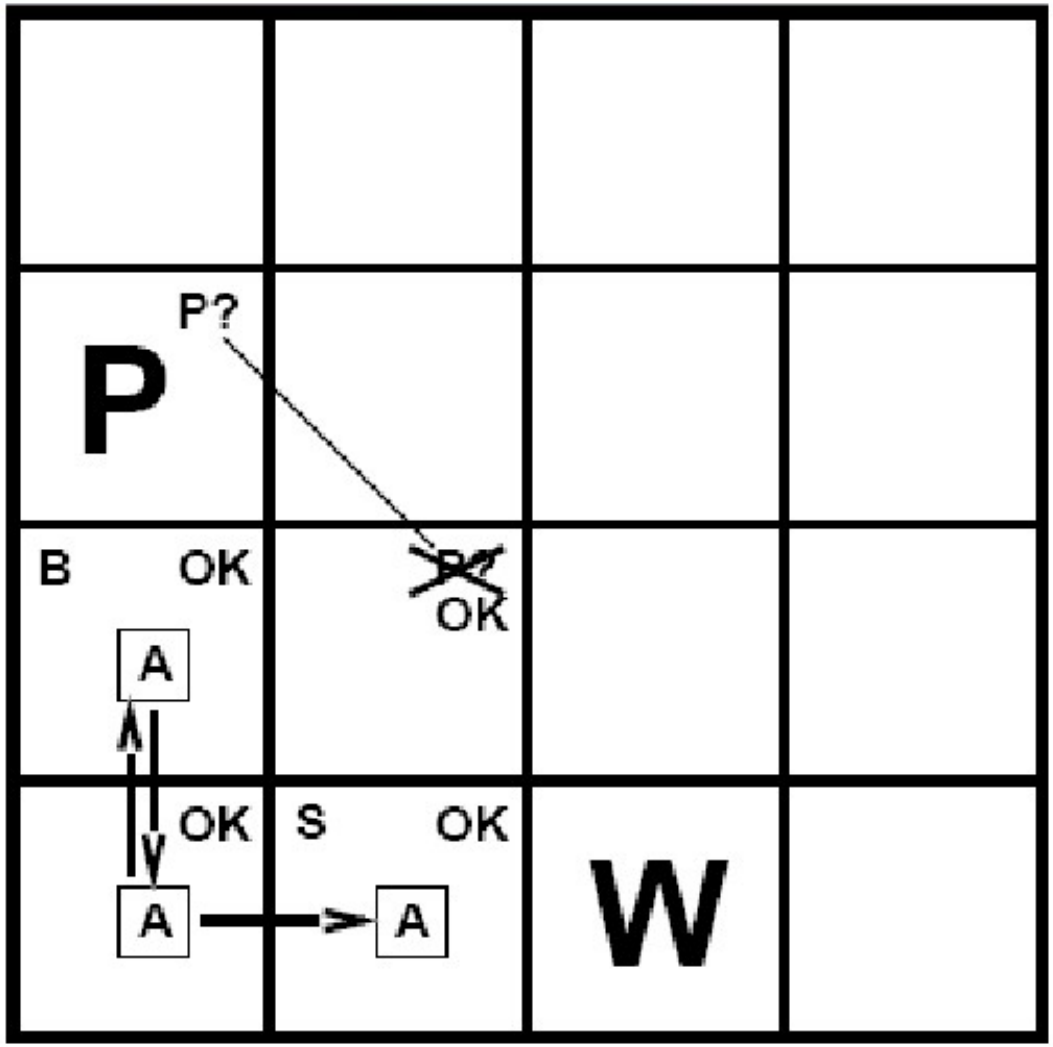
A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

Exploring a wumpus world



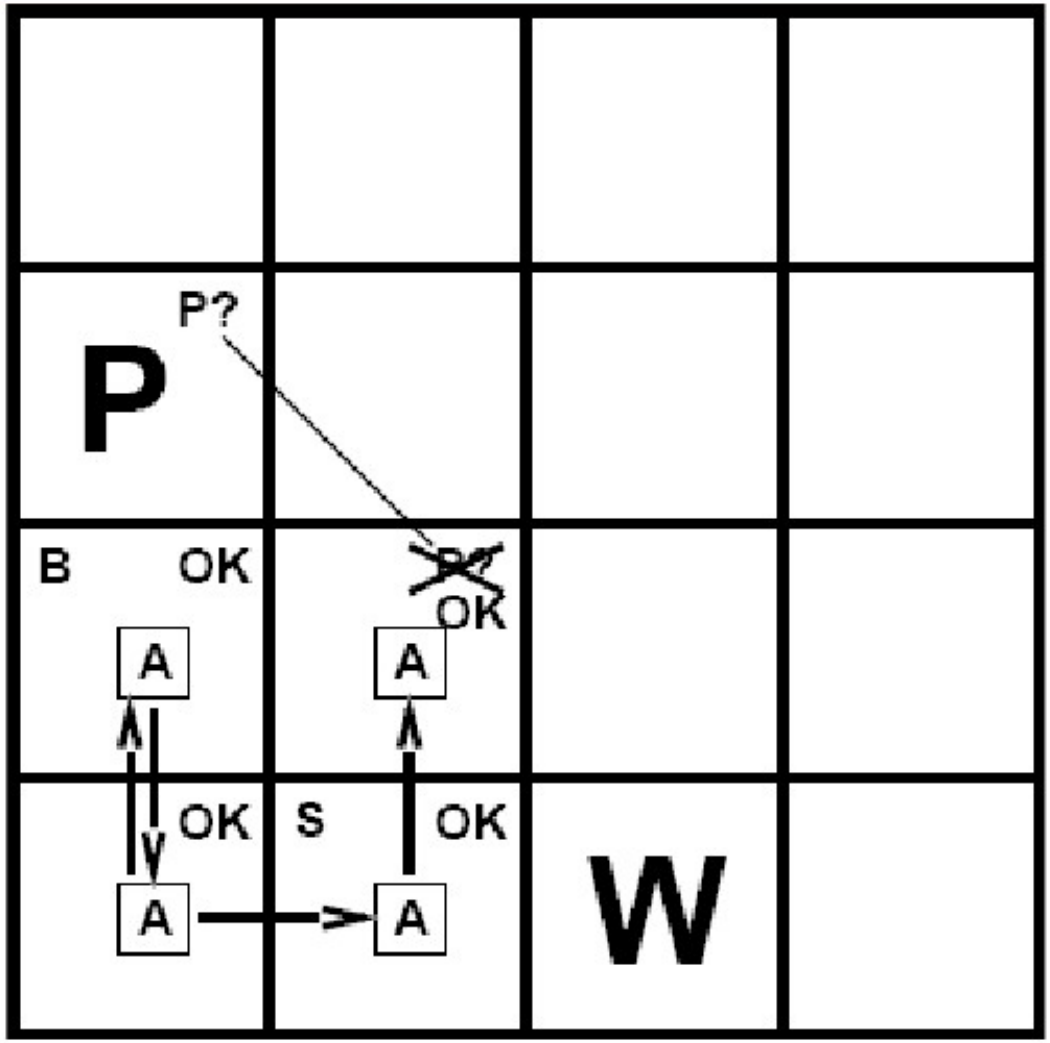
A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

Exploring a wumpus world



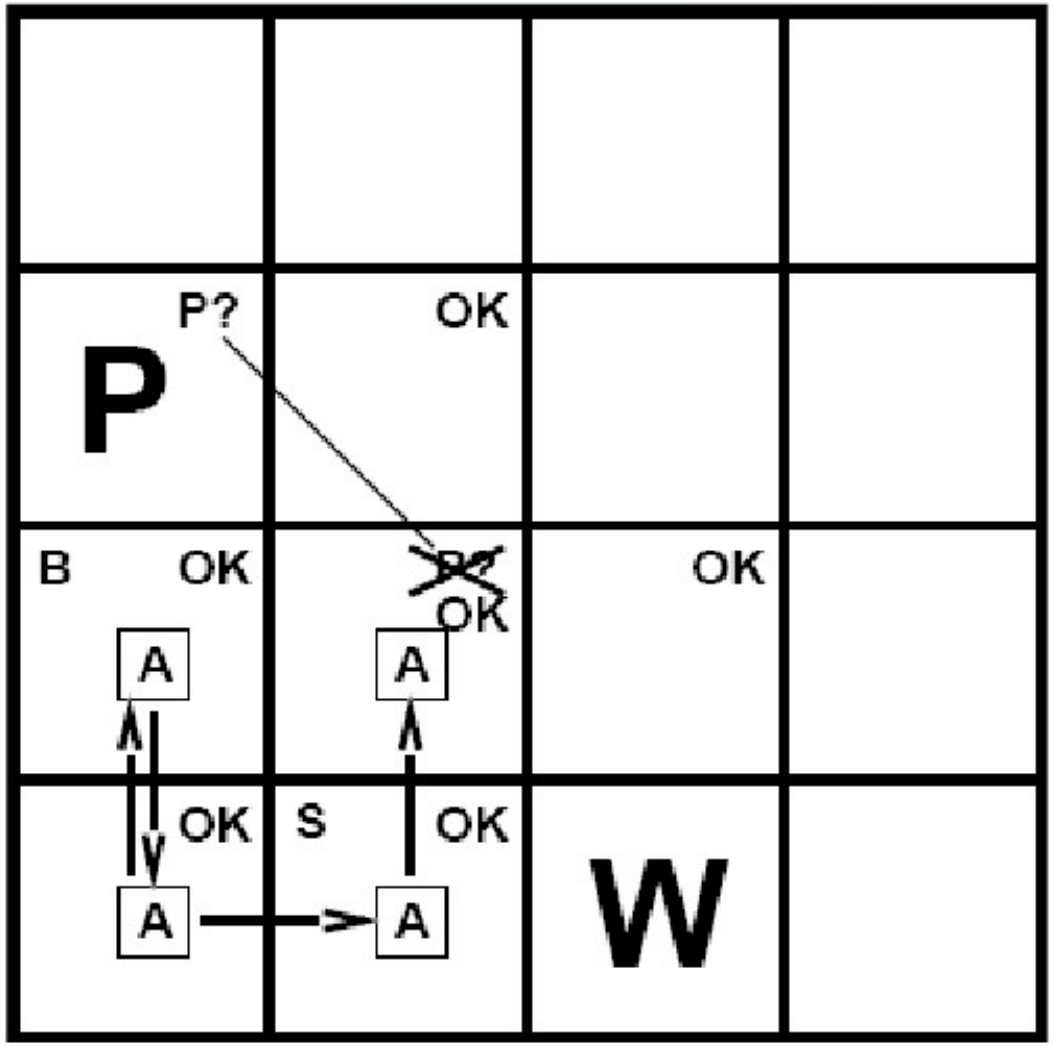
A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

