LECTURE NOTES–UNIT III

KNOWLEDGE AND REASONING

ECS302

ARTIFICIAL

INTELLIGENCE

Module III Lecture Notes
[*Knowledge and Reasoning*]

Syllabus

***Knowledge Representation*:** Logical Agents: Knowledge based agents, the wumpus world, logic.

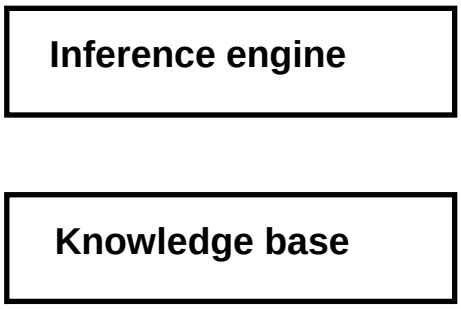
Propositional Logic: A very simple logic, reasoning patterns in propositional logic, effective propositional inference;

First-Order Logic: Representation revisited, syntax and semantics of first order logic, using first-Order logic.

Knowledge Based Agents

A **knowledge-based agent** needs a KB and an inference mechanism. It operates by storing sentences in its **knowledge** base, inferring new sentences with the inference mechanism, and using them to deduce which actions to take. ... The interpretation of a sentence is the fact to which it refers.

Knowledge Bases:



Domain−independent algorithms

Domain−specific content

Knowledge base = set of sentences in a formal language

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

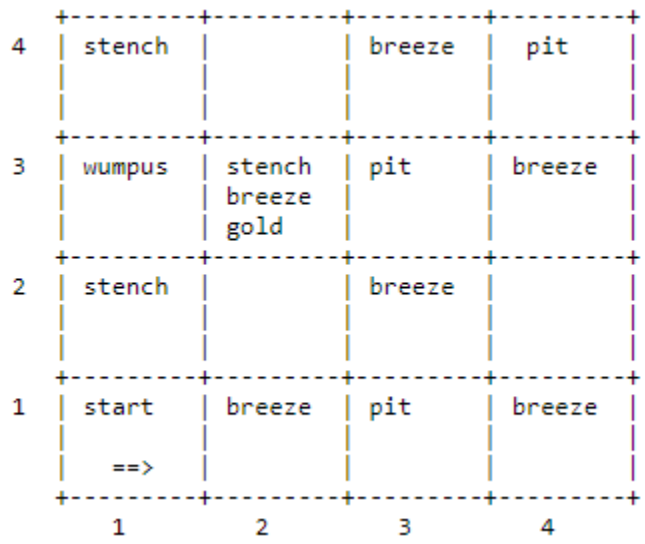
Tell it what it needs to know - Then it can Ask itself what to do—answers should follow from the KB Agents can be viewed at the knowledge level i.e., what they know, regardless of how implemented or at the implementation level i.e.,data structures in KB and algorithms that manipulate them.

The Wumpus World:

A variety of "worlds" are being used as examples for Knowledge Representation, Reasoning, and Planning. Among them the Vacuum World, the Block World, and the Wumpus World.

The Wumpus World was introduced by Genesereth, and is discussed in Russell-Norvig. The Wumpus World is a simple world (as is the Block World) for which to represent knowledge and to reason.

It is a cave with a number of rooms, represented as a 4x4 square.



Rules of the Wumpus World

The **neighborhood** of a node consists of the four squares north, south, east, and west of the given square.

In a square the agent gets a vector of percepts, with components

Stench, Breeze, Glitter, Bump, Scream

For example [***Stench, None, Glitter, None, None***]

- ***Stench*** is perceived at a square iff the wumpus is at this square or in its neighborhood.
- ***Breeze*** is perceived at a square iff a pit is in the neighborhood of this square.
- ***Glitter*** is perceived at a square iff gold is in this square

- ***Bump*** is perceived at a square iff the agent goes Forward into a wall
- ***Scream*** is perceived at a square iff the wumpus is killed anywhere in the cave

An agent can do the following actions (one at a time):

Turn (Right), Turn (Left), Forward, Shoot, Grab, Release, Climb

- The agent can go forward in the direction it is currently facing, or Turn Right, or Turn Left. Going forward into a wall will generate a Bump percept.
- The agent has a single arrow that it can shoot. It will go straight in the direction faced by the agent until it hits (and kills) the wumpus, or hits (and is absorbed by) a wall.
- The agent can grab a portable object at the current square or it can Release an object that it is holding.
- The agent can climb out of the cave if at the Start square. The Start square is (1,1) and initially the agent is facing east. The agent dies if it is in the same square as the wumpus. The objective of the game is to kill the wumpus, to pick up the gold, and to climb out with it.

Representing our Knowledge about the Wumpus World

Percept(x, y)

Where x must be a percept vector and y must be a situation. It means that at situation y the agent perceives x. For convenience we introduce the following definitions:

- Percept([***Stench,y,z,w,v***],t) = > Stench(t)
- Percept([***x,Breeze,z,w,v***],t) = > Breeze(t)
- Percept([***x,y,Glitter,w,v***],t) = > AtGold(t)

Holding(x, y)

Where x is an object and y is a situation. It means that the agent is holding the object x in situation y.

Action(x, y)

Where x must be an action (i.e. Turn (Right), Turn (Left), Forward,) and y must be a situation. It means that at situation y the agent takes action x.

At(x, y, z)

Where x is an object, y is a Location, i.e. a pair [u, v] with u and v in {1, 2, 3, 4}, and z is a situation. It means that the agent x in situation z is at location y.

Present(x, s)

Means that object x is in the current room in the situation s.

Result(x, y)

It means that the result of applying action x to the situation y is the situation Result(x,y). Note that Result(x,y) is a term, not a statement.
For example we can say

- Result(Forward, S0) = S1
- Result(Turn(Right),S1) = S2

These definitions could be made more general. Since in the Wumpus World there is a single agent, there is no reason for us to make predicates and functions relative to a specific agent. In other "worlds" we should change things appropriately.

Wumpus World: PEAS Description

Performance measure

Gold +1000, death−1000
−1perstep,−10forusingthearrow

Environment

Squares adjacent to wumpus are smelly Squares adjacent to pit are breezy Glitter if gold is in the same square

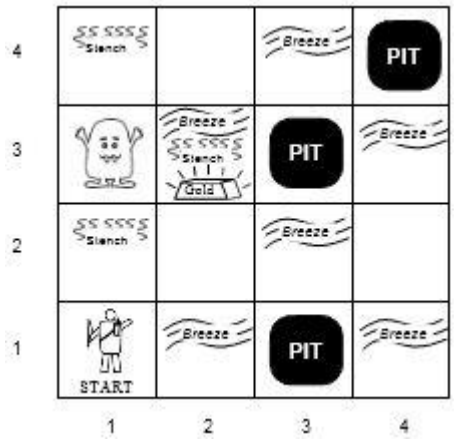
Shooting kills wumpus if you are facing it Shooting uses up the only arrow Grabbing picks up gold if in same square Releasing drops the gold in same square

Actuators

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

Sensors

Breeze, Glitter, Smell



Wumpus World Characterization

Observable?? No—only local perception

Deterministic?? Yes—outcomes exactly specified

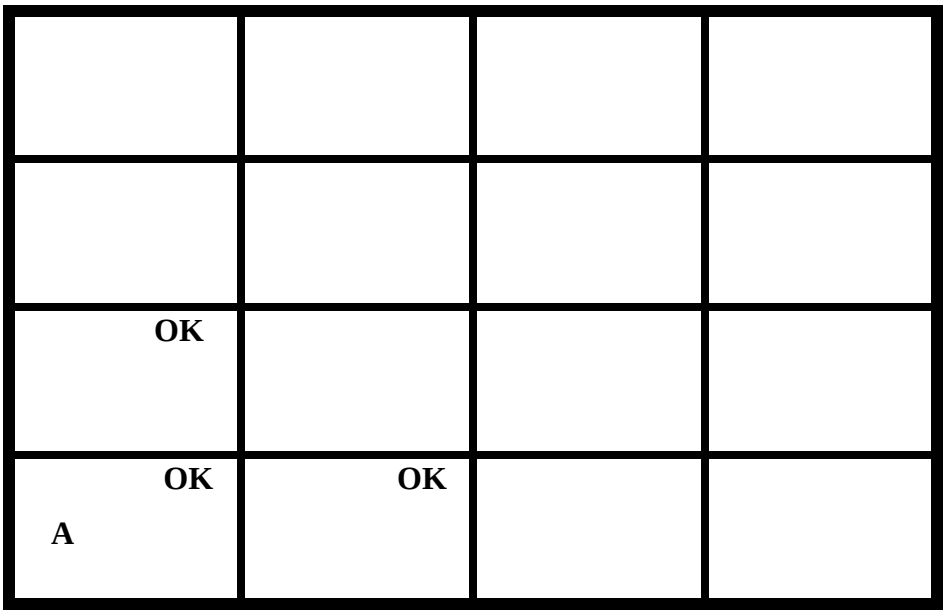
Episodic?? No—sequential at the level of actions

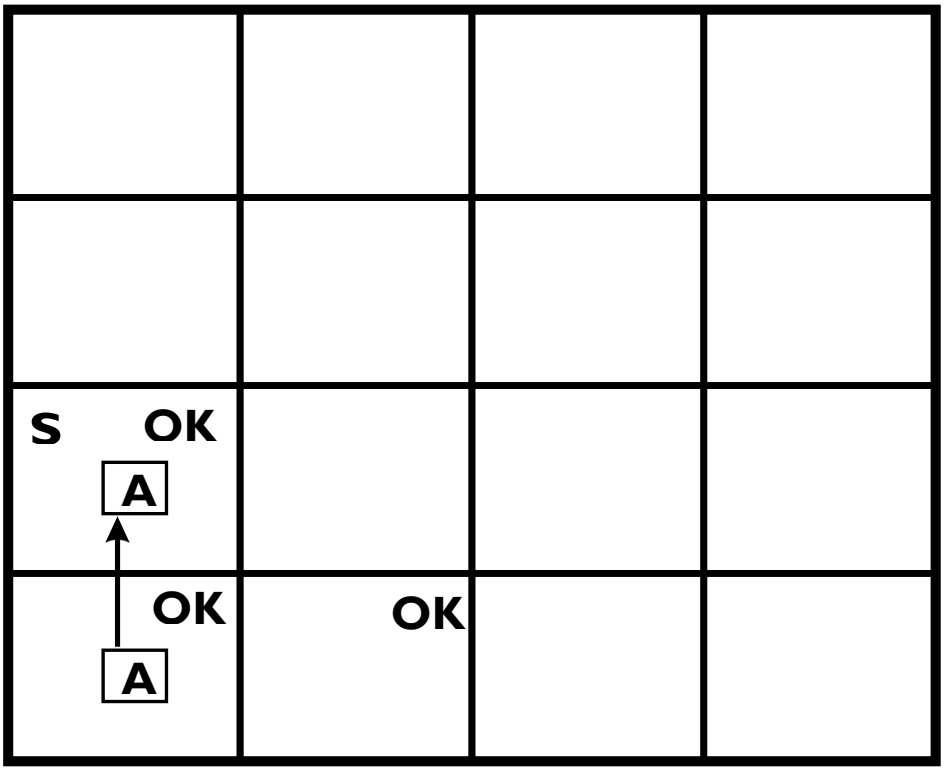
Static?? Yes—Wumpus and Pits do not move

