Artificial Intelligence
ENCS 434

Logic

PL + FOL

Aziz M. Qaroush - Birzeit University

Introduction

- **Problem-solving Agent by search**
  - Figure out what exactly the problem is
  - Design an algorithm to solve the problem (by search)
  - Execute the program
- Searching algorithm can find the shortest path from Arad to Bucharest, but can't easily adjust/adapt when the road from Vilcea to Pitesti is closed for maintenance.
- **Knowledge-Based Agents**
  - Identify the knowledge to solve the problem
  - Write down/Represent the knowledge in a machine readable form
  - Use logical inference/Reasoning to solve the problem

Logical Agents

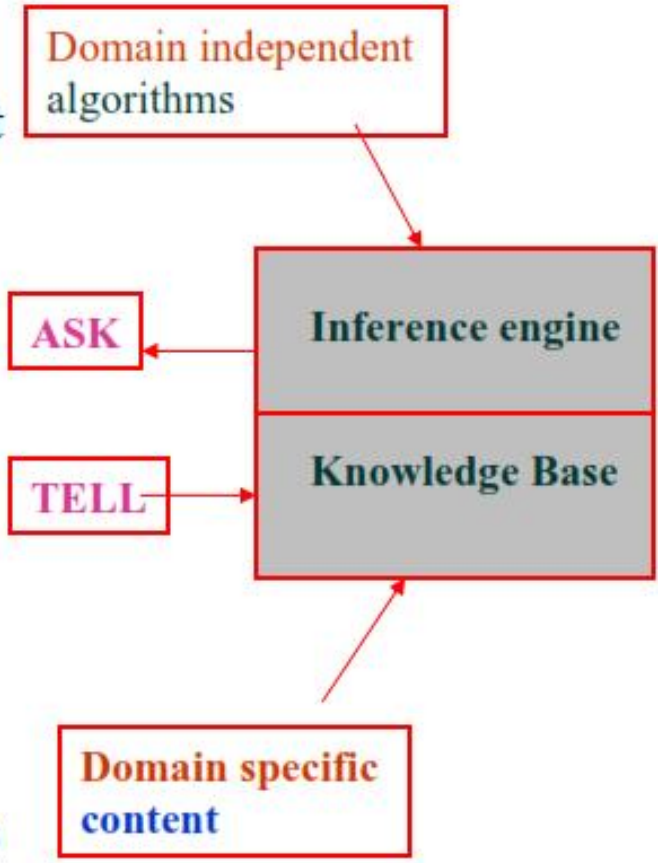
- Humans can know "things" and "reason"
  - Representation: How are the things stored?
  - Reasoning: How is the knowledge used?
    - To solve a problem…
    - To generate more knowledge…

- Knowledge and reasoning are important to artificial agents because they enable successful behaviors difficult to achieve otherwise
  - Useful in partially observable environments

- Can benefit from knowledge in very general forms, combining and recombining information

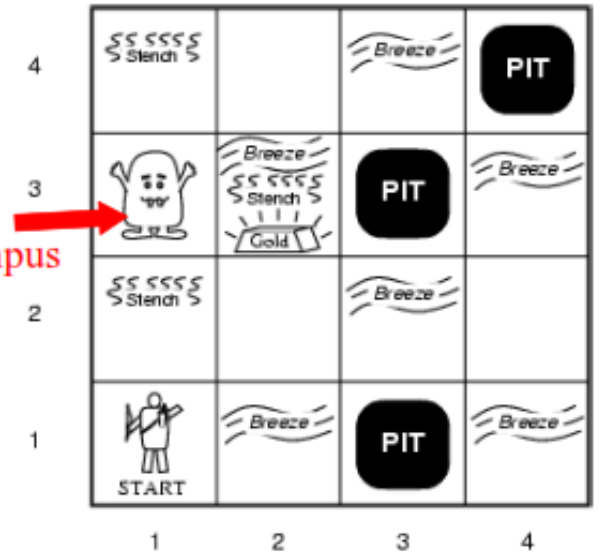
Knowledge Bases Agents (KBA)

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



The Wumpus World Game

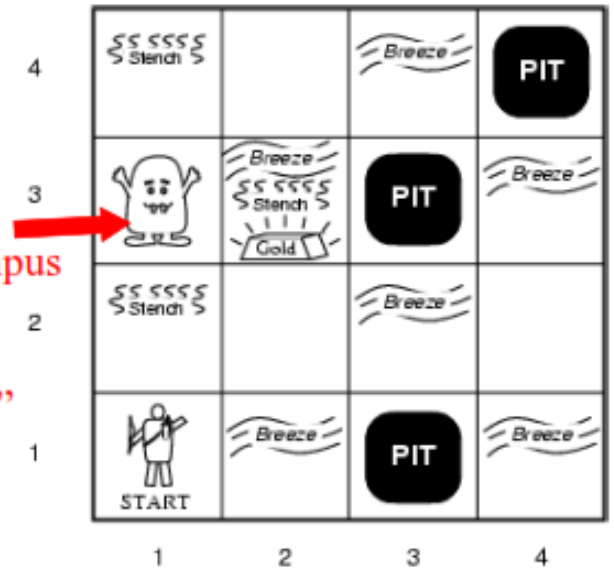
- The Wumpus "وحش" computer game
- The agent explores a cave "الكهفِ" consisting of rooms connected by passageways.
- The Wumpus is monitoring/lurking "التسنط" somewhere in the cave, it eats any agent that enters its room.
- Some rooms contain bottomless Pits "الحُفر" that trap any agent that wanders into the room.
- Occasionally, there is a heap of Gold in a room.
- The goal is to collect the gold and exit the world/cave without being eaten by the Wumpus.



- 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 PEAS Description

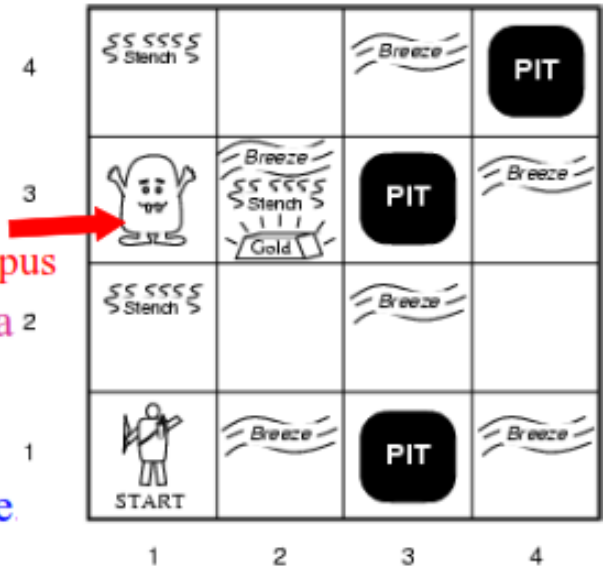
- Performance measure
  - +1000 for getting the gold,
  - -1000 for being dead,
  - -1 for each action taken,
  - -10 for using up the arrow
- Environment
  - Squares adjacent to wumpus are smelly "كريه الرائحة"
  - Squares adjacent to pit are breezy
  - Glitter iff gold is in the same square
  - Shooting kills wumpus if you are facing it. It screams
  - Shooting uses up the only arrow
  - Grabbing picks up gold if in same square
  - Releasing drops the gold in same square
  - You bump if you walk into a wall
- Sensors: Stench, Breeze, Glitter, Bump, Scream
- Actuators: Left turn, Right turn, Forward, Grab, Release, Shoot.
- Goal: bring back gold as quickly as possible



1. Stench الرائحة الكريهة
2. Breeze النسيم
3. Glitter الألق
4. Bump الصدمة
5. Scream الصيحة
6. Grabbing المَسْك

Wumpus World Characterization

- The agent has 5 sensors, each of which gives a single bit of information as follows:
  - In the square containing the wumpus and the directory (not diagonally) adjacent squares the agent will perceive a stench.
  - In the squares directly adjacent to a pit, the agent will perceive a breeze.
  - In the square where the gold is, the agent will perceive a glitter.
  - When the agent walks into a wall, it will perceive a bump.
  - When the wumpus is killed, it emits a woeful scream that can be perceived any where in the cave.
- The percepts will be given to the agent in the form of a list of five symbols [X1, X2, X3, X4, X5].
- For example if there is a stench and a breeze but no glitter, bump or scream, the agent will receive the precept [Stench, Breeze, None, None, None]



Wumpus Environment Characterization

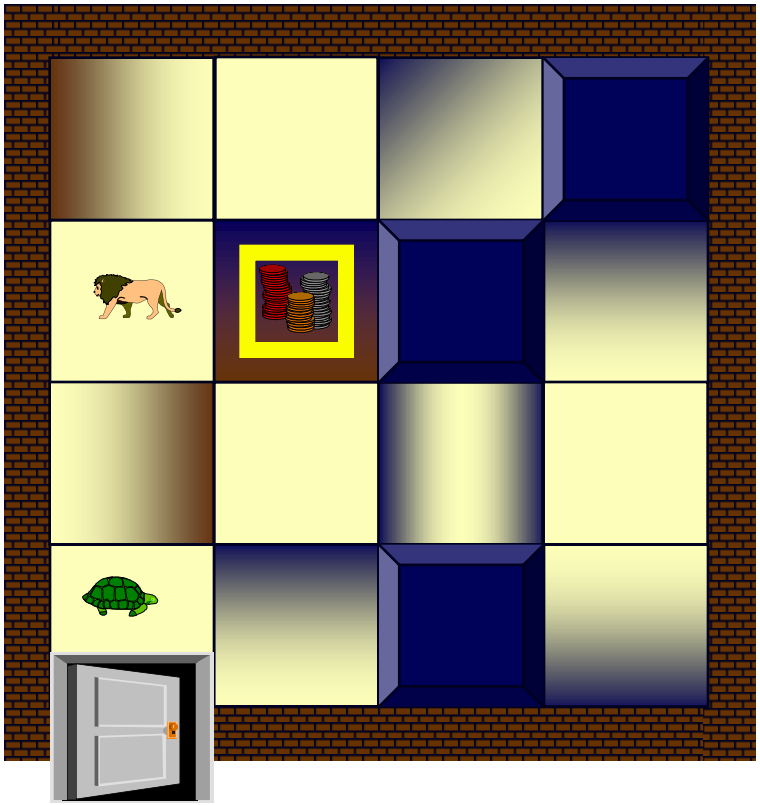
Fully Observable	No – only local perception
Deterministic	Yes – outcomes exactly specified
Static	Yes – Wumpus and Pits do not move
Discrete	Yes
Single-agent	Yes – Wumpus is essentially a natural feature

Life in the Wumpus World

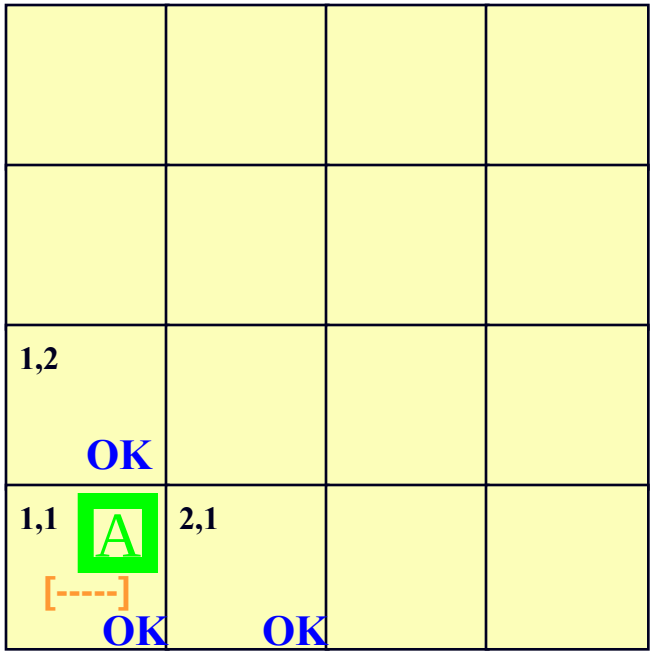
- before performing an action, it is advisable for the agent to "think" about it
  - perceive current state
  - avoid danger
    - wumpus, pits
  - seek rewards
    - gold
  - keep track of the environment
    - internal map, properties of squares
    - escape route

