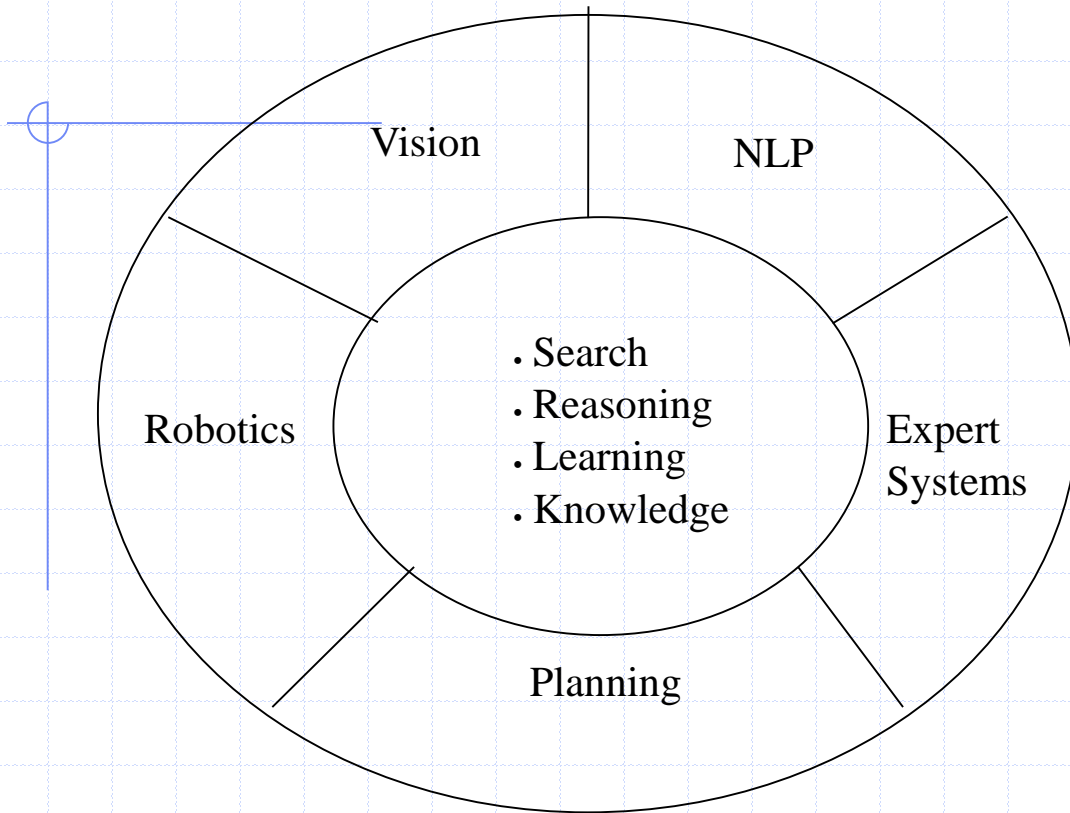


CS 561/571: Logical Reasoning

Asif Ekbal
IIT Patna

Logic and inferencing



Obtaining implication of given facts and rules -- **Hallmark of intelligence**

Knowledge and reasoning: enable successful behaviors, hard to achieve

Introduction: KR

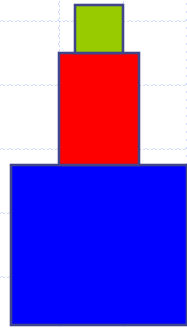
- ◆ Knowledge Representation means:
 - Capturing human knowledge
 - In a form computer can reason about
- ◆ Why?
 - Model human cognition
 - Add power to search-based methods
- ◆ Actually a component of all software development

Knowledge Representation (KR)

Given the world

- Express the general facts or beliefs using a language
- Determine what else we should (not) believe

Example



◆ Given:

- "The red block is above the blue block"
- "The green block is above the red block"

◆ Infer:

- "The green block is above the blue block"
- "The blocks form a tower"

Characteristics of a good KR:

◆ It should

- Be able to represent the knowledge important to the problem

- Reflect the structure of knowledge in the domain

 - Otherwise our development is a constant process of distorting things to make them fit.

- Capture knowledge at the appropriate level of granularity

- Support incremental, iterative development

◆ It should *not*

- Be too difficult to reason about

- Require that more knowledge be represented than is needed to solve the problem

A KR language needs to be

- ◆ expressive (*should be able to describe common sense knowledge*)
- ◆ unambiguous (*should be able to express the meaning of any instance uniquely*)
- ◆ flexible (*easy to represent different facts and beliefs*)

The inference procedures need to be

- ◆ Correct (sound) (*all probable statements are true*)
- ◆ Complete (*any entailed sentence can be proved*)
- ◆ Efficient (*requires less time to derive*)

Kinds of Knowledge

Things we need to talk about and reason about; what do we know?

- ◆ Objects
 - Descriptions
 - Classifications
- ◆ Events
 - Time sequence
 - Cause and effect
- ◆ Relationships
 - Among objects
 - Between objects and events
- ◆ Meta-knowledge

Distinguish between knowledge and its representation

Mappings are not one-to-one

Never gets it complete or exactly right

Knowledge engineering!

- ◆ Modeling the “right” conditions and the “right” effects at the “right” level of abstraction is very difficult
- ◆ Knowledge engineering (creating and maintaining knowledge bases for intelligent reasoning) is an entire field of investigation
- ◆ Many researchers hope that automated knowledge acquisition and machine learning tools can fill the gap:
 - Our intelligent systems should be able to learn about the conditions and effects, just like we do!
 - Our intelligent systems should be able to learn when to pay attention to, or reason about, certain aspects of processes, depending on the context!

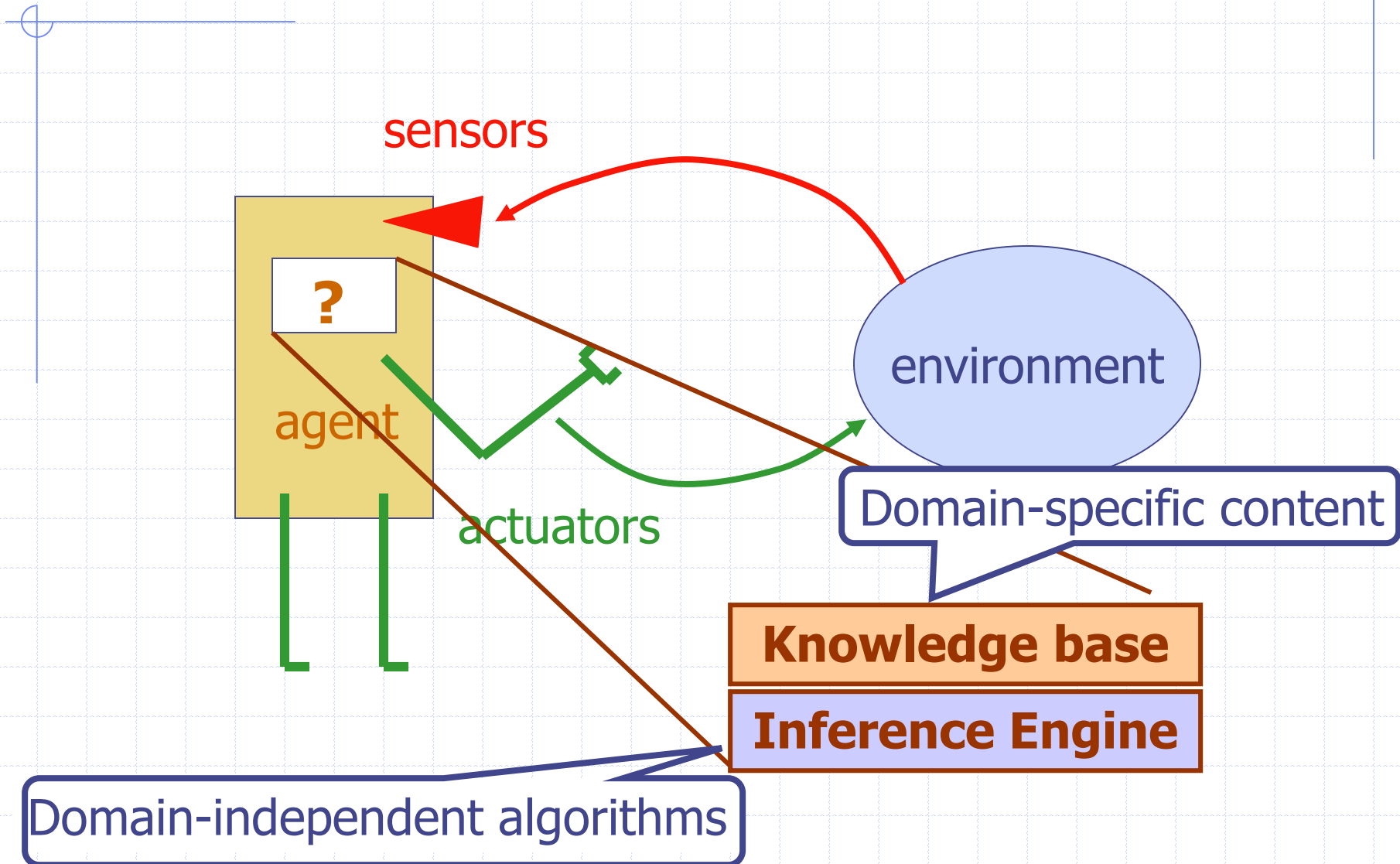
Some Typical Kinds of KR

- ◆ Logic and predicate calculus
- ◆ **Rules:** production systems
- ◆ Description logics, semantic nets, frames
- ◆ Scripts
- ◆ Ontologies
- ◆ Knowledge graphs

“Thinking Rationally”

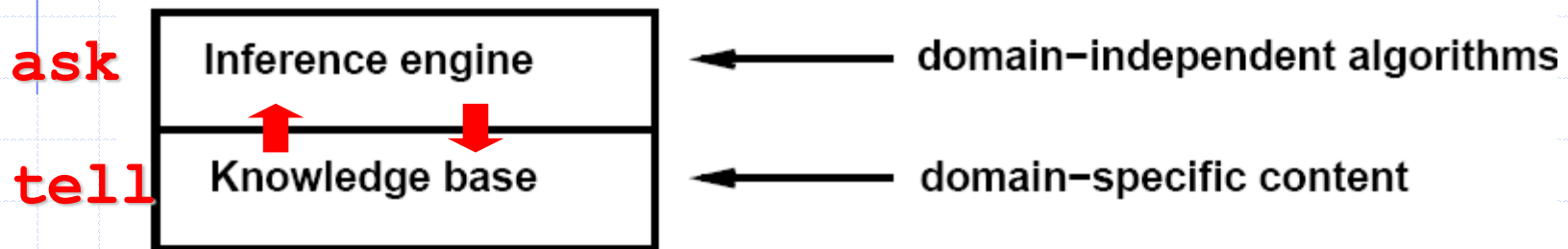
- ◆ Computational models of human “thought” processes
- ◆ Computational models of human behavior
- ◆ Computational systems that “think” rationally
- ◆ Computational systems that behave rationally

Knowledge-Based Agent



Knowledge Base

Knowledge Base: set of sentences represented in a knowledge representation language and represents assertions about the world



Inference rule: when one ASKs questions of the KB, the answer should *follow* from what has been TELLED to the KB previously.