Exploring a wumpus world



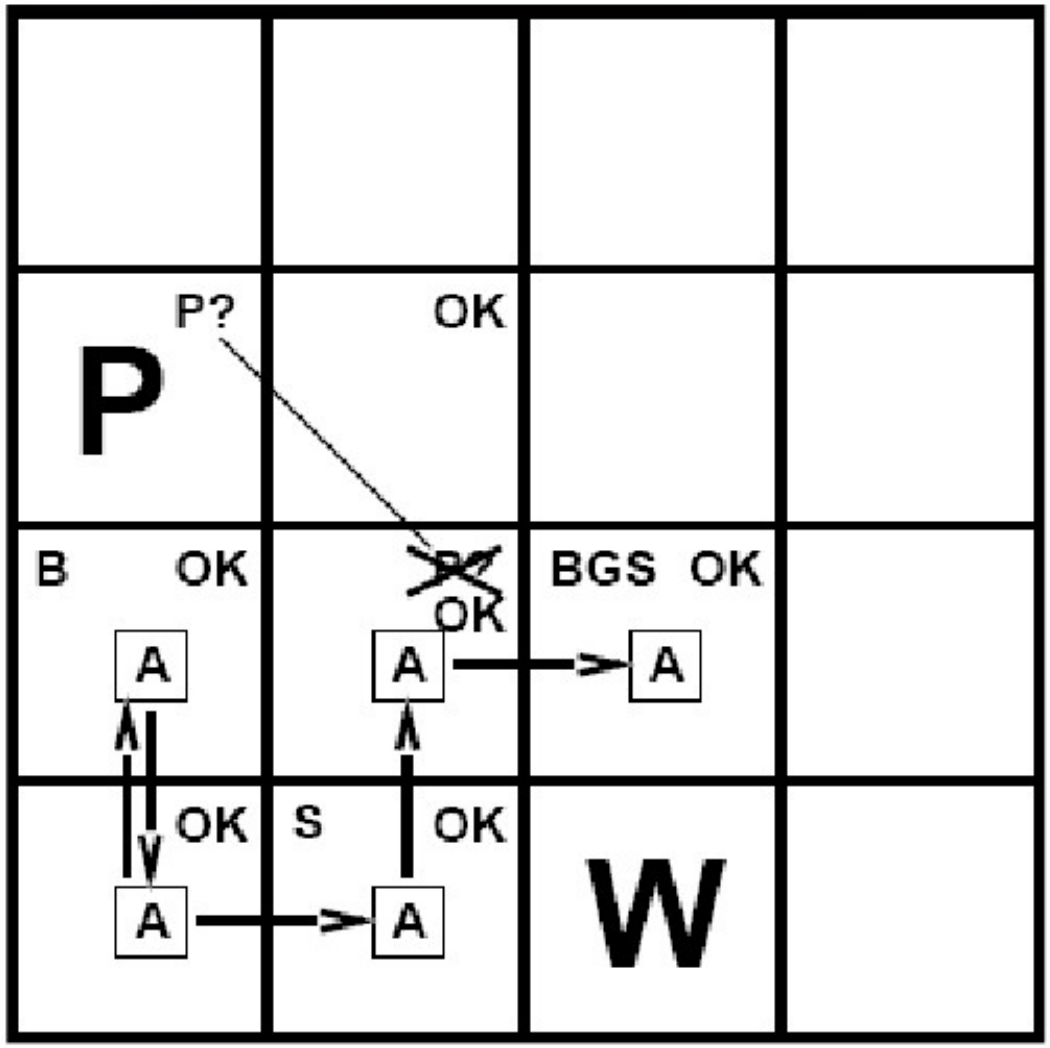
A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

Exploring a wumpus world



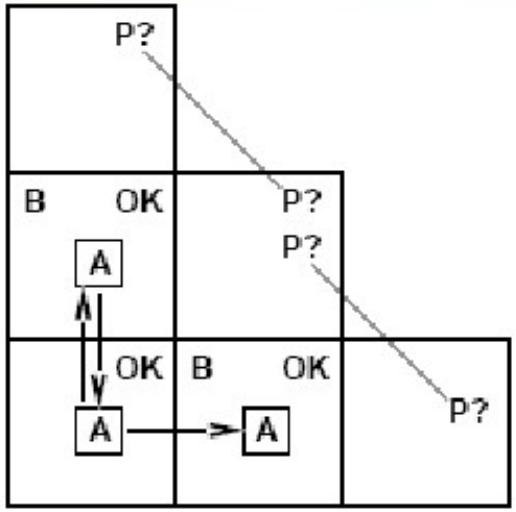
A= Agent
B= Breeze
S= Smell
P= Pit
W= Wumpus
OK = Safe
V = Visited
G = Glitter

Exploring a wumpus world



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

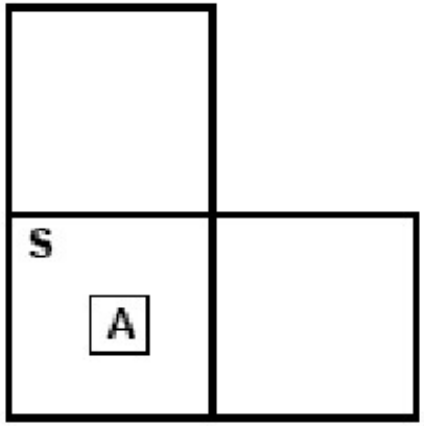
Other tight spots



Breeze in (1,2) and (2,1)

è no safe actions

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



Smell in (1,1)

è cannot move

Can use a strategy of coercion:

shoot straight ahead

wumpus was there è dead è safe

wumpus wasn't there è safe

Example solution

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

S in 1,2 à 1,3 or 2,2 has W

No S in 2,1 à 2,2 OK

2,2 OK à 1,3 W

No B in 1,2 & B in 2,1 à 3,1 P

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].

$P_{1;1}$

$B_{1;1}$

$B_{2;1}$

"Pits cause breezes in adjacent squares"

$B_{1;1}$, ($P_{1;2}$ _ $P_{2;1}$)

$B_{2;1}$, ($P_{1;1}$ _ $P_{2;2}$ _ $P_{3;1}$)

"A square is breezy if and only if there is an adjacent pit"

Wumpus world sentences

The knowledge base includes the following sentences, each one labeled for convenience:

- There is no pit in [1,1]:

$$R_1 : \quad \neg P_{1,1} \ .$$

- A square is breezy if and only if there is a pit in a neighboring square. This has to be stated for each square; for now, we include just the relevant squares:

$$R_2 : \quad B_{1,1} \quad \Leftrightarrow \quad (P_{1,2} \vee P_{2,1}) \ .$$
$$R_3 : \quad B_{2,1} \quad \Leftrightarrow \quad (P_{1,1} \vee P_{2,2} \vee P_{3,1}) \ .$$

- The preceding sentences are true in all wumpus worlds. Now we include the breeze percepts for the first two squares visited in the specific world the agent is in, leading up to the situation in Figure 7.3(b).

$$R_4 : \quad \neg B_{1,1} \ .$$
$$R_5 : \quad B_{2,1} \ .$$

The knowledge base, then, consists of sentences R_1 through R_5. It can also be considered as a single sentence—the conjunction $R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5$—because it asserts that all the individual sentences are true.

Wumpus world sentences

Let us see how these inference rules and equivalences can be used in the wumpus world. We start with the knowledge base containing R_1 through R_5, and show how to prove $\neg P_{1,2}$, that is, there is no pit in [1,2]. First, we apply biconditional elimination to R_2 to obtain

$$R_6 : \quad (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}) .$$

Then we apply And-Elimination to R_6 to obtain

$$R_7 : \quad ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}) .$$

Logical equivalence for contrapositives gives

$$R_8 : \quad (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})) .$$

Now we can apply Modus Ponens with R_8 and the percept R_4 (i.e., $\neg B_{1,1}$), to obtain

$$R_9 : \quad \neg(P_{1,2} \vee P_{2,1}) .$$

Finally, we apply De Morgan's rule, giving the conclusion

$$R_{10} : \quad \neg P_{1,2} \wedge \neg P_{2,1} .$$

That is, neither [1,2] nor [2,1] contains a pit.

Resolution

We have argued that the inference rules covered so far are *sound*, but we have not discussed the question of *completeness* for the inference algorithms that use them. Search algorithms such as iterative deepening search (page 78) are complete in the sense that they will find any reachable goal, but if the available inference rules are inadequate, then the goal is not reachable—no proof exists that uses only those inference rules. For example, if we removed the biconditional elimination rule, the proof in the preceding section would not go through. The current section introduces a single inference rule, **resolution**, that yields a complete inference algorithm when coupled with any complete search algorithm.

We begin by using a simple version of the resolution rule in the wumpus world. Let us consider the steps leading up to Figure 7.4(a): the agent returns from [2,1] to [1,1] and then goes to [1,2], where it perceives a stench, but no breeze. We add the following facts to the knowledge base:

$$R_{11}: \quad \neg B_{1,2} \,.$$
$$R_{12}: \quad B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3}) \,.$$

By the same process that led to R_{10} earlier, we can now derive the absence of pits in [2,2] and [1,3] (remember that [1,1] is already known to be pitless):

$$R_{13}: \quad \neg P_{2,2} \,.$$
$$R_{14}: \quad \neg P_{1,3} \,.$$

We can also apply biconditional elimination to R_3, followed by modus ponens with R_5, to obtain the fact that there is a pit in [1,1], [2,2], or [3,1]:

$$R_{15}: \quad P_{1,1} \vee P_{2,2} \vee P_{3,1} \,.$$

Resolution

Now comes the first application of the resolution rule: the literal $\neg P_{2,2}$ in R_{13} *resolves with* the literal $P_{2,2}$ in R_{15} to give

$$R_{16}: \quad P_{1,1} \vee P_{3,1} \ .$$

In English; if there's a pit in one of [1,1], [2,2], and [3,1], and it's not in [2,2], then it's in [1,1] or [3,1]. Similarly, the literal $\neg P_{1,1}$ in R_1 resolves with the literal $P_{1,1}$ in R_{16} to give

$$R_{17}: \quad P_{3,1} \ .$$

In English: if there's a pit in [1,1] or [3,1], and it's not in [1,1], then it's in [3,1]. These last two inference steps are examples of the **unit resolution** inference rule,

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k} \ ,$$

where each ℓ is a literal and ℓ_i and m are **complementary literals** (i.e., one is the negation of the other). Thus, the unit resolution rule takes a **clause**—a disjunction of literals—and a literal and produces a new clause. Note that a single literal can be viewed as a disjunction of one literal, also known as a **unit clause**.

The unit resolution rule can be generalized to the full **resolution** rule,

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

where ℓ_i and m_j are complementary literals. If we were dealing only with clauses of length

Resolution

Conjunctive Normal Form (CNF—universal)

$$\text{conjunction of } \underbrace{\text{disjunctions of literals}}_{\text{clauses}}$$