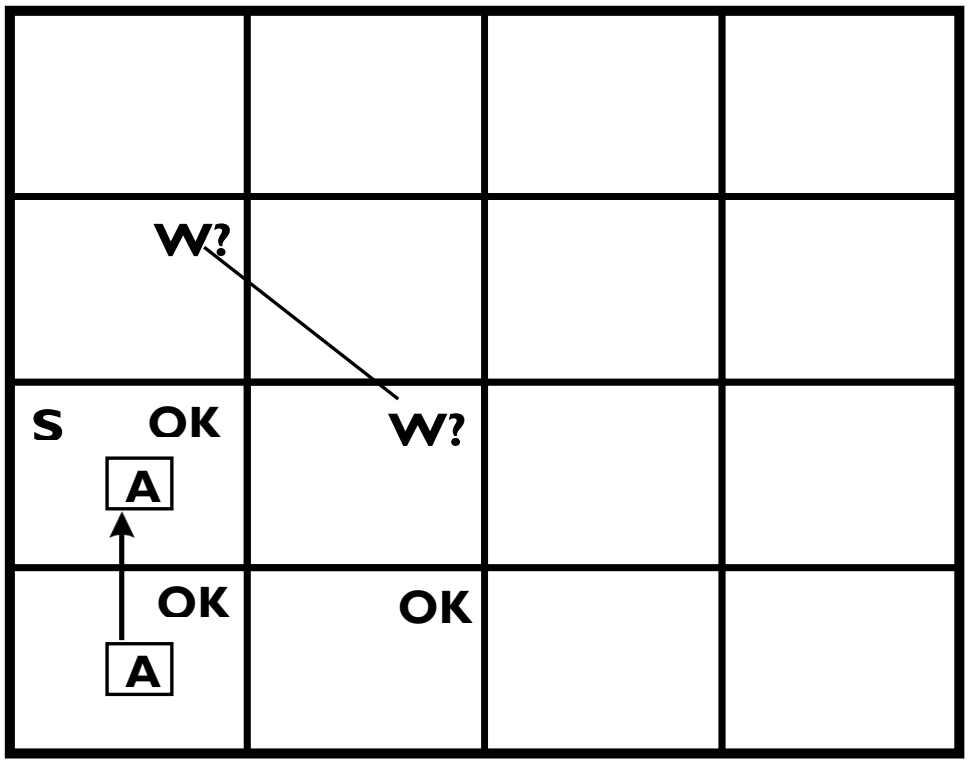
Discrete?? Yes

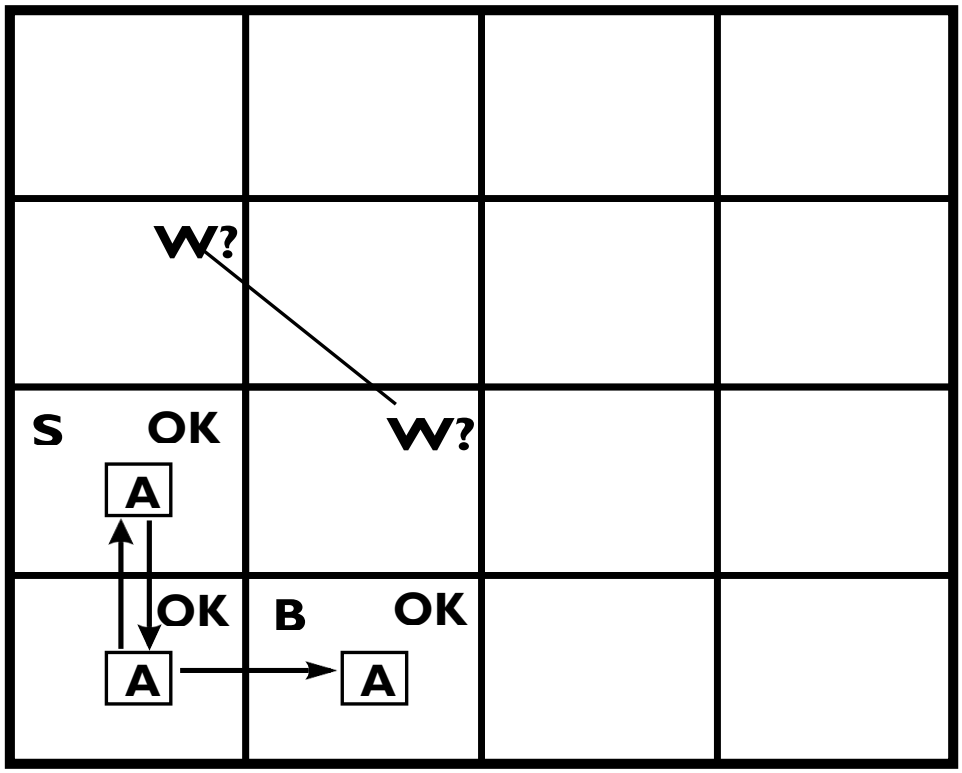
Single-agent?? Yes—Wumpus is essentially a natural feature

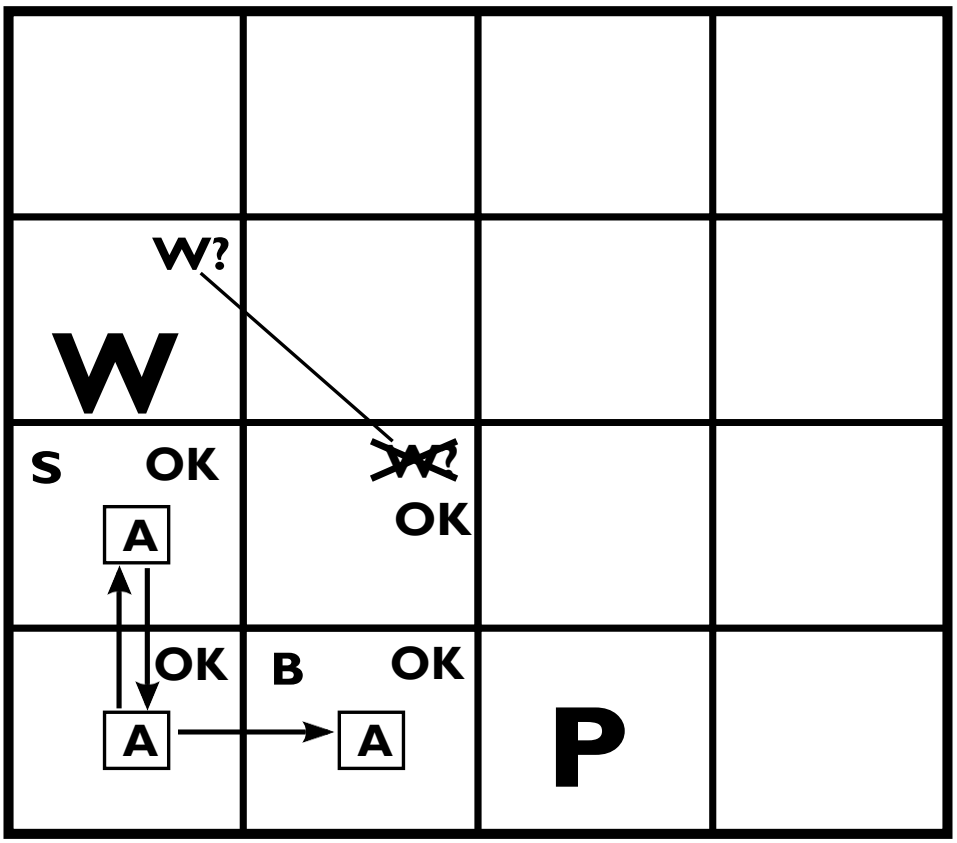
Exploring a Wumpus World

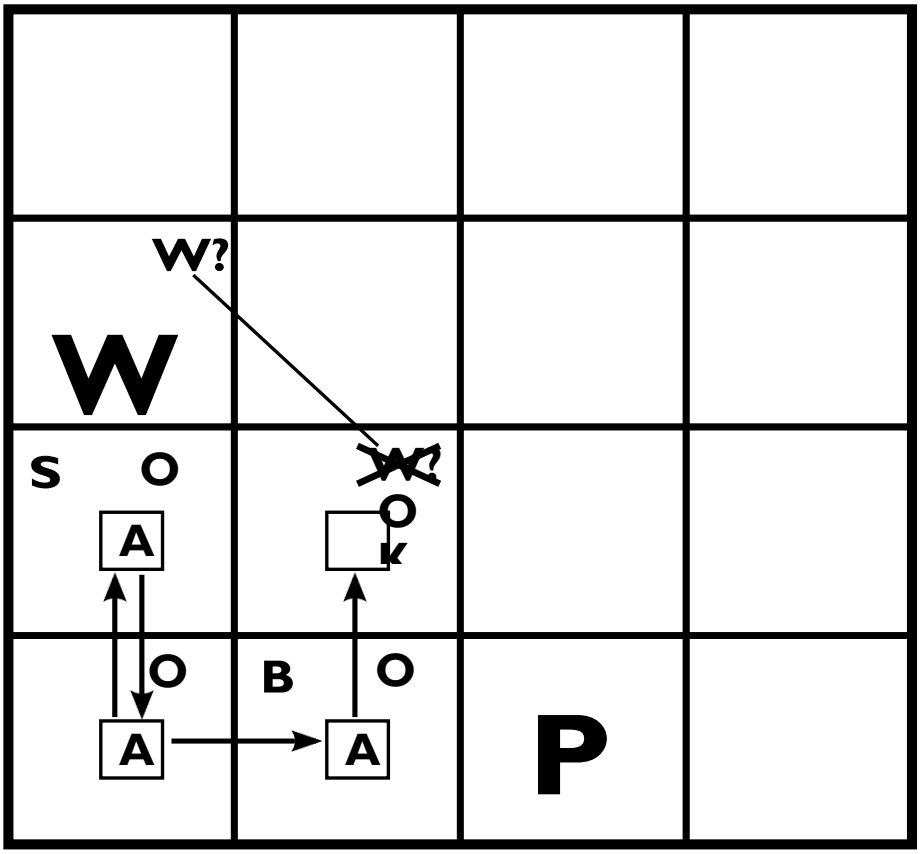

OK
OK
OK
A

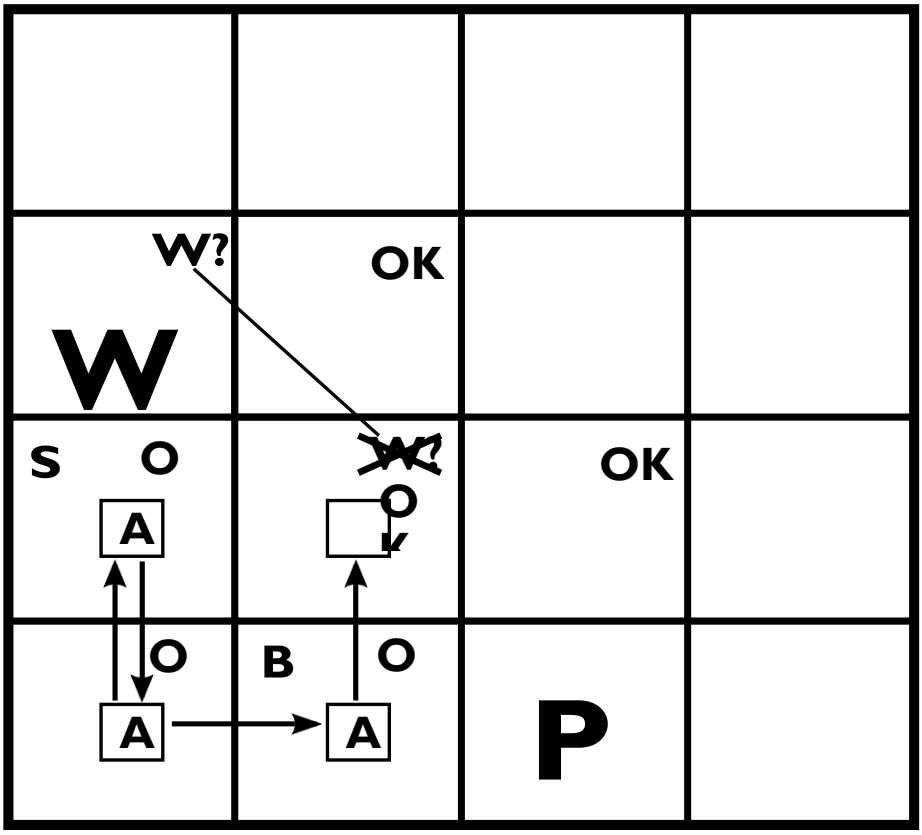

S
OK
A
OK
OK
A

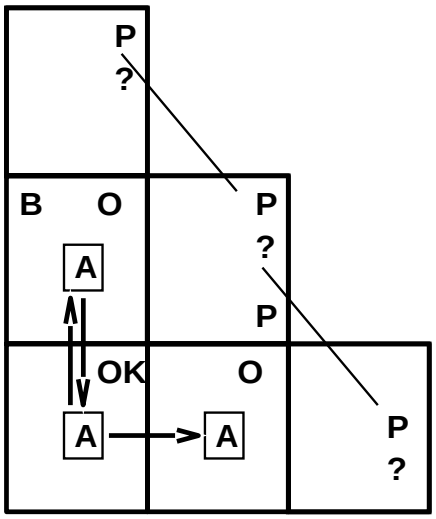

W?
S
OK
W?
A
OK
OK
A


W?
S
OK
W?
A
OK
B
OK
A
A


W?
W
S
OK
W?
OK
A
OK
B
OK
A
A
P


W?
W
S
O
A
W?
O
K
O
B
O
A
A
P





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

⇒ no safe actions

Assumingpitsuniformlydistributed,

(2,2) has pit w/ prob 0.86, vs. 0.31

Smell in (1,1)

⇒cannot move

Can use a strategy of coercion:

shoot straight ahead

S

wumpus was there ⇒dead

A

Wumpus wasn't there⇒ safe

Logic in general

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

Syntaxdefinesthesentencesinthelanguage

Semantics define the "meaning" ofsentences; 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 trueiff the number$x+2$ isnolessthanthenumber$y$ $x+2\geq y$ 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: $KB \models \alpha$

A knowledge base KB entails a sentence α if and only if α is true in all worlds where KB is true.