Generic KB-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 \leftarrow ASK(*KB*, MAKE-ACTION-QUERY(*t*))


TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))

t \leftarrow *t* + 1

return *action*

- KB: initially contains background knowledge
- TELLS KB what it perceives
- ASKs KB what action should perform
- In the process of query, extensive reasoning may be done about the current state of the world, about outcomes of possible action sequences , and so on
- After action is chosen, agent records its choice with TELL and executes action

Abilities of KB agent

- 
- Represents states and actions
 - Incorporate new percepts
 - Update internal representation of the world
 - Deduce hidden properties of the world
 - Deduce appropriate actions

Description level

- ◆ KB agent == agents with internal state
- ◆ Description of agents at different levels
 - **Knowledge level**
 - ◆ What they know, regardless of the actual implementation (Declarative description)
 - ◆ E.g: automated taxi might have a goal, might know its actual location etc.
 - **Implementation level**
 - ◆ Data structures in KB and algorithms that manipulate them e.g propositional logic and resolution

Types of Knowledge



Procedural, e.g.: functions

- Encodes desired behaviors directly as program code
- Minimize the role of explicit representation
- Reasoning can result in a much more efficient way
- Such knowledge can only be used in one way -- by executing it



Declarative, e.g.: constraints and rules

- Initial program built by adding the sentences (knowledge about environment) one by one
- It can be used to perform many different sorts of inferences

The Wumpus World

- ◆ The Wumpus computer game
- ◆ The agent explores a cave consisting of rooms connected by passageways
- ◆ Lurking somewhere in the cave is the Wumpus, a beast that eats any agent that enters its room
- ◆ Some rooms contain bottomless pits that trap any agent that wanders into the room
- ◆ Occasionally, there is a heap of gold in a room
- ◆ The goal is to collect the gold and exit the world without being eaten

History of “Hunt the Wumpus”

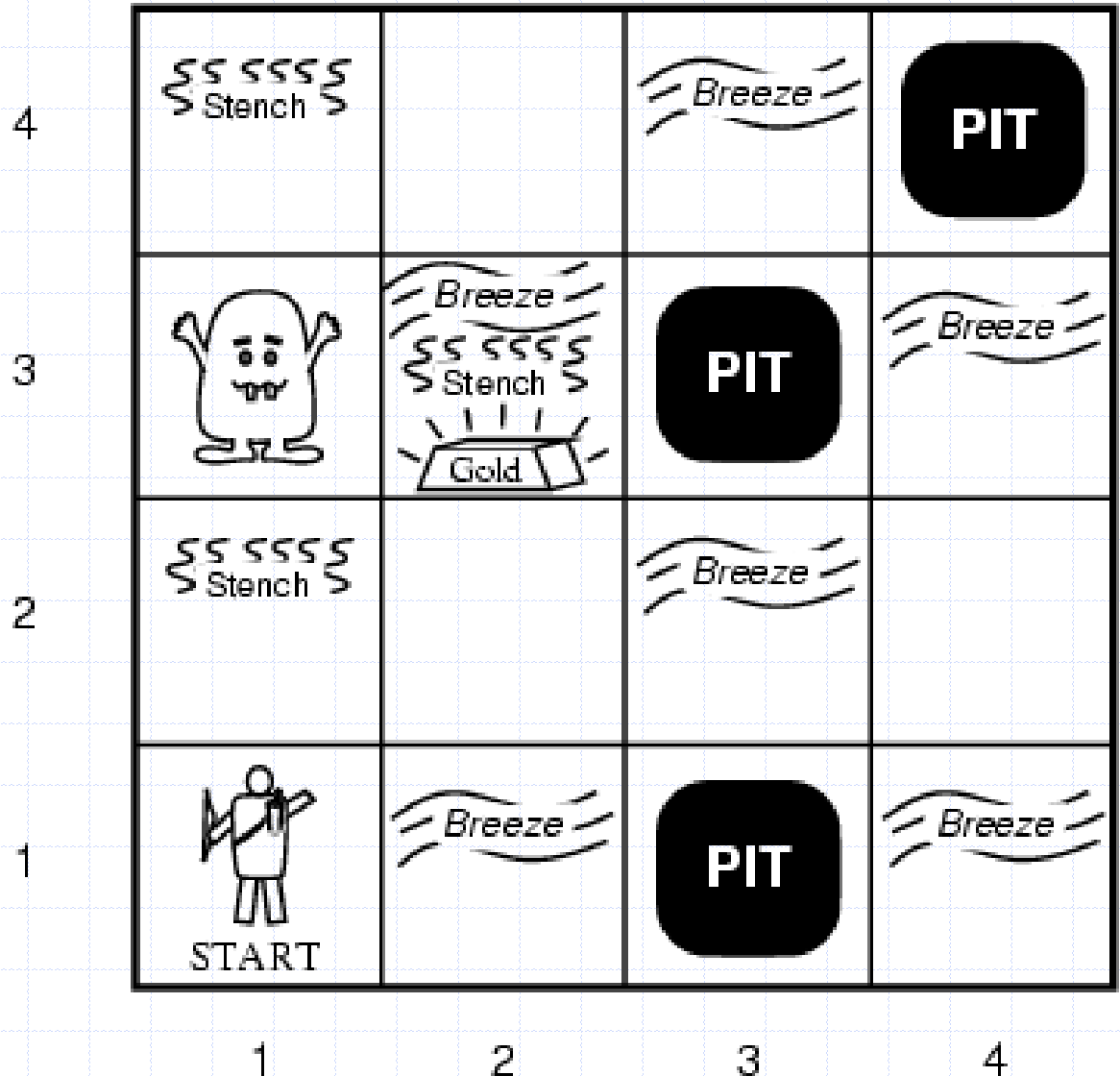
- ◆ WUMPUS /wuhm'p*s/ n. The central monster (and, in many versions, the name) of a famous family of very early computer games called “Hunt The Wumpus,” dating back at least to 1972 (several years before ADVENT) on the Dartmouth Time-Sharing System. The wumpus lived somewhere in a cave with the topology of a dodecahedron's edge/vertex graph (later versions supported other topologies, including an icosahedron and Mobius strip). The player started somewhere at random in the cave with five “crooked arrows”; these could be shot through up to three connected rooms, and would kill the wumpus on a hit (later versions introduced the wounded wumpus, which got very angry). Unfortunately for players, the movement necessary to map the maze was made hazardous not merely by the wumpus (which would eat you if you stepped on him) but also by bottomless pits and colonies of super bats that would pick you up and drop you at a random location (later versions added “anaerobic termites” that ate arrows, bat migrations, and earthquakes that randomly changed pit locations).
- ◆ This game appears to have been the first to use a non-random graph-structured map (as opposed to a rectangular grid like the even older Star Trek games). In this respect, as in the dungeon-like setting and its terse, amusing messages, it prefigured ADVENT and Zork and was directly ancestral to both. (Zork acknowledged this heritage by including a super-bat colony.) Today, a port is distributed with SunOS and as freeware for the Mac. A C emulation of the original Basic game is in circulation as freeware on the net.

Wumpus PEAS description

- ◆ **Performance measure:**
gold +1000, death -1000,
-1 per step, -10 use arrow
- ◆ **Environment:**
Squares adjacent to wumpus are smelly
Squares adjacent to pit are breezy
Glitter iff gold is in the same square
Bump iff move into a wall
Woeful scream iff the wumpus is killed
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
- ◆ **Sensors:** Stench, Breeze, Glitter, Bump, Scream
- ◆ **Actuators:** Left turn, Right turn, Forward, Grab, Release, Shoot

A typical Wumpus world

- ◆ The agent always starts in [1,1].
- ◆ The task of the agent is to find the gold, return to the field [1,1] and climb out of the cave.



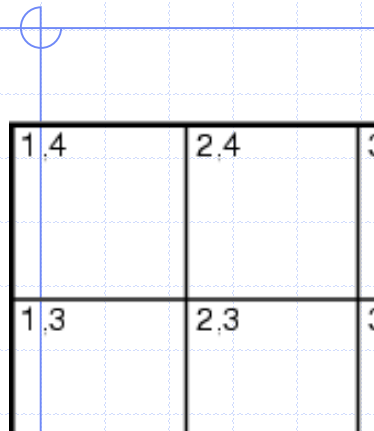
Wumpus World Characteristics

- ◆ Observable?
- ◆ Deterministic?
- ◆ Episodic or Sequential?
- ◆ Static?
- ◆ Discrete?
- ◆ Single-agent?

Wumpus World Characterization

- ◆ Observable?
 - No, only local perception
- ◆ Deterministic?
 - Yes, outcome exactly specified
- ◆ Episodic?
 - No, sequential at the level of actions (**current action may have future consequences**)
- ◆ Static?
 - Yes, Wumpus and pits do not move
- ◆ Discrete?
 - Yes
- ◆ Single-agent?
 - Yes, Wumpus is essentially a natural feature

The Wumpus agent's first step



1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
1,1 A OK	2,1	3,1	4,1

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

(a)

Sensors: Stench, Breeze, Glitter, Bump, Scream

- [1,1] The KB initially contains the rules of the environment. The first percept is [none, none, none, none, none], move to safe cell e.g. 2,1
- [2,1] breeze which indicates that there is a pit in [2,2] or [3,1], return to [1,1] to try next safe cell

The Wumpus agent's first step

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
1,1 A OK	2,1	3,1	4,1

(a)

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

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
1,1 V OK	2,1	3,1 P?	4,1

(b)

- [1,1] The KB initially contains the rules of the environment. The first percept is [*none, none, none, none, none*], move to safe cell e.g. 2,1
- [2,1] breeze which indicates that there is a pit in [2,2] or [3,1], return to [1,1] to try next safe cell

Next....

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1	4,1

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

(a)

[1,2] Stench in cell which means that wumpus is in [1,3] or [2,2]
 YET ... not in [1,1]
 YET ... not in [2,2] or stench would have been detected in [2,1]
 THUS ... wumpus is in [1,3]
 THUS [2,2] is safe because of lack of breeze in [1,2]
 THUS pit in [3,1]
 move to next safe cell [2,2]

Then...

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

(b)

[2,2] move to [2,3]
 [2,3] detect glitter , smell, breeze
 THUS pick up gold
 THUS pit in [3,3] or [2,4]

World-wide web wumpi

- ◆ [http://en.wikipedia.org/wiki/Hunt the Wumpus](http://en.wikipedia.org/wiki/Hunt_the_Wumpus)
- ◆ <http://www.atariarchives.org/bcc1/showpage.php?page=247>
- ◆ <http://www.ifiction.org/games/play.phpz>
- ◆ <http://www.taylor.org/~patrick/wumpus/>
- ◆ <http://www.inthe70s.com/games/wumpus/index.shtml#>

What is a logic?

◆ A formal language

- Syntax – what expressions are legal (well-formed)
- Semantics – what legal expressions mean
 - ◆ in logic the truth of each sentence with respect to each possible world

◆ E.g: *the language of arithmetic*

- $X+2 \geq y$ is a sentence but x^2+y is not
- $X+2 \geq y$ is true in a world where $x=7$ and $y=1$
- $X+2 \geq y$ is false in a world where $x=0$ and $y=6$

Syntax vs. Semantics issue

Refers to

FORM VS. CONTENT

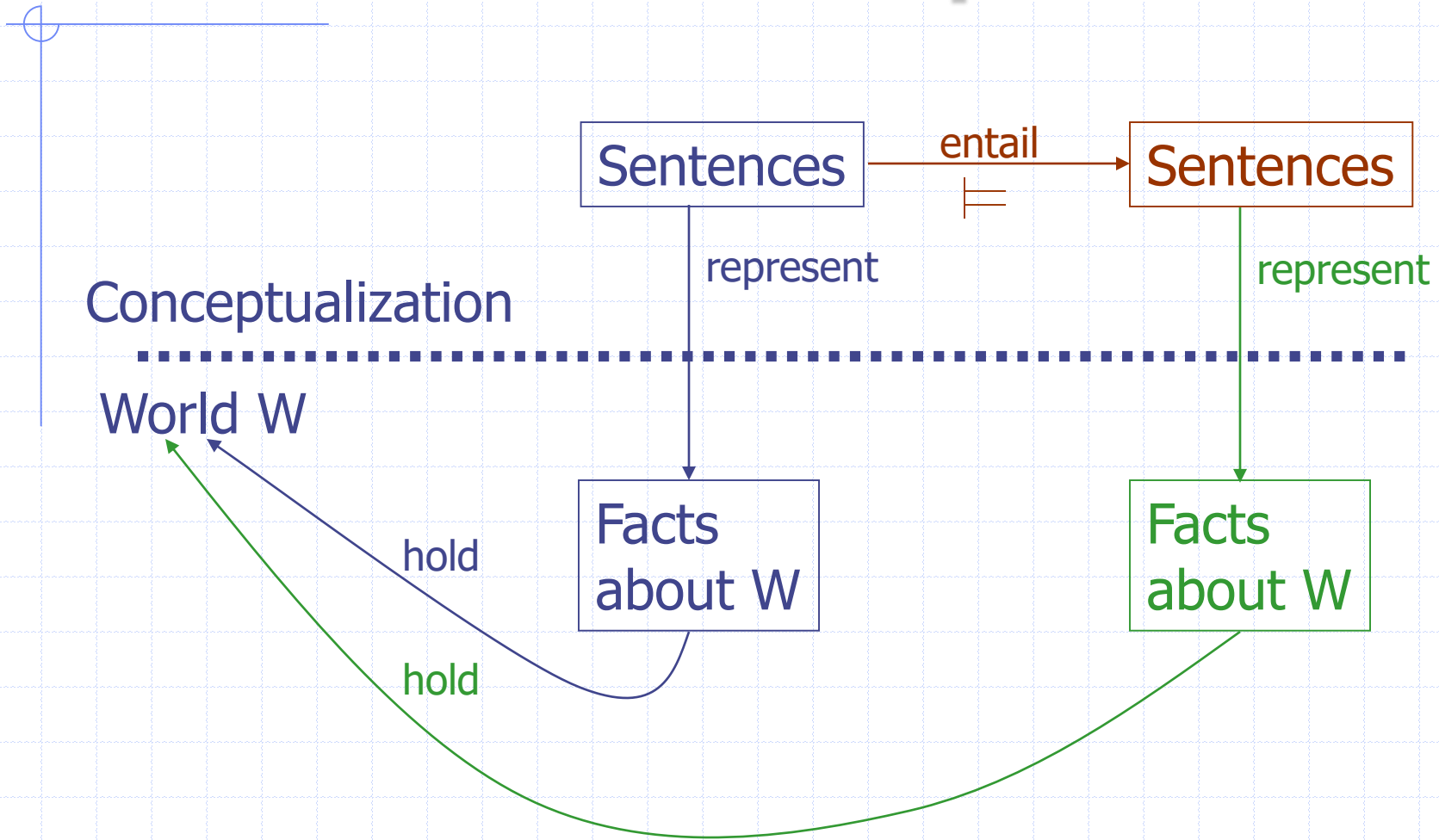
Form



Tea

(Content)