Wumpus World Exploration 1

World State



Agent's View



Position: [1,1]

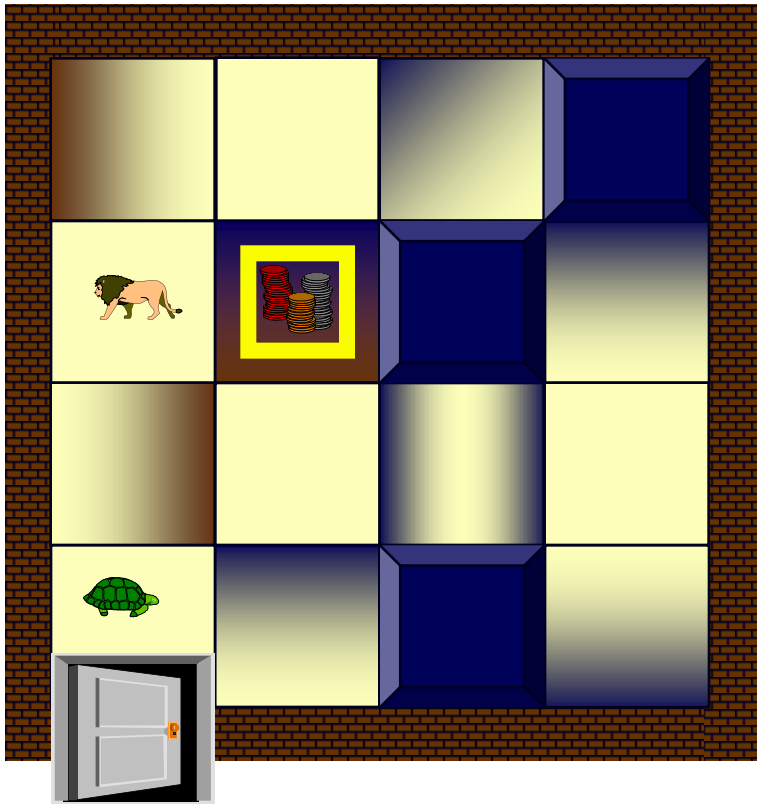
Percept:
[None, None, None, None, None]

Action: Turn right, forward

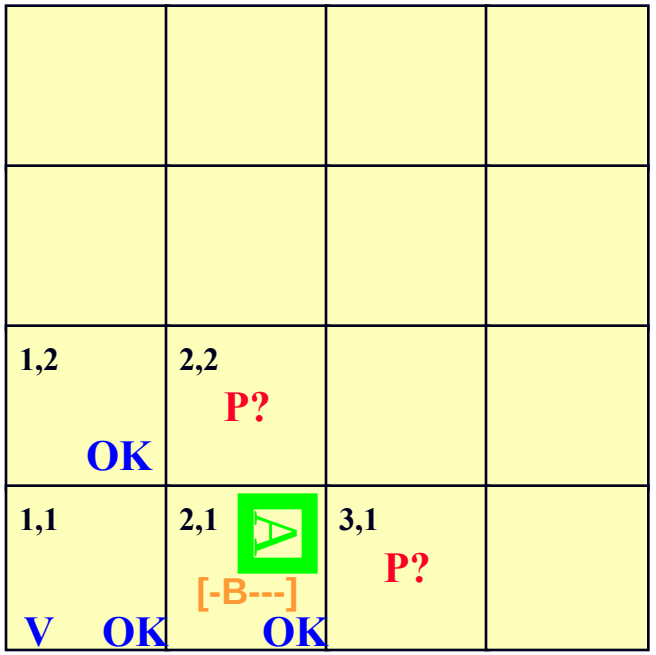
Inferences:current position is safe
adjacent positions are safe

Wumpus World Exploration 2

World State



Agent's View



Inferences: current position is safe adjacent positions may be pits because of a perceived breeze

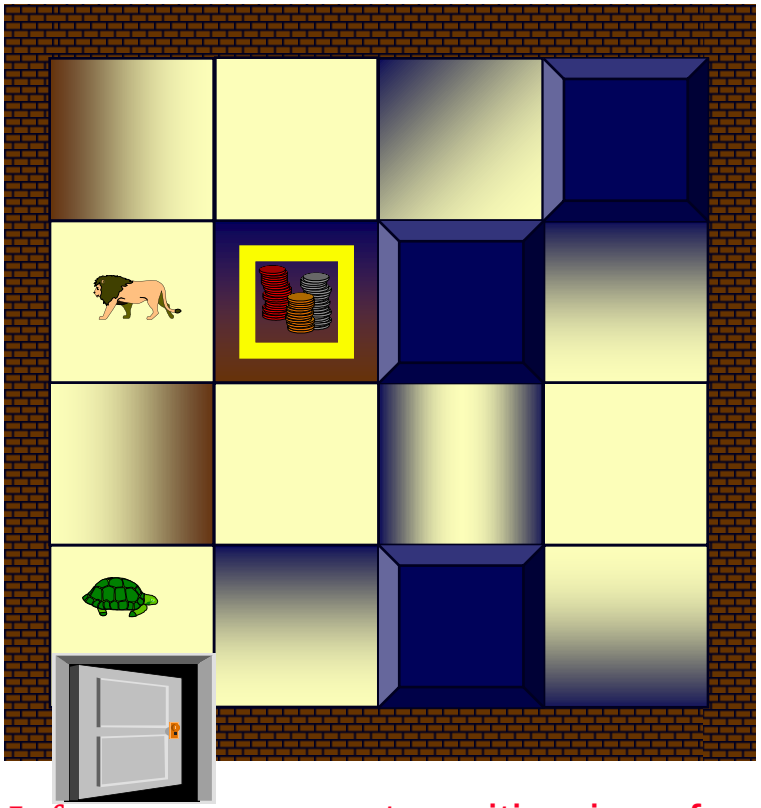
Position: [2,1]

Percept: [None, Breeze, None, None, None]

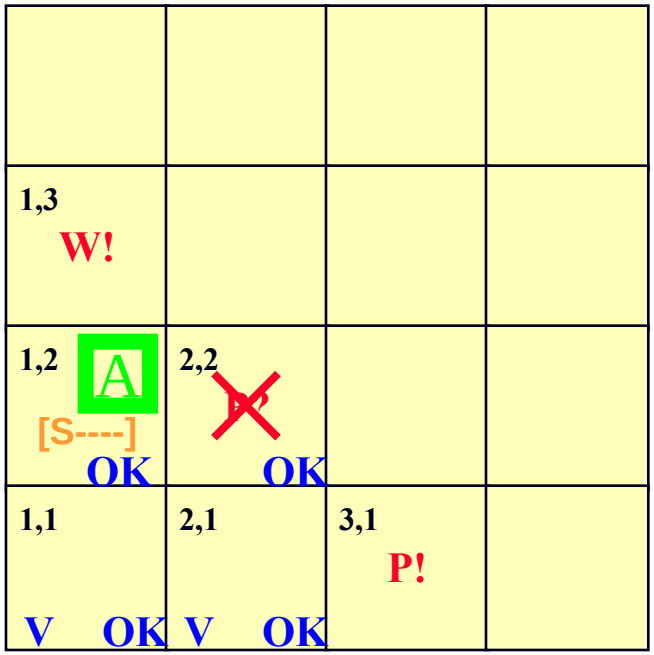
Action: Turn right, turn right, forward, turn right,forward

Wumpus World Exploration 3

World State



Agent's View

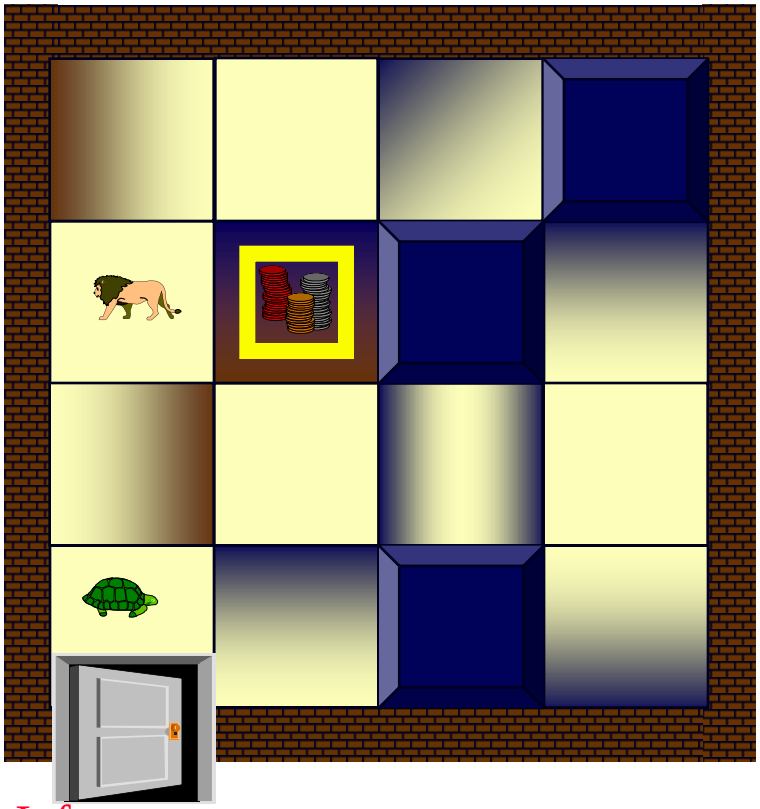


Inferences: current position is safe
[2,2] not a pit, no breeze;
hence [3,1] must be a pit
[1,3] wumpus because of stench

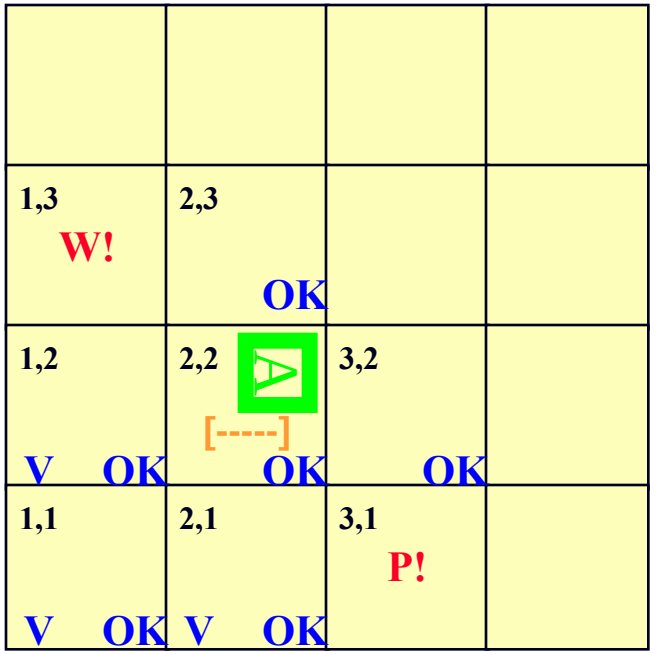
Position: [1,2]
Percept: [Stench, None, None, None, None]
Action: Turn right, forward

Wumpus World Exploration 4

World State



Agent's View



Position: [2,2]