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

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

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

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 ^ : ®** unsatisfiable

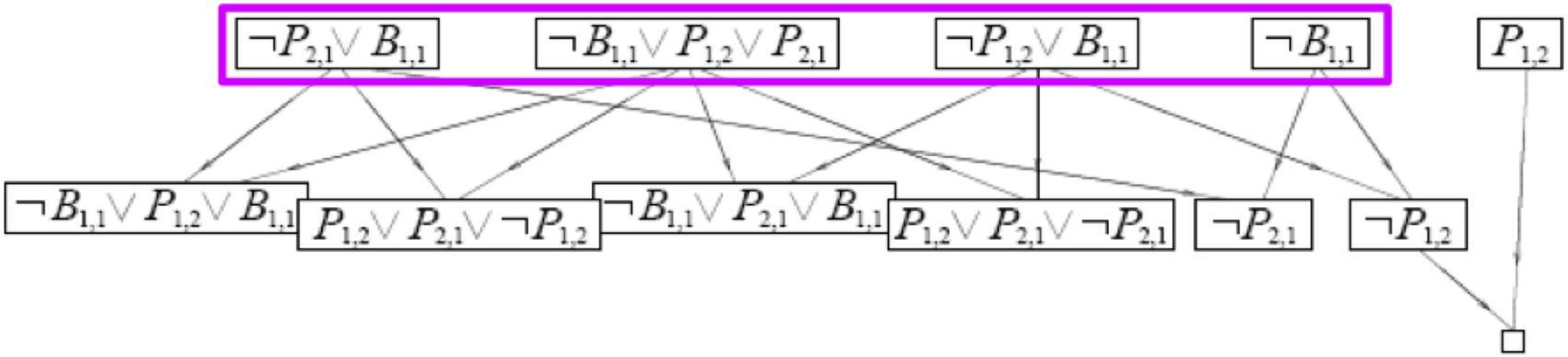
```
function PL-RESOLUTION(KB, α) returns true or false
    inputs: KB, the knowledge base, a sentence in propositional logic
            α, the query, a sentence in propositional logic

    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each 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

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

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



Limitations of Propositional Logic

1. It is too weak, i.e., has very limited expressiveness:

   Each rule has to be represented for each situation:

   e.g., "don't go forward if the wumpus is in front of you" takes 64 rules

2. It cannot keep track of changes:

   If one needs to track changes, e.g., where the agent has been before then we need a timed-version of each rule. To track 100 steps we'll then need 6400 rules for the previous example.

Its **hard to write and maintain** such a huge rule-base

Inference becomes intractable

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

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

- Propositional logic lacks expressive power

Why first-order logic?

Pros and cons of propositional logic

- Propositional logic is declarative: pieces of syntax correspond to facts
- Propositional logic allows partial/disjunctive/negated information (unlike most data structures and databases)
- Propositional logic is compositional: meaning of **B1;1 ^ P1;2** is derived from meaning of **B1;1** and of **P1;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

First-order logic (FOL)

- Whereas propositional logic assumes world contains facts, first-order logic (like natural language) assumes the world contains:
  - **Objects**: wheel, door, body, engine, seat, car, passenger, driver, people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries, …
  - **Predicates**: Inside(car, passenger), Beside(driver, passenger), BrotherOf(person, person), BiggerThan(object, object), Inside(), part of(), HasColor(), OccurredAfter(), Owns(), ComesBetween(), …
  - **Functions**: ColorOf(car), FatherOf(person), BestFriend(person), ThirdInningOf(), OneMoreThan(), EndOf(), …
- Functions are relations with single value for each object

Logics in general

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 Commitment
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 logic	**facts**	**degree of belief 0…1**
Fuzzy logic	**facts, degree of truth**	**known interval value**

Semantics/ Interpretation

there is a correspondence between

- functions, which return values
- predicates, which are true or false

Function: father_of(Mary) = Bill
Predicate: father_of(Mary, Bill)

Examples:

- "One plus two equals three"
  Objects:
  Predicates:
  Functions:

- "Squares neighboring the Wumpus are smelly"
  Objects:
  Predicates:
  Functions:

Examples:

- "One plus two equals three"

  Objects: one, two, three, one plus two

  Predicates: equals

  Functions: plus ("one plus two" is the name of the object obtained by applying function plus to one and two; three is another name for this object)

- "Squares neighboring the Wumpus are smelly"

  Objects: Wumpus, square

  Predicates: neighboring, smelly

  Functions: --

FOL: Syntax of basic elements

- **Constant symbols:** 1, 5, A, B, USC, JPL, Alex, Manos, …
- **Predicate symbols:** >, Friend, Student, Colleague, …
- **Function symbols:** +, sqrt, SchoolOf, TeacherOf, ClassOf, …
- **Variables:** *x, y, z, next, first, last, …*
- **Connectives:** $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$
- **Quantifiers:** $\forall$, $\exists$
- **Equality:** =

FOL: Atomic sentences

AtomicSentence = **predicate(term$_1$, ..., term$_n$)**
or **term$_1$= term$_2$**

Term = **function(term$_1$, ..., term$_n$)**
or **constant** or **variable**

- Examples:
  - **SchoolOf (Manos)**
  - **Colleague(TeacherOf (Alex) , TeacherOf (Manos))**
  - **> ((+ x y) , x)**

FOL: Complex sentences

Complex sentences are made from atomic sentences using connectives

$\neg S$, $S_1 \wedge S_2$, $S_1 _ S_2$, S_1) S_2, S_1 , S_2

Sentence → AtomicSentence
| Sentence Connective Sentence
| Quantifier Variable, ... Sentence
| ¬ Sentence
| (Sentence)

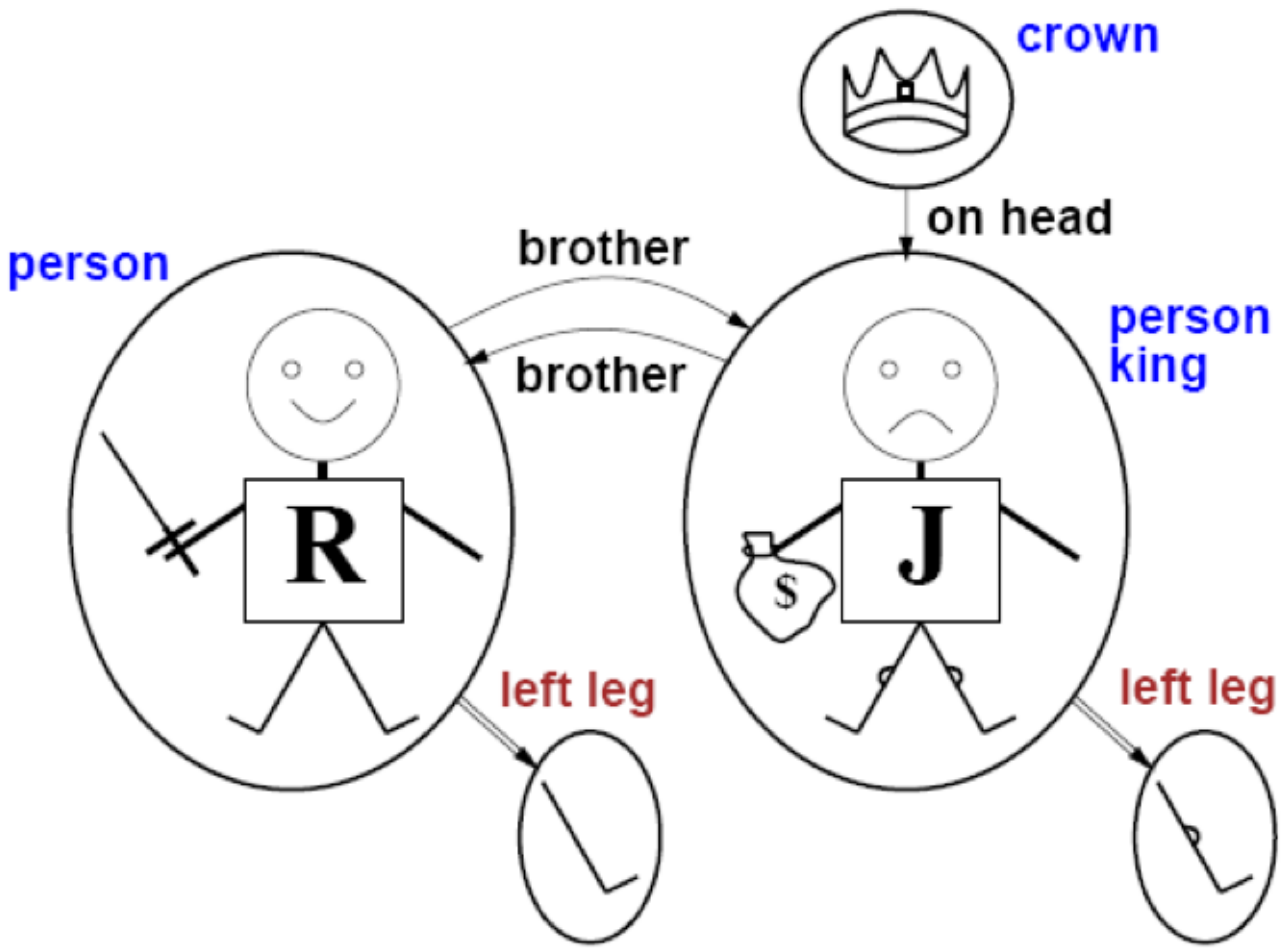
- Examples:
  - Colleague(Paolo; Maja)) Colleague(Maja; Paolo)
    Student(Alex; Paolo)) Teacher(Paolo; Alex)

Truth in first-order logic

- Sentences in FOL are interpreted with respect to a model and an interpretation
- Model contains ≥ 1 objects (domain elements) and relations among them
- Interpretation specifies refererents for
  - Constant symbols: refer to objects
  - Predicate symbols: refer to relations
  - Function symbols: refer to 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**

Models for FOL: Example



Truth example

- Consider the interpretation in which
- **Richard** → Richard the Lionheart
- **John** → the evil King John
- **Brother** → the brother relation

- Under this interpretation, **Brother**(**Richard**; **John**) is true just in case Richard the Lionheart and the evil King John are in the brother relation in the model

Models for FOL: Lots!

- Entailment in propositional logic can be computed by enumerating models
- We can enumerate the FOL models for a given KB vocabulary:
- For each number of domain elements n from 1 to 1

  For each k-ary predicate P_k in the vocabulary

  For each possible k-ary relation on n objects

  For each constant symbol C in the vocabulary

  For each choice of referent for C from n objects ...

Computing entailment by enumerating FOL models is not easy!

Quantifiers

- Expressing sentences about **collections** of objects without enumeration (naming individuals)
- E.g., All Trojans are clever

  Someone in the class is sleeping
- Universal quantification (for all): $\forall$
- Existential quantification (there exists): $\exists$

Universal quantification (for all): $\forall$

$\forall$ <variables> <sentence>

- *"Every one in the cs561 class is smart"*:
  $\forall x \quad In(cs561; x) \Rightarrow Smart(x)$

- **$\forall$ P corresponds to the conjunction of instantiations of P**
  $In(cs561; Manos) \Rightarrow Smart(Manos) \wedge$
  $In(cs561; Dan) \Rightarrow Smart(Dan) \wedge$
  ...
  $In(cs561; Bush) \Rightarrow Smart(Bush)$

Universal quantification (for all): $\forall$

$\Rightarrow$ is a natural connective to use with $\forall$

Common mistake: to use $\wedge$ in conjunction with $\forall$