Connection World-Representation



Entailment

- ◆ Logical reasoning

- Involves the relation between sentences

- ◆ One sentence logically follows from another

$$KB \models \alpha$$

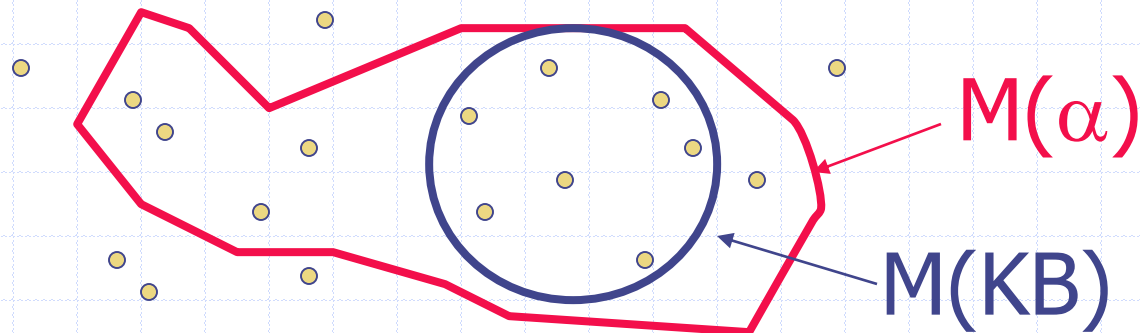
- ◆ KB entails sentence α *if and only if* α is true in worlds where KB is true

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

- ◆ Entailment is a relationship between sentences that is based on semantics

Models

- ◆ Models are formal definitions of possible states of the world
 - ◆ 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
- $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

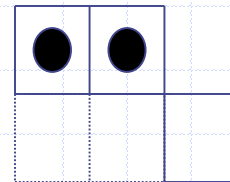
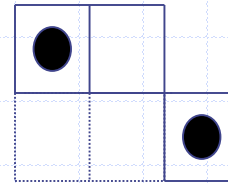
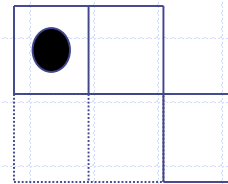
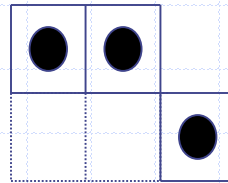
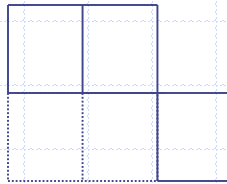
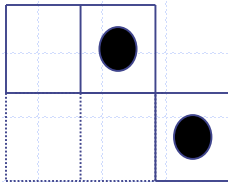
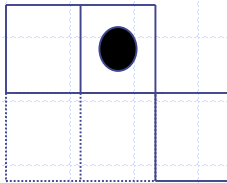
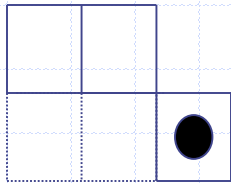


Entailment in the Wumpus World

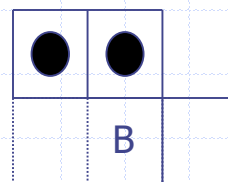
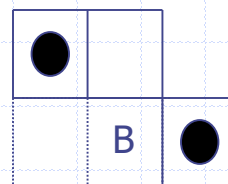
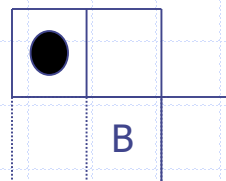
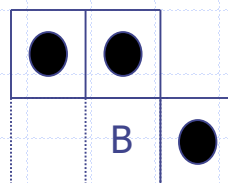
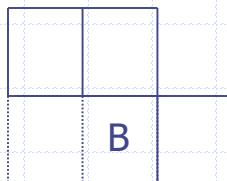
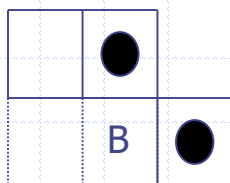
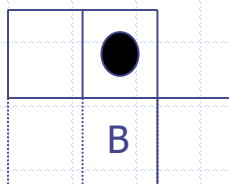
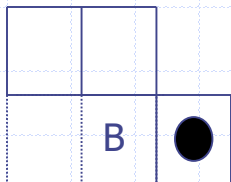
- ◆ Situation after detecting nothing in [1,1], moving right, breeze in [2,1]
- ◆ What are possible models for ? – assume only possibility pit or no pit.

?	?		
V	B V	?	

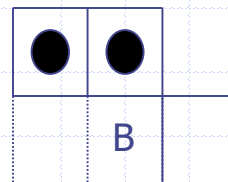
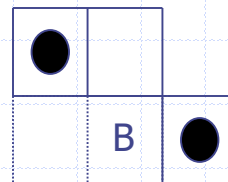
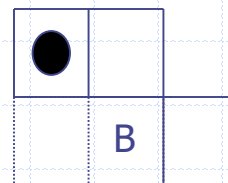
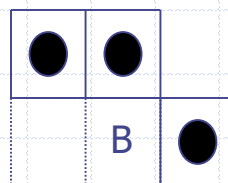
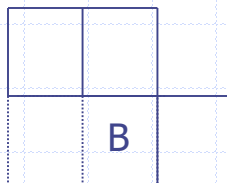
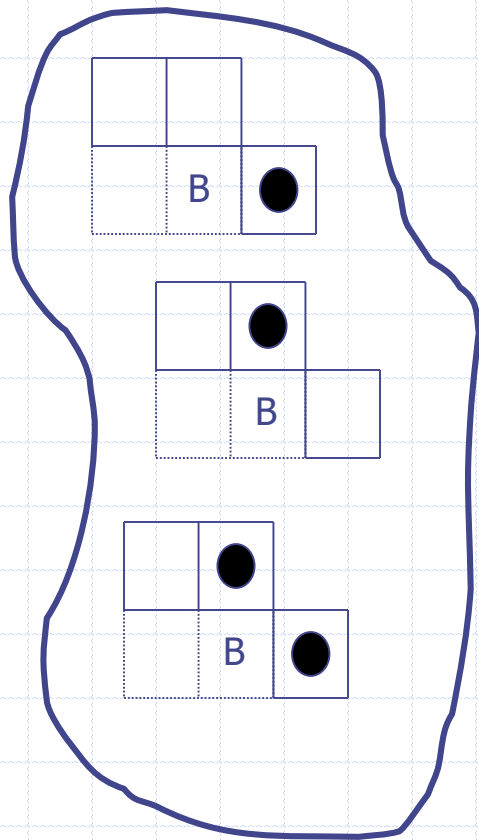
Wumpus Models



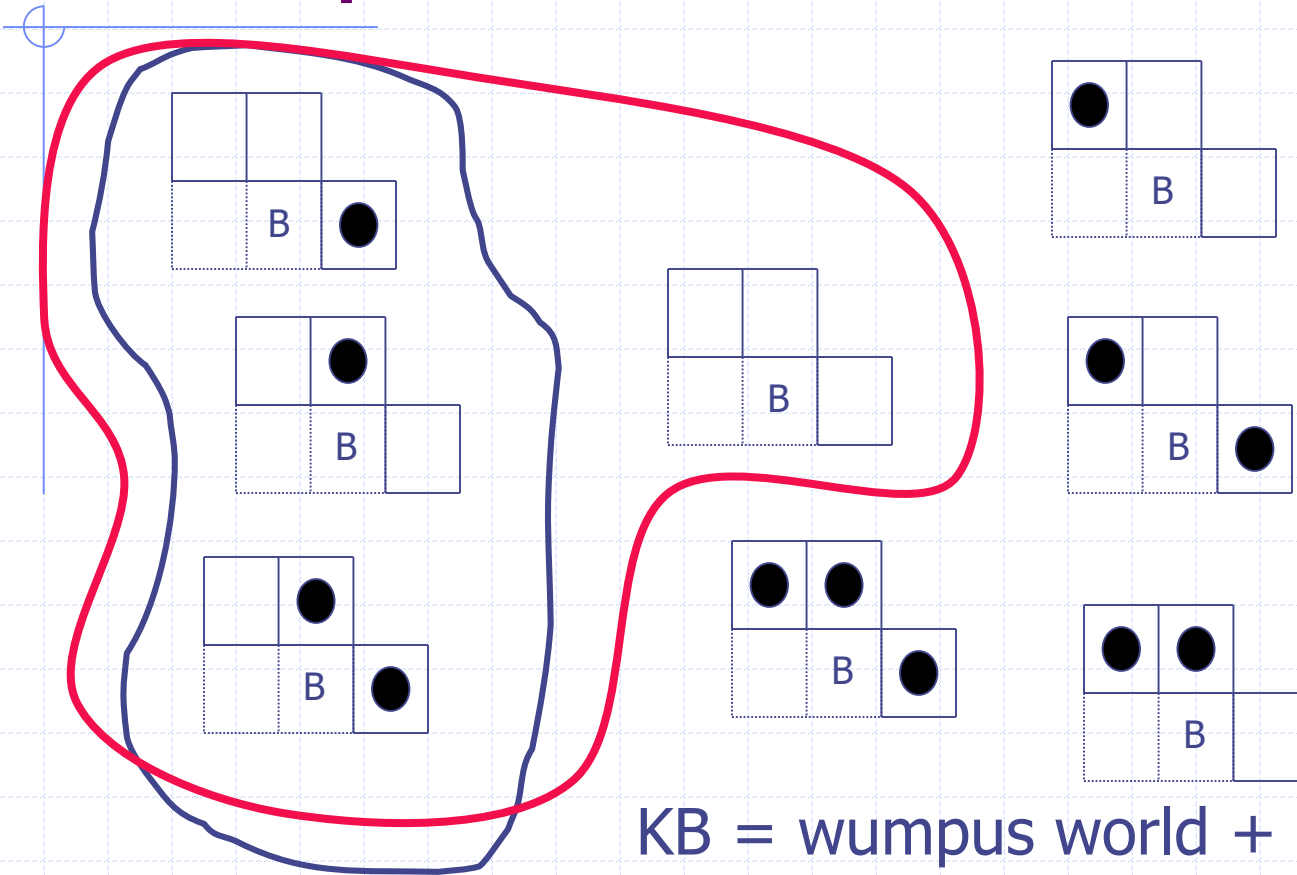
Wumpus Models



Wumpus Models



Wumpus Models

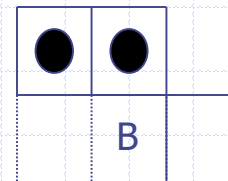
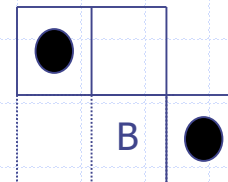
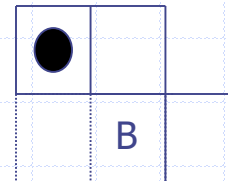
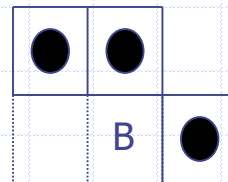
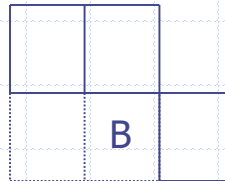
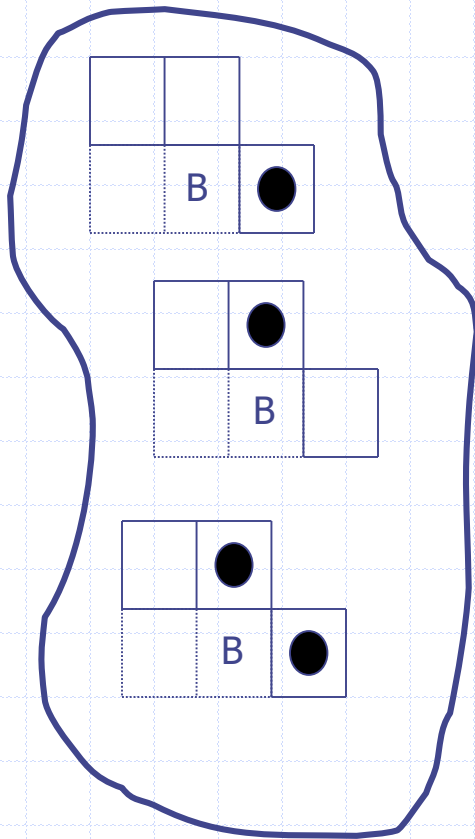