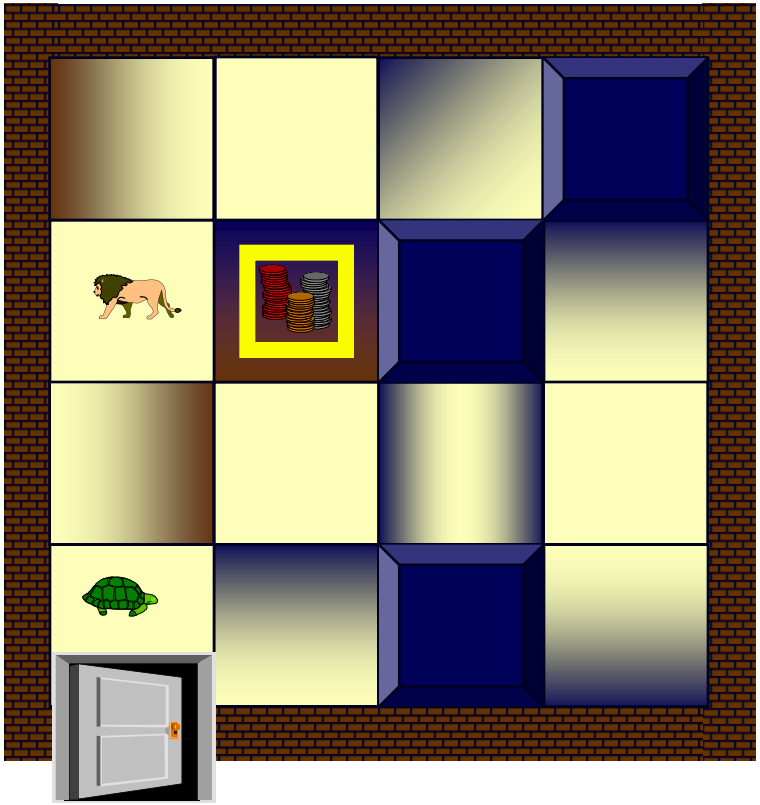
Percept:
[None, None, None, None, None]

Action: Turn right, forward

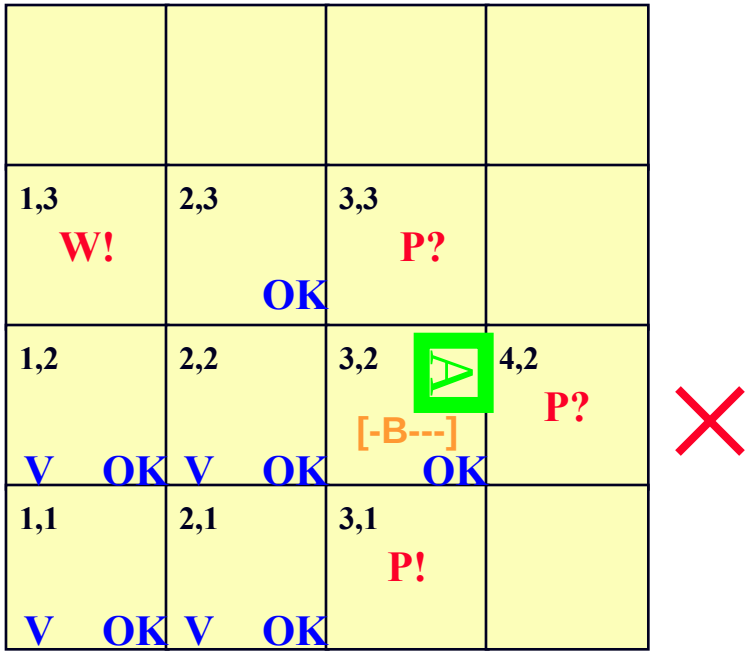
Inferences:
current position is safe
[2,2] not a pit, no breeze;
hence [3,1] must be a pit
[1,3] wumpus because of stench

Wumpus World Exploration 5

World State



Agent's View



Position: [3,2]

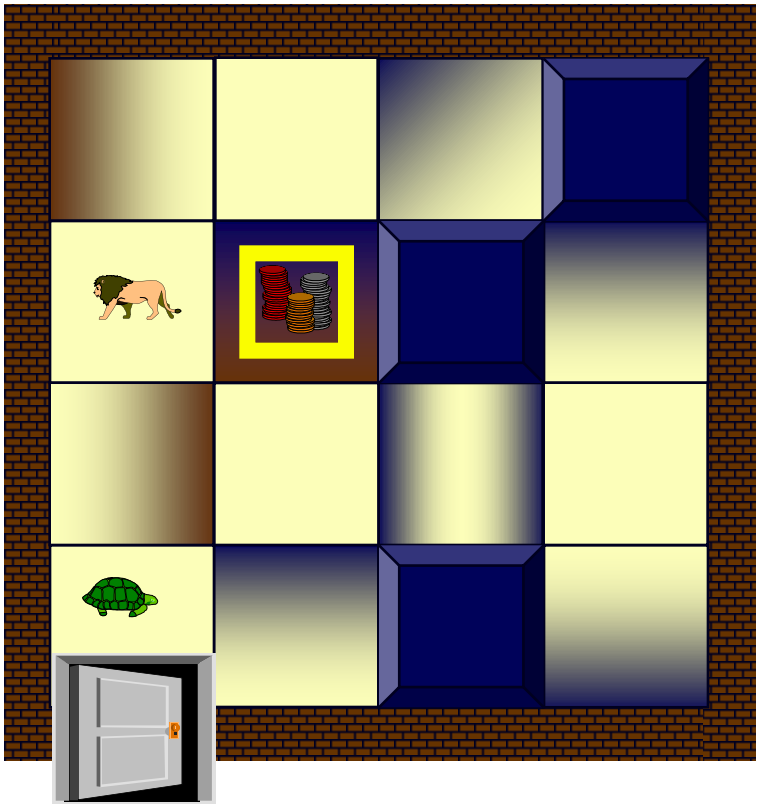
Percept:
[None, Breeze, None, None, None]

Action: Turn left, turn left, forward, turn right, forward

Inferences:
current position is safe
[3,3], [4,2] may be pits
because of breeze;

Wumpus World Exploration 6

World State



Agent's View

	2,4 P?		
1,3 W!	2,3 A [SBG--] OK	3,3 P?	
1,2 V OK	2,2 V OK	3,2 V OK	4,2 P?
1,1 V OK	2,1 V OK	3,1 P!	

Position: [3,2]

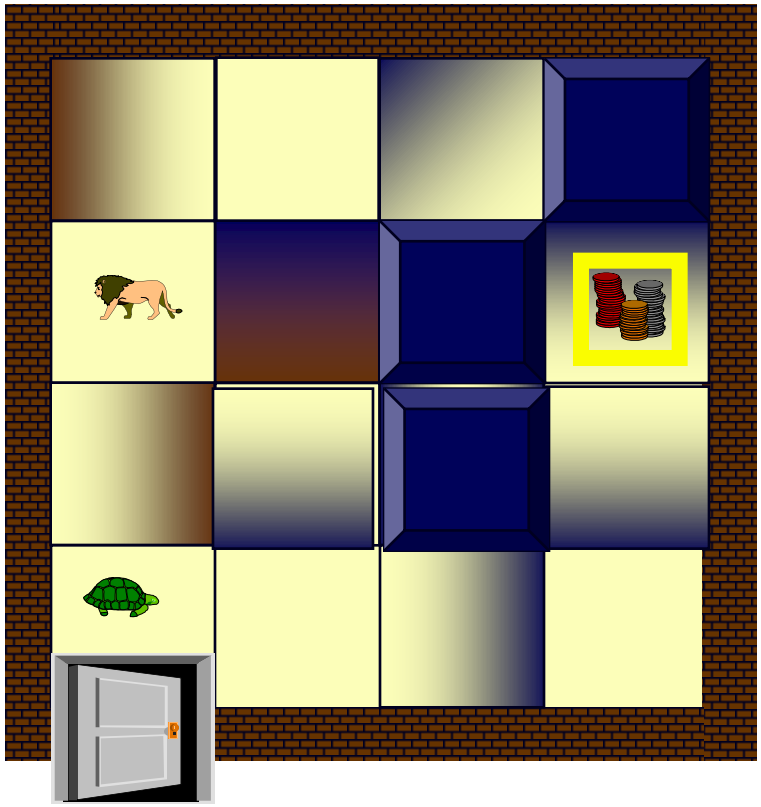
Percept:
[Stench, Breeze, Glitter, None, None]

Action: Grab gold, left, left, forward, right, forward, left, forward, climb out

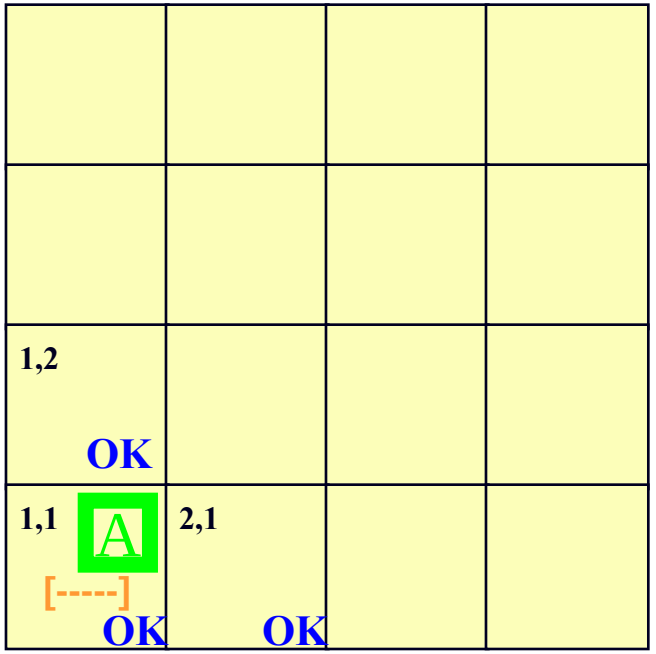
Inferences:
current position is safe
[2,4], [3,3] may be pits because of breeze;
[1,3] wumpus

Wumpus Example

World State



Agent's View



Position: [1,1]

Percept:
[None, None, None, None, None]

Action: Turn right, forward

Inferences:current position is safe
adjacent positions are safe

Wumpus World Observations

- many of the reasoning steps seem trivial to humans, but are not so trivial for computers
  - knowledge gained in different places at different times must be combined
  - absence of percepts is used to draw conclusions
    - sometimes the "closed-world assumption" is used: everything that is not explicitly stated is assumed to be false
    - not always realistic
- reasoning methods should be generalized
  - ad hoc representation and methods may be sufficient for one situation, but may have to be augmented for others
    - e.g grid-based world vs. graph-based world

Why Logic in the Wumpus World

- survival in the wumpus world requires advanced skills
  - explore the environment
  - remember information about the environment
  - connect different pieces of information
  - make decisions
  - evaluate risks
- most animals are not "smart" enough to do well in the wumpus world
- computers can perform the above activities
  - but some are difficult (the last three above)
  - an algorithmic solution may be possible, but not very flexible
  - logic provides a framework for knowledge representation and reasoning

Knowledge Representation

- One of the core problems in developing an agent is knowledge representation:
  - How do you represent knowledge?
  - How do you reason using that knowledge?
- A knowledge representation is a formal scheme that dictates how an agent is going to represent its knowledge in the knowledge base.
- **Representation = Syntax + Semantics + Reasoning**
  - Syntax: Rules that determine all possible sequences of symbols that constitute sentences of the language (grammar to form sentences, the alphabet of symbols and how they could be combined) in the language.
  - Semantics: Rules that determine a mapping from sentences in the representation to situations in the world.
- Programming language
  - Advantages: independent of context, effective procedure
  - Disadvantages: unable to describe problems with incomplete information (i.e., not expressive enough) E.g., There is a pit in [2,2] or [3,1]
- Natural language
  - Advantages: expressive enough
  - Disadvantages: suffer from ambiguity (e.g., Small dogs and cats)
- A good knowledge representation should combine both the advantages of natural languages and formal languages.