E.g., the KB containing "There's a pit ahead" and "There's gold to the left" entails "Either there's a pit ahead or gold to the left"

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

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

Models

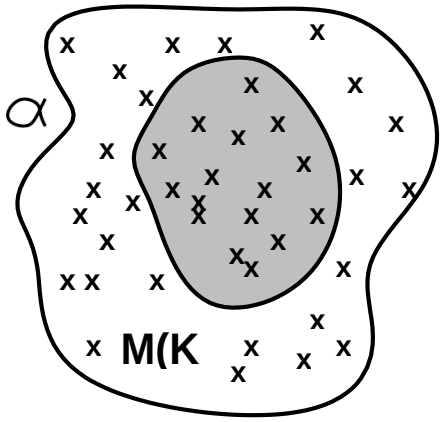
Logicians= typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

We say m is a model of a sentence α if α is true in m $M(\alpha)$ is the set of all models of α

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

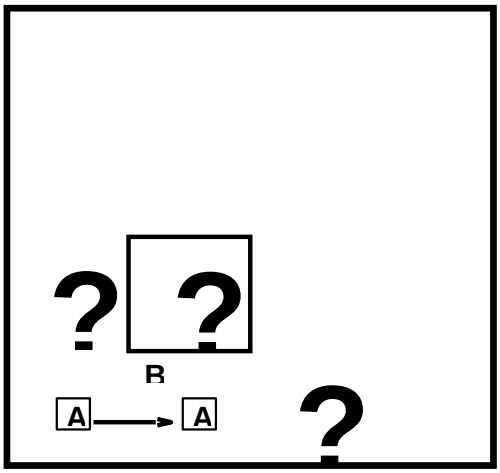
E.g. KB = {there's a pit ahead, there's gold to the left }

α = there's gold to the left



Entailment in the Wumpus World

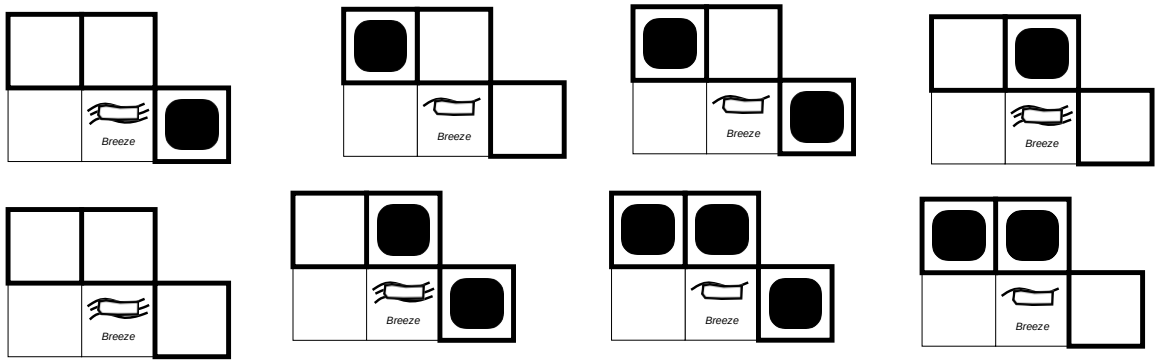
Situation after detecting nothing in[1,1], moving right, breeze in [2,1]

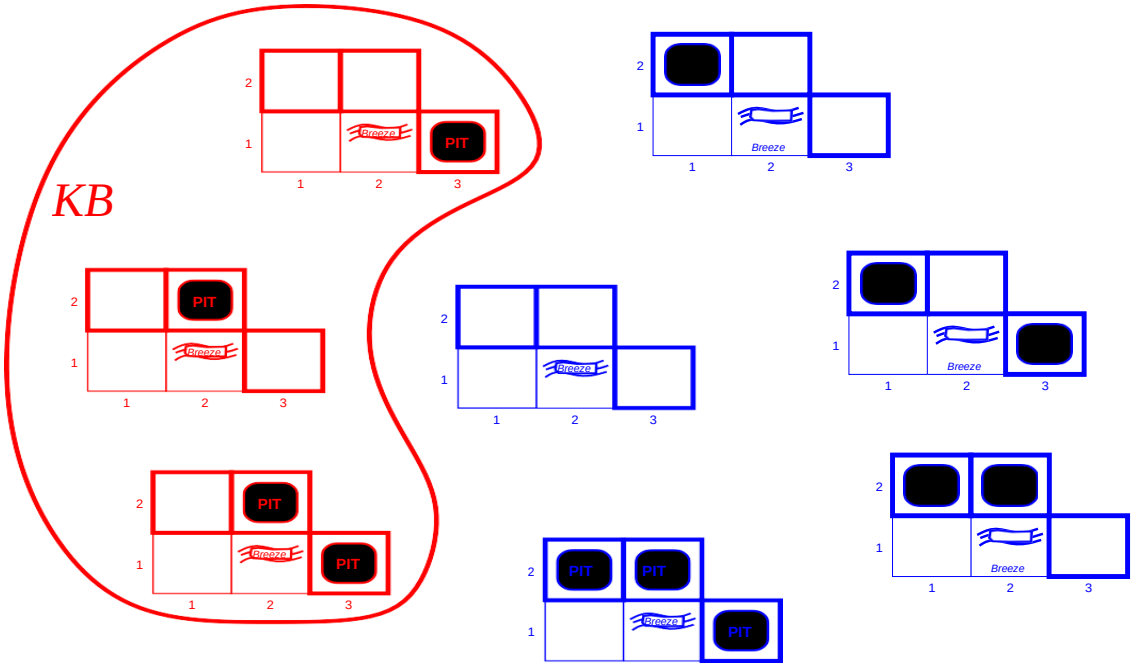


Consider possible models for ? ' s assuming only pits

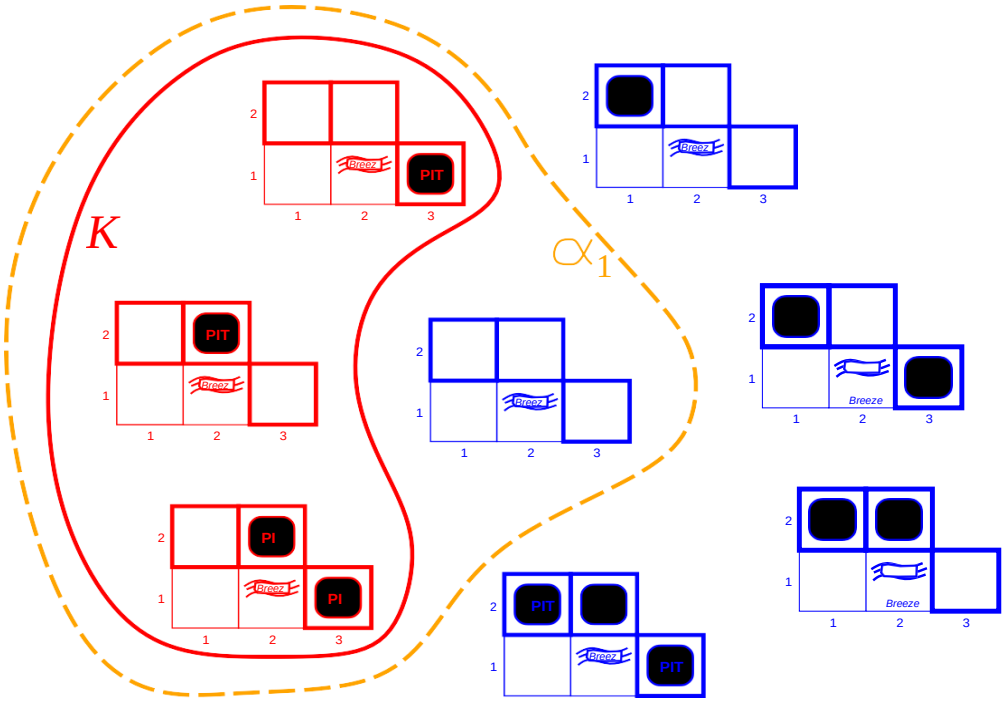
3Booleanchoices ⇒ 8 possible model

Wumpus Models



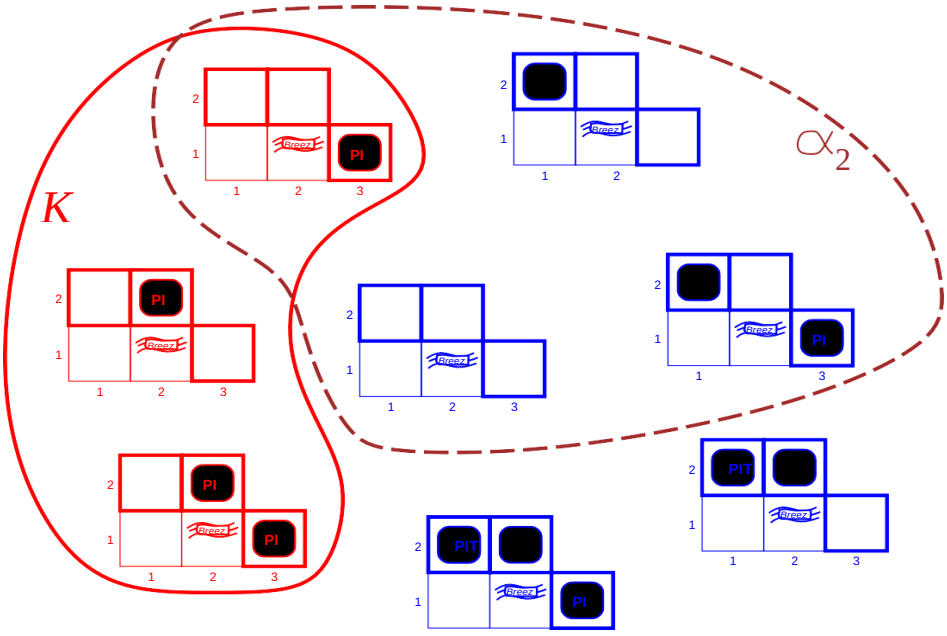


KB = wumpus-world rules + observations



KB = wumpus-world rules + observations

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



KB = wumpus-world rules + observations

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

Inference

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

Consequences of KB are a hay stack; α is a needle.

Entailment=needle in hay stack;

inference=finding it

Soundness: i is sound if

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

Completeness: i is complete if

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

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

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

Propositional logic: Syntax

Propositional logic is the simplest logic—illustrates basic ideas The proposition symbols P_1, P_2 etc are sentences

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

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

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

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

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

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g.	$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
	TRUE	TRUE	FALSE

(With these symbols, 8 possible models, can be enumerated automatically.) Rules for evaluating truth with respect to a model m:

$\neg S$ is true iff S is false

$S_1 \wedge S_2$ is true iff S_1 is true and S_2 is true

$S_1 \vee S_2$ is true iff S_1 is true or S_2 is true

$S_1 \Rightarrow S_2$ is true iff S_1 is false or S_2 is true

i.e., is false iff S_1 is true and S_2 is false

$S_1 \Leftrightarrow S_2$ is true iff $S_1 \Rightarrow S_2$ is true and $S_2 \Rightarrow S_1$ is true

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 $[i,j]$.