KB = wumpus world + observations

α_1 = "[1,2] is safe"

KB $\models \alpha_1$

Wumpus Models

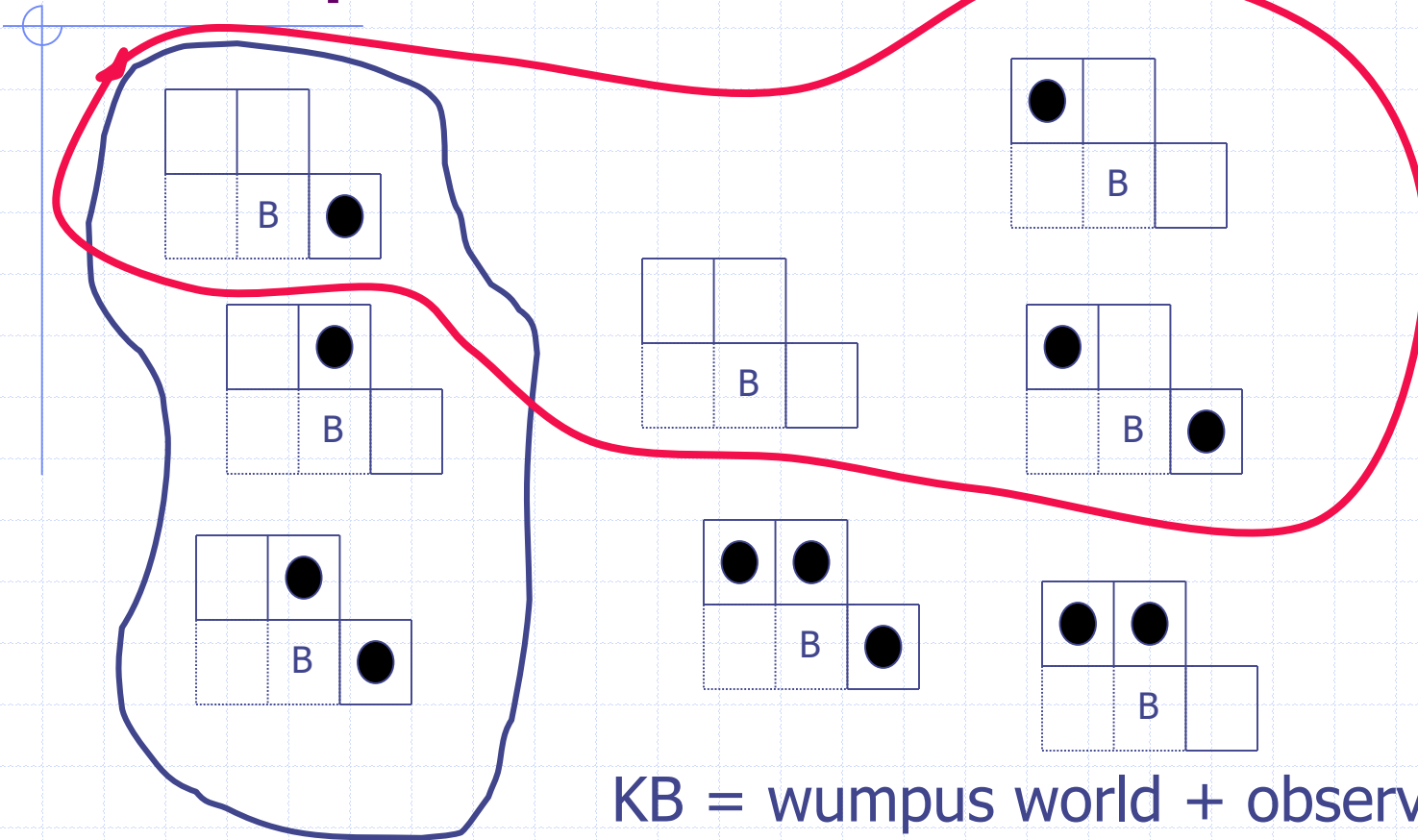


KB = wumpus world + observations

α_2 = "[2,2] is safe"

KB \models α_2 ??

Wumpus Models

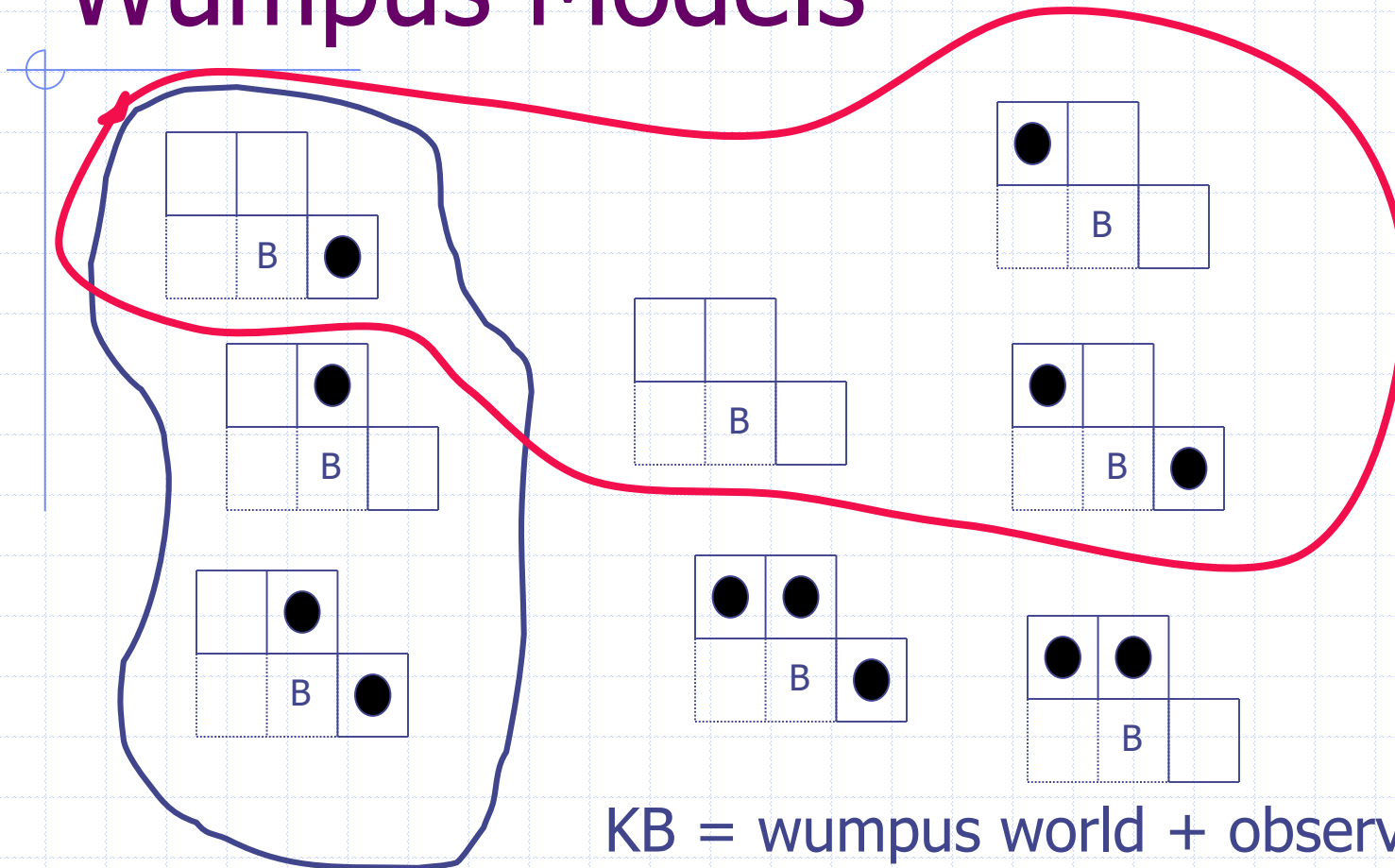


KB = wumpus world + observations

α_2 = "[2,2] is safe"

KB \models α_2 ??

Wumpus Models



KB = wumpus world + observations

α_2 = "[2,2] is safe"

KB $\not\models$ α_2 NOT!

Logical inference

- ◆ Logical inference: Notion of entailment can be used for deriving conclusions
 - Model checking (for Wumpus example):
enumerate all possible models and check whether α is true
- ◆ Distinction between entailment and inference
 - $KB \vdash_i \alpha$: *inference algo i derives α from KB*
- ◆ If an algorithm only derives entailed sentences it is called *sound* or *truth preserving*
 - Otherwise it just makes things up
 - i is sound if whenever $KB \vdash_i \alpha$ it is also true that $KB \models \alpha$*

Logical inference

◆ Completeness

- the algorithm can derive any sentence that is entailed
- i is complete if whenever $KB \models \alpha$ it is also true that $KB \vdash_i \alpha$*

If KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world

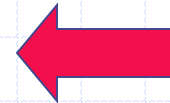
Examples of Logics

◆ Propositional logic

$$A \wedge B \Rightarrow C$$

◆ First-order logic

$$(\forall x) (\exists y) \text{ Mother}(y, x)$$



Symbols of Propositional Logic

- Propositional symbols, e.g., P, Q, R, \dots

True, False

- Connectives:

- ◆ \neg (negation): $\neg A$

- ◆ \wedge (and): $A \wedge B$

- ◆ \vee (or): $A \vee B$

- ◆ \Rightarrow (implies): $A \Rightarrow B$

- ◆ \Leftrightarrow (*biconditional*, i.e. if and only if): $A \Leftrightarrow B$

Syntax of Propositional Logic

sentence \rightarrow atomic sentence | complex sentence

atomic sentence \rightarrow propositional symbol | *True* | *False*

symbol $\rightarrow P$ | Q | R |

complex sentence \rightarrow \neg sentence
| (sentence \wedge sentence)
| (sentence \vee sentence)
| (sentence \Rightarrow sentence)
| (sentence \Leftrightarrow sentence)

◆ Examples:

- $((P \wedge Q) \Rightarrow R)$
- $(A \Rightarrow B) \vee (\neg C)$

Order of Precedence

(highest to lowest): \neg \wedge \vee \Rightarrow \Leftrightarrow

Examples:

- $\neg A \vee B \Rightarrow C$ is equivalent to $((\neg A) \vee B) \Rightarrow C$

Models in Propositional Logic

- ◆ Assignment of a truth value – true or false – to every atomic sentence
- ◆ Examples:
 - Let A, B, C, and D be the propositional symbols
 - is $m = \{A=\text{true}, B=\text{false}, C=\text{false}, D=\text{true}\}$ a model?
 - is $m' = \{A=\text{true}, B=\text{false}, C=\text{false}\}$ a model?
- ◆ How many models can be defined over n propositional symbols?

Semantics of Propositional Logic

- ◆ Specifies how to determine the truth value of any sentence in a model **m**
- ◆ Truth value of *True* : *True*
- ◆ Truth value of *False*: *False*
- ◆ The truth value of each atomic sentence is given by **m**
- ◆ The truth value of every other sentence is obtained recursively by using **truth tables**

Truth Tables

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$
<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>	<i>True</i>	<i>True</i>
<i>True</i>	<i>False</i>	<i>False</i>	<i>False</i>	<i>True</i>	<i>False</i>
<i>False</i>	<i>False</i>	<i>True</i>	<i>False</i>	<i>False</i>	<i>True</i>
<i>False</i>	<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>	<i>True</i>

About ($P \Rightarrow Q$)

- ◆ **Propositional logic does not require any relation of causation or relevance between P and Q**

- ◆ $ODD(5) \Rightarrow CAPITAL(\text{Japan}, \text{Tokyo})$

- ◆ $EVEN(5) \Rightarrow SMART(\text{Sam})$

Observation-1: 5 is odd implies Tokyo is the capital of Japan is a true sentence (*however it is decidedly odd sentence in English language*)

Observation-2: "5 is even implies Sam is smart" is true regardless whether Sam is smart (**Problem:** *implication is true whenever antecedent is false*)

- ◆ Read $A \Rightarrow B$ as:

"If A IS True, then I claim that B is True, otherwise I make no claim."