Logic

- Logics are formal languages for representing information such that conclusions can be drawn.
- Each sentence is defined by a **syntax** and a **semantic.**
- **Syntax** defines the sentences in the language. It specifies well formed sentences.
- **Semantics** define the ``meaning'' of sentences;
  i.e., in logic it defines the **truth of a sentence** in a **possible world.**

Logic

- For example, the language of arithmetic
  - $\mathbf{x + 2 \geq y}$ is a sentence.
  - $\mathbf{x + y >}$ is not a sentence.
  - $\mathbf{x + 2 \geq y}$ is true iff the number **x+2** is not less than the number **y**.
  - $\boldsymbol{x + 2 \geq y}$ is true in a world where $\boldsymbol{x = 7, y = 1.}$
  - $\boldsymbol{x + 2 \geq y}$ is false in a world where $\boldsymbol{x = 0, y = 6.}$

Semantics and Interpretations

- A sentence does not mean anything by itself.
  - How do we establish the correspondence between sentences and facts ?
    - The writer has to provide an interpretation for it.
- Interpretation: a way of matching objects in the world to symbols in the sentence.
- We want to be able to test sentences for truth
  - But truth depends both on interpretation of the sentence and on the actual state of the world.
  - A sentence may be true in one interpretation and false in another
    - E.g., x+y = 4 is true when x=2 & y=2, but is false when x=1 & y=1
  - e.g. 2*x < y means:
    - is true iff the number 2 times x is less than the number y
    - is true in a world where x=11, y=33
    - is false in a world where x=3, y=4

General Logic

- Logics are characterized by what they commit to as "primitives".

Logic	What Exists in World	Knowledge States
Propositional	**facts**	**true/false/unknown**
First-Order	facts, objects, relations	true/false/unknown
Temporal	facts, objects, relations, times	true/false/unknown
Probability Theory	facts	degree of belief 0..1
Fuzzy	degree of truth	degree of belief 0..1

Propositional Logic

Propositional Logic

- Propositions: assertions about an aspect of a world that can be assigned either a true or false value.
- a relatively simple framework for reasoning
- can be extended for more expressiveness at the cost of computational overhead
- important aspects
  - syntax
  - semantics
  - validity and inference
  - models
  - inference rules
  - complexity

Syntax

- symbols
  - logical constants True, False
  - propositional symbols P, Q, …
  - logical connectives
    - conjunction ∧, disjunction ∨,
    - negation ¬,
    - implication ⇒, equivalence ⇔
  - parentheses (,)
- sentences
  - constructed from simple sentences
  - conjunction, disjunction, implication, equivalence, negation

BNF Grammar Propositional Logic

Sentence → *AtomicSentence* | *ComplexSentence*

AtomicSentence → True | False | P | Q | R | …

ComplexSentence → (*Sentence*)

| *Sentence Connective Sentence*

| ¬ *Sentence*

Connective → ∧ | ∨ | ⇒ | ⇔

ambiguities are resolved through precedence ¬ ∧ ∨ ⇒ ⇔ or parentheses

e.g. ¬ *P* ∨ *Q* ∧ *R* ⇒ *S* is equivalent to (¬ *P*) ∨ (*Q* ∧ *R*)) ⇒ *S*

Logical Connectives

- $\neg S$ **Negation** (not)
- $S_1 \wedge S_2$ **Conjunction** (and)

  S_1 and S_2 are conjuncts
- $S_1 \vee S_2$ **Disjunction** (or)

  S_1 and S_2 are disjuncts
- $S_1 \Rightarrow S_2$ **Implication/conditional** (if-then)

  S_1 is the antecedent/premise

  S_2 is the consequent/conclusion
- $S_1 \Leftrightarrow S_2$ **Equivalence/bi-conditional** (if and only if)
- Ambiguities are resolved through Parentheses

  e.g. $\neg P \vee Q \wedge R \Rightarrow S$ is equivalent to $(\neg P) \vee (Q \wedge R)) \Rightarrow S$
- **Operator priority:** (highest) $\neg \wedge \vee \Rightarrow \Leftrightarrow$ (lowest)

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 True	False
True	True	False	True	True		True

Semantics

- **SEMANTIC:** It defines the rules for determining the truth of a sentence with respect to a particular model.

- **The question:**

How to compute the truth value of any sentence given a model?

Validity and Satisfiability

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

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

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is satisfiable if it is true in some model

e.g., $A \vee B$, C

A sentence is unsatisfiable if it is false in all 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

(there is no model for which KB=true and α is false)

Validity Example

- known facts about the Wumpus World
  - there is a wumpus in [1,3] or in [2,2]
  - there is no wumpus in [2,2]
- question (hypothesis)
  - is there a wumpus in [1,3]
- task
  - prove or disprove the validity of the question
- approach
  - construct a sentence that combines the above statements in an appropriate manner
    - so that it answers the questions
  - construct a truth table that shows if the sentence is valid
    - incremental approach with truth tables for sub-sentences

Validity Example

P	Q	$P \vee Q$
False	False	False
False	True	True
True	False	True
True	True	True

W_{13}
False
False
True
True

$\vee$

W_{22}
False
True
False
True

$W_{13} \vee W_{22}$
False
True
True
True

Interpretation:

W_{13} Wumpus in [1,3]

W_{22} Wumpus in [2,2]

Facts:

- there is a wumpus in [1,3] or in [2,2]

Validity Example

P	Q	$P \wedge Q$
False	False	False
False	True	False
True	False	False
True	True	True

$W_{13} \vee W_{22}$		$\neg W_{22}$
False		True
True	$\wedge$	False
True		True
True		False

Interpretation:

W_{13} Wumpus in [1,3]

W_{22} Wumpus in [2,2]

Facts:

- there is a wumpus in [1,3] or in [2,2]
- there is no wumpus in [2,2]

Validity Example

P	Q	$P \Rightarrow Q$
False	False	True
False	True	True
True	False	False
True	True	True

$W_{13} \vee W_{22}$		$\neg W_{22}$
False		True
True	$\wedge$	False
True		True
True		False

$(W_{13} \vee W_{22}) \wedge \neg W_{22}$		W_{13}
False		False
False	$\Rightarrow$	False
True		True
False		True

Question:

- can we conclude that the wumpus is in [1,3]?

Validity Example

$W_{13} \vee W_{22}$		$\neg W_{22}$
False		*True*
True	$\wedge$	*False*
True		*True*
True		*False*

$(W_{13} \vee W_{22}) \wedge \neg W_{22}$		W_{13}
False		*False*
False	$\Rightarrow$	*False*
True		*True*
False		*True*

$((W_{13} \vee W_{22}) \wedge \neg W_{22}) \Rightarrow W_{13}$
True
True
True
True

Valid Sentence:
For all possible combinations, the value of the sentence is true.

Question: Can we conclude that the wumpus is in [1,3]? **YES**

Entailment

- Entailment means that one thing follows from another:
- Entailment is used in propositional logic and predicate logic to describe a relationship between two sentences or sets of sentences.
- KB |= α

  A knowledge base KB entails sentence α if and only if α is true in all (models) where KB is true.
  - E.g., x+y = 4 entails 4 = x+y
- **For example:**
  - **KB:** "sky is blue" = true, "sun is shining" = true
  - **entails α:** "sky is blue and sun is shining" = true
  - α represents a true fact as long as facts represented in KB are true
  - If it is night then KB isn't the true state then α would not represent a true fact.
- **Entailment requires sentences in KB to be true.**

Entailment

- Let $\alpha = A \vee B$ and
  $KB = (A \vee C) \wedge (B \vee \neg C)$
- Is it the case that KB $\models \alpha$?
- Check all possible models -- α must be true whenever KB is true.

A	B	C	**KB** $(A \vee C) \wedge (B \vee \neg C)$	α $A \vee B$
False	False	False	False	False
False	False	True	False	False
False	True	False	False	True
False	True	True	True	True
True	False	False	True	True
True	False	True	False	True
True	True	False	True	True
True	True	True	True	True

Entailment

A	B	C	**KB** $(A \vee C) \wedge (B \vee \neg C)$	**α** $A \vee B$
False	False	False	False	False
False	False	True	False	False
False	True	False	False	True
False	True	True	True	True
True	False	False	True	True
True	False	True	False	True
True	True	False	True	True
True	True	True	True	True

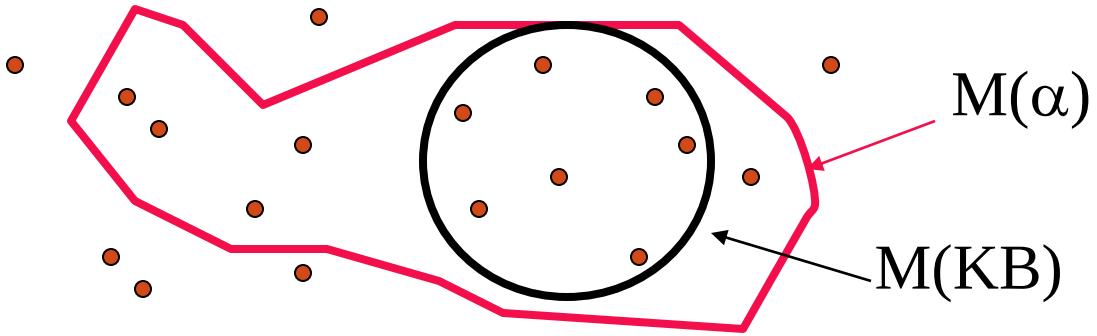
Entailment

A	B	C	**KB** $(A \vee C) \wedge (B \vee \neg C)$	α $A \vee B$
False	False	False	False	False
False	False	True	False	False
False	True	False	False	True
False	True	True	True	True
True	False	False	True	True
True	False	True	False	True
True	True	False	True	True
True	True	True	True	True

$$KB \models \alpha$$

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



logical Inference

- Formal Definition of Inference (e.g., A |– B);
  - It can be read as "B can be proven/inferred from A".
  - We say that the set A of sentences logically imply (or infer) the set B of sentences, if one can derive all sentences in B by assuming all sentences in A and applying a finite sequence of inference rules to them (for example, those from propositional logic)
  - In cases where multiple logics are under discussion, it may be useful to put a subscript on the symbol. (e.g., A $|-_i$ B)
- Inference procedure: KB $|-_i \alpha$
  - Sentence α can be derived from KB using inference procedure i

Inference Methods

- **Model Checking:**
  1. **Truth-Table Enumeration**
     - **Sound** and **complete** for propositional logic

- **Inference Rules**

  (Application of Syntactic Operations):
  - **Sound** generation of new sentences from old
  - Could use inference rules as operators for search more in FOL.

Inference by Truth-Table Enumeration

LET: $KB = A \vee C, B \vee \neg C$ $\quad \alpha = A \vee B$

Does: $KB \models \alpha$?

A	B	C
false	false	false
false	false	true
false	true	false
false	true	true
true	false	false
true	false	true
true	true	false
true	true	true

RECALL: The computer doesn't know the meaning of the proposition symbols (and neither do we at this point).

So all logically distinct cases must be checked to prove that a sentence can be derived from a KB.

Inference by Truth-Table Enumeration

LET: $KB = A \vee C, B \vee \neg C$ $\quad \alpha = A \vee B$

Does: $KB \models \alpha$?

A	B	C	A ∨ C	B ∨ ¬ C	KB
false	false	false	false	true	false
false	false	true	true	false	false
false	true	false	false	true	false
false	true	true	true	true	true
true	false	false	true	true	true
true	false	true	true	false	false
true	true	false	true	true	true
true	true	true	true	true	true

Rows where all of sentences in KB are true are the models of KB

Inference by Truth-Table Enumeration

LET: $KB = A \vee C, B \vee \neg C$ $\quad \alpha = A \vee B$

Does: $KB \models \alpha$? *YES!*

A	B	C	$A \vee C$	$B \vee \neg C$	KB	$A \vee B$	$KB \Rightarrow \alpha$
false	false	false	false	true	false	false	true
false	false	true	true	false	false	false	true
false	true	false	false	true	false	true	true
false	true	true	true	true	true →	true	true
true	false	false	true	true	true →	true	true
true	false	true	true	false	false	true	true
true	true	false	true	true	true →	true	true
true	true	true	true	true	true →	true	true

In other words KB $\models \alpha$ is valid.