$\neg P_{1,1}$

$\neg B_{1,1}$

$B_{2,1}$

How do we encode "pits cause breezes in adjacent squares"?

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

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

In other words, "a square is breezy if and only if there is an adjacent pit"

Truth Tables for Inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
:	:	:	:	:	:	:	:	:	:	:	:	:
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
:	:	:	:	:	:	:	:	:	:	:	:	:
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

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

Inference by Enumeration

Depth-first enumeration of all models is sound and complete

```
function TT-Entails?(KB, α) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
          α, the query, a sentence in propositional logic
  symbols ← 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

Logical Equivalence

Two sentences are logically equivalent iff they are true in the same models:

$\alpha \equiv \beta$ iff $\alpha \models \beta$ 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., $True$, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

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 un satisfiable if it is true in no models e.g., $A \wedge \neg A$

Satisfiability is connected to inference via the following:

$KB \models \alpha$ iff $(KB \wedge \neg\alpha)$ is un satisfiable i.e., prove α by *reduction and absurdum*

Proof Methods

Proof methods divide into (roughly)two kinds:

Application of inference rules

- Legitimate(sound)generationofnewsentencesfromold
- Proof = a sequence of inference rule applications canuseinferencerules as operators in a standard search algorithm
- Typically require translation of sentences into a normalform

Model checking

- Truth table enumeration (always exponential in n)
- Improvedbacktracking,e.g.,Davis–Putnam–Loge Mann–Loveland
- Heuristic search inmodelspace(soundbutincomplete) e.g.,min-conflicts-likehill-climbing algorithms

Forward and Backward Chaining

Horn Form (restricted)

KB = conjunction of Horn clauses

Horn clause =

- proposition symbol; or
- (conjunctionofsymbols) ⇒ symbol Example KB: $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

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

$$\alpha_1, \ldots, \alpha_n, \alpha_1 \wedge \cdots \wedge \alpha \Rightarrow \beta \beta$$

Can be used with forwardchaining or backwardchaining. These algorithms are very natural and run in linear time.,

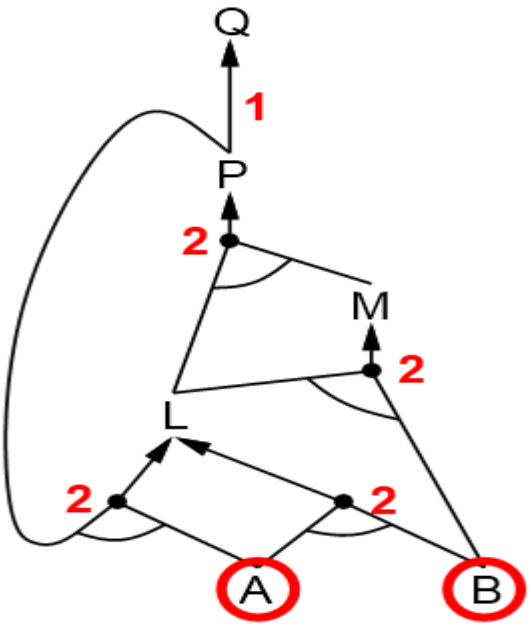
ForwardChaining

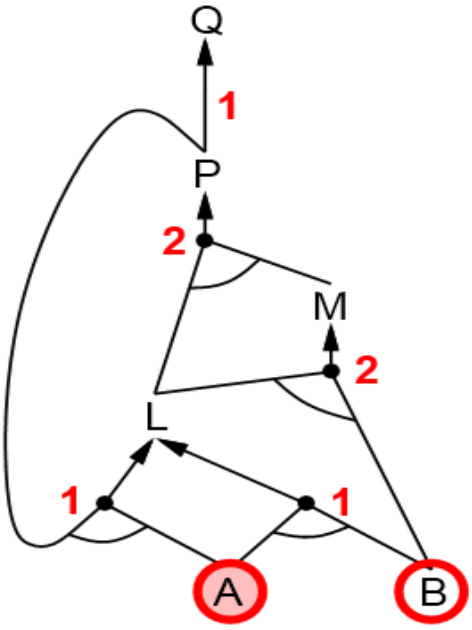
Idea: If any rule whose premisesaresatisfiedintheKB, additsconclusiontotheKB,untilqueryisfound

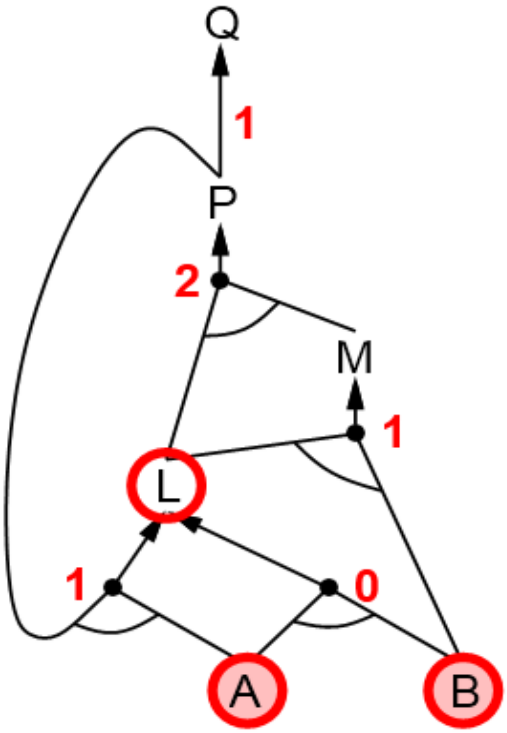
ForwardChaining Algorithm

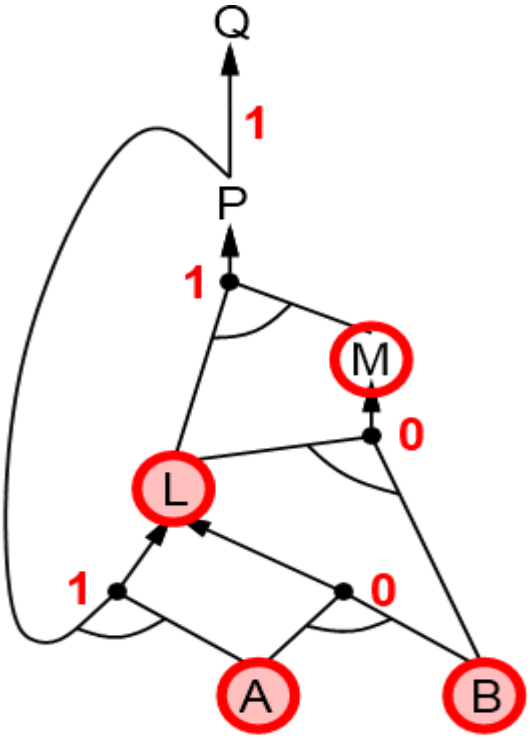
```
function PL-FC-Entails?(KB,q) returns true or false
  inputs: KB, the knowledge base, a set of propositional Horn clauses
          q, the query, a proposition symbol
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known in KB
  while agenda is not empty
    do p ← Pop(agenda)
    unless inferred[p] do
      inferred[p] ← true
      for each Horn clause c in whose premise p appears do
        decrement count[c]
```

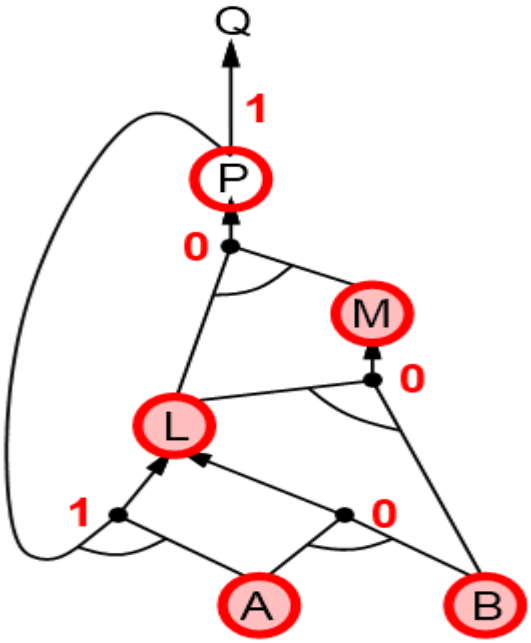
ForwardChaining Example