$$\forall x \quad In(cs561; x) \wedge Smart(x)$$

means "*every one is in cs561 and everyone is smart*"

Existential quantification (there exists): ∃

∃ <variables> <sentence>

- *"Someone in the cs561 class is smart"*:
  ∃ x In(cs561; x) ^ Smart(x)

- ∃ x P is true in a model m iff P is true with x being some possible object in the model

- **∃ P corresponds to the disjunction of instantiations of P**
  (In(cs561; Manos) ^ Smart(Manos)) _
  (In(cs561; Dan) ^ Smart(Dan)) _
  ...
  (In(cs561; Bush) ^ Smart(Bush))

Existential quantification (there exists): $\exists$

- $\wedge$ is a natural connective to use with $\exists$
- Common mistake: to use $\Rightarrow$ in conjunction with $\exists$

$$\exists\, x \quad In(cs561; x) \Rightarrow Smart(x)$$

- is true if there is anyone that is not in cs561! (remember, false $\Rightarrow$ true is valid).

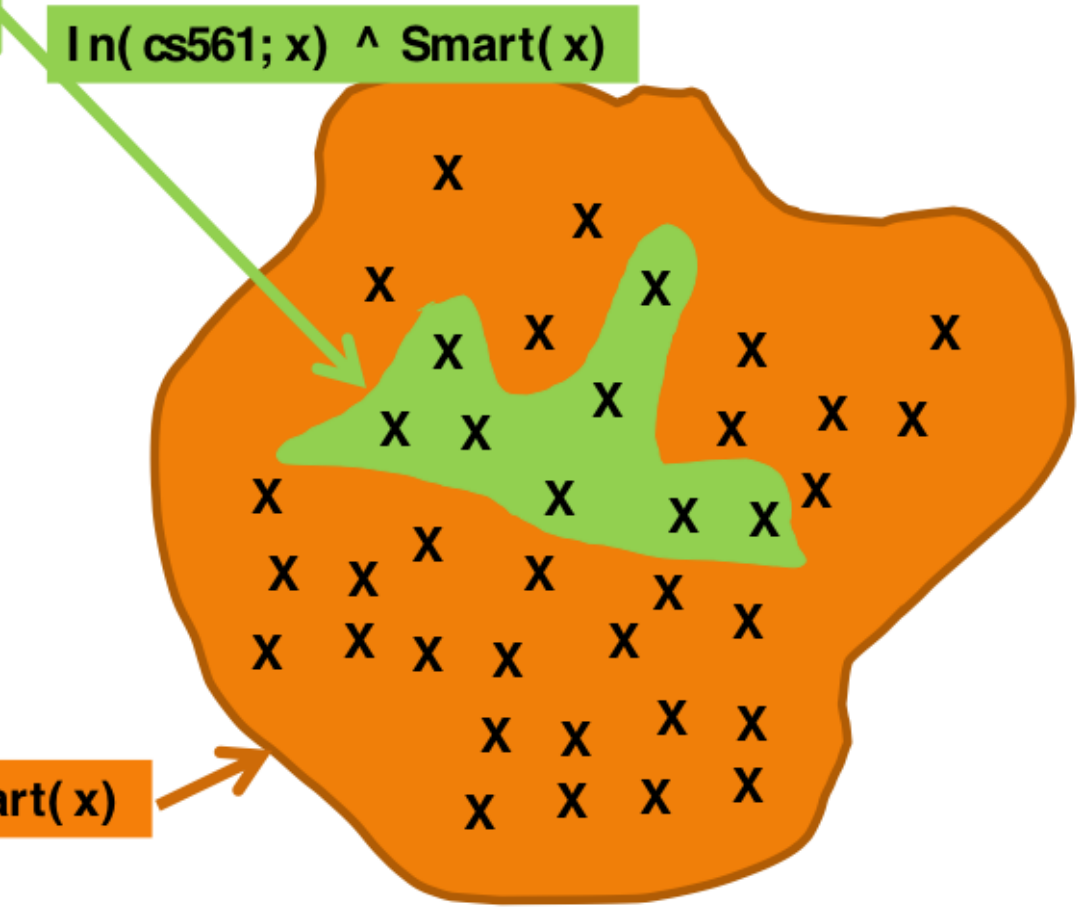
Examples...

$\forall x \;\; In(cs561; x) \Rightarrow Smart(x)$

$\forall x \;\; In(cs561; x) \wedge Smart(x)$

$\exists x \;\; In(cs561; x) \wedge Smart(x)$

$\exists x \;\; In(cs561; x) \Rightarrow Smart(x)$



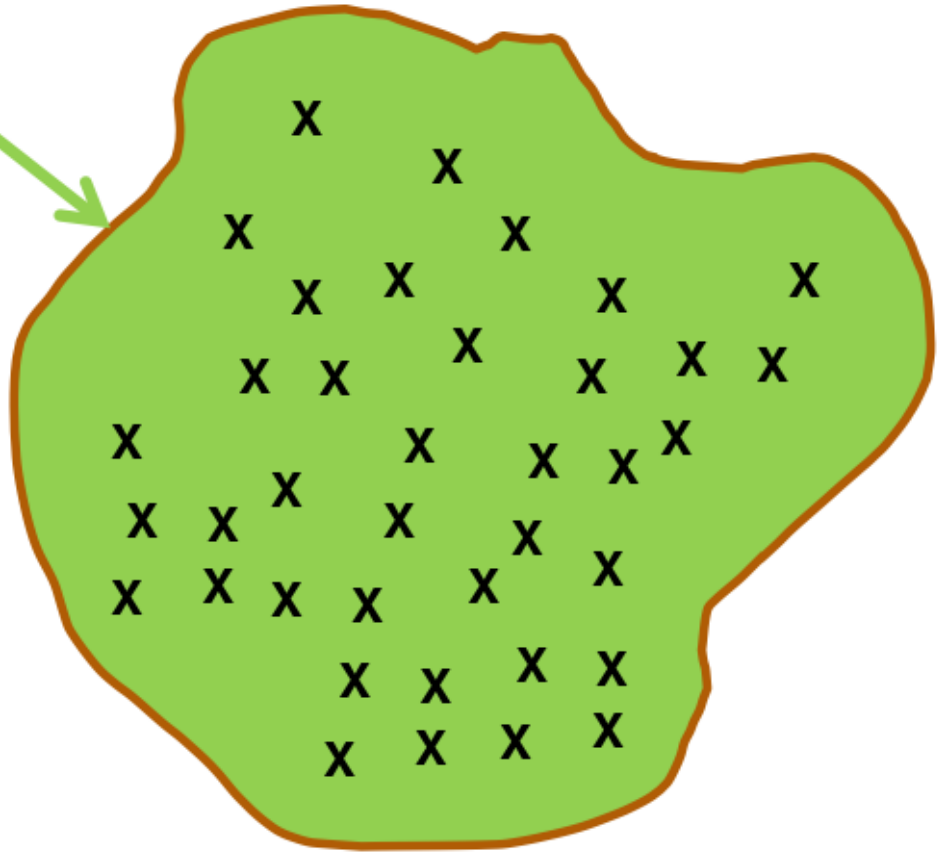
Examples...

$\forall x \;\; In(cs561; x) \Rightarrow Smart(x)$

$\forall x \;\; In(cs561; x) \wedge Smart(x)$

$\exists x \;\; In(cs561; x) \wedge Smart(x)$

$\exists x \;\; In(cs561; x) \Rightarrow Smart(x)$



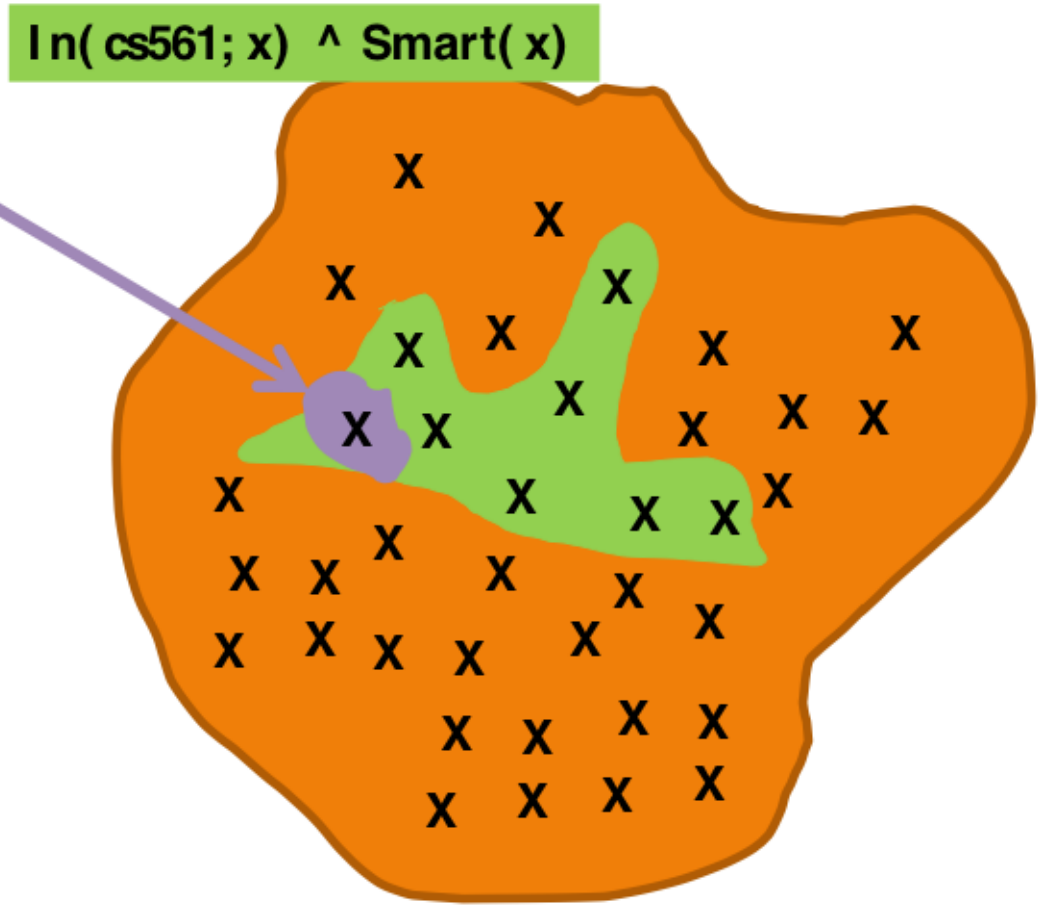
Examples...

$\forall x \;\; In(cs561; x) \Rightarrow Smart(x)$

$\forall x \;\; In(cs561; x) \wedge Smart(x)$

$\exists x \;\; In(cs561; x) \wedge Smart(x)$

$\exists x \;\; In(cs561; x) \Rightarrow Smart(x)$



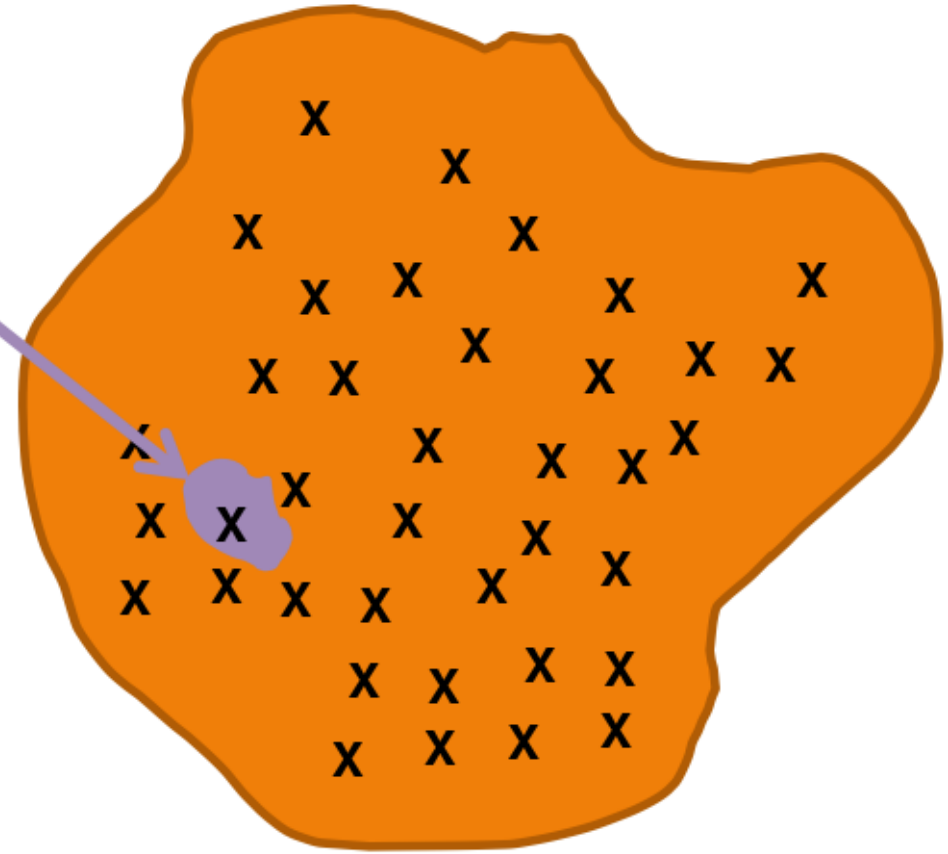
Examples...

$\forall x \;\; In(cs561; x) \Rightarrow Smart(x)$

$\forall x \;\; In(cs561; x) \wedge Smart(x)$

$\exists x \;\; In(cs561; x) \wedge Smart(x)$

$\exists x \;\; In(cs561; x) \Rightarrow Smart(x)$



$\neg In(cs561; x)$

Properties of quantifiers

$\forall x \ \forall y$ is the same as $\forall y \ \forall x$ (why??)

$\exists x \ \exists y$ is the same as $\exists y \ \exists x$ (why??)

$\exists x \ \forall y$ is **not** the same as $\forall y \ \exists x$

$\exists x \ \forall y \ Loves(x, y)$
"There is a person who loves everyone in the world"

$\forall y \ \exists x \ Loves(x, y)$
"Everyone in the world is loved by at least one person"

Quantifier duality: each can be expressed using the other

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

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

Fun with sentences

- Brothers are siblings
  $\forall x;y \quad Brother(x;y) \Rightarrow Sibling(x;y)$

- "Sibling" is symmetric
  $\forall x;y \quad Sibling(x;y) , \quad Sibling(y;x)$

- One's mother is one's female parent
  $\forall x;y \; Mother(x;y) , \quad (Female(x) \wedge Parent(x;y))$

- A first cousin is a child of a parent's sibling
  $\forall x;y \; FirstCousin(x;y) , \quad \exists p;ps \; Parent(p;x)$
  $\wedge Sibling(ps;p) \wedge Parent(ps;y))$

Translating English to FOL

- Every gardener likes the sun.
  $\forall x \; gardener(x) \Rightarrow likes(x, Sun)$

- You can fool some of the people all of the time.
  $\exists x \; \forall t \; (person(x) \wedge time(t)) \Rightarrow can\text{-}fool(x, t)$

Translating English to FOL

- You can fool all of the people some of the time.

  $\forall x \; \exists t \; (person(x) \wedge time(t)) \Rightarrow can\text{-}fool(x, t)$

  $\exists t \; \forall x \; (person(x) \wedge time(t)) \Rightarrow can\text{-}fool(x, t)$

  → You can fool all of the people ***at the same time.***

- All purple mushrooms are poisonous.

  $\forall x \; (mushroom(x) \wedge purple(x)) \Rightarrow poisonous(x)$

Translating English to FOL...

- No purple mushroom is poisonous.

$\neg(\exists x)\ purple(x) \wedge mushroom(x) \wedge poisonous(x)$

or, equivalently,

$(\forall x)\ (mushroom(x) \wedge purple(x)) \Rightarrow \neg poisonous(x)$

Negating Quantified Expressions: De Morgan Laws

- Practice Exercises
- What are the negations of the following statements:
- “There is an honest politician.”
- “All americans eat cheeseburgers.”
- What are the negations of
- 宓x($x^2 > x$) and 宝x($x^2 = 2$)? Ans: (宝x($x^2 <= x$)) and 宓x($x^2 \neq 2$)
- Show that
- ¬宓x(P (x) → Q(x)) and 宝x(P (x) 穿 ¬Q(x)) are logically equivalent.

Negating Quantified Expressions: De Morgan Laws

- Let H (x) denote "x is honest." Then the statement
- "There is an honest politician" is represented by 宝x H(x), where the domain consists of all politicians.
- The negation of this statement is 宝x H(x), which is equivalent to 宓x ¬H(x).
- This negation can be expressed, as "Every politician is dishonest."
- (Note: In English, the statement "All politicians are not honest" is ambiguous. In common usage, this statement often means, "Not all politicians are honest." Consequently, we do not use this statement to express this negation.)

Negating Quantified Expressions: De Morgan Laws

- Let C(x) denote "x eats cheeseburgers." Then the statement "All Americans eat cheese-burgers" is represented by 宓x C(x), where the domain consists of all Americans. The negation of this statement is ¬宓x C(x), which is equivalent to 宝x ¬C(x).
- This negation can be expressed in several different ways, including "Some American does not eat cheeseburgers" and "There is an American who does not eat cheeseburgers."

Negation

$\neg(\forall x\ P(x))$ is logically equivalent to $\exists x\ (\neg P(x))$.

$\neg(\exists x\ P(x))$ is logically equivalent to $\forall x\ (\neg P(x))$.

This is de Morgan's law for quantifiers

Negation

Examples

Not all roses are red

$\neg\forall x\ (Rose(x) \wedge Red(x))$

$\exists x\ (Rose(x) \wedge \neg Red(x))$

Nobody is perfect

$\neg\exists x\ (Person(x) \wedge Perfect(x))$

"$\forall\ (Person(x) \wedge \neg Perfect(x))$

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

- E.g.,

  $\forall$ x (* Sqrt(x) Sqrt(x)) = x is satisfiable

  2 = 2 is valid

- E.g., definition of (full) Sibling in terms of Parent:

  $\forall$ x;y Sibling(x; y) , [: (x=y) ^ 9 m; f : (m=f) ^ Parent(m; x) ^ Parent(f ; x) ^ Parent(m; y) ^ Parent(f ; y)]

Higher-order logic?

First-order logic allows us to quantify over objects (= the first-order entities that exist in the world).

Higher-order logic also allows quantification over relations and functions.

e.g., "two objects are equal iff all properties applied to them are equivalent":

" x,y (x=y) Û (" p, p(x) Û p(y))

Higher-order logics are more expressive than first-order; however, so far we have little understanding on how to effectively reason with sentences in higher-order logic.

Logical agents for the Wumpus world

Remember: generic 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
```