"α is entailed by KB, if all models of KB are models of α"
i.e. all rows where **KB** is true, α is true

Inference by Truth-Table Enumeration

☞ **Why is this technique impractical?**

- The proofs using truth-table enumeration grow exponentially in length as the number of symbols increases.

  $O(2^n)$ where n is the number of symbols

- There must be a better way.

- **Natural deduction** is an inference procedure that uses sound inference rules to derive new sentences from the KB and any previously derived sentences until the conclusion sentence is derived.

Inference and Derivation

- inference rules allow the construction of new sentences from existing sentences
  - notation: a sentence β can be derived from α

$$\alpha \mid\!- \beta \quad \text{or} \quad \frac{\alpha}{\beta}$$

- an **inference procedure** generates new sentences on the basis of inference rules
- **Soundness**: An inference procedure is sound if every sentence X it produces from a set of sentences S logically follows from S. (No contradiction is created).

$$\text{if } S \mid\!- X \text{ then } S \mid\!= X$$

- **Completeness**: A inference procedure is complete, if it is able to produce every sentence that logically follows from any give S.

$$\text{if } S \mid\!= X \text{ then } S \mid\!- X$$

Inference Rules

- **modus ponens**
  - from an implication and its premise one can infer the conclusion
- **and-elimination**
  - from a conjunct, one can infer any of the conjuncts
- **and-introduction**
  - from a list of sentences, one can infer their conjunction
- **or-introduction**
  - from a sentence, one can infer its disjunction with anything else

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

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

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

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

Inference Rules

- **double-negation elimination**
  - a double negations infers the positive sentence

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

- **unit resolution**
  - if one of the disjuncts in a disjunction is false, then the other one must be true

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

- **resolution**
  - β cannot be true and false, so one of the other disjuncts must be true
  - can also be restated as "implication is transitive"

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

$$\frac{\neg\alpha \Rightarrow \beta, \ \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Logical Equivalence

- These equivalence rules will be used to convert into Conjunctive Normal Form **(CNF)** while proofing without using truth tables; Resolution.

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$
$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$
$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$
$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$
$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}$$
$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{de Morgan}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{de Morgan}$$
$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$
$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

Proofing

- A proof is a sequence of sentences, where each sentence is either a premise or a sentence derived from earlier sentences in the proof by one of the rules of inference.
- The last sentence is one that we want to prove. (also called goal or query)
- Example for the "weather problem":

1.	Hu	Premise	"It is humid"
2.	Hu→Ho	Premise	"If it is humid, it is hot"
3.	Ho	Modus Ponens(1,2)	"It is hot"
4.	(Ho∧Hu)→R	Premise	"If it's hot & humid, it's raining"
5.	Ho∧Hu	And Introduction(1,3)	"It is hot and humid"
6.	R	Modus Ponens(4,5)	"It is raining"

Wumpus Logic

- An agent can use propositional logic to reason about the Wumpus world. KB contains
  - Percepts
  - Rules

- Some atomic propositions:
  S12 = There is a stench in cell (1,2)
  B34 = There is a breeze in cell (3,4)
  W22 = The Wumpus is in cell (2,2)
  V11 = We have visited cell (1,1)
  OK11 = Cell (1,1) is safe.
  etc
- Some rules:
  (**R1**) $\neg S11 \rightarrow \neg W11 \wedge \neg W12 \wedge \neg W21$
  (**R2**) $\neg S21 \rightarrow \neg W11 \wedge \neg W21 \wedge \neg W22 \wedge \neg W31$
  (**R3**) $\neg S12 \rightarrow \neg W11 \wedge \neg W12 \wedge \neg W22 \wedge \neg W13$
  (**R4**) $S12 \rightarrow W13 \vee W12 \vee W22 \vee W11$
  etc
- We may give similar rules for other cells

Finding the Wumpus

- two options
  - construct truth table to show that $W_{1,3}$ is a valid sentence
    - rather tedious
  - use inference rules
    - apply some inference rules to sentences already in the knowledge base

Proofing W13

- Apply MP with ¬S11 and R1:
  ¬ W11 ∧ ¬ W12 ∧ ¬ W21
- Apply And-Elimination to this, yielding 3 sentences:
  ¬ W11, ¬ W12, ¬ W21
- Apply MP to ~S21 and R2, then apply And-elimination:
  ¬ W22, ¬ W21, ¬ W31
- Apply MP to S12 and R4 to obtain:
  W13 ∨ W12 ∨ W22 ∨ W11
- Apply Unit resolution on (W13 ∨ W12 ∨ W22 ∨ W11) and ¬W11:
  W13 ∨ W12 ∨ W22
- Apply Unit Resolution with (W13 ∨ W12 ∨ W22) and ¬W22:
  W13 ∨ W12
- Apply UR with (W13 ∨ W12) and ¬W12:
  W13

Action in the Wumpus World

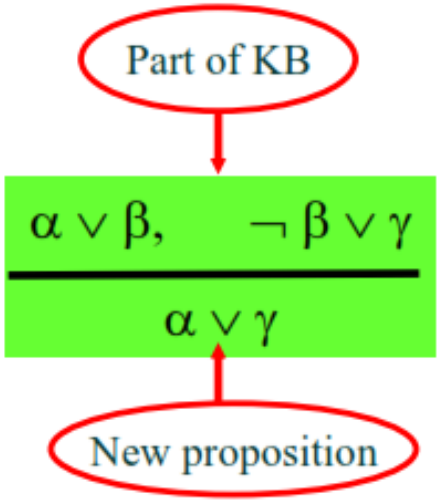
- The agent also needs to Ask the knowledge base what to do
  - Must ask specific questions
    - Can I go forward?
  - General questions are not possible in propositional logic
    - Where should I go?
- Additional rules are required to determine actions for the agent

RM: $\mathbf{A}_{1,1} \wedge \mathbf{East}_A \wedge W_{2,1} \Rightarrow \neg\, Forward_A$
RM + 1: . . .
. . .

- The size of the knowledge base even for a small wumpus world becomes immense
  - Explicit statements about the state of each square
  - Additional statements for actions, time
  - Easily reaches thousands of sentences
- Completely unmanageable for humans

Inference by: Resolution

- The resolution process is a simple iterative process:
  - At each step, two clauses are compared (resolved), yielding a new clause that has been inferred from them.
    - Resolution operates by taking two clauses and producing a new clause containing all the literals of the two originals except the two complementary literals (i.e., one in positive form and the other in negative form).
- **Steps of inference by Resolution**
- Represent your knowledge base in CNF.
- Find two clauses that contain complementary literals
  - Delete both literals
  - Delete multiple copies of remaining literals
  - Produce a new clause with the remaining ones
  - Iterate the process



$$\frac{\neg \alpha \Rightarrow \beta, \ \beta \Rightarrow \gamma}{\neg \alpha \Rightarrow \gamma}$$

Inference by: Resolution

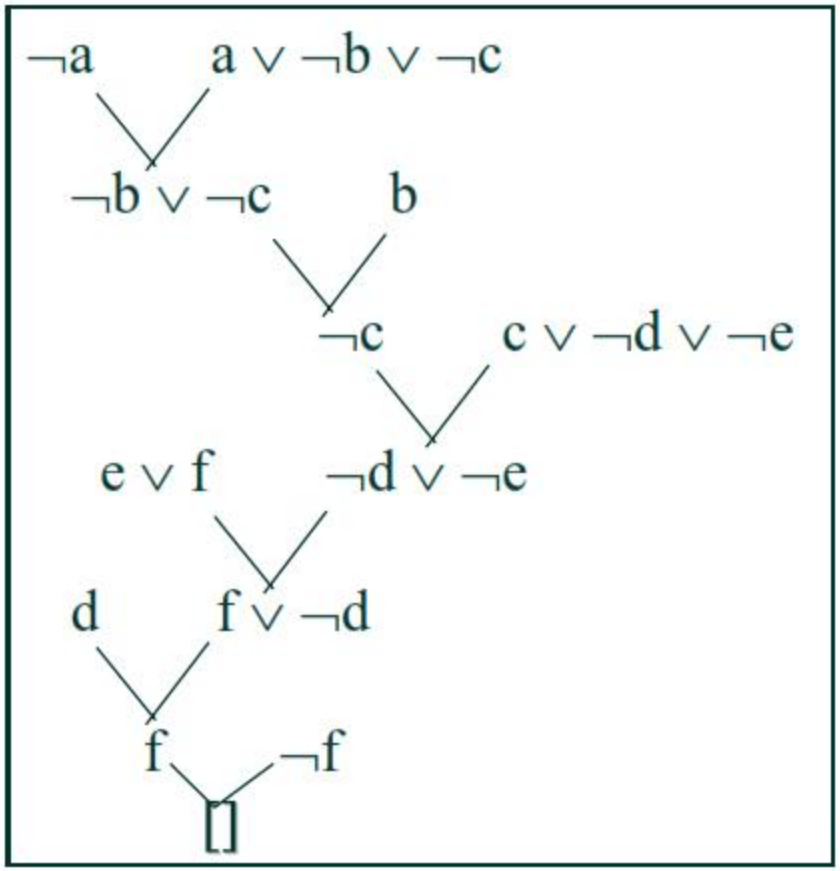
- **Binary Resolution Step**
  - For any two clauses C_1 and C_2, if there is a literal L_1 in C_1 that is complementary to a literal L_2 in C_2, then delete L_1 and L_2 from C_1 and C_2 respectively, and construct the disjunction of the remaining clauses.
  - The constructed clause is a resolvent of C_1 and C_2.
- **Examples of Resolution Step**
  - $C_1 = a \vee \neg b$, $C_2 = b \vee c$
    - Complementary literals : $\neg b$, b
    - **Resolvent**: $a \vee c$
  - $C_1 = \neg a \vee b \vee c$, $C_2 = \neg b \vee d$
    - Complementary literals : b, $\neg b$
    - **Resolvent** : $\neg a \vee c \vee d$

Resolution Procedure

- To prove P
  - Assume ~P
  - Add ~P to KB
  - Resolve in KB till resulting:
    1. In KB (therefore P is false)
    2. Empty (therefore ~P is contradictory so P is true)

Resolution Procedure

- Proof by resolution



- **Theorem proving as search**
  - Start node: the set of given premises/axioms (KB + Input)
  - Operator: inference rule (add a new sentence into parent node)
  - Goal: a state that contains the theorem asked to prove
  - Solution: a path from start node to a goal

Conjunctive Normal Form

- **Disjunctive Normal Form (DNF)**
  - Any sentence can be written as a disjunction of conjunctions of literals.
  - Example: A^B v C^D v P^Q^R;
  - Widely used in logical circuit design (simplification)
- **Conjunctive Normal Form (CNF)**
  - Any sentence can be written as a conjunction of disjunctions of literals.
  - Example: (A v B) ^ (C v D) ^ (P v Q v R);
- **Normal forms can be obtained by applying equivalence laws**
- A formula in conjunctive normal form is unsatisfiable if for every interpretation I, there is a clause C that is false in I.
- A formula in CNF is satisfiable if there is an interpretation I that makes all clauses true.