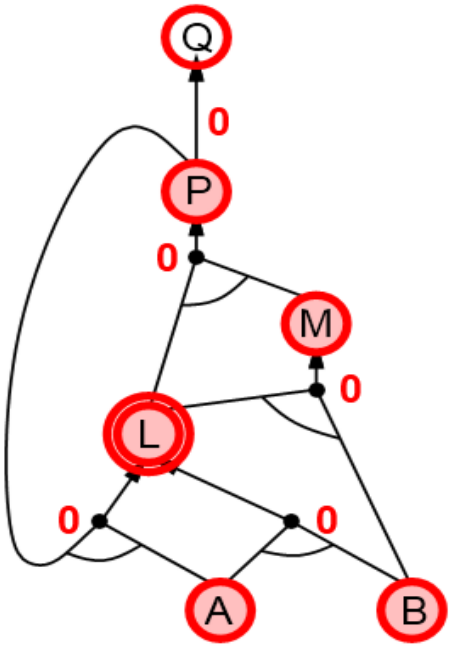

Q
1
P
2
M
2
L
2
2
A
B

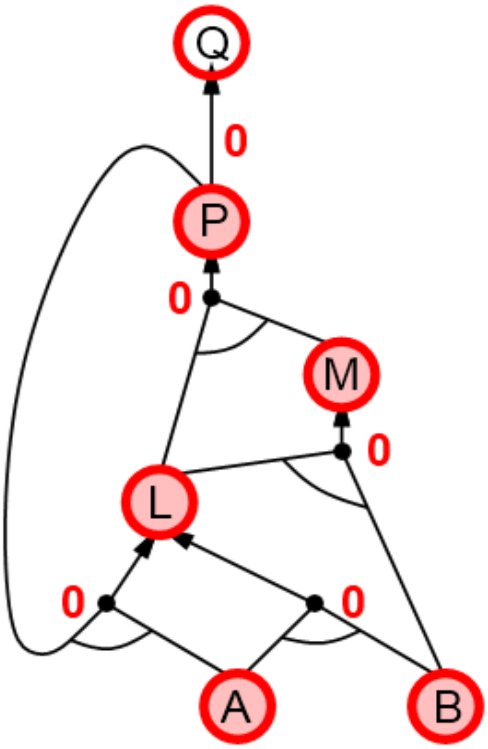

Q
1
P
2
M
2
L
1
1
A
B

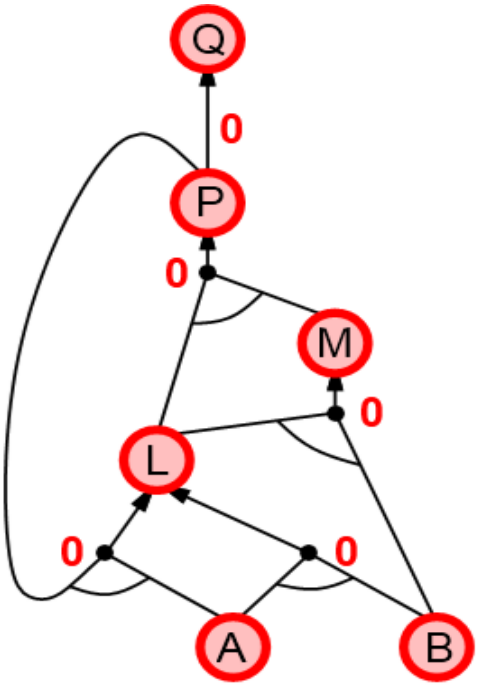

Q
1
P
2
M
1
L
1
0
A
B


Q
1
P
1
M
0
L
1
0
A
B


Q
1
P
0
M
0
L
1
0
A
B


Q
0
P
0
M
0
L
0
0
A
B


Q
0
P
0
M
0
L
0
0
A
B


Q
0
P
0
M
0
L
0
0
A
B

Proof of Completeness

FC derives every atomic sentence that is entailed by KB

1. FC reaches a fixed point where none new atomic sentences are derived
2. Consider the final state as a model m, assigning true/false to symbols
3. Every clause in the original KB is true in m
    i. Proof: Suppose a clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false in m Then $a_1 \wedge \ldots \wedge a_k$ is true in m and b is false in m Therefore the algorithm has not reached a fixed point!
4. Hence m is a model of KB
5. If $KB \models q$, then q is true in every model of KB, including m
    a. General idea: construct any model of KB by sound inference, check α

Backward Chaining

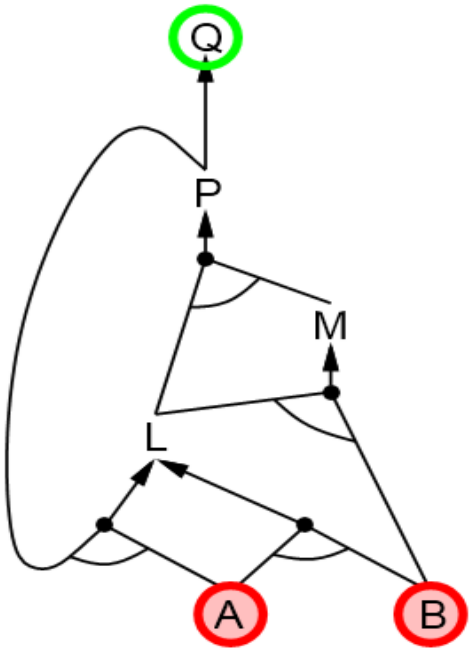
Idea: work backwards from the query q: to prove q by BC,

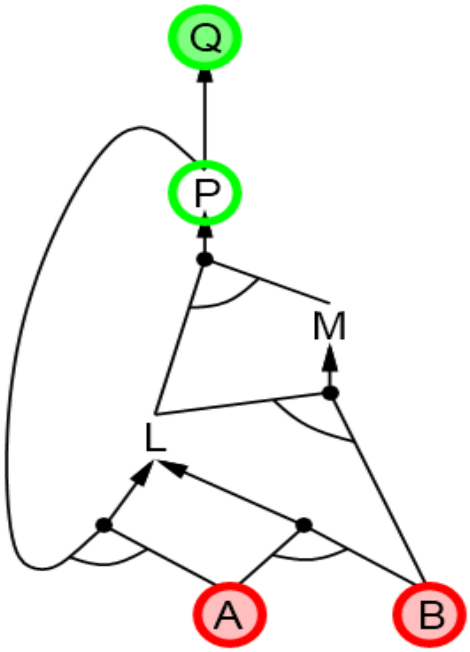
check if q is known already, or

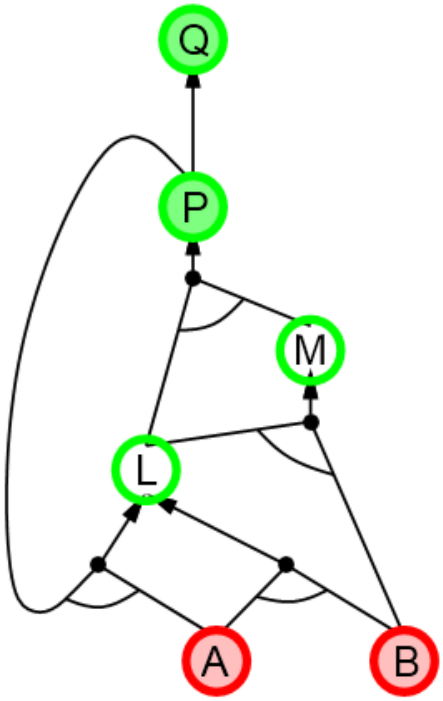
prove by BC all premises of some rule concluding q

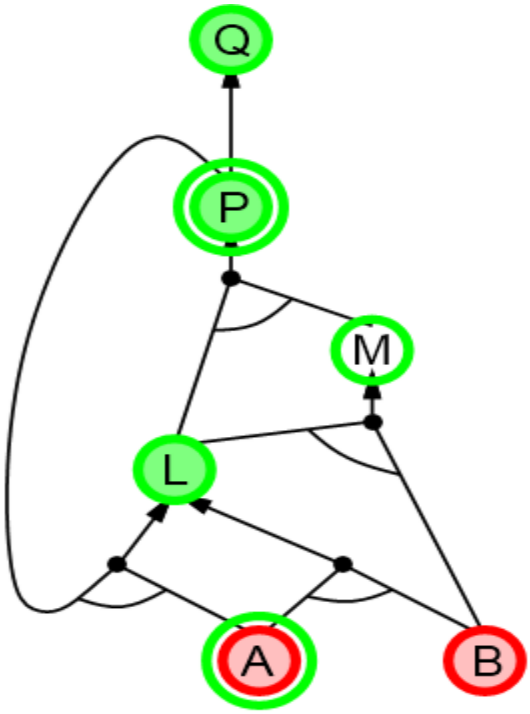
Avoid loops: check if new sub goal is already on the goal stack Avoid repeated work: check if new sub goal

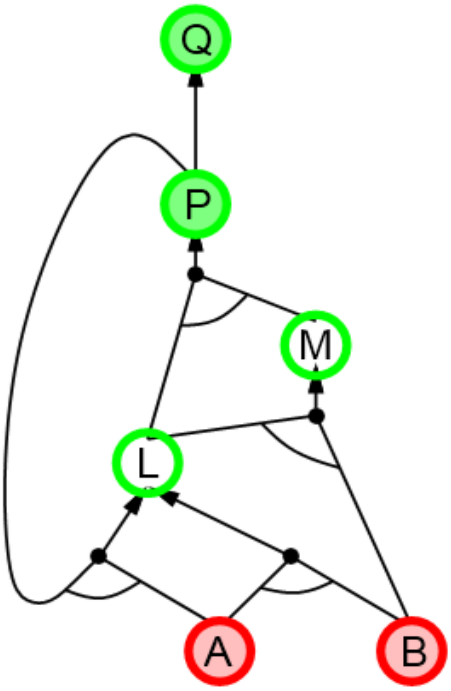
1. has already been proved true, or
2. has already failed

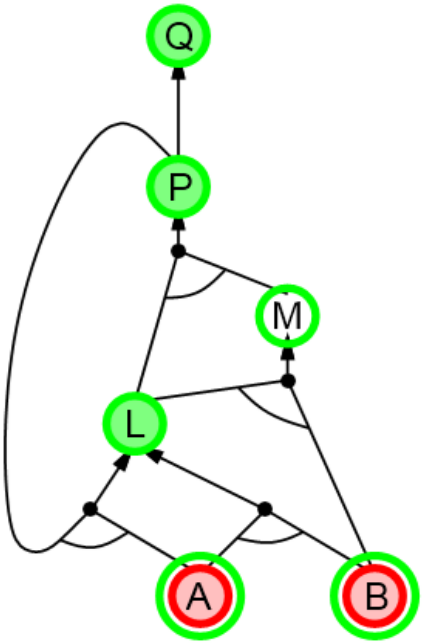

Q
P
M
L
A
B

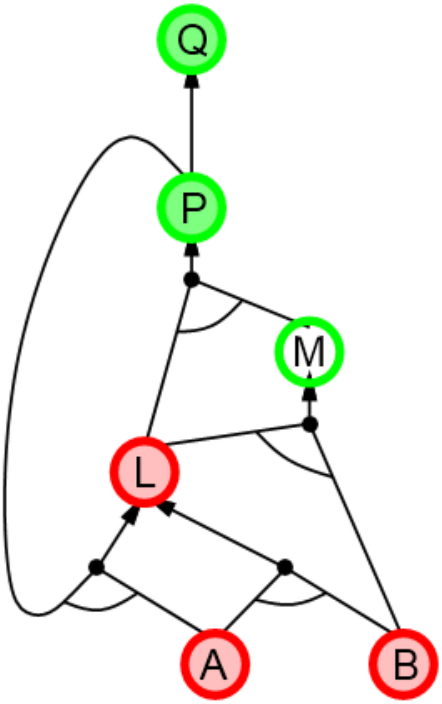

Q
P
M
L
A
B

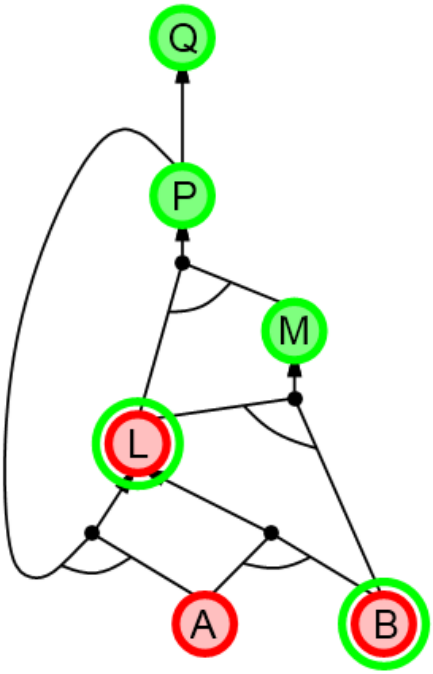

Q
P
M
L
A
B

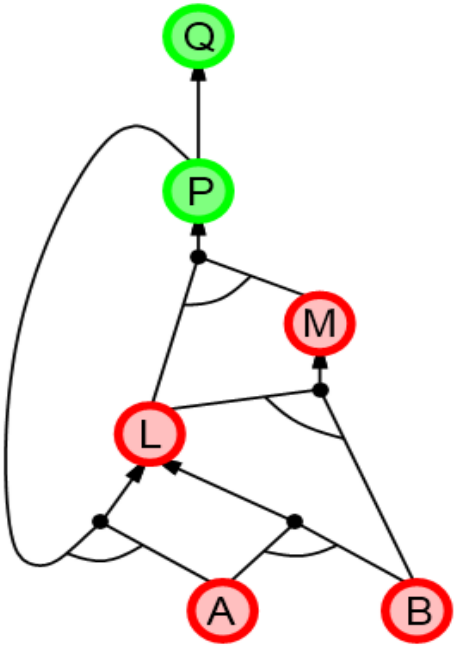

Q
P
M
L
A
B

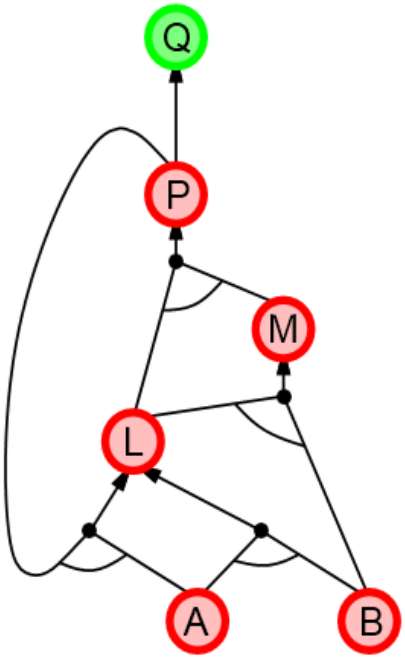

Q
P
M
L
A
B

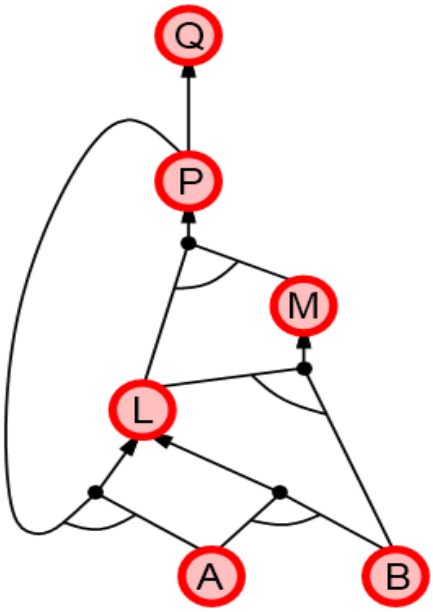

Q
P
M
L
A
B


Q
P
M
L
A
B


Q
P
M
L
A
B


Q
P
M
L
A
B


Q
P
M
L
A
B


Q
P
M
L
A
B

Forward vs Backward Chaining

FC is data-driven, cf. automatic, unconscious processing, e.g.,objectrecognition,routinedecisions

Maydolotsofworkthatisirrelevanttothegoal BC is goal-driven, appropriate forproblem-solving,

e.g., Where are my keys? How do I get into a PhD program?

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

Resolution

Conjunctive Normal Form (CNF—universal)

Conjunction of disjunctions of literals / Clauses

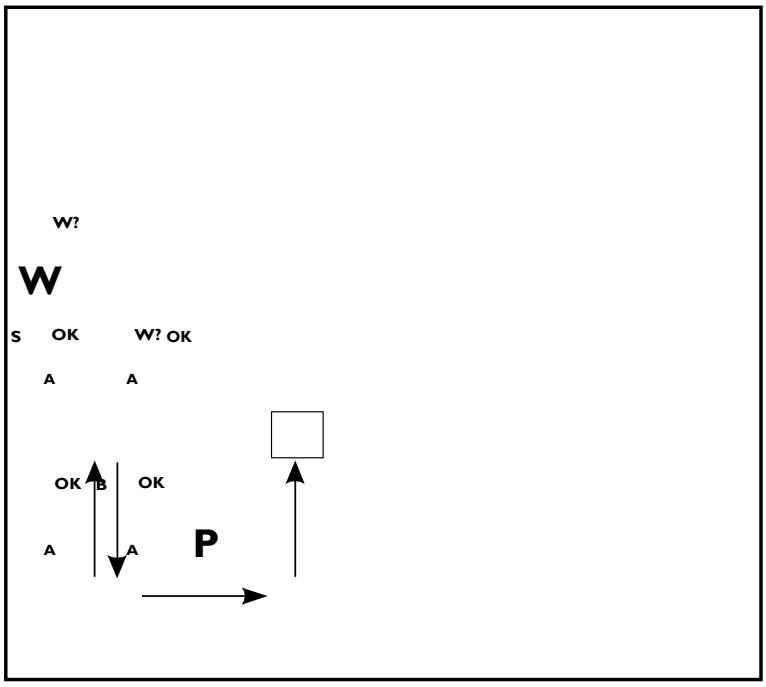
E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Resolution inference rule (CNF): complete for propositional logic

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

where $\ell_i \equiv \neg m_j$ are complementary literals. E.g.,

$$\frac{W_{1,3} \vee W_{2,2} \ , \ \neg W_{2,2}}{W_{1,3}}$$



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 $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
   i. $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.
   i. $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. Move $\neg$ inwards using de Morgan's rules and double-negation:
   i. $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. Apply distributivity law ($\vee$ over $\wedge$) and flatten:

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

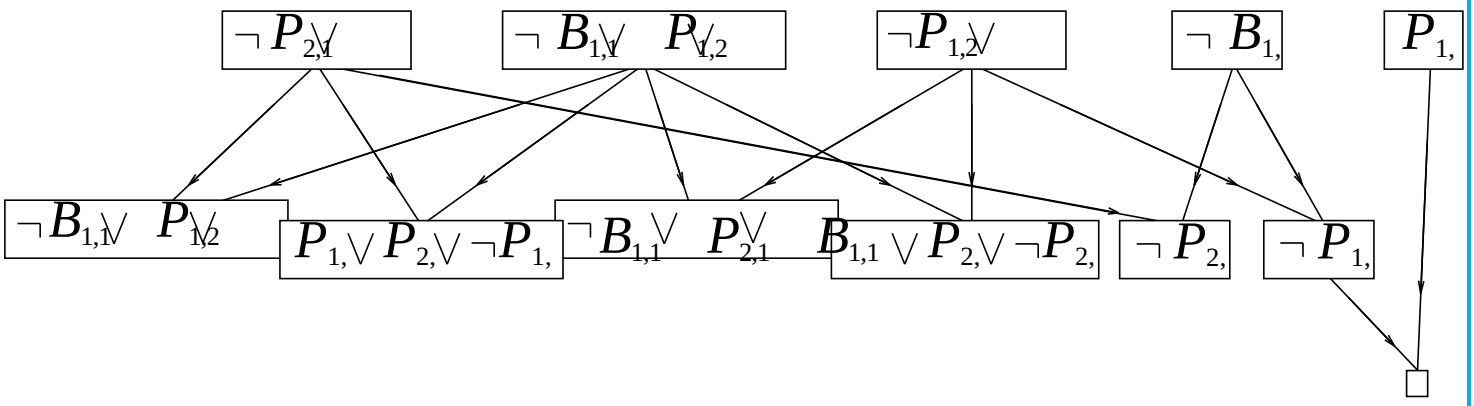
Resolution Algorithm

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