Wumpus world (symbols and knowledge base)

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 $[i, j]$.

R1: $\neg P_{1,1}$

R2: $\neg B_{1,1}$

R3: $B_{2,1}$

"Pits cause breezes in adjacent squares"

R4: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

R5: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

KB: $R1 \wedge \mathbf{R2} \wedge \mathbf{R3} \wedge \mathbf{R4} \wedge \mathbf{R5}$

$\neg P_{1,2}$ is entailed ??

$B_{1,1}$	B_2
<i>false</i>	<i>false</i>
<i>false</i>	<i>false</i>
\vdots	\vdots
<i>false</i>	<i>true</i>

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	α_1
<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>
<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>true</i>
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<u><i>true</i></u>	<u><i>true</i></u>
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<u><i>true</i></u>	<u><i>true</i></u>
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>	<u><i>true</i></u>	<u><i>true</i></u>
<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>false</i>

Logical equivalence

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

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

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

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

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

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

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

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

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

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

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

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

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

Terminology

A sentence is **valid** if it is true in **all** models (also known as *tautologies*),
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 (**NP-complete problem**)

A sentence is **unsatisfiable** if it is true in **no** models
e.g., $A \wedge \neg A$

Satisfiability is connected to inference via the following:
 $KB \models \alpha$ if and only if $(KB \wedge \neg \alpha)$ is **unsatisfiable**

Rules of Inference

Valid Rules of Inference:

- Modus Ponens
- And-Elimination
- And-Introduction
- Or-Introduction
- Double Negation
- Unit Resolution
- Resolution

Rules of Inference

◆ **Modus Ponens** (*Latin word: made that affirms*) :

$\alpha \Rightarrow \beta, \alpha \quad \beta$

$(\text{WumpusAhead} \wedge \text{WumpusAlive}) \Rightarrow \text{Shoot},$
 $(\text{WumpusAhead} \wedge \text{WumpusAlive})$
 $\quad \text{Shoot}$

$\alpha \Rightarrow \beta$
 α

 β

◆ **And-Elimination**: $\alpha \wedge \beta \quad \alpha$

$(\text{WumpusAhead} \wedge \text{WumpusAlive})$
 $\quad \text{WumpusAlive}$

$\alpha \wedge \beta$

◆ **Resolution**: $\alpha \vee \beta, \neg \beta \vee \gamma \quad \alpha \vee \gamma$
 $(\text{WumpusDead} \vee \text{WumpusAhead}),$
 $(\neg \text{WumpusAhead} \vee \text{Shoot})$
 $\quad (\text{WumpusDead} \vee \text{Shoot})$

α

$\alpha \vee \beta$
 $\neg \beta \vee \gamma$

 $\alpha \vee \gamma$

Proof Using Rules of Inference

Prove $A \Rightarrow B$, $(A \wedge B) \Rightarrow C$, Therefore $A \Rightarrow C$

⑩ $A \Rightarrow B \quad \neg A \vee B$

⑩ $A \wedge B \Rightarrow C \quad \neg (A \wedge B) \vee C \quad \neg A \vee \neg B \vee C$

⑩ So $\neg A \vee B$ resolves with $\neg A \vee \neg B \vee C$ deriving
 $\neg A \vee C$

⑩ Equivalent to $A \Rightarrow C$

Rules of Inference (continued)

◆ And-Introduction

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

◆ Or-Introduction

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

◆ Double Negation

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

◆ Unit Resolution (special case of resolution)

$$\frac{\alpha \vee \beta \quad \neg \beta}{\alpha} \quad \text{Alternatively} \quad \frac{\neg \alpha \Rightarrow \beta \quad \neg \beta}{\alpha}$$

Wumpus World KB

◆ Proposition Symbols for each i,j :

- Let $P_{i,j}$ be true if there is a pit in square i,j
- Let $B_{i,j}$ be true if there is a breeze in square i,j

◆ Sentences in KB

- “There is no pit in square 1,1”

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

- “A square is breezy iff pit in a neighboring square”

$$R_2: B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

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

- “Square 1,1 has no breeze”

$$R_4: \neg B_{1,1}$$

- Square 1,2 has a breeze”

$$R_5: B_{1,2}$$

prove $\neg P_{1,2}$

Inference in Wumpus World

- ◆ Apply biconditional elimination to R_2 :
 $R_6: (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- ◆ Apply AE to R_6 :
 $R_7: ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- ◆ Contrapositive of R_7 :
 $R_8: (\neg B_{1,1}) \Rightarrow \neg (P_{1,2} \vee P_{2,1})$
- ◆ Modus Ponens with R_8 and $R_4 (\neg B_{1,1})$:
 $R_9: \neg (P_{1,2} \vee P_{2,1})$
- ◆ de Morgan:
 $R_{10}: \neg P_{1,2} \wedge \neg P_{2,1}$

Therefore, neither [1,2] nor [2,1] contains pit

Wumpus world (symbols and knowledge base)

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 $[i, j]$.

$$R1: \neg P_{1,1}$$

$$R2: \neg P_{1,2}$$

$$R3: B_{2,1}$$

"Pits cause breezes in adjacent squares"

$$R4: B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$R5: B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

$$KB: R1 \wedge R2 \wedge R3 \wedge R4 \wedge R5$$

Inference in PL

- ◆ Inference in PL is NP-complete
 - Worst case: enumerate all the models
- ◆ Solution: *ignore irrelevant propositions*
- ◆ E.g.
 - proof leading to : $P_{1,2} \wedge \neg P_{2,1}$
 - Does not mention $P_{1,1}$ $P_{2,2}$ $P_{3,1}$ $B_{2,1}$
 - Ignore these
 - Goal, $P_{1,2}$ appears only in R4 and R2
 - Not necessary to keep R1, R3, and R5

Monotonicity property

◆ Set of entailed sentences can only *increase*

$KB \models \alpha$ then $KB \wedge \Phi \models \alpha$

Knowledge: Suppose, Φ : there are 8 pits in the world

- helps to draw additional conclusions
- can't *invalidate* any conclusion α *already inferred*
(e.g., no pit in [1,2])

Monotonicity:

- Inference rules can be applied whenever suitable premises are found in KB
- Conclusion of rule must follow regardless of what else is there in KB

Resolution

Conjunctive Normal Form (CNF)

conjunction of disjunctions of literal clauses

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

◆ Resolution inference rule (for CNF):

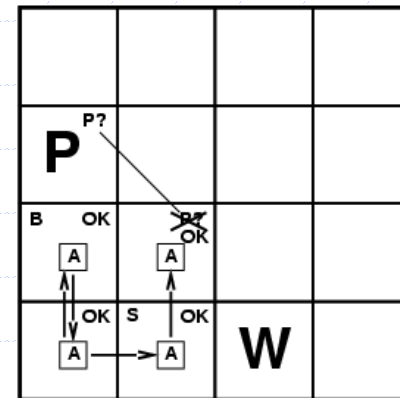
$$\frac{l_1 \vee \dots \vee l_k, \quad m_1 \vee \dots \vee m_n}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

where l_i and m_j are complementary literals.

E.g., $P_{1,3} \vee P_{2,2}, \quad \neg P_{2,2}$

$P_{1,3}$

- ⑩ Resolution is sound and complete for propositional logic



Resolution

Soundness of resolution inference rule:

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

Resolution (Refutation Completeness)

Given A is true

- ❑ Not possible to generate consequence $A \vee B$ with resolution
- ❑ Resolution can be applied to answer whether $A \vee B$ true !
 - ❑ **Refutation completeness**: either confirm or refute a sentence, but can't enumerate true sentences

Conjunctive Normal Form (CNF)

- Resolution rule applies only to disjunctions of literals
- How can lead to complete inference procedure for PL?
 - *Every sentence of PL is logically equivalent to a conjunction of disjunction of literals*

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 distributive law (\wedge over \vee) and flatten:

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

Resolution algorithm

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

function PL-RESOLUTION(KB, α) **returns** *true or false*

clauses \leftarrow the set of clauses in the CNF representation of $KB \wedge \neg \alpha$

new $\leftarrow \{ \}$

loop do

for each C_i, C_j **in** *clauses* **do**

resolvents \leftarrow PL-RESOLVE(C_i, C_j)

if *resolvents* contains the empty clause **then return** *true*

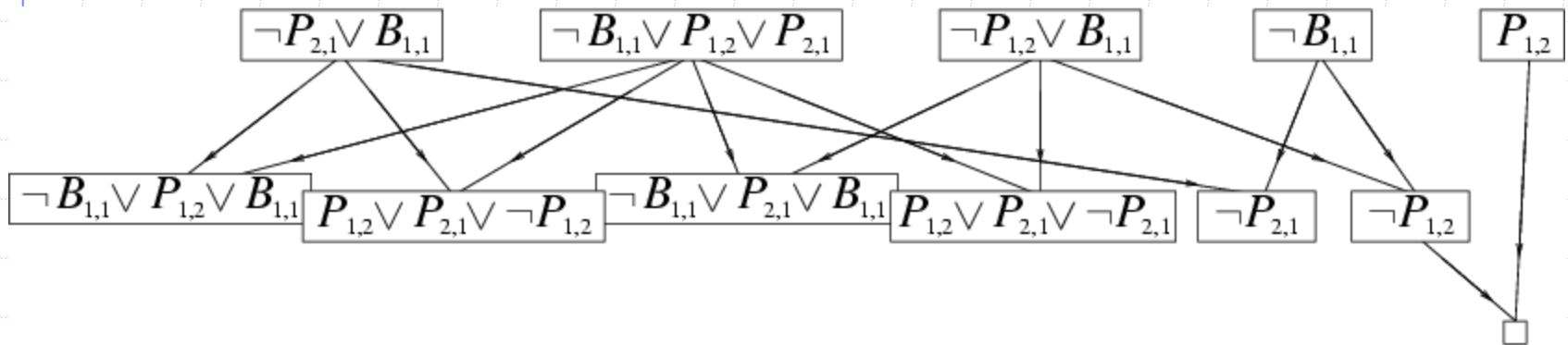
new \leftarrow *new* \cup *resolvents*

if *new* \subseteq *clauses* **then return** *false*

clauses \leftarrow *clauses* \cup *new*

Resolution example

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



Resolution (completeness)

Resolution closure: $RC(S)$

- set of all clauses S derivable by repeated application of the resolution rule to clauses in S or their derivatives
- Finite: only finitely many distinct clauses can be constructed from the symbols

■ Ground resolution theorem:

If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause

■ Proof:

contrapositive: If the closure $RC(S)$ does not contain empty clause then S is satisfiable

Resolution (completeness)

Construct a model for S with truth values for P_1, P_2, \dots, P_k

For $i = 1$ to k

- If there is a clause in $RC(S)$ containing the literal $\neg P_i$ such that all its other literals are false under the assignment chosen for P_1, P_2, \dots, P_{i-1} then assign false to P_i
- otherwise, assign true to P_i

Forward and backward chaining

- ◆ Horn Form (restricted): in practical situations full resolution is not required

KB = conjunction of Horn clauses

- Horn clause (disjunction of literals of which at most one is positive)
 - ◆ proposition symbol; or
 - ◆ (conjunction of symbols) \Rightarrow symbol
- E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

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

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

- ⑩ Can be used with forward chaining or backward chaining
- ⑩ These algorithms are very natural and run in linear time

Some properties of horn clauses

- ◆ Can be written as an implication
 - Premise: conjunction of positive literals
 - Conclusion: single positive literale.g. $\neg A \vee \neg B \vee C \equiv (A \wedge B) \Rightarrow C$
- ◆ Definite clause
 - Horn clause with exactly one positive literal
 - Head: positive literal
 - Body: negative literals
 - *Fact* if does not contain any negative literal
- ◆ Integrity constraints
 - Horn clause with no positive literale.g. $\neg A \vee \neg B$
 $A \wedge B \Rightarrow \text{False}$