Resolution Example:

1. Convert 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 $\neg\alpha \vee \beta$.**
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

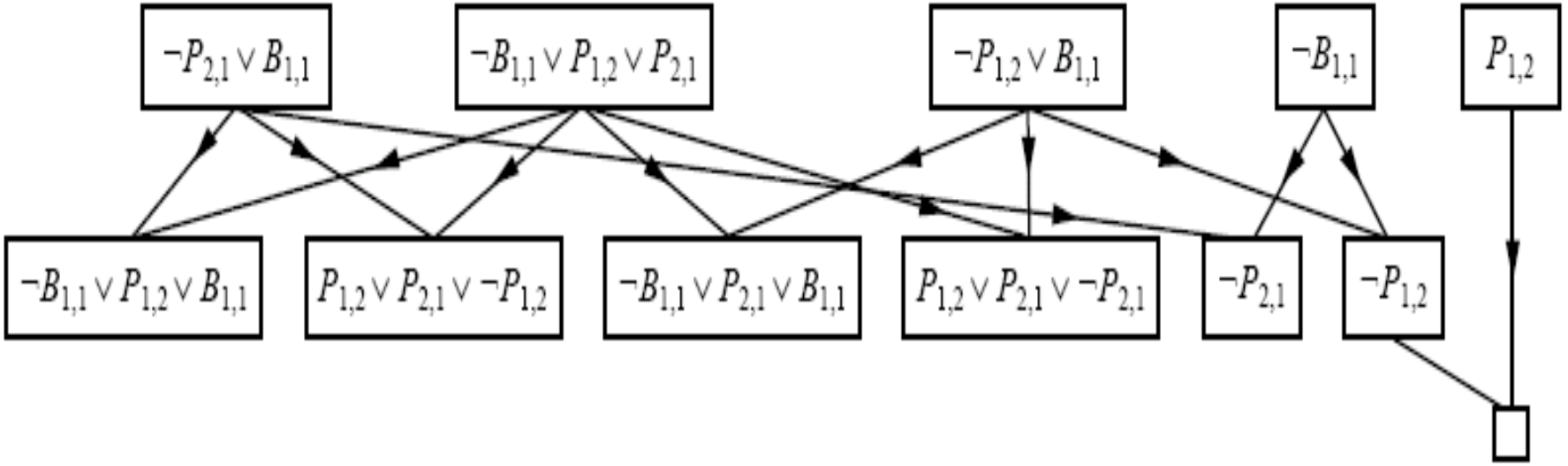
3. **Move $\neg$ inwards using de Morgan's rules and double-negation:**
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

4. **Apply distributive law ($\wedge$ over $\vee$) and flatten:**
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

$$\neg(\alpha \vee \beta) = \neg\alpha \wedge \neg\beta$$

Resolution Example:

2. Resolution Procedure



Horn Clauses

- Resolution can be exponential in space and time unless we have Horn clauses.
- Linear time algorithms exist when knowledge bases are restricted to Horn clauses
- IF we can reduce all clauses to "Horn clauses" THEN resolution will be linear in space and time.
- Horn clauses: A disjunction of literals of which at most one is positive

A clause with at most 1 positive literal.

e.g. $A \vee \neg B \vee \neg C$

- Every Horn clause can be rewritten as an implication with a conjunction of positive literals in the premises and a single positive literal as a conclusion.

e.g. $B \wedge C \Rightarrow A$ $\quad (\neg A \vee \neg B) \equiv (A \wedge B \Rightarrow False)$

- Forward Chaining and Backward chaining are sound and complete with Horn clauses and run linear in space and time.

Inference Methods

- Forward Chaining
  - Data driven
  - Proofs start with the given axioms/premises in KB, deriving new sentences using GMP until the goal/query sentence is derived
  - It moves "forward" from the KB to the goal
  - Inference using GMP is complete for KBs containing only Horn clauses

- Backward Chaining
  - Goal-directed: we have a goal to prove.
  - We first start with the goal,
  - Find implication sentences that enable us to conclude it,
  - Finally try to establish their premises; use backward chaining on premises with unknown values.

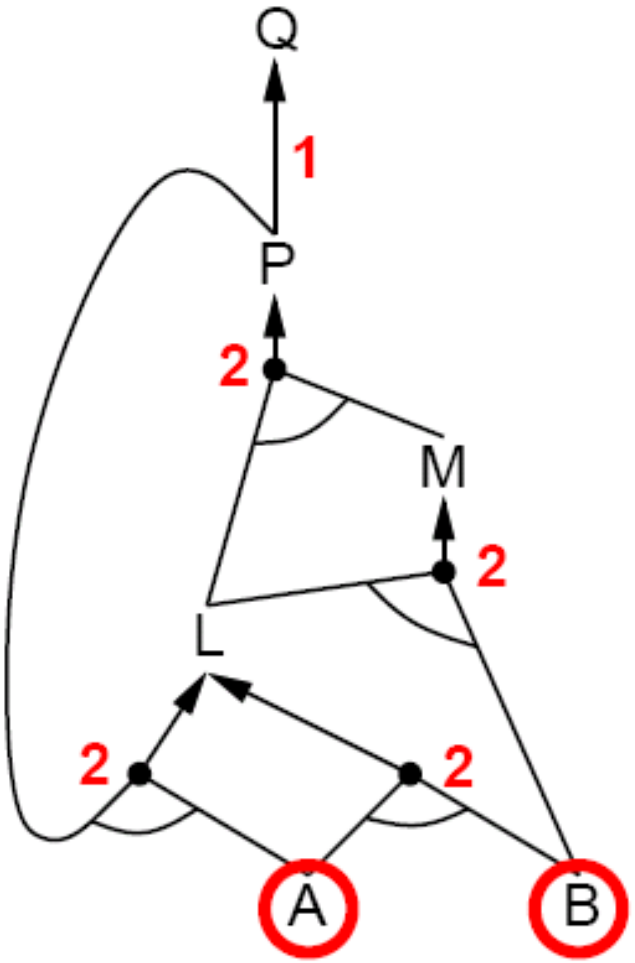
Forward Chaining

- Starting from the current state, matching the premises of the rules (the IF parts), and performing the corresponding actions (the then parts) that usually update the knowledge base or working memory.
- The process continues until no more rules can be applied, or some cycle limit is met.
- Forward Chaining Algorithm:

  KB = conjunction of Horn clauses

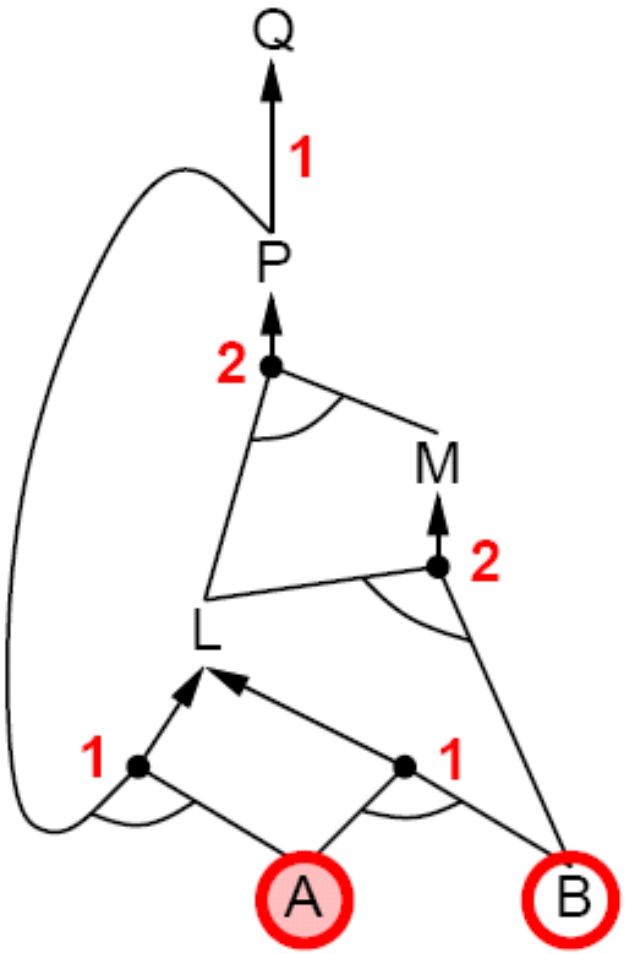
  - Horn clause = proposition symbol or (conjunction of symbols) $\Rightarrow$ symbol
  - E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$
  - When a new sentence α is added into the knowledge base KB
  - Look for all sentences share literals with α
  - Perform resolution
  - Add new sentence to KB and continue
- Forward chaining is data-driven
- In Forwarding chaining: New facts are inferred as soon as possible

Forward Chaining Example



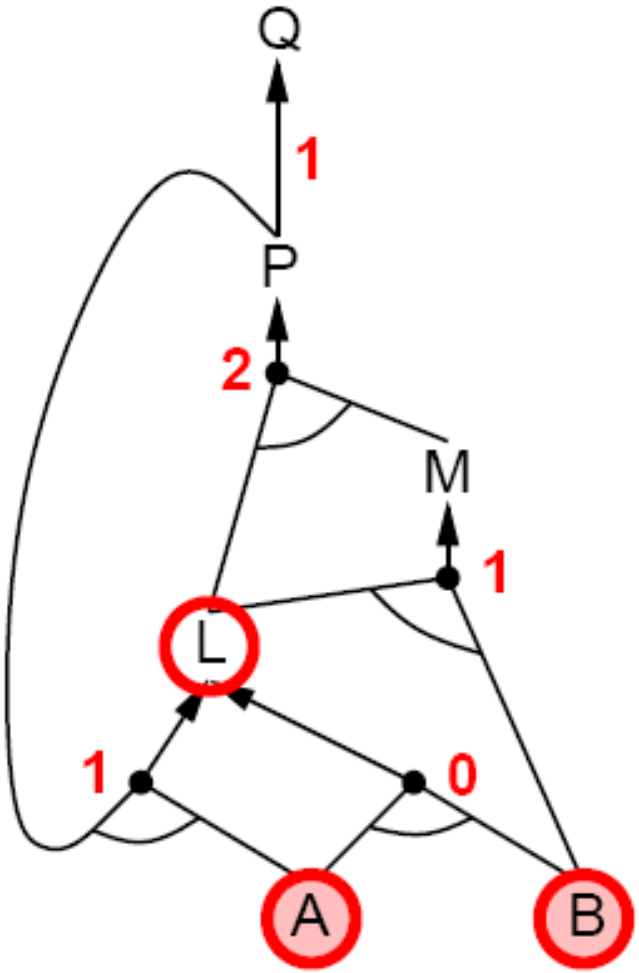
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Forward Chaining Example



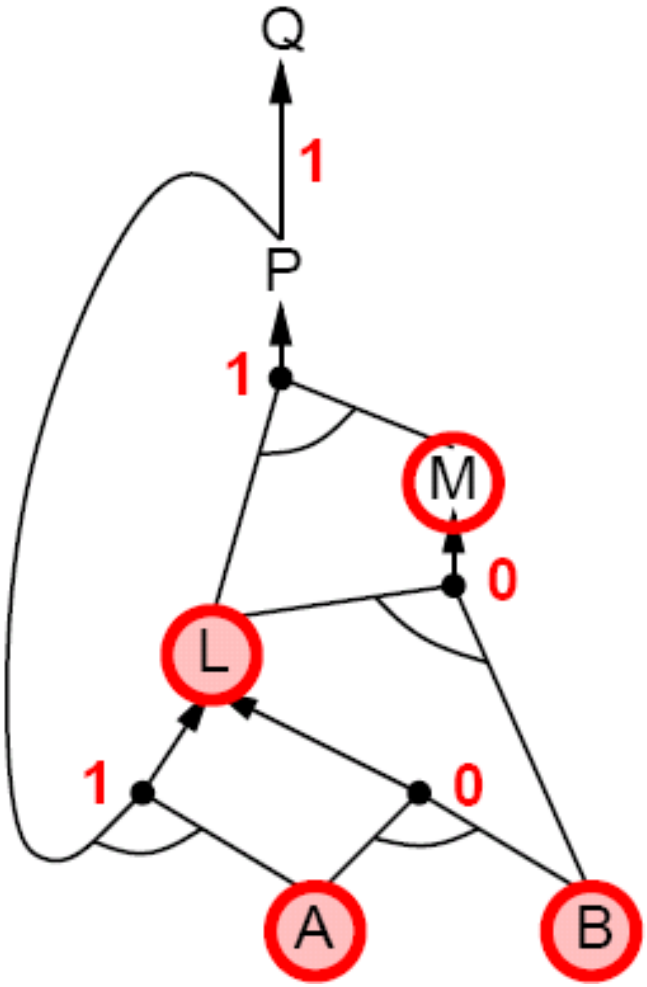
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Forward Chaining Example