```
function PL-Resolution(KB,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

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



Summary

Logical agents apply inference to a knowledge base toderivenewinformationandmakedecisions

Basic concepts of logic:

- syntax: formal structure of sentences

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

- ✓ Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.
- ✓ Forward, backward chaining are linear-time, complete for Horn clauses Resolution is complete for propositional logic
- ✓ Propositional logic lacks expressive power

FIRST ORDER LOGIC

PROCEDURAL LANGUAGES AND PROPOSITIONAL LOGIC:

Drawbacks of Procedural Languages

- ✓ Programming languages (such as C++ or Java or Lisp) are by far the largest class of formal languages in common use. Programs themselves represent only computational processes. Data structures within programs can represent facts.

  For example, a program could use a 4 × 4 array to represent the contents of the wumpus world. Thus, the programming language statement World[2,2]← Pit is a fairly natural way to assert that there is a pit in square [2,2].

  What programming languages lack is any general mechanism for deriving facts from other facts; each update to a data structure is done by a domain-specific procedure whose details are derived by the programmer from his or her own knowledge of the domain.

- ✓ A second drawback of is the lack the expressiveness required to handle partial information . For example data structures in programs lack the easy way to say, “There is a pit in [2,2] or [3,1]” or “If the wumpus is in [1,1] then he is not in [2,2].”

Advantages of Propositional Logic

- ✓ The declarative nature of propositional logic, specify that knowledge and inference are separate, and inference is entirely domain-independent.
- ✓ Propositional logic is a declarative language because its semantics is based on a truth relation between sentences and possible worlds.
- ✓ It also has sufficient expressive power to deal with partial information, using disjunction and negation.
- ✓ Propositional logic has a third COMPOSITIONALITY property that is desirable in representation languages, namely, compositionality. In a compositional language, the meaning of a sentence is a function of the meaning of its parts. For example, the meaning of "S1,4 ∧ S1,2" is related to the meanings of "S1,4" and "S1,2.

Drawbacks of Propositional Logic

- ✓ Propositional logic lacks the expressive power to concisely describe an environment with many objects.

  For example, we were forced to write a separate rule about breezes and pits for each square, such as $B1,1 \Leftrightarrow (P1,2 \vee P2,1)$.

- ✓ In English, it seems easy enough to say, "Squares adjacent to pits are breezy."
- ✓ The syntax and semantics of English somehow make it possible to describe the environment concisely.

NATURAL LANGUAGES

Advantage of Natural Languages

Natural languages are very expressive. There is a long tradition in linguistics and the philosophy of language that views natural language as essentially a declarative knowledge representation language and attempts to pin down its formal semantics. Such a research program, if successful, would be of great value to artificial intelligence because it would allow a natural language (or some derivative) to be used within representation and reasoning systems.

Drawbacks of Natural Languages

- ✓ The modern view of natural language is as a medium for communication rather than pure representation.

  For Example: When a speaker points and says, "Look!" ,the meaning of the sentence depends both on the sentence itself and on the context in which the sentence was spoken. Clearly, one could not store a sentence such as "Look!" in a knowledge base and expect to recover its meaning without also storing a representation of the context-which raises the question of how the context itself can be represented.
- ✓ Natural languages are also non compositional-the meaning of a sentence such as "Then she saw it" can depend on a context constructed by many preceding and succeeding sentences.
- ✓ Natural languages also suffer from Ambiguity, a problem for a representation language.

  For example, “When people think about spring, surely they are not confused as to whether they are thinking about a season or something else”. If one word can correspond to two thoughts, thoughts can’t be words.”

Combining Propositional Logic with elements of Natural Language

The foundation of propositional logic (-a declarative, compositional semantics that is context-independent and unambiguous-) can be adopted and borrow representational ideas from natural language (while avoiding its drawbacks) to build a more expressive logic on that foundation.

The elements of Natural Language are

- ✓ Nouns and noun phrases that refer to objects

  Example: (squares, pits, wumpuses)
- ✓ Verbs and verb phrases that refer to relations among objects

  Example: (is breezy, is adjacent to, FUNCTIONS shoots).
- ✓ Some relations are functions-relations in which there is only one "value" for a given "input."

Any assertion can be thought of as referring to objects and properties or relations.

Some examples:

- ✓ "One plus two equals three"
  Objects: one, two, three,
  Relation: equals;
  Function: plus.

- ✓ "Squares neighboring the wumpus are smelly."
  Objects: wumpus, squares;
  Property: smelly;
  Relation: neighboring.

Difference between Propositional and First Order Logic

The primary difference between propositional and first-order logic lies in the ontological commitment made by each language-that is, what it assumes about the nature of reality.

For example, propositional logic assumes that there are facts that either hold or do not hold in the world. Each fact can be in one of two states: true or false. First-order logic assumes more; namely, that the world consists of objects with certain relations among them that do or do not hold.

A logic can also be characterized by its epistemological commitments-the possible states of knowledge that it allows with respect to each fact. In both propositional and first order logic, a sentence represents a fact and the agent either believes the sentence to be true, believes it to be false, or has no opinion.

Language	**Ontological Commitment** (***What exists in the world***)	**Epistemological Commitment** (***What an agent believes about facts***)
Propositional logic	facts	True / false / unknown
First-order logic	facts, objects, relations	True / false / unknown
Temporal Logic	facts, objects, relations	True / false / unknown
Predicate Logic	Facts	Degree of belief E [O,l]
Fuzzy Logic	facts with degree of truth E [O,1]	Known Interval Value

SYNTAX AND SEMANTICS OF FIRST-ORDER LOGIC

Models for first-order logic

The models of a logical language are the formal structures that constitute the possible worlds under consideration. Each model links the vocabulary of the logical sentences to elements of the possible world, so that the truth of any sentence can be determined. Thus, models for propositional logic link proposition symbols to predefined truth values.

Models for first-order logic have objects. The **domain** of a model is the set of objects or **domain elements** it contains. The domain is required to be *nonempty*—every possible world must contain at least one object.

A relation is just the set of **tuples** of objects that are related.

- ✓ Unary Relation: Relations relates to single Object
- ✓ Binary Relation: Relation Relates to multiple objects

Certain kinds of relationships are best considered as functions, in that a given object must be related to exactly one object.

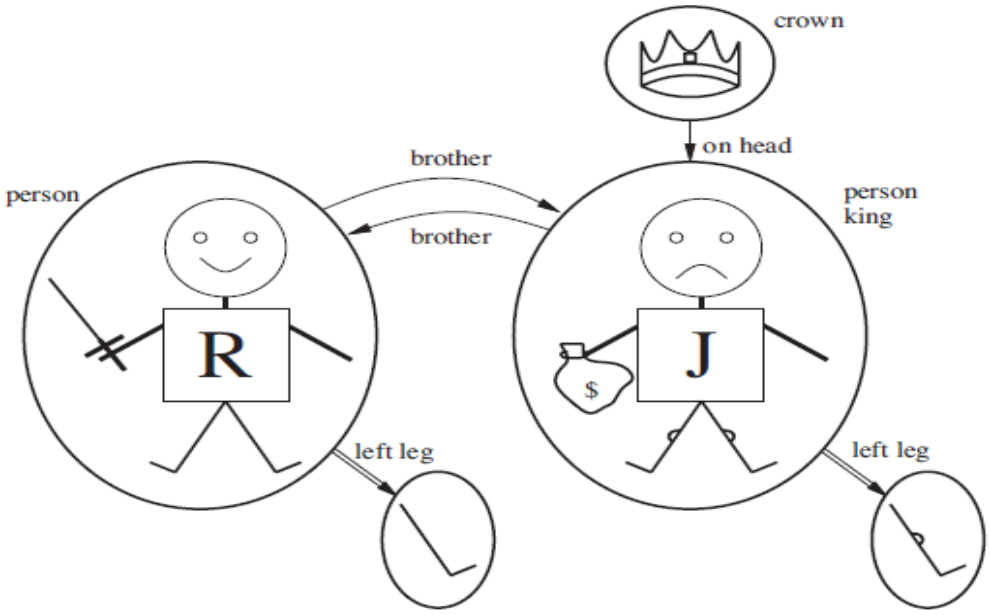
For Example:

Richard the Lionheart, King of England from 1189 to 1199;

His younger brother, the evil King John, who ruled from 1199 to 1215;

the left legs of Richard and John;

crown



Unary Relation : John is a king

Binary Relation :crown is ***on head*** of john , Richard is ***brother of*** john

The unary "left leg" function includes the following mappings:

(Richard the Lionheart) ➔ Richard's left leg

(King John) ➔ John's left leg .