1. TELL KB what was perceived
   Uses a KRL to insert new sentences, representations of facts, into KB

2. ASK KB what to do.
   Uses logical reasoning to examine actions and select best.

Interacting with FOL KBs

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

Tell(KB; Percept([Smell;Breeze;None]; 5))

Ask(KB; 9a Action(a; 5))

I.e., does KB entail any particular actions at t = 5?

Answer: Yes; {a/Shoot} ¬ substitution (binding list)

Set of solutions

Given a sentence S and a substitution ®,

S denotes the result of plugging into S; e.g.,

S = Smarter(x; y)

® = {x/Hillary; y/Bill}

S® = Smarter(Hillary;Bill)

Ask(KB; S) returns some/all ® such that KB ⊨ S®

Knowledge base for the wumpus world

"Perception"

8 b; g; t Percept([Smell; b; g]; t)) Smelt(t)

8 s; b; t Percept([s; b; Glitter]; t)) AtGold(t)

Reflex: 8 t AtGold(t)) Action(Grab; t)

Reflex with internal state: do we have the gold already?

8 t AtGold(t) ^ :Holding(Gold; t)) Action(Grab; t)

Holding(Gold; t) cannot be observed

) keeping track of change is essential

Deducing hidden properties

Properties of locations:

8 x; t At(Agent; x; t) ^ Smelt(t)) Smelly(x)

8 x; t At(Agent; x; t) ^ Breeze(t)) Breezy(x)

Squares are breezy near a pit:

Diagnostic rule—infer cause from effect

8 y Breezy(y)) 9 x Pit(x) ^ Adjacent(x; y)

Causal rule—infer effect from cause

8 x; y Pit(x) ^ Adjacent(x; y)) Breezy(y)

Neither of these is complete—e.g., the causal rule doesn't say whether squares far away from pits can be breezy

Definition for the Breezy predicate:

8 y Breezy(y) , [9 x Pit(x) ^ Adjacent(x; y)]

Situation calculus (Keeping track of change)

Facts hold in situations, rather than eternally

E.g., Holding(Gold;Now) rather than just Holding(Gold)

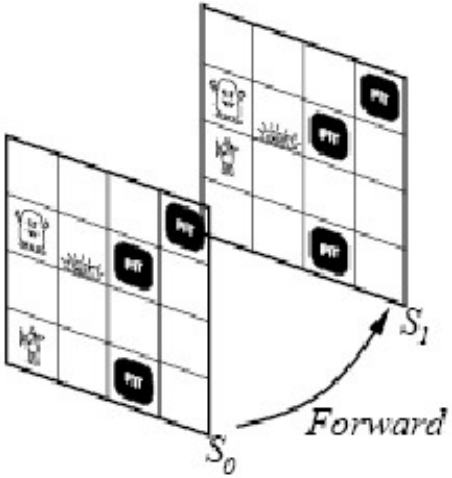
Situation calculus is one way to represent change in FOL:

Adds a situation argument to each non-eternal predicate

E.g., Now in Holding(Gold;Now) denotes a situation

Situations are connected by the Result function

Result(a; s) is the situation that results from doing a in s



Describing actions

"Effect" axiom—describe changes due to action

8 s AtGold(s)) Holding(Gold;Result(Grab; s))

"Frame" axiom—describe non-changes due to action

8 s HaveArrow(s)) HaveArrow(Result(Grab; s))

Frame problem: find an elegant way to handle non-change

(a) representation—avoid frame axioms

(b) inference—avoid repeated "copy-overs" to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—what if gold is slippery or nailed down or …

Ramification problem: real actions have many secondary consequences—what about the dust on the gold, wear and tear on gloves, …

Describing actions (cont'd)

Successor-state axioms solve the representational frame problem

Each axiom is "about" a predicate (not an action per se):

P true afterwards , [an action made P true
_ P true already and no action made P false]

For holding the gold:

8 a; s Holding(Gold;Result(a; s)) ,
[(a=Grab ^ AtGold(s))
_ (Holding(Gold; s) ^ a ¹ Release)]

Making plans

Initial condition in KB:

At(Agent; [1; 1]; S0)

At(Gold; [1; 2]; S0)

Query: Ask(KB; 9 s Holding(Gold; s))

i.e., in what situation will I be holding the gold?

Answer: {s/Result(Grab;Result(Forward; S0))}

i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at S0 and that S0 is the only situation described in the KB

Making plans: A better way

Represent plans as action sequences [a_1; a_2 ; … ; a_n]

PlanResult(p; s) is the result of executing p in s

Then the query Ask(KB; 9 p Holding(Gold;PlanResult(p; S0)))
has the solution {p/[Forward; Grab]}

Definition of PlanResult in terms of Result:

8 s PlanResult([]; s) = s

8 a; p; s PlanResult([ajp]; s) = PlanResult(p;Result(a; s))

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

Summary

First-order logic:

- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define wumpus world

Situation calculus:

- conventions for describing actions and change in FOL
- can formulate planning as inference on a situation calculus KB

Inference in FOL [AIMA Ch. 9]

Reducing first-order inference to propositional inference

Unification

Generalized Modus Ponens

Forward and backward chaining

Logic programming

Resolution

A brief history of reasoning

450b.c.	**Stoics**	propositional logic, inference (maybe)
322b.c.	**Aristotle**	"syllogisms" (inference rules), quantifiers
1565	**Cardano**	probability theory (propositional logic+ uncertainty)
1847	**Boole**	propositional logic (again)
1879	**Frege**	first-order logic
1922	**Wittgenstein**	proof by truth tables
1930	**Gödel**	宝 complete algorithm for FOL
1930	**Herbrand**	complete algorithm for FOL (reduce to propositional)
1931	**Gödel**	¬宝 complete algorithm for arithmetic
1960	**Davis/Putnam**	"practical" algorithm for propositional logic
1965	**Robinson**	"practical" algorithm for FOL—resolution

Universal instantiation (UI)

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

$$\frac{\forall v\ \alpha}{\textbf{SUBST}(\{v/g\}, \alpha)}$$

for any variable v and ground term g

E.g., $\forall x\ King(x) \Rightarrow Greedy(x) \Rightarrow Evil(x)$ yields

$King(John) \Rightarrow Greedy(John) \Rightarrow Evil(John)$
$King(Richard) \Rightarrow Greedy(Richard) \Rightarrow Evil(Richard)$
$King(Father(John)) \Rightarrow Greedy(Father(John)) \Rightarrow Evil(Father(John))$
...