Forward chaining

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

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

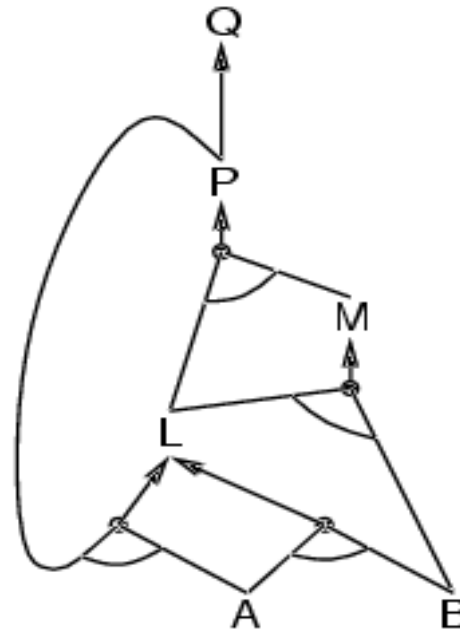
$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

A

B



Forward chaining algorithm

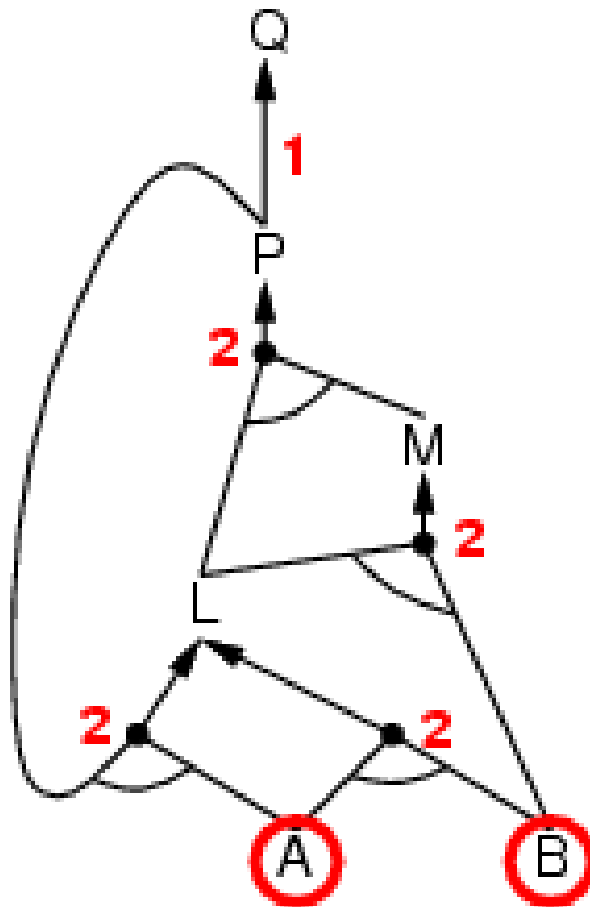
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                  inferred, a table, indexed by symbol, each entry initially false
                  agenda, a list of symbols, initially the symbols known to be true