Symbols and interpretations

Symbols are the basic syntactic elements of first-order logic.

Symbols stand for objects, relations, and functions.

The symbols are of three kinds:

- ✓ Constant symbols which stand for objects;
  Example: John, Richard
- ✓ Predicate symbols, which stand for relations;
  Example: OnHead, Person, King, and Crown
- ✓ Function symbols, which stand for functions.
  Example: left leg

Symbols will begin with uppercase letters.

Interpretation

The semantics must relate sentences to models in order to determine truth. For this to happen, we need an interpretation that specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.

For Example:

- ✓ Richard refers to Richard the Lionheart and John refers to the evil king John.
- ✓ Brother refers to the brotherhood relation
- ✓ OnHead refers to the "on head relation that holds between the crown and King John;
- ✓ Person, King, and Crown refer to the sets of objects that are persons, kings, and crowns.
- ✓ LeftLeg refers to the "left leg" function,

The truth of any sentence is determined by a model and an interpretation for the sentence's symbols. Therefore, entailment, validity, and so on are defined in terms of all possible models and all possible interpretations. The number of domain elements in each model may be unbounded-for example, the domain elements may be integers or real numbers. Hence, the number of possible models is anbounded, as is the number of interpretations.

Term

A **term** is a logical expression that refers to an object.

Constant symbols are therefore terms.

Complex Terms

A complex term is just a complicated kind of name.

A complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol

For example: "King John's left leg"
Instead of using a constant symbol, we use *LeftLeg(John).*

The formal semantics of terms :

Consider a term f *(tl,. . . , t,).* The function symbol *f* refers to some function in the model (F); the argument terms refer to objects in the domain (call them *d1 ....dn);* and the term as a whole refers to the object that is the value of the function *F* applied to *dl, . . . , d,.*

For example,: the *LeftLeg* function symbol refers to the function " (King John) -+ John's left leg" and *John* refers to King John, then *LeftLeg(John)* refers to King John's left leg. In this way, the interpretation fixes the referent of every term.

Atomic sentences

An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms:
For Example: Brother(Richard, John).

Atomic sentences can have complex terms as arguments.
For Example: Married (Father(Richard), Mother(John)).

An atomic sentence is true in a given model, under a given interpretation, if the relation referred to by the predicate symbol holds among the objects referred to by the arguments

Complex sentences

Complex sentences can be constructed using logical Connectives, just as in propositional calculus.
For Example:

- ✓ ¬Brother (LeftLeg(Richard), John)
- ✓ Brother (Richard , John) ∧ Brother (John, Richard)
- ✓ King(Richard) ∨ King(John)
- ✓ ¬King(Richard) ⇒ King(John)

Quantifiers

Quantifiers express properties of entire collections of objects, instead of enumerating the objects by name.

First-order logic contains two standard quantifiers:

1. Universal Quantifier
2. Existential Quantifier

Universal Quantifier

Universal quantifier is defined as follows:

"Given a sentence ∀x P, where P is any logical expression, says that P is true for every object x."

More precisely, ∀x P is true in a given model if P is true in all possible **extended interpretations** constructed from the interpretation given in the model, where each extended interpretation specifies a domain element to which x refers.

For Example: "All kings are persons," is written in first-order logic as

∀ x King(x) ⇒ Person(x) .

∀ is usually pronounced "For all"

Thus, the sentence says, "For all x, if x is a king, then x is a person."

The symbol x is called a **variable**. Variables are lowercase letters.

A variable is a term all by itself, and can also serve as the argument of a function

A term with no variables is called a **ground term**.

Assume we can extend the interpretation in different ways:

x → Richard the Lionheart,

x → King John,

x → Richard's left leg,

x → John's left leg,

x → the crown

The universally quantified sentence ∀ x King(x) ⇒ Person(x) is true in the original model if the sentence King(x) ⇒ Person(x) is true under each of the five extended interpretations.

That is, the universally quantified sentence is equivalent to asserting the following five sentences:

Richard the Lionheart is a king ⇒ Richard the Lionheart is a person.
King John is a king ⇒ King John is a person.
Richard's left leg is a king ⇒ Richard's left leg is a person.
John's left leg is a king ⇒ John's left leg is a person.
The crown is a king ⇒ the crown is a person.

Existential quantification (∃)

Universal quantification makes statements about every object. Similarly, we can make a statement about *some* object in the universe without naming it, by using an existential quantifier.

"The sentence ∃x P says that P is true for at least one object x. More precisely, ∃x P is true in a given model if P is true in *at least one* extended interpretation that assigns x to a domain element."
∃x is pronounced "There exists an x such that . . ." or "For some x . . .".

For example, that King John has a crown on his head, we write
∃ x Crown(x) ∧ OnHead(x, John) .

Given assertions:

Richard the Lionheart is a crown ∧ Richard the Lionheart is on John's head;
King John is a crown ∧ King John is on John's head;
Richard's left leg is a crown ∧ Richard's left leg is on John's head;
John's left leg is a crown ∧ John's left leg is on John's head;
The crown is a crown ∧ the crown is on John's head.
The fifth assertion is true in the model, so the original existentially quantified sentence is true in the model.
Just as ⇒ appears to be the natural connective to use with ∀, ∧ is the natural connective to use with ∃.

Nested quantifiers

One can express more complex sentences using multiple quantifiers.

For example, “Brothers are siblings” can be written as

∀ x ∀ y Brother (x, y) ⇒ Sibling(x, y) .

Consecutive quantifiers of the same type can be written as one quantifier with several variables.

For example, to say that siblinghood is a **symmetric relationship**,

we can write **∀ x, y Sibling(x, y) ⇔ Sibling(y, x)** .

In other cases we will have mixtures.

For example:

1. “Everybody loves somebody” means that for every person, there is someone that person loves:

   ∀ x ∃ y Loves(x, y) .
2. On the other hand, to say “There is someone who is loved by everyone,” we write

   ∃ y ∀ x Loves(x, y) .

Connections between ∀ and ∃

Universal and Existential quantifiers are actually intimately connected with each other, through negation.

Example assertions:

1. “ Everyone dislikes medicine” is the same as asserting “ there does not exist someone who likes medicine” , and vice versa:

   “**∀ x ¬Likes(x, medicine)**” is equivalent to **“¬∃ x Likes(x, medicine)”**.
2. “Everyone likes ice cream” means that “ there is no one who does not like ice cream” :

   ∀ x Likes(x, IceCream) is equivalent to ¬∃ x ¬Likes(x, IceCream) .

Because ∀ is really a conjunction over the universe of objects and ∃ is a disjunction that they obey De Morgan's rules. The De Morgan rules for quantified and unquantified sentences are as follows:

$$\forall x \ \neg P \equiv \neg \exists x \ P \qquad \neg(P \vee Q) \equiv \neg P \wedge \neg Q$$
$$\neg \forall x \ P \equiv \exists x \ \neg P \qquad \neg(P \wedge Q) \equiv \neg P \vee \neg Q$$
$$\forall x \ P \equiv \neg \exists x \ \neg P \qquad P \wedge Q \equiv \neg(\neg P \vee \neg Q)$$
$$\exists x \ P \equiv \neg \forall x \ \neg P \qquad P \vee Q \equiv \neg(\neg P \wedge \neg Q) .$$

Thus, Quantifiers are important in terms of readability.

Equality

First-order logic includes one more way to make atomic sentences, other than using a predicate and terms .We can use the **equality symbol** to signify that two terms refer to the same object.

For example,

"Father (John) =Henry" says that the object referred to by Father (John) and the object referred to by Henry are the same.

Because an interpretation fixes the referent of any term, determining the truth of an equality sentence is simply a matter of seeing that the referents of the two terms are the same object. The equality symbol can be used to state facts about a given function. It can also be used with negation to insist that two terms are not the same object.

For example,

"Richard has at least two brothers" can be written as,

∃ x, y Brother (x,Richard) ∧ Brother (y,Richard) ∧¬(x=y) .

The sentence

∃ x, y Brother (x,Richard) ∧ Brother (y,Richard) does not have the intended meaning.
In particular, it is true only in the model where Richard has only one brother considering the extended interpretation in which both x and y are assigned to King John. The addition of ¬(x=y) rules out such models.

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{Predicate} \mid \textit{Predicate}(\textit{Term}, \ldots) \mid \textit{Term} = \textit{Term} \\
\textit{ComplexSentence} &\rightarrow (\ \textit{Sentence}\ ) \mid [\ \textit{Sentence}\ ] \\
&\mid \neg\ \textit{Sentence} \\
&\mid \textit{Sentence} \wedge \textit{Sentence} \\
&\mid \textit{Sentence} \vee \textit{Sentence} \\
&\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
&\mid \textit{Quantifier}\ \textit{Variable}, \ldots\ \textit{Sentence} \\
\textit{Term} &\rightarrow \textit{Function}(\textit{Term}, \ldots) \\
&\mid \textit{Constant} \\
&\mid \textit{Variable} \\
\textit{Quantifier} &\rightarrow \forall \mid \exists \\
\textit{Constant} &\rightarrow A \mid X_1 \mid \textit{John} \mid \cdots \\
\textit{Variable} &\rightarrow a \mid x \mid s \mid \cdots \\
\textit{Predicate} &\rightarrow \textit{True} \mid \textit{False} \mid \textit{After} \mid \textit{Loves} \mid \textit{Raining} \mid \cdots \\
\textit{Function} &\rightarrow \textit{Mother} \mid \textit{LeftLeg} \mid \cdots \\
\text{OPERATOR PRECEDENCE} &: \neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{aligned}
$$

Backus Naur Form for First Order Logic

USING FIRST ORDER LOGIC

Assertions and queries in first-order logic

Assertions:

Sentences are added to a knowledge base using TELL, exactly as in propositional logic. Such sentences are called **assertions**.

For example,

John is a king, TELL (KB, King (John)).
Richard is a person. TELL (KB, Person (Richard)).
All kings are persons: TELL (KB, ∀ x King(x) ⇒ Person(x)).

Asking Queries:

We can ask questions of the knowledge base using ASK. Questions asked with ASK are called **queries** or **goals**.

For example,

ASK (KB, King (John)) returns true.

Any query that is logically entailed by the knowledge base should be answered affirmatively.

For example, given the two preceding assertions, the query:

"ASK (KB, Person (John))" should also return true.

Substitution or **binding list**

We can ask quantified queries, such as ASK (KB, ∃ x Person(x)) .

The answer is true, but this is perhaps not as helpful as we would like. It is rather like answering "Can you tell me the time?" with "Yes."

If we want to know what value of x makes the sentence true, we will need a different function, ASKVARS, which we call with ASKVARS (KB, Person(x)) and which yields a stream of answers.

In this case there will be two answers: {x/John} and {x/Richard}. Such an answer is called a substitution or binding list.

ASKVARS is usually reserved for knowledge bases consisting solely of Horn clauses, because in such knowledge bases every way of making the query true will bind the variables to specific values.

The kinship domain

The objects in Kinship domain are people.

We have two unary predicates, Male and Female.

Kinship relations—parenthood, brotherhood, marriage, and so on—are represented by binary predicates: Parent, Sibling, Brother, Sister, Child, Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, and Uncle.

We use functions for Mother and Father, because every person has exactly one of each of these.

We can represent each function and predicate, writing down what we know in terms of the other symbols.

For example:-

1. one's mother is one's female parent:

   ∀ m, c Mother (c)=m ⇔ Female(m) ∧ Parent(m, c) .

2. One's husband is one's male spouse:

   ∀ w, h Husband(h,w) ⇔ Male(h) ∧ Spouse(h,w) .

3. Male and female are disjoint categories:

   ∀ x Male(x) ⇔ ¬Female(x) .

4. Parent and child are inverse relations:

   ∀ p, c Parent(p, c) ⇔ Child (c, p) .