Existential instantiation (EI)

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

$$\frac{\exists v\ \alpha}{\textbf{SUBST}(\{v/k\}, \alpha)}$$

E.g., $\exists x\ Crown(x) \land OnHead(x, John)$ yields
$Crown(C_1) \land OnHead(C_1, John)$

provided C_1 is a new constant symbol, called a Skolem constant

Another example: from $\exists x\ d(x^y)/dy = x^y$ we obtain
$d(e^y)/dy = e^y$

provided e is a new constant symbol

Existential instantition contd.

UI can be applied several times to add new sentences; the new KB is logically equivalent to the old

EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old, but is satisfiable iff the old KB was satisfiable

Reduction to propositional inference

Suppose the KB contains just the following:

宓x King(x) 弁 Greedy(x) 宅 Evil(x)
King(John)
Greedy(John)
Brother(Richard, John)

Instantiating the universal sentence in all possible ways, we have

King(John) 弁 Greedy(John) 宅 Evil(John)
King(Richard) 弁 Greedy(Richard) 宅 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.

Claim: a ground sentence is entailed by new KB iff entailed by original KB

Claim: every FOL KB can be propositionalized so as to preserve entailment

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)))

Reduction contd.

Theorem: Herbrand (1930). If a sentence α is entailed by an FOL KB, it is entailed by a finite subset of the propositional KB

Idea: For n **= 0 to** ∞ do

create a propositional KB by instantiating with depth-n terms and see if α is entailed by KB

Problem: works if α is entailed, loops if α is not entailed

Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable

Problems with propositionalization

Propositionalization seems to generate lots of irrelevant sentences.

E.g., from

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

$King(John)$

$\forall y\ Greedy(y)$

$Brother(Richard, John)$

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

With p k-ary predicates and n constants, there are $p \cdot n^k$ instantiations

With function symbols, it gets much much worse!

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 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)	fail

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

Generalized Modus Ponens (GMP)

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

where $p_i'\theta = p_i\theta$ for all i

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)
$q\theta$ is Evil(John)	

GMP used with KB of definite clauses (exactly one positive literal)

All variables assumed universally quantified

Horn Form

The completeness of resolution makes it a very important inference method. In many practical situations, however, the full power of resolution is not needed. Real-world knowledge bases often contain only clauses of a restricted kind called **Horn clauses**. A Horn clause is a disjunction of literals of which *at most one is positive*. For example, the clause $(\neg L_{1,1} \lor \neg Breeze \lor B_{1,1})$, where $L_{1,1}$ means that the agent's location is [1,1], is a Horn clause, whereas $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1})$ is not.

The restriction to just one positive literal may seem somewhat arbitrary and uninteresting, but it is actually very important for three reasons:

1. Every Horn clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal. (See Exercise 7.12.) For example, the Horn clause $(\neg L_{1,1} \lor \neg Breeze \lor B_{1,1})$ can be written as the implication $(L_{1,1} \land Breeze) \Rightarrow B_{1,1}$. In the latter form, the sentence is much easier to read: it says that if the agent is in [1,1] and there is a breeze, then [1,1] is breezy. People find it easy to read and write sentences in this form for many domains of knowledge.

2. Inference with Horn clauses can be done through the **forward chaining** and **backward chaining** algorithms, which we explain next. Both of these algorithms are very natural, in that the inference steps are obvious and easy to follow for humans.

3. Deciding entailment with Horn clauses can be done in time that is *linear* in the size of the knowledge base.

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 Col. West is a criminal

Example knowledge base contd.

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

American(x) ⊓ Weapon(y) ⊓ Sells(x,y,z) ⊓ Hostile(z) ⊒ Criminal(x)

Nono … has some missiles, i.e.,∃x Owns(Nono,x) ⊓ Missile(x):

Owns(Nono,M$_{\boxed{1}}$) **and** Missile(M$_{\boxed{1}}$)

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

∀x Missile(x) ⊓ Owns(Nono,x) ⊒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⊒ Weapon(x)

An enemy of America counts as "hostile":

Enemy(x,America) ⊒ Hostile(x)

West, who is American …

American(west)

The country Non, an enemy of America

Enemy(Nono,America)

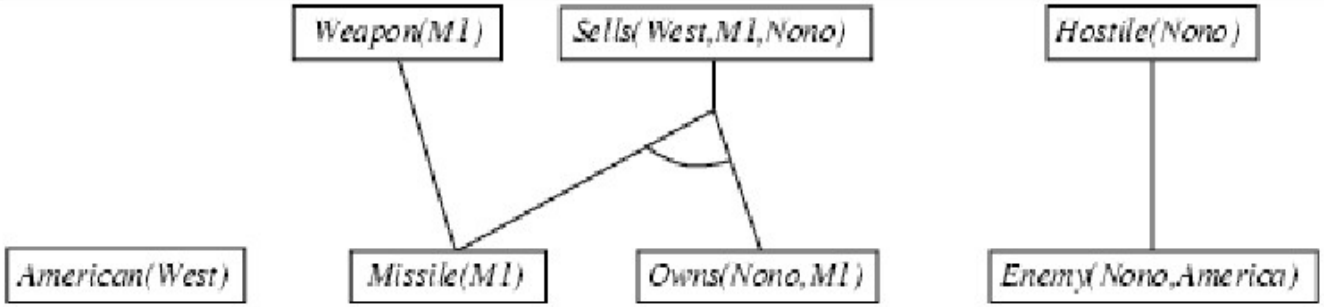
Forward chaining algorithm

```
function FOL-FC-ASK(KB, α) returns a substitution or false
  repeat until new is empty
      new ← { }
      for each sentence r in KB do
          ( p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-APART(r)
          for each θ such that (p1 ∧ ... ∧ pn)θ = (p'1 ∧ ... ∧ p'n)θ
                      for some p'1, ..., p'n in KB
              q' ← SUBST(θ, q)
              if q' is not a renaming of a sentence already in KB or new then do
                  add q' to new
                  φ ← UNIFY(q', α)
                  if φ is not fail then return φ
      add new to KB
  return false
```

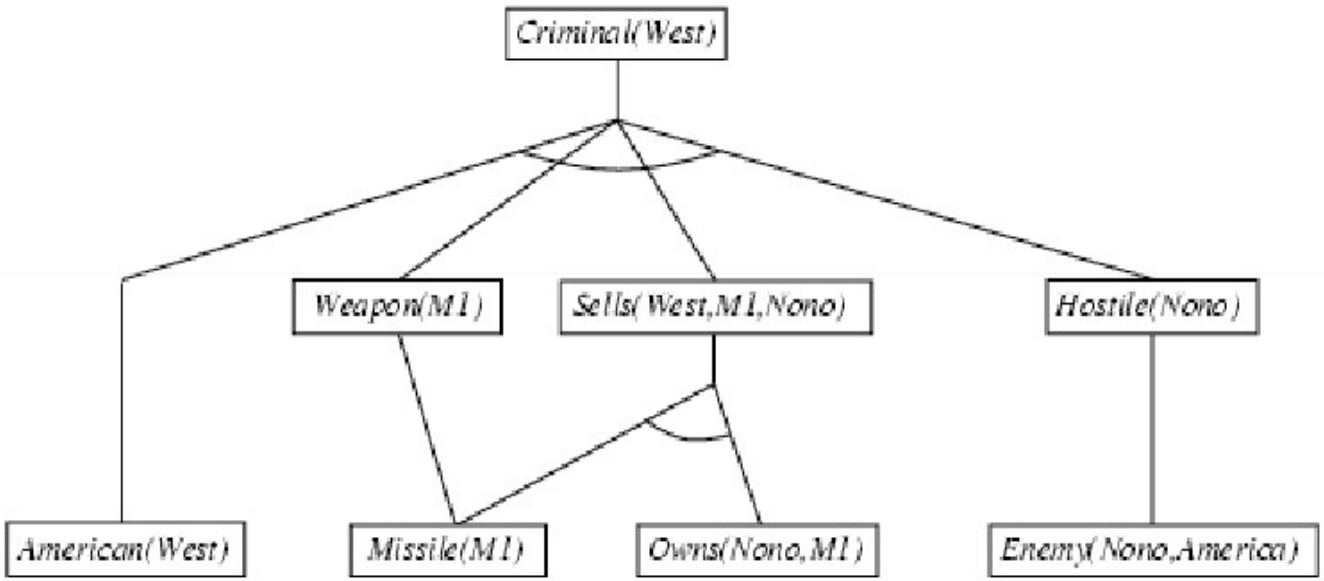
Forward chaining proof

$American(West)$ $Missile(M1)$ $Owns(Nono, M1)$ $Enemy(Nono, America)$

Forward chaining proof



Forward chaining proof



Properties of forward chaining

Sound and complete for first-order definite clauses (proof similar to propositional proof)

Datalog = first-order definite clauses + no functions (e.g., crime KB)

FC terminates for Datalog in poly iterations: at most $p \cdot n^k$ literals

May not terminate in general if α is not entailed

This is unavoidable: entailment with definite clauses is semidecidable

Efficiency of forward chaining

Simple observation: no need to match a rule on iteration k if a premise wasn't added on iteration k-1

宰 match each rule whose premise contains a newly added literal

Matching itself can be expensive

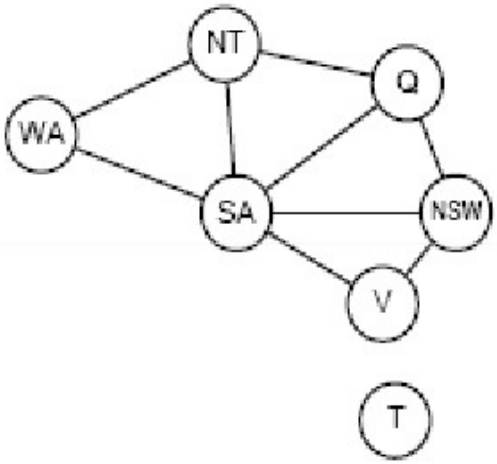
Database indexing allows O(1) retrieval of known facts

e.g., query Missile(x) retrieves Missile(M$\boxed{\text{W}}$)

Matching conjunctive premises against known facts is NP-hard

Forward chaining is widely used in deductive databases

Hard matching example



$$Diff(wa, nt) \wedge Diff(wa, sa) \wedge$$
$$Diff(nt, q) Diff(nt, sa) \wedge$$
$$Diff(q, nsw) \wedge Diff(q, sa) \wedge$$
$$Diff(nsw, v) \wedge Diff(nsw, sa) \wedge$$
$$Diff(v, sa) \Rightarrow Colorable()$$

$$Diff(Red, Blue) \quad Diff(Red, Green)$$
$$Diff(Green, Red) \quad Diff(Green, Blue)$$
$$Diff(Blue, Red) \quad Diff(Blue, Green)$$

Colorable() is inferred iff the CSP has a solution
CSPs include 3SAT as a special case, hence matching is NP-hard