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

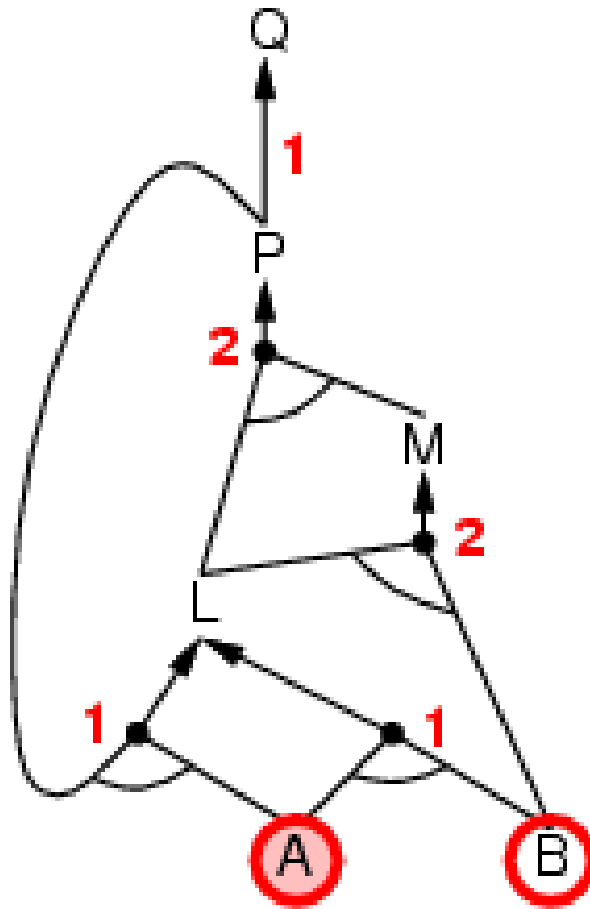
  return false
```

Forward chaining is sound and complete for Horn KB

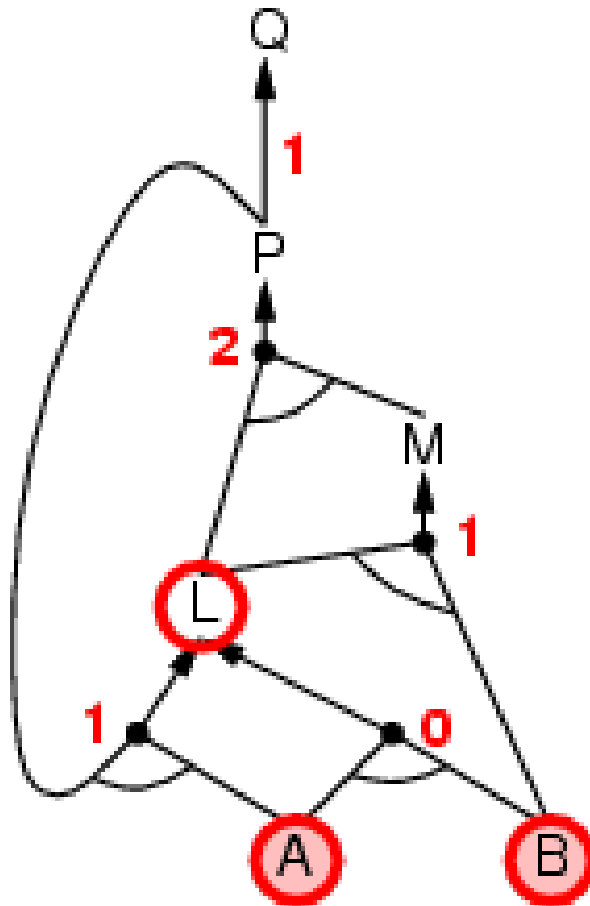
Forward chaining example



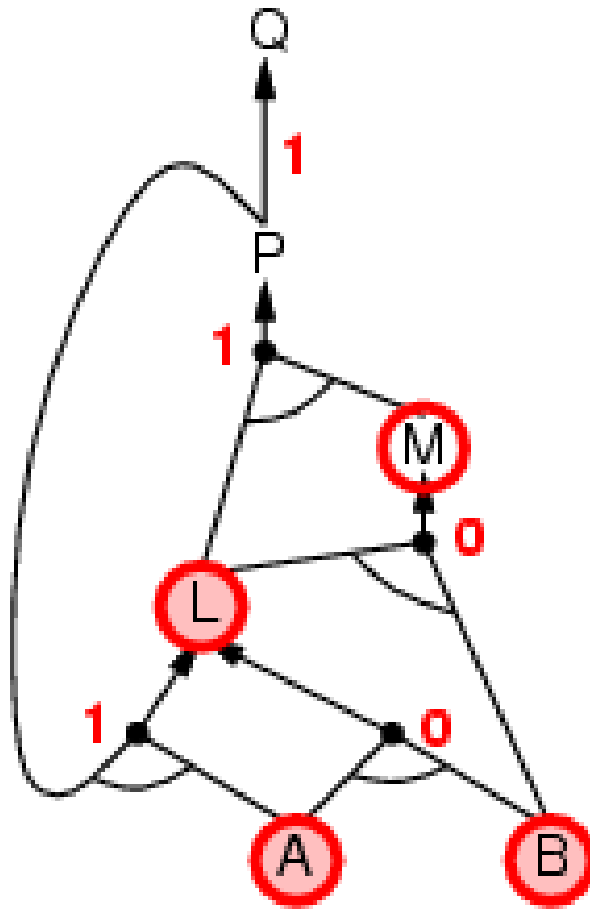
Forward chaining example



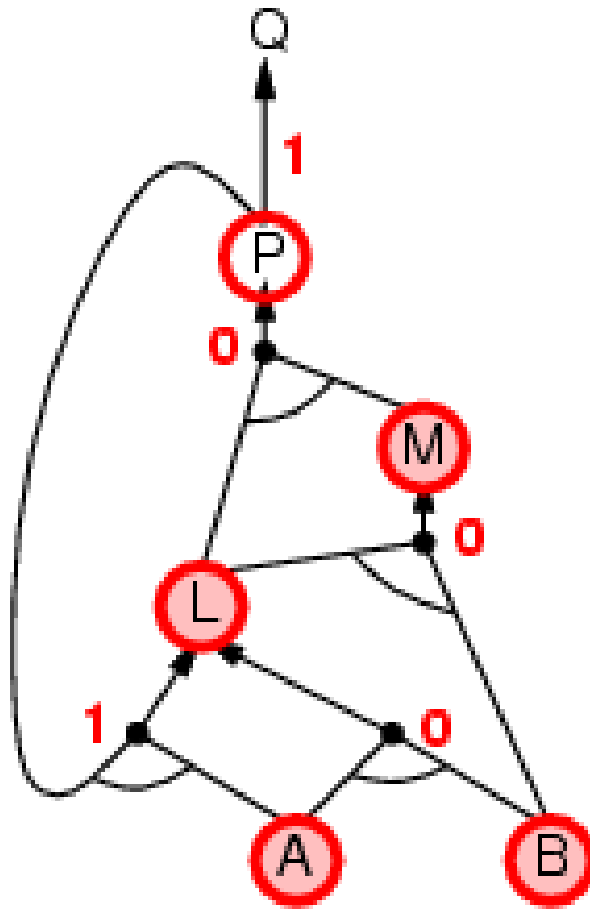
Forward chaining example



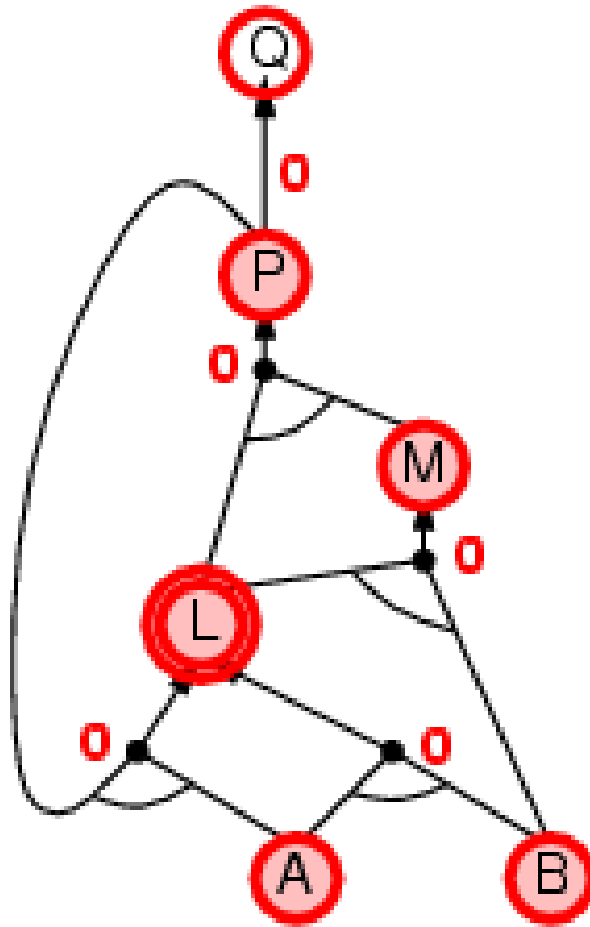
Forward chaining example



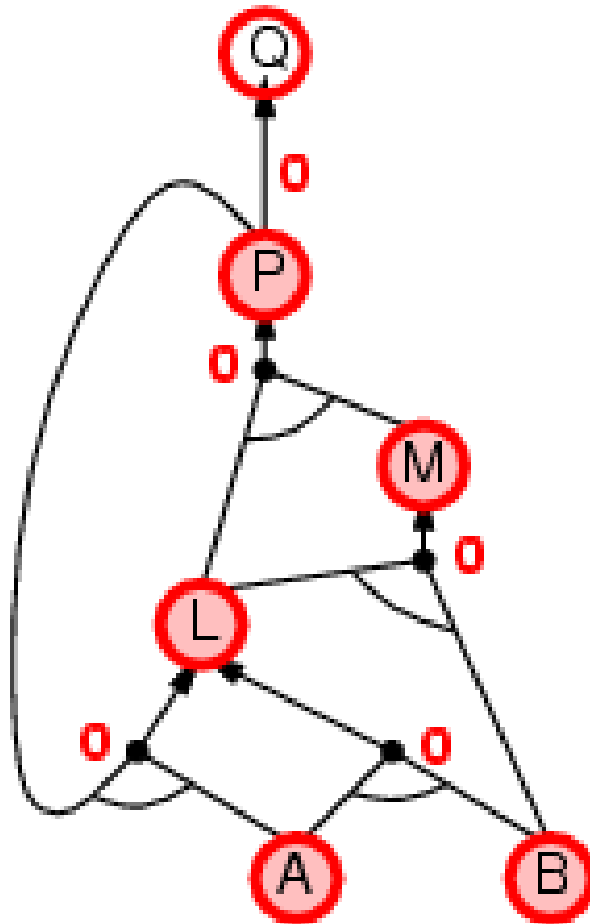
Forward chaining example



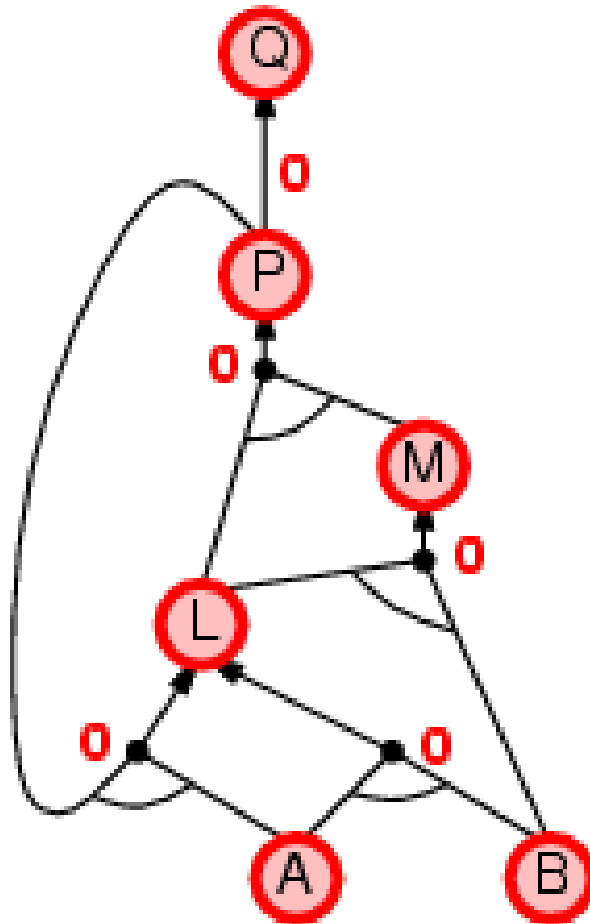
Forward chaining example



Forward chaining example



Forward chaining example



Proof of completeness

FC derives every atomic sentence that is entailed by KB

1. FC reaches a **fixed point** where no new atomic sentences are derived
2. Consider the final state as a model m , assigning true/false to symbols
3. Every definite clause in the original KB is true in m

$a_1 \wedge \dots \wedge a_k \Rightarrow b$ (let us assume this as false in m)

$\rightarrow a_1 \wedge \dots \wedge a_k$: true in m

$\rightarrow b$: false in m

Contradicts that the algorithm has reached a fixed point !

Proof of completeness

4. Hence, set of atomic sentences inferred at the fixed point defines a model (i.e. m) of KB

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

Backward chaining

Idea: work backwards from the query q :

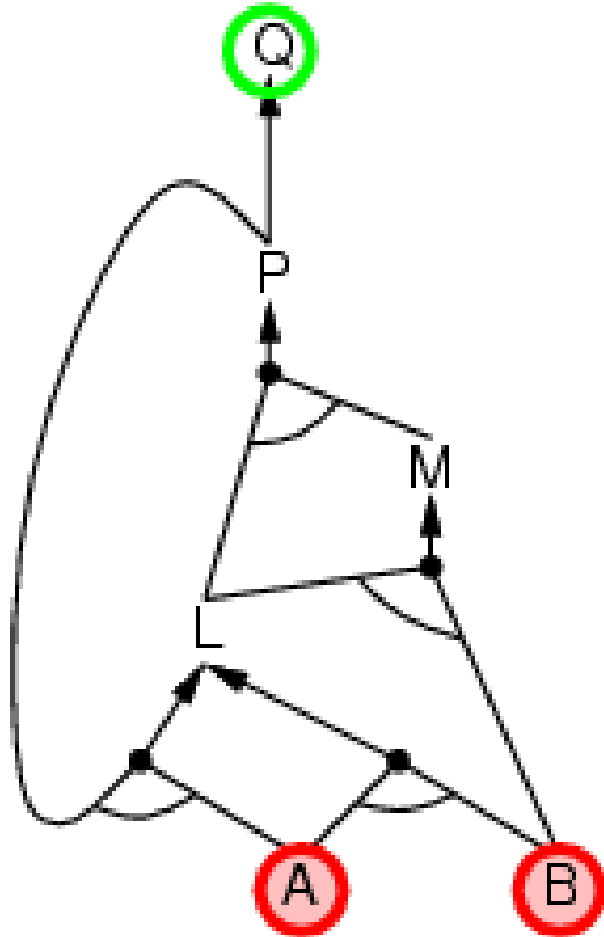
to prove q by BC,
check if q is known already, or
prove by BC all premises of some rule concluding q

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

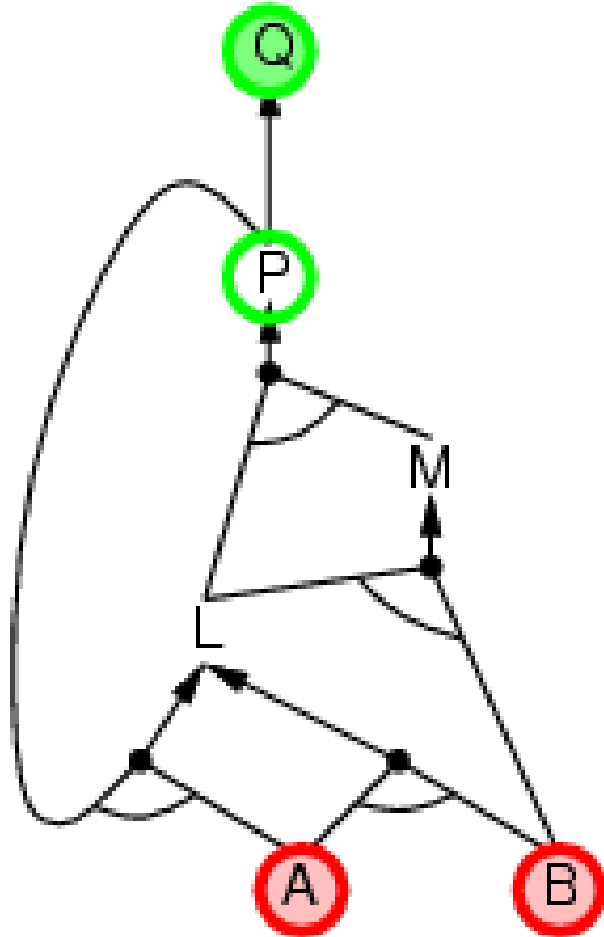
Avoid repeated work: check if new sub-goal

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

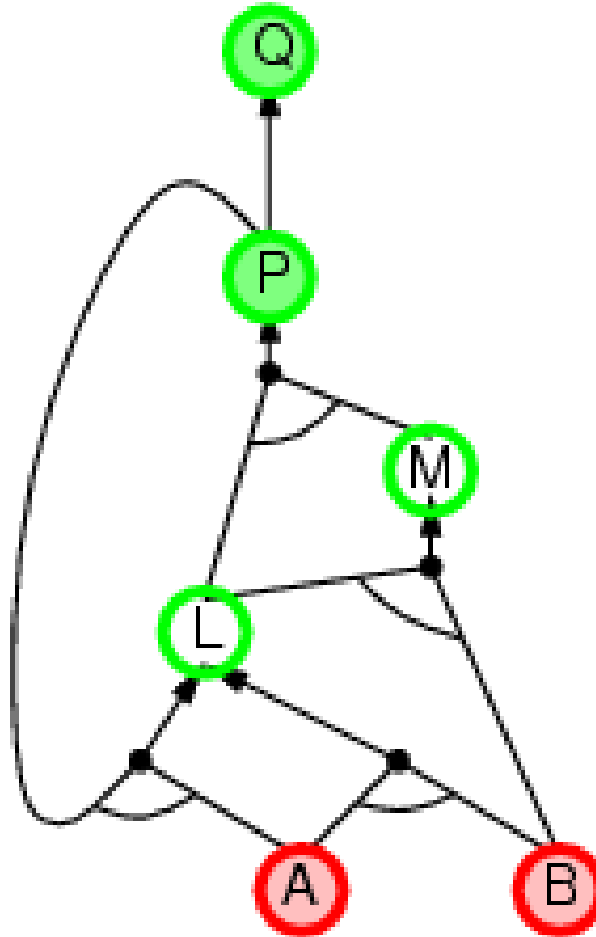
Backward chaining example



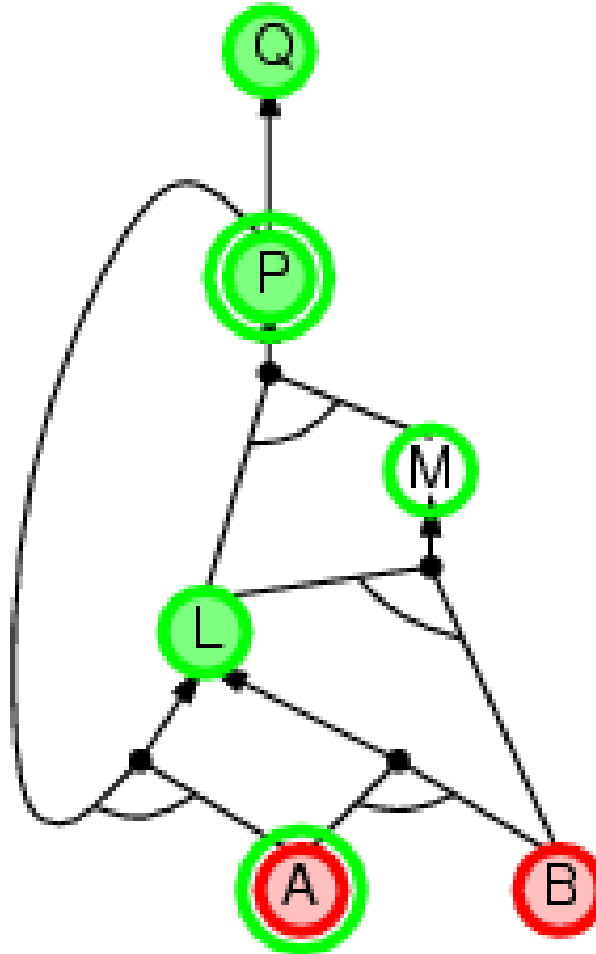
Backward chaining example



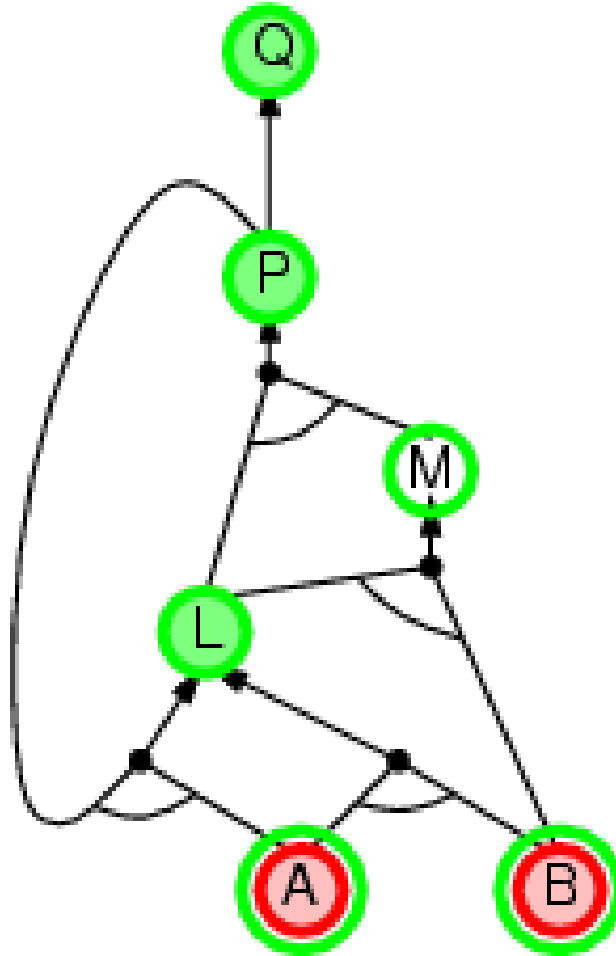
Backward chaining example



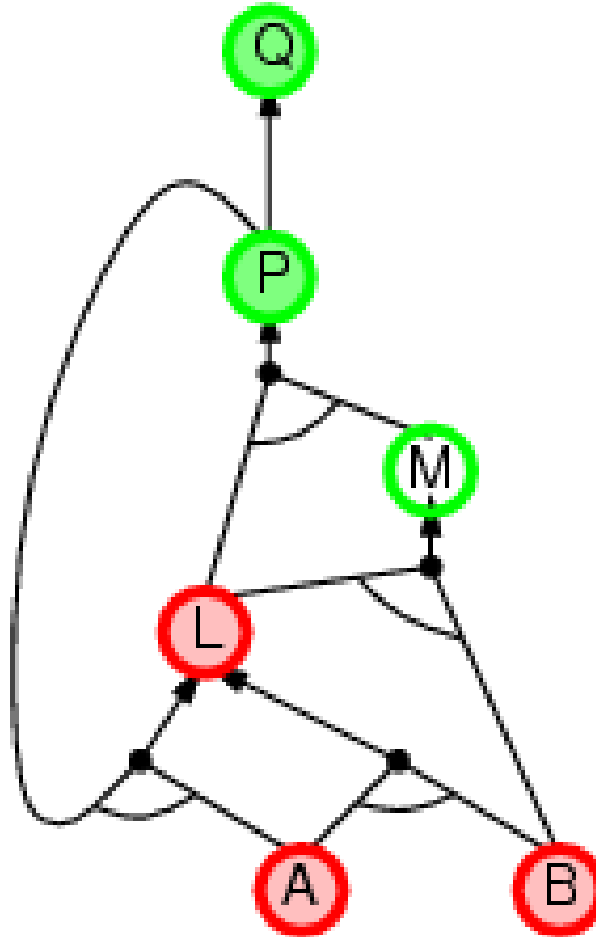
Backward chaining example



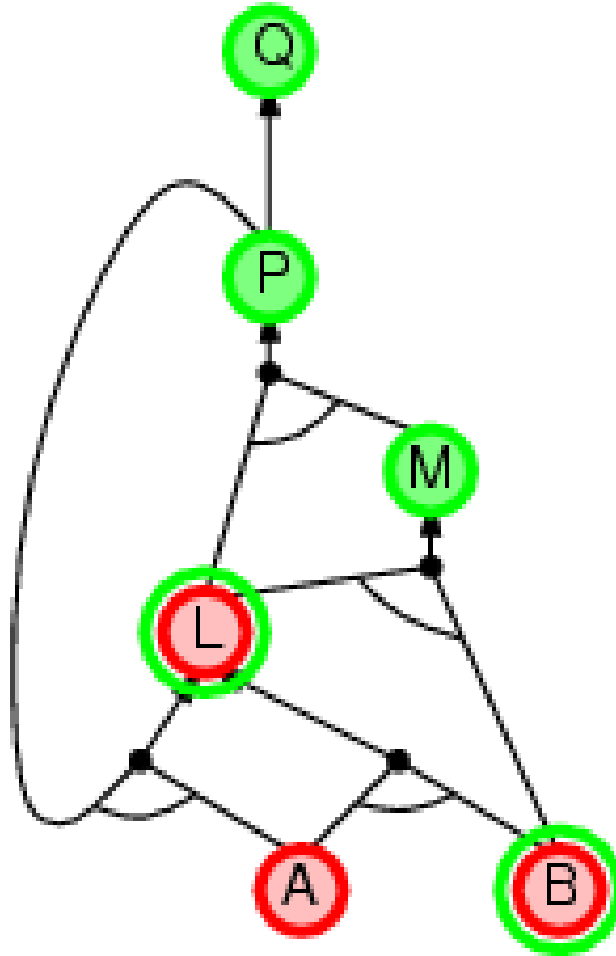
Backward chaining example



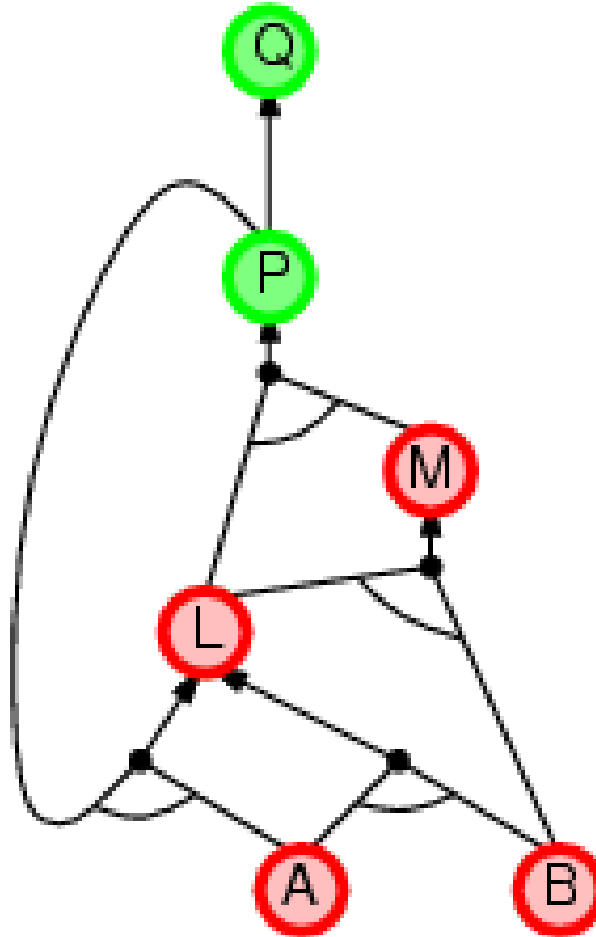
Backward chaining example



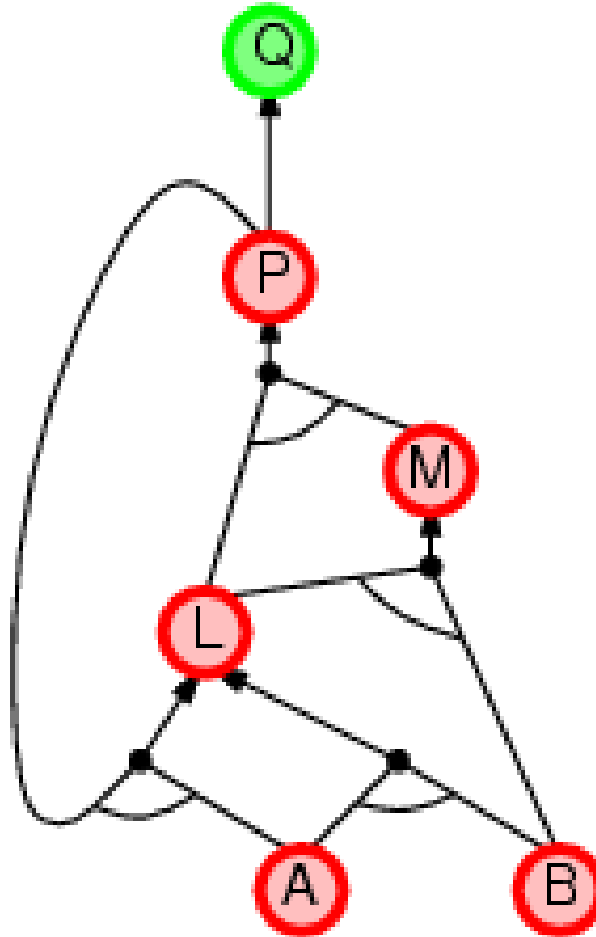
Backward chaining example



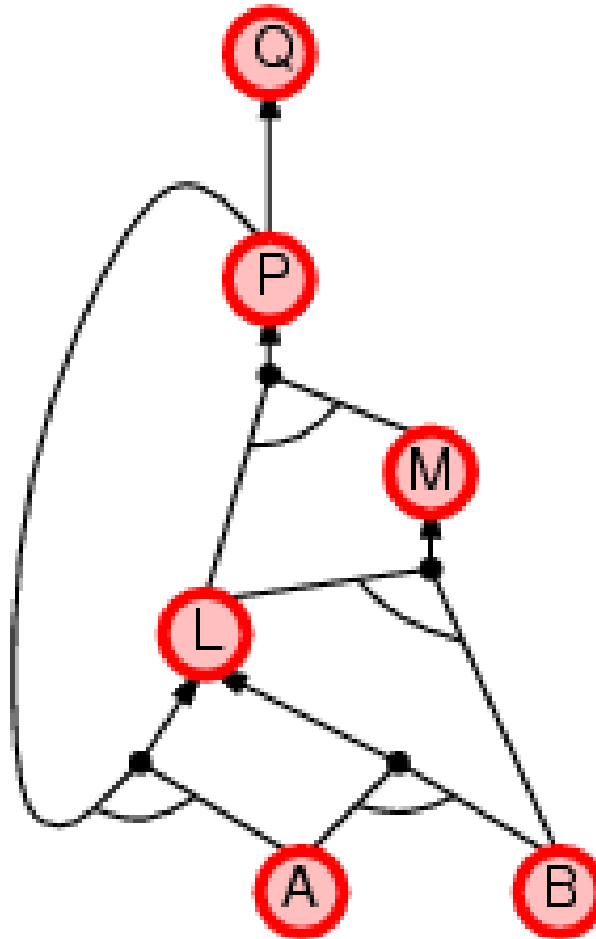
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

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

Efficient propositional inference

Two families of efficient algorithms for propositional inference:

- ◆ DPLL algorithm (Davis, Putnam, Logemann, Loveland)
 - 1960 (1962)
 - Complete, Backtracking algorithm
 - Recursive, Depth-first Search
- ◆ Incomplete local search algorithms
 - **WalkSAT algorithm**
 - **Hill Climbing**

The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable

Improvements over truth table enumeration:

1. Early termination

A clause is true if any literal is true