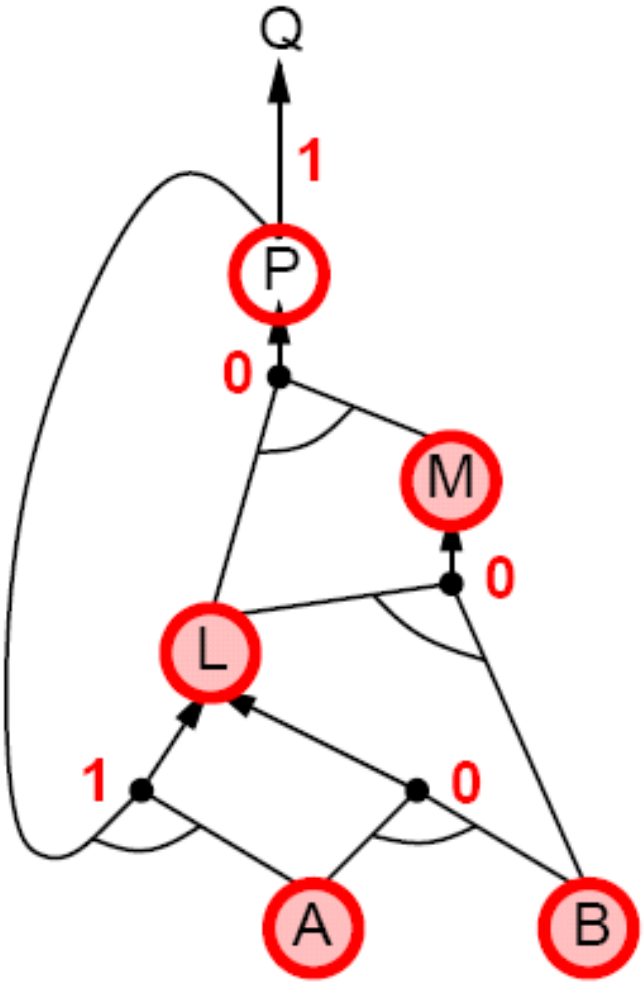
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Forward Chaining Example



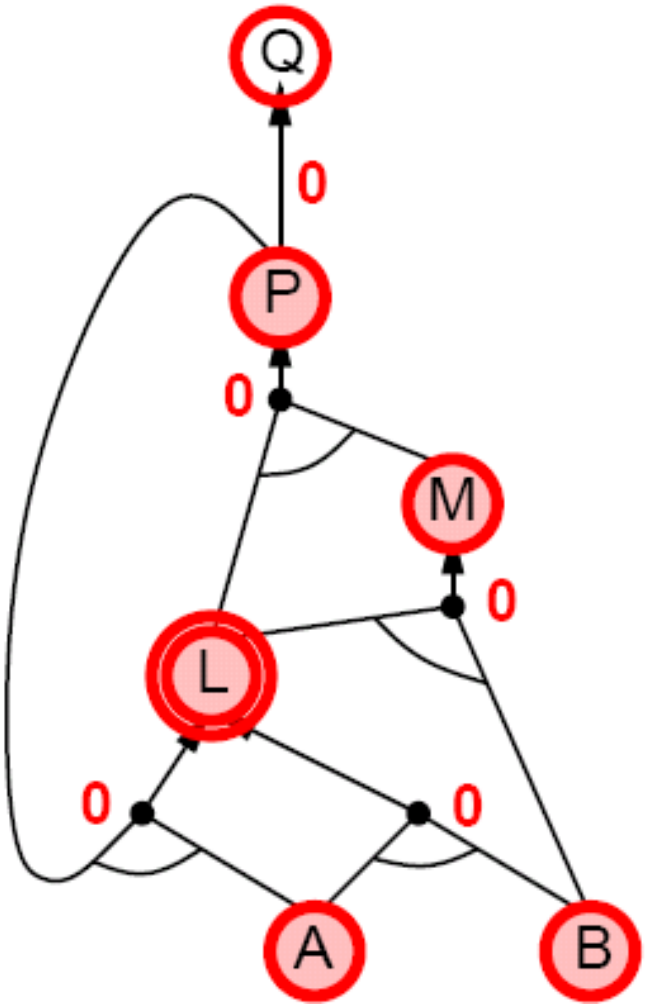
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Forward Chaining Example



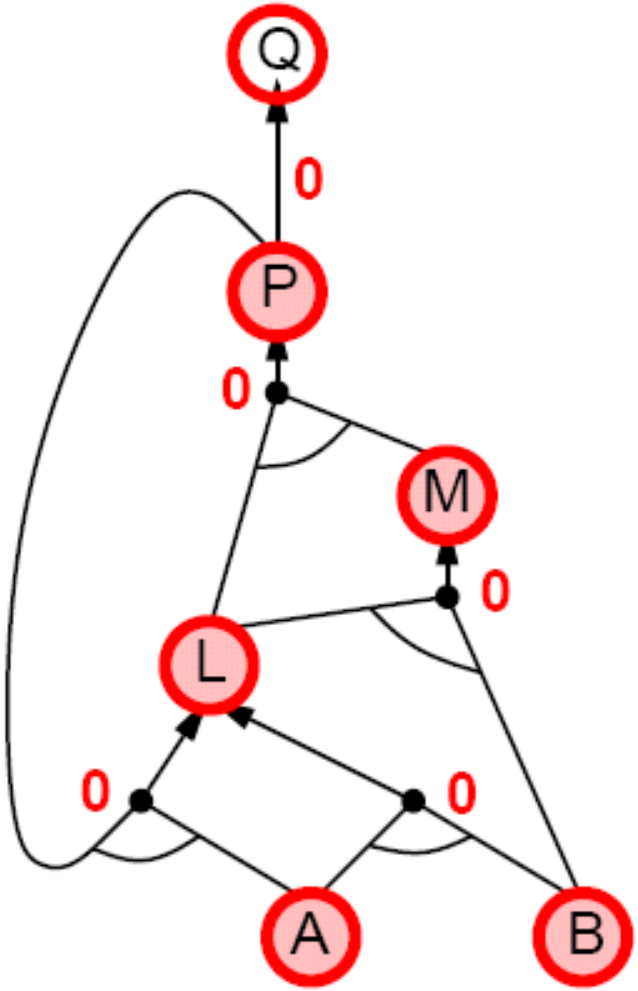
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Forward Chaining Example



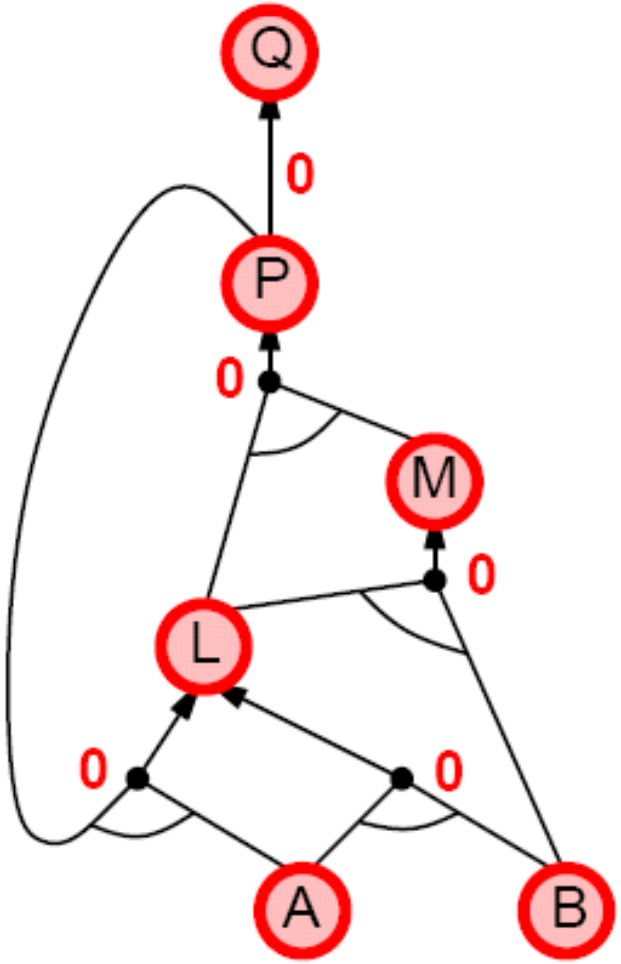
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Forward Chaining Example



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

Forward Chaining Example



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

Backward Chaining

- Backward chaining or goal-driven inference works towards a final state by looking at the working memory to see if the sub-goal states already exist there.
- If not, the actions (the THEN parts) of the rules that will establish the sub-goals are identified and new sub-goals are set up for achieving the premises of those rules (the IF parts).

Backward Chaining Algorithm

- Establish a goal, determine what must be found out in order to reach that goal, and try to find that knowledge

Idea: work backwards from the query *q*

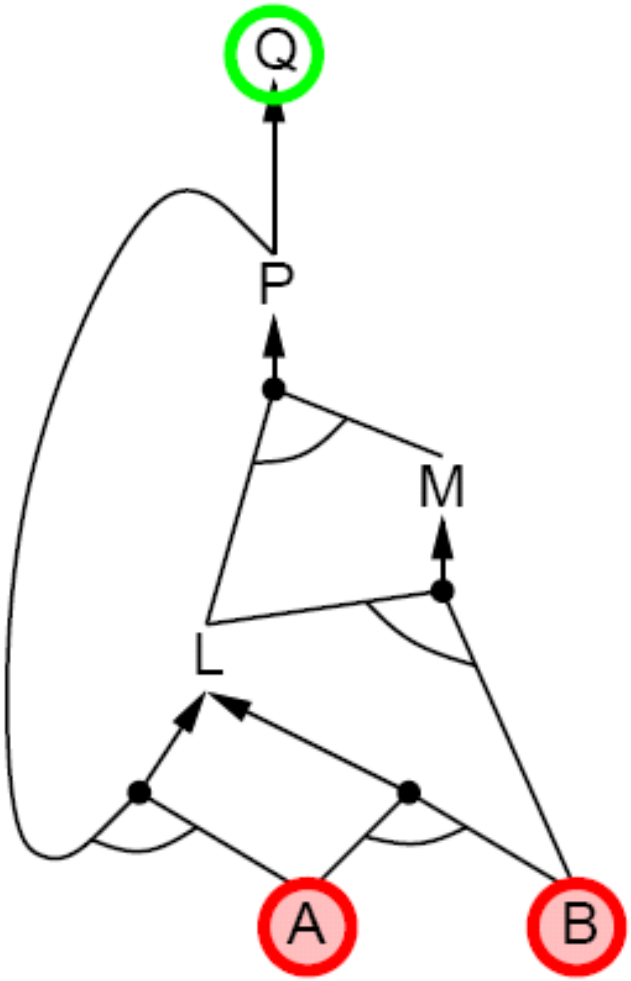
- Check if *q* is known already, or
- Prove by BC all premises of some rule concluding *q*
- Hence BC maintains a stack of sub-goals that need to be proved to get to q.