Backward chaining algorithm

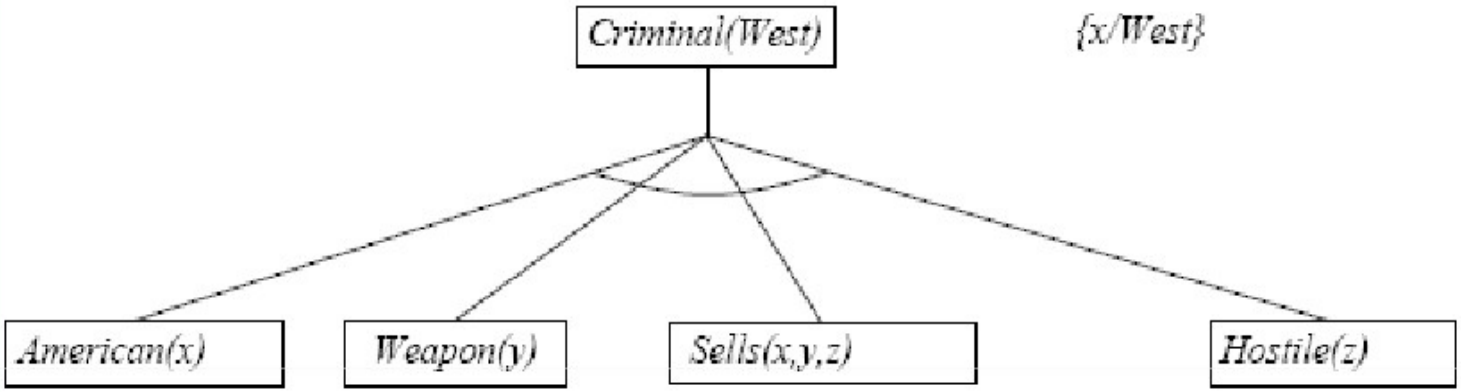
```
function FOL-BC-ASK(KB, goals, θ) returns a set of substitutions
   inputs: KB, a knowledge base
           goals, a list of conjuncts forming a query (θ already applied)
           θ, the current substitution, initially the empty substitution { }
   local variables: answers, a set of substitutions, initially empty

   if goals is empty then return {θ}
   q' ← SUBST(θ, FIRST(goals))
   for each sentence r in KB
           where STANDARDIZE-APART(r) = ( p1 ∧ ... ∧ pn ⇒ q)
           and θ' ← UNIFY(q, q') succeeds
       new_goals ← [p1, ..., pn | REST(goals)]
       answers ← FOL-BC-ASK(KB, new_goals, COMPOSE(θ', θ)) ∪ answers
   return answers
```