A sentence is false if any clause is false

(*reduces no of comparisons*)

2. Pure symbol heuristic

Pure symbol: always appears with the same "sign" in all clauses.

e.g., In the three clauses $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, A and B are pure, C is impure.

Make a pure symbol literal true.

To ensure that a sentence has a model

The DPLL algorithm

- **Unit clause heuristic**

Unit clause: only one literal in the clause
The only literal in a unit clause must be true.

Let $B = \text{false}$; then $B \vee \sim C = \sim C$ (an unit clause)

C must be set to *false*, if $\sim C$ has to be true

*Any attempt to prove a literal that is already there in the KB
Will succeed immediately*

Assigning one unit clause will generate another

e.g. when C is set to false, $C \vee A$ becomes a unit clause,
causing A to be assigned true (***equivalent to unit
propagation like forward chaining algorithm***)

The DPLL algorithm

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

inputs: *s*, a sentence in propositional logic

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

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

return DPLL(*clauses*, *symbols*, [])

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

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

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

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

if *P* is non-null **then return** DPLL(*clauses*, *symbols* - *P*, [*P* = *value* | *model*])

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

if *P* is non-null **then return** DPLL(*clauses*, *symbols* - *P*, [*P* = *value* | *model*])

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

return DPLL(*clauses*, *rest*, [*P* = *true* | *model*]) **or**

DPLL(*clauses*, *rest*, [*P* = *false* | *model*])

The WalkSAT algorithm

- ◆ Incomplete, local search algorithm
- ◆ Evaluation function: *The min-conflict heuristic of minimizing the number of unsatisfied clauses*
- ◆ Balance between greediness and randomness
(*many local minima; to escape various forms of randomness required*)

The WalkSAT algorithm

function WALKSAT(*clauses*, *p*, *max-flips*) **returns** a satisfying model or *failure*

inputs: *clauses*, a set of clauses in propositional logic

p, the probability of choosing to do a “random walk” move

max-flips, number of flips allowed before giving up

model \leftarrow a random assignment of *true/false* to the symbols in *clauses*

for *i* = 1 **to** *max-flips* **do**

if *model* satisfies *clauses* **then return** *model*

clause \leftarrow a randomly selected clause from *clauses* that is false in *model*

with probability *p* **flip** the value in *model* of a randomly selected symbol
 from *clause*

else flip whichever symbol in *clause* maximizes the number of satisfied clauses

return *failure*