5. A grandparent is a parent of one's parent:

   ∀ g, c Grandparent (g, c) ⇔ ∃p Parent(g, p) ∧ Parent(p, c) .

6. A sibling is another child of one's parents:

   ∀ x, y Sibling(x, y) ⇔ x _= y ∧ ∃p Parent(p, x) ∧ Parent(p, y) .

Axioms:

Each of these sentences can be viewed as an **axiom** of the kinship domain. **Axioms** are commonly associated with purely mathematical domains. They provide the basic factual information from which useful conclusions can be derived.

Kinship axioms are also **definitions**; they have the form ∀ x, y P(x, y) ⇔ . . ..

The axioms define the Mother function, Husband, Male, Parent, Grandparent, and Sibling predicates in terms of other predicates.

Our definitions "bottom out" at a basic set of predicates (*Child, Spouse, and Female*) in terms of which the others are ultimately defined. This is a natural way in which to build up the representation of a domain, and it is analogous to the way in which software packages are built up by successive definitions of subroutines from primitive library functions.

Theorems:

Not all logical sentences about a domain are axioms.
Some are **theorems**—that is, they are entailed by the axioms.

For example, consider the assertion that siblinghood is symmetric:
∀ x, y Sibling(x, y) ⇔ Sibling(y, x) .

It is a theorem that follows logically from the axiom that defines siblinghood.
If we ASK the knowledge base this sentence, it should return true.
From a purely logical point of view, a knowledge base need contain only axioms and no theorems, because the theorems do not increase the set of conclusions that follow from the knowledge base.
From a practical point of view, theorems are essential to reduce the computational cost of deriving new sentences. Without them, a reasoning system has to start from first principles every time.

Axioms without Definition

Not all axioms are definitions. Some provide more general information about certain predicates without constituting a definition. Indeed, some predicates have no complete definition because we do not know enough to characterize them fully.

For example, there is no obvious definitive way to complete the sentence

∀ x Person(x) ⇔ . . .

Fortunately, first-order logic allows us to make use of the Person predicate without completely defining it. Instead, we can write partial specifications of properties that every person has and properties that make something a person:

∀ x Person(x) ⇒ . . .
∀x . . . ⇒ Person(x) .

Axioms can also be "just plain facts," such as Male (Jim) and Spouse (Jim, Laura). Such facts form the descriptions of specific problem instances, enabling specific questions to be answered. The answers to these questions will then be theorems that follow from the axioms.

Numbers, sets, and lists

Number theory

Numbers are perhaps the most vivid example of how a large theory can be built up from NATURAL NUMBERS a tiny kernel of axioms. We describe here the theory of natural numbers or non-negative integers. We need:

- ✓ predicate NatNum that will be true of natural numbers;
- ✓ one PEANO AXIOMS constant symbol, 0;
- ✓ One function symbol, S (successor).
- ✓ The Peano axioms define natural numbers and addition.

Natural numbers are defined recursively:

NatNum(0) .

∀ n NatNum(n) ⇒ NatNum(S(n)) .

That is, 0 is a natural number, and for every object n, if n is a natural number, then S(n) is a natural number.

So the natural numbers are 0, S(0), S(S(0)), and so on.

We also need axioms to constrain the successor function:

∀ n 0 != S(n) .

∀ m, n m != n ⇒ S(m) != S(n) .

Now we can define addition in terms of the successor function:

∀ m NatNum(m) ⇒ + (0, m) = m .

∀ m, n NatNum(m) ∧ NatNum(n) ⇒ + (S(m), n) = S(+(m, n))

The first of these axioms says that adding 0 to any natural number m gives m itself. Addition is represented using the binary function symbol "+" in the term + (m, 0);

To make our sentences about numbers easier to read, we allow the use of infix notation.
We can also write S(n) as n + 1, so the second axiom becomes :

∀ m, n NatNum (m) ∧ NatNum(n) ⇒ (m + 1) + n = (m + n)+1 .

This axiom reduces addition to repeated application of the successor function. Once we have addition, it is straightforward to define multiplication as repeated addition, exponentiation as repeated multiplication, integer division and remainders, prime numbers, and so on. Thus, the whole of number theory (including cryptography) can be built up from one constant, one function, one predicate and four axioms.

Sets

The domain of sets is also fundamental to mathematics as well as to commonsense reasoning. Sets can be represented as individual sets, including empty sets.

Sets can be built up by:

- ✓ adding an element to a set or
- ✓ Taking the union or intersection of two sets.

Operations that can be performed on sets are:

- ✓ To know whether an element is a member of a set

- ✓ Distinguish sets from objects that are not sets.

Vocabulary of set theory:

The empty set is a constant written as { }.
There is one unary predicate, Set, which is true of sets.
The binary predicates are

- ✓ x∈ s (x is a member of set s)
- ✓ s1 ⊆ s2 (set s1 is a subset, not necessarily proper, of set s2).

The binary functions are

- ✓ s1 ∩ s2 (the intersection of two sets),
- ✓ s1 ∪ s2 (the union of two sets), and
- ✓ {x|s} (the set resulting from adjoining element x to set s).

One possible set of axioms is as follows:

- ✓ The only sets are the empty set and those made by adjoining something to a set:
  ∀ s Set(s) ⇔ (s={}) ∨ (∃ x, s2 Set(s2) ∧ s={x|s2}) .
- ✓ The empty set has no elements adjoined into it. In other words, there is no way to decompose {} into a smaller set and an element:
  ¬∃ x, s {x|s}={} .
- ✓ Adjoining an element already in the set has no effect:
  ∀ x, s x∈ s ⇔ s={x|s} .
- ✓ The only members of a set are the elements that were adjoined into it. We express this recursively, saying that x is a member of s if and only if s is equal to some set s2 adjoined with some element y, where either y is the same as x or x is a member of s2:
  ∀ x, s x∈ s ⇔ ∃y, s2 (s={y|s2} ∧ (x=y ∨ x∈ s2))

- ✓ A set is a subset of another set if and only if all of the first set's members are members of the second set:
  $\forall$ s1, s2 s1 $\subseteq$ s2 $\Leftrightarrow$ ($\forall$x x$\in$ s1 $\Rightarrow$ x$\in$ s2)
- ✓ Two sets are equal if and only if each is a subset of the other:
  $\forall$ s1, s2 (s1 =s2) $\Leftrightarrow$ (s1 $\subseteq$ s2 $\wedge$ s2 $\subseteq$ s1)
- ✓ An object is in the intersection of two sets if and only if it is a member of both sets:
  $\forall$ x, s1, s2 x$\in$ (s1 $\cap$ s2) $\Leftrightarrow$ (x$\in$ s1 $\wedge$ x$\in$s2)
- ✓ An object is in the union of two sets if and only if it is a member of either set:
  $\forall$ x, s1, s2 x$\in$ (s1 $\cup$ s2) $\Leftrightarrow$ (x$\in$ s1 $\vee$ x$\in$s2)

Lists :

are similar to sets. The differences are that lists are ordered and the same element can appear more than once in a list. We can use the vocabulary of Lisp for lists:

- ✓ Nil is the constant list with no elements;
- ✓ Cons, Append, First, and Rest are functions;
- ✓ Find is the predicate that does for lists what Member does for sets.
- ✓ List? is a predicate that is true only of lists.
- ✓ The empty list is [ ].
- ✓ The term Cons(x, y), where y is a nonempty list, is written [x|y].
- ✓ The term Cons(x, Nil) (i.e., the list containing the element x) is written as [x].
- ✓ A list of several elements, such as [A,B,C], corresponds to the nested term
- ✓ Cons(A, Cons(B, Cons(C, Nil))).

The wumpus world

Agents Percepts and Actions

The wumpus agent receives a percept vector with five elements. The corresponding first-order sentence stored in the knowledge base must include both the percept and the time at which it

occurred; otherwise, the agent will get confused about when it saw what.We use integers for time steps. A typical percept sentence would be

Percept ([***Stench, Breeze, Glitter, None, None***], 5).

Here, Percept is a binary predicate, and Stench and so on are constants placed in a list.
The actions in the wumpus world can be represented by logical terms:

Turn (Right), Turn (Left), Forward, Shoot, Grab, Climb.

To determine which is best, the agent program executes the query:

ASKVARS (∃ a BestAction (a, 5)), which returns a binding list such as {a/Grab}.

The agent program can then return Grab as the action to take.

The raw percept data implies certain facts about the current state.

For example:
∀ t, s, g, m, c Percept ([***s, Breeze, g,m, c***], t) ⇒ Breeze(t) ,
∀ t, s, b, m, c Percept ([***s, b, Glitter,m, c***], t) ⇒ Glitter (t) ,

These rules exhibit a trivial form of the reasoning process called **perception.**

Simple "reflex" behavior can also be implemented by quantified implication sentences.

For example, we have
∀ t Glitter (t) ⇒ BestAction(Grab, t) .

Given the percept and rules from the preceding paragraphs, this would yield the desired conclusion Best Action (Grab, 5)—that is, Grab is the right thing to do.

Environment Representation

Objects are squares, pits, and the wumpus.
Each square could be named—Square1,2 and so on—but then the fact that Square1,2 and Square1,3 are adjacent would have to be an "extra" fact, and this needs one such fact for each pair of squares. It is better to use a complex term in which the row and column appear as integers;

For example, we can simply use the list term [1, 2].

Adjacency of any two squares can be defined as:

$\forall$ x, y, a, b Adjacent ([x, y], [a, b]) $\Leftrightarrow$
$(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee (y = b \wedge (x = a - 1 \vee x = a + 1))$.

Each pit need not be distinguished with each other. The unary predicate Pit is true of squares containing pits.

Since there is exactly one wumpus, a constant Wumpus is just as good as a unary predicate.
The agent's location changes over time, so we write At (Agent, s, t) to mean that the agent is at square s at time t.

To specify the Wumpus location (for example) at [2, 2] we can write
$\forall$t At (Wumpus, [2, 2], t).

Objects can only be at one location at a time:
$\forall$ x, s1, s2, t At(x, s1, t) $\wedge$ At(x, s2, t) $\Rightarrow$ s1 = s2 .

Given its current location, the agent can infer properties of the square from properties of its current percept.

For example, if the agent is at a square and perceives a breeze, then that square is breezy:

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

It is useful to know that a *square* is breezy because we know that the pits cannot move about.
Breezy has no time argument.

Having discovered which places are breezy (or smelly) and, very importantly, *not* breezy (or *not* smelly), the agent can deduce where the pits =e (and where the wumpus is).

There are two kinds of synchronic rules that could allow such deductions:

Diagnostic rules:

Diagnostic rules lead from observed effects to hidden causes.
For finding pits, the obvious diagnostic rules say that if a square is breezy, some adjacent square must contain a pit, or

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

and that if a square is not breezy, no adjacent square contains a pit:
$\forall$ s $\neg$*Breezy (s)* $\Rightarrow \neg \exists$r *Adjacent (r, s)* $\wedge$ *Pit* (,r)

Combining these two, we obtain the biconditional sentence
$\forall$ s *Breezy (s)* $\Leftrightarrow \exists$r *Adjacent(r, s)* $\wedge$ *Pit (r)* .

Causal rules:

Causal rules reflect the assumed direction of causality in the world: some hidden property of the world causes certain percepts to be generated.
For example, a pit causes all adjacent squares to be breezy:

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

and if all squares adjacent to a given square are pitless, the square will not be breezy:

$\forall$ s [$\forall$r *Adjacent (r, s)* $\Rightarrow\neg$*Pit (r)]* $\Rightarrow\neg$*Breezy (s)* .

It is possible to show that these two sentences together are logically equivalent to the biconditional sentence " $\forall$ s *Breezy (s)* $\Leftrightarrow$ $\exists$r *Adjacent(r, s)* $\wedge$ *Pit (r)"* .

The biconditional itself can also be thought of as causal, because it states how the truth value of *Breezy* is generated from the world state.

Systems that reason with causal rules are called **model-based reasoning systems**, because the causal rules form a model of how the environment operates.

Whichever kind of representation the agent uses, if the axioms correctly and completely describe the way the world works and the way that percepts are produced, then any complete logical inference procedure will infer the strongest possible description of the world state, given the available percepts. Thus, the agent designer can concentrate on getting the knowledge right, without worrying too much about the processes of deduction.