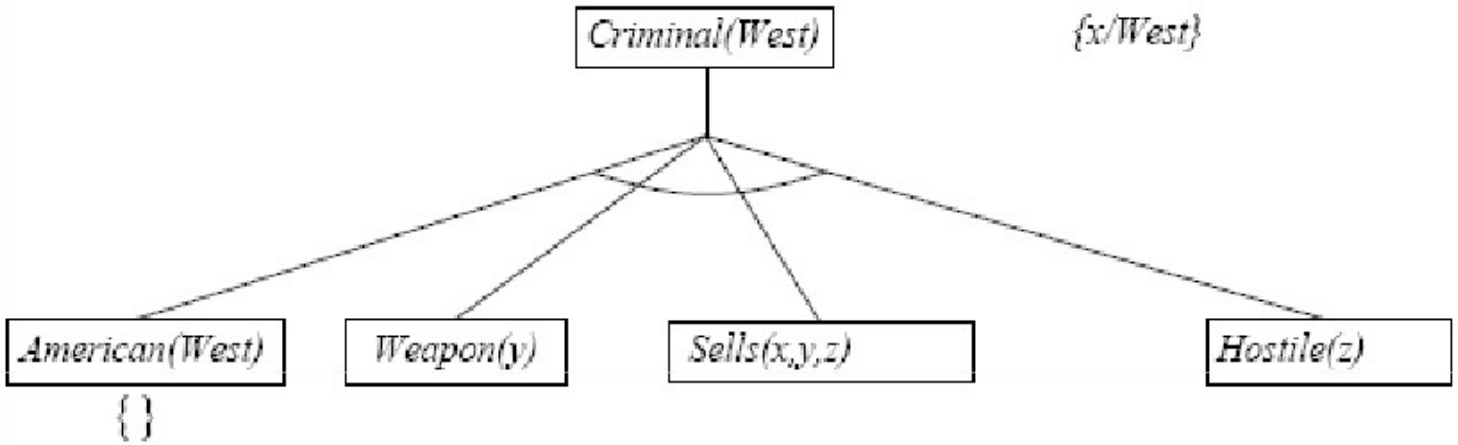
Backward chaining example

$Criminal(West)$

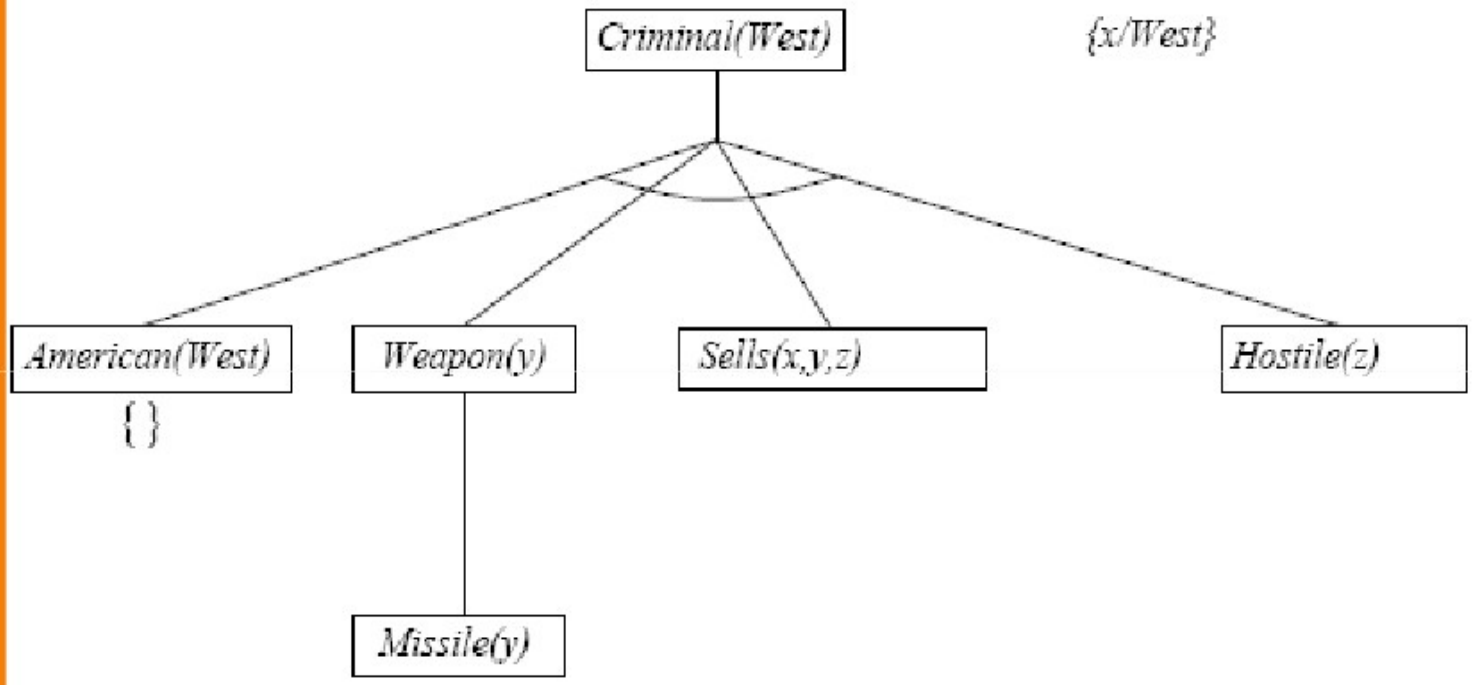
Backward chaining example



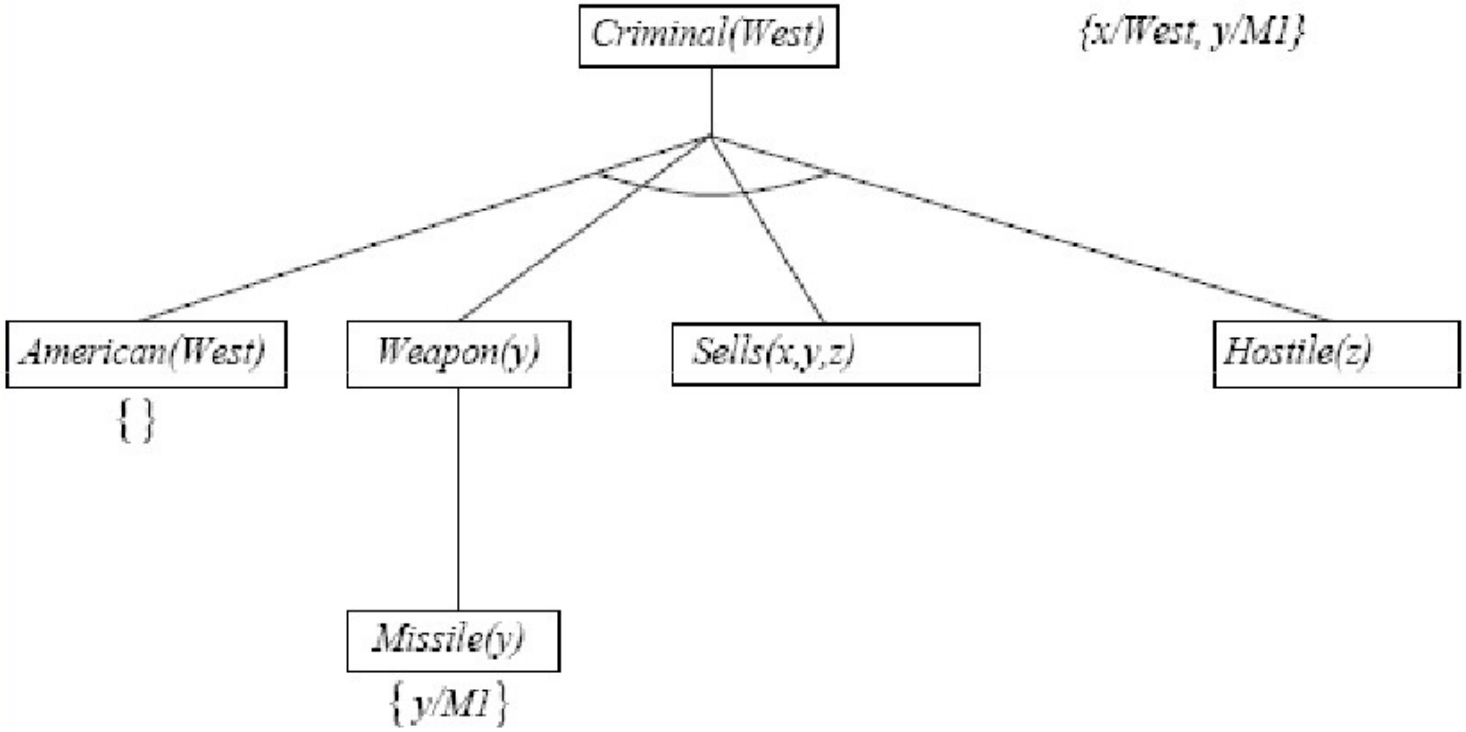
Backward chaining example



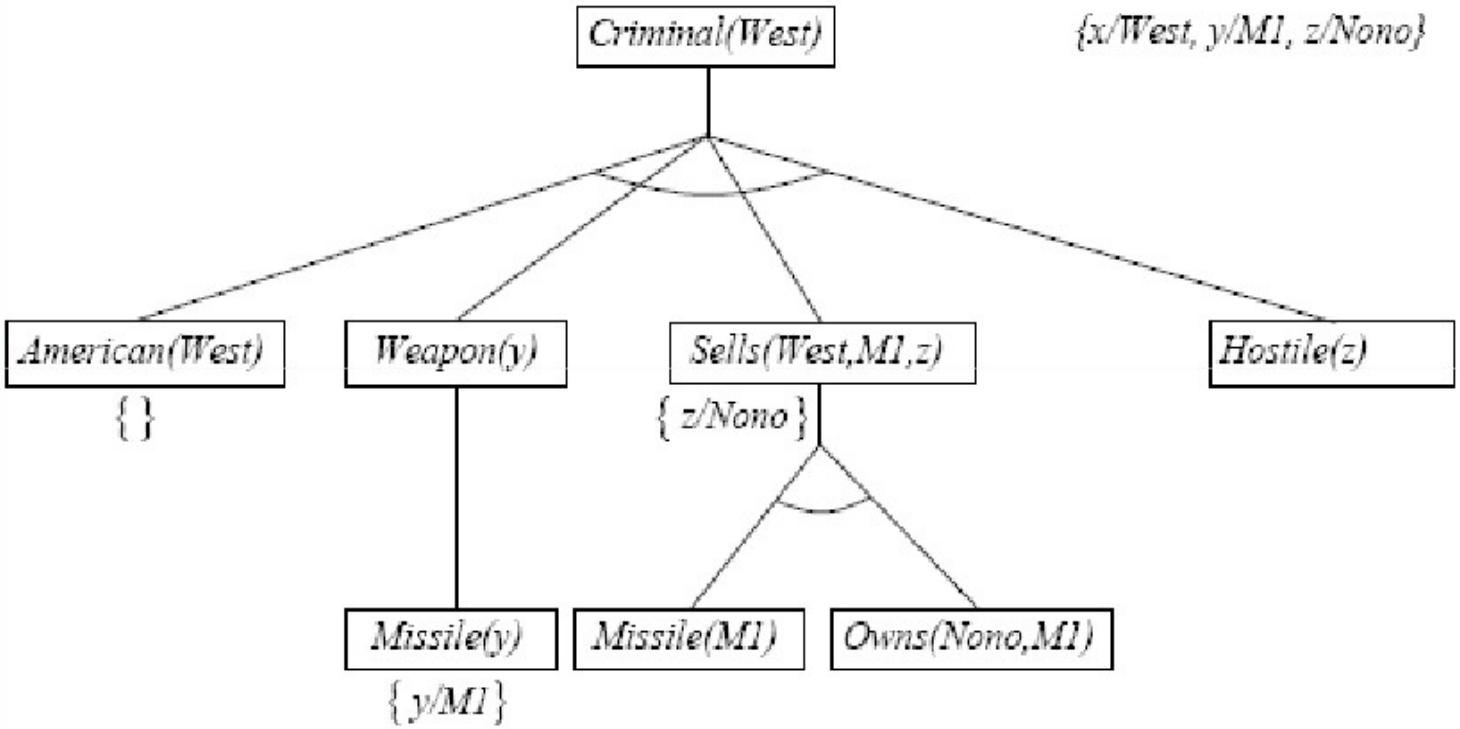
Backward chaining example



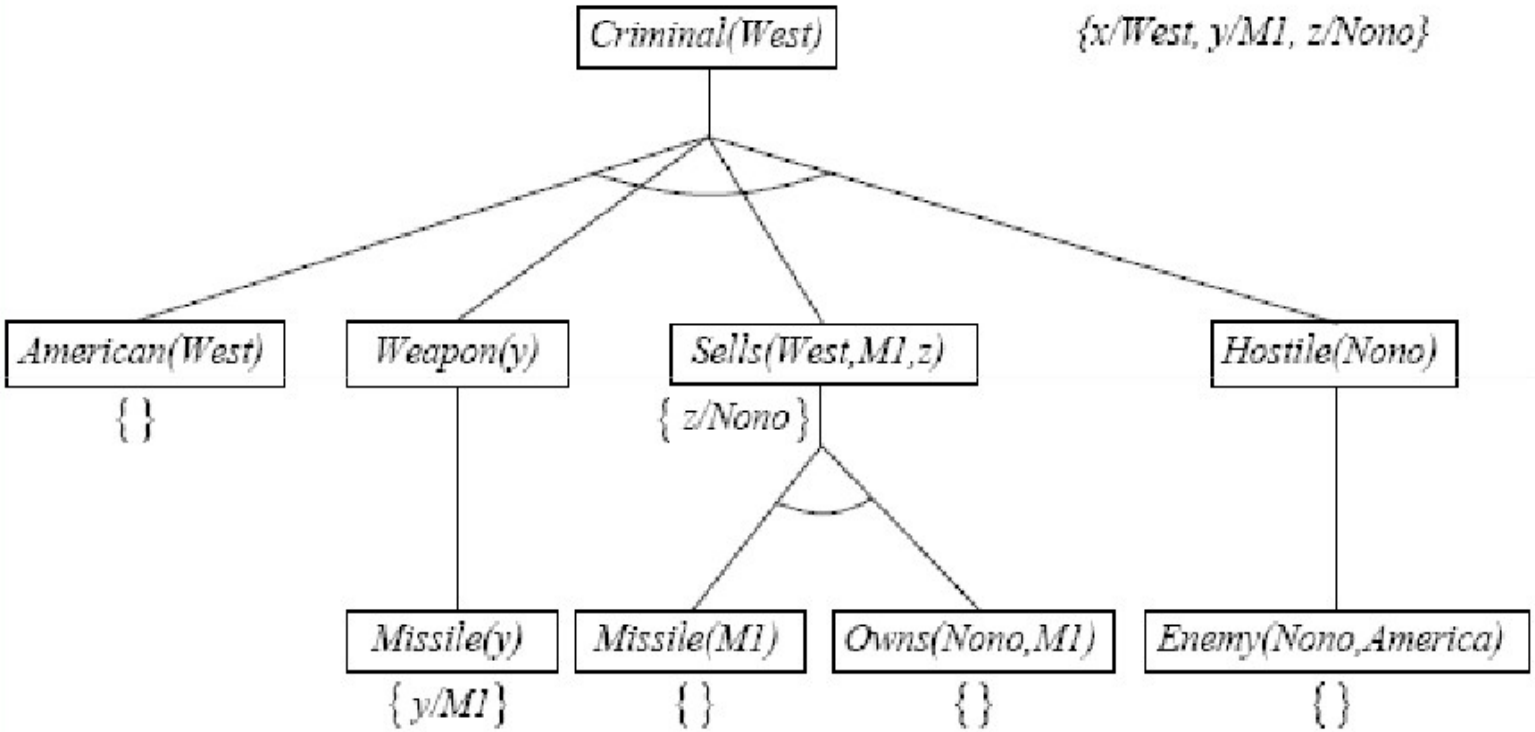
Backward chaining example



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
  宰 fix by checking current goal against every goal on stack

Inefficient due to repeated subgoals (both success and failure)
  宰 fix using caching of previous results (extra space!)

Widely used (without improvements!) for logic programming

Logic programming

Sound bite: computation as inference on logical KBs

Logic programming	Ordinary programming
1. Identify problem	Identify problem
2. Assemble information	Assemble information
3. Tea break	Figure out solution
4. Encode information in KB	Program solution
5. Encode problem as facts	Encode problem as data
6. Ask queries	Apply program to data
7. Find false facts	Debug procedural errors

Should be easier to debug Capital(NewYork,US) than x := x + 2!

Resolution: brief summary

Full first-order version:

$$\frac{l_1 \vee \ldots \vee l_k,\ m_1 \vee \ldots \vee m_n}{(l_1 \vee \ldots\ l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)\theta}$$

where $\text{UNIFY}(l_i, \neg m_j)=\theta$.

For example,

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

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

Apply resolution steps to $CNF(KB \vee \neg\alpha)$; complete for FOL

Conversion to CNF

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: $\neg\forall x, p \equiv \exists x \; \neg p$, $\neg\exists x, p \equiv \forall x \; \neg p$:

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

Conversion to CNF contd.

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

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

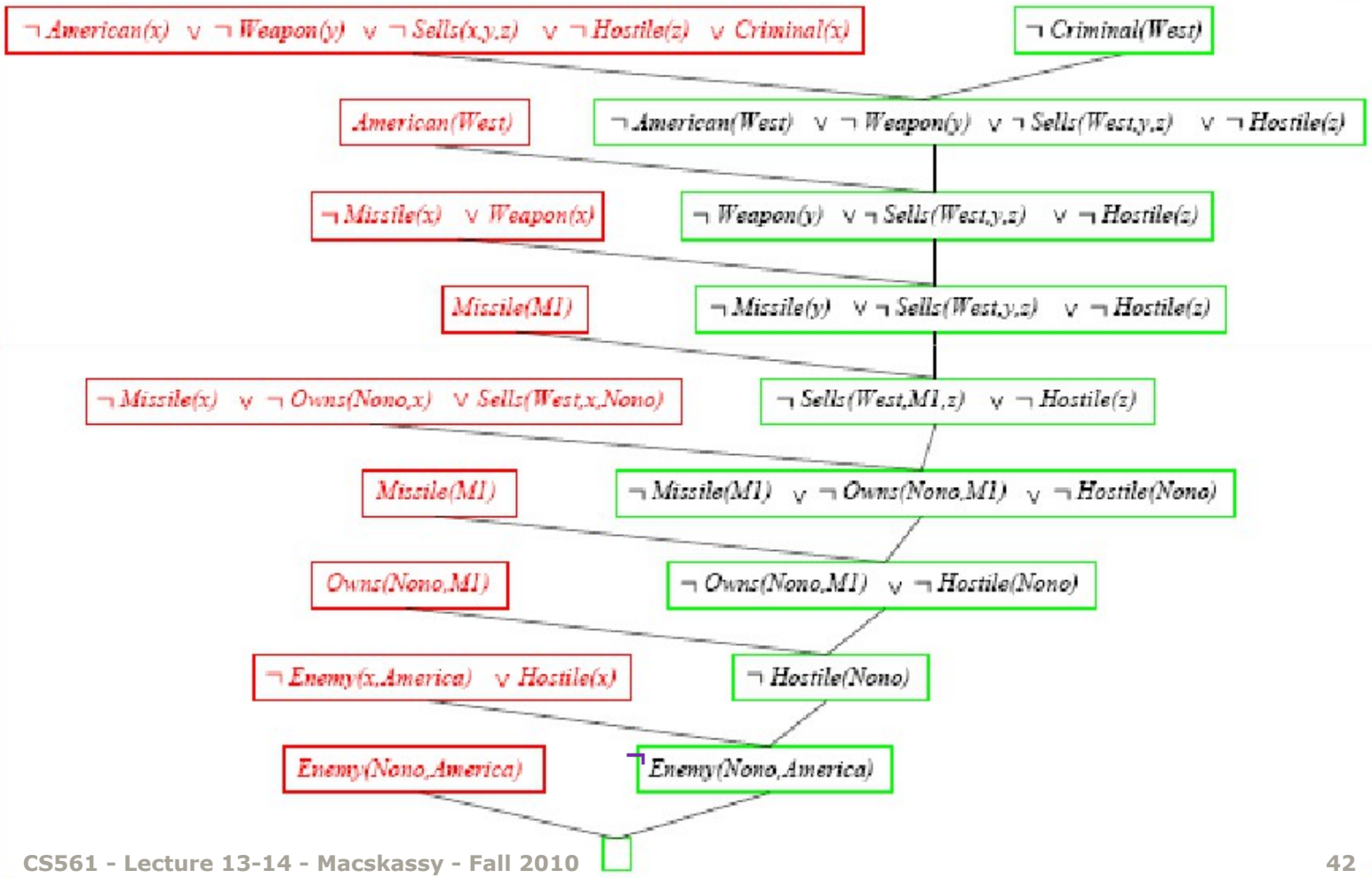
4. Skolemize: a more general form of existential instantiation.
   Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

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

5. Drop universal quantifiers:

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

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

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

Resolution proof: definite clauses