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

Avoid repeated work: check if new sub-goal:

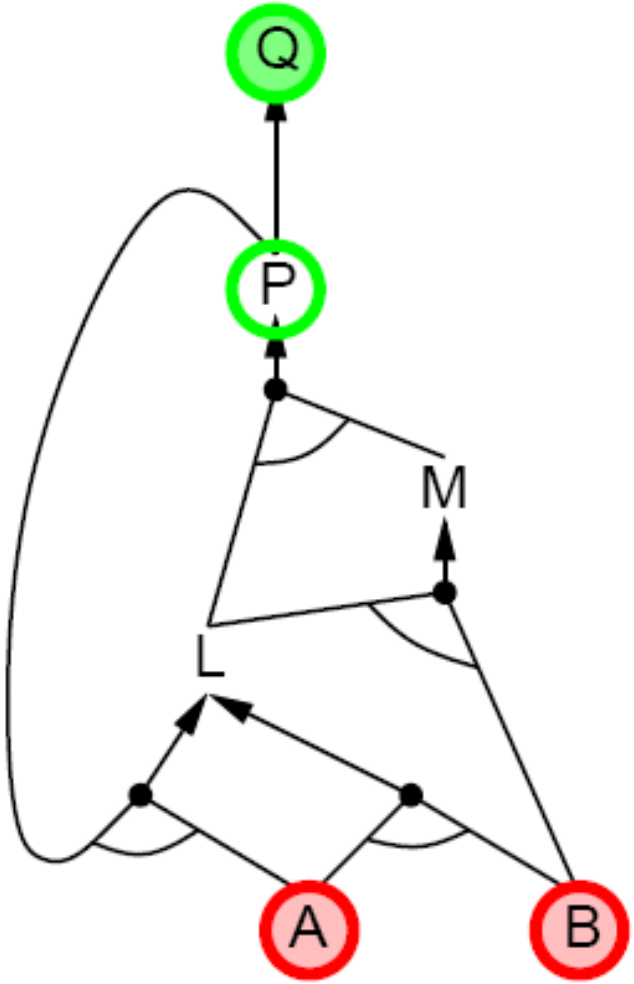
1. Has already been proved true, or
2. Has already failed

Backward Chaining Example



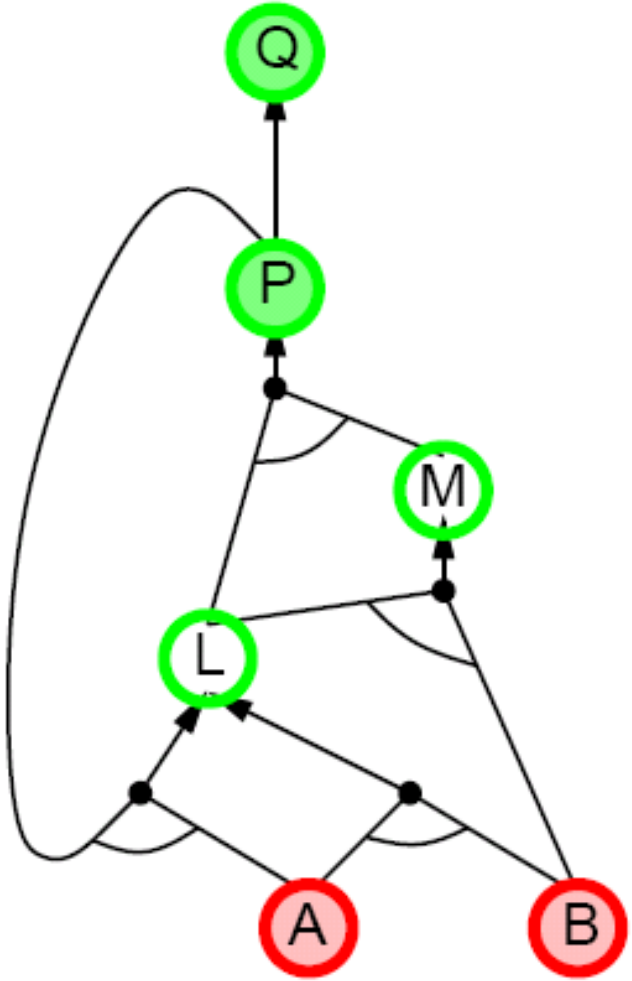
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Backward Chaining Example



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

Backward Chaining Example



$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

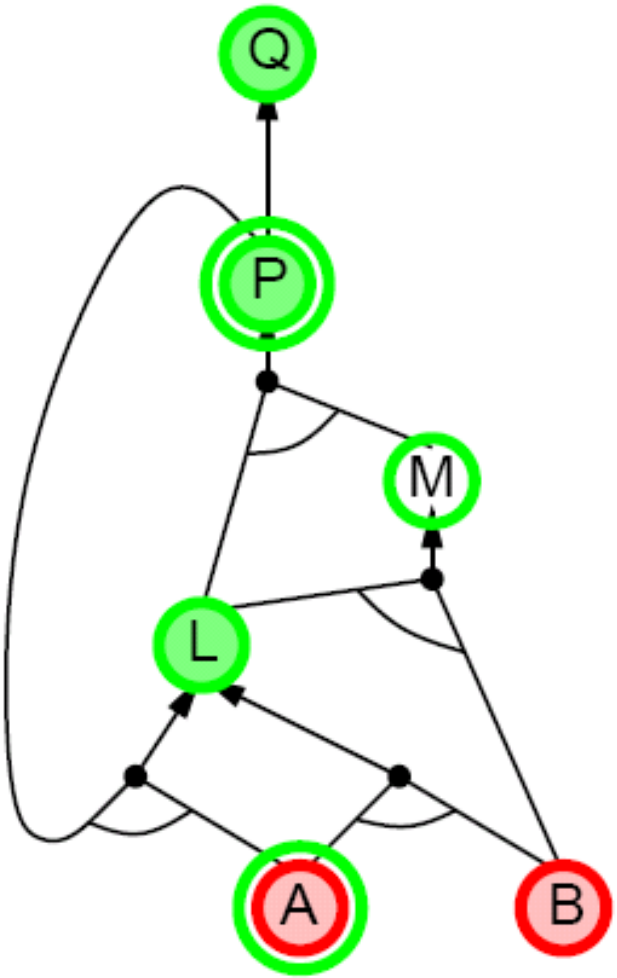
$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

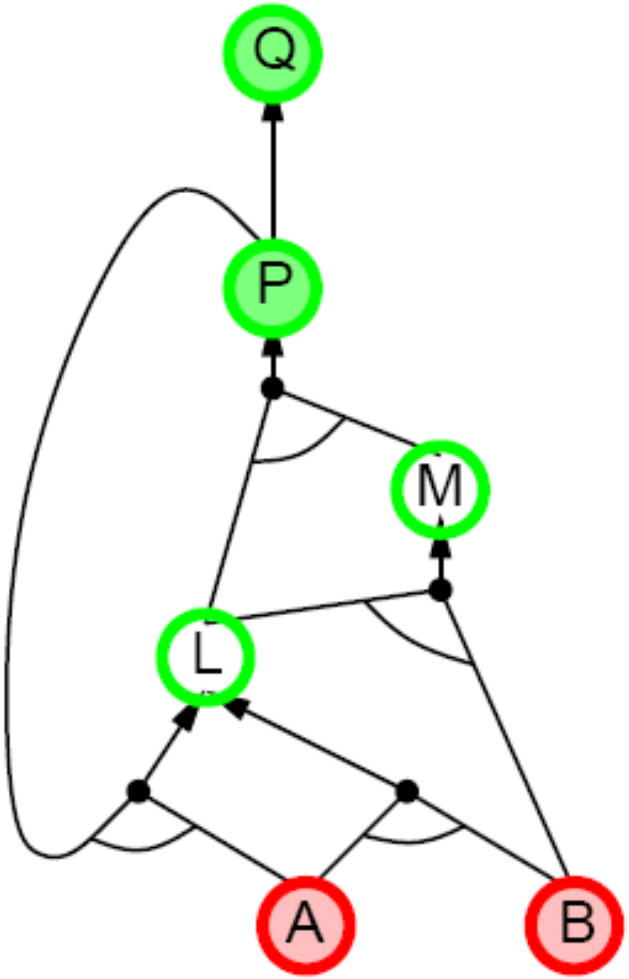
$$B$$

Backward Chaining Example



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

Backward Chaining Example



$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

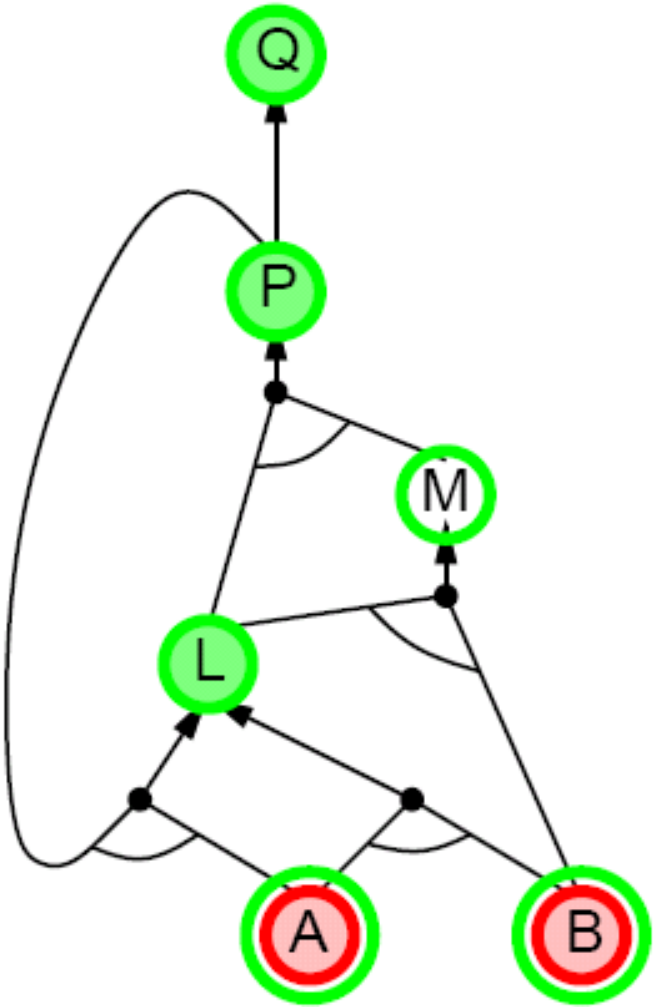
$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

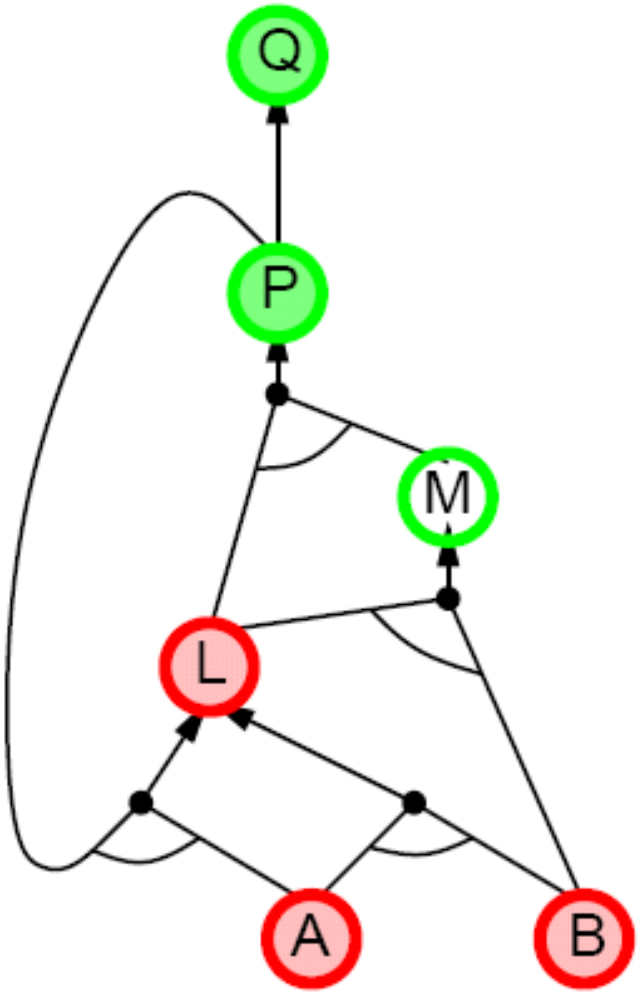
$$B$$

Backward Chaining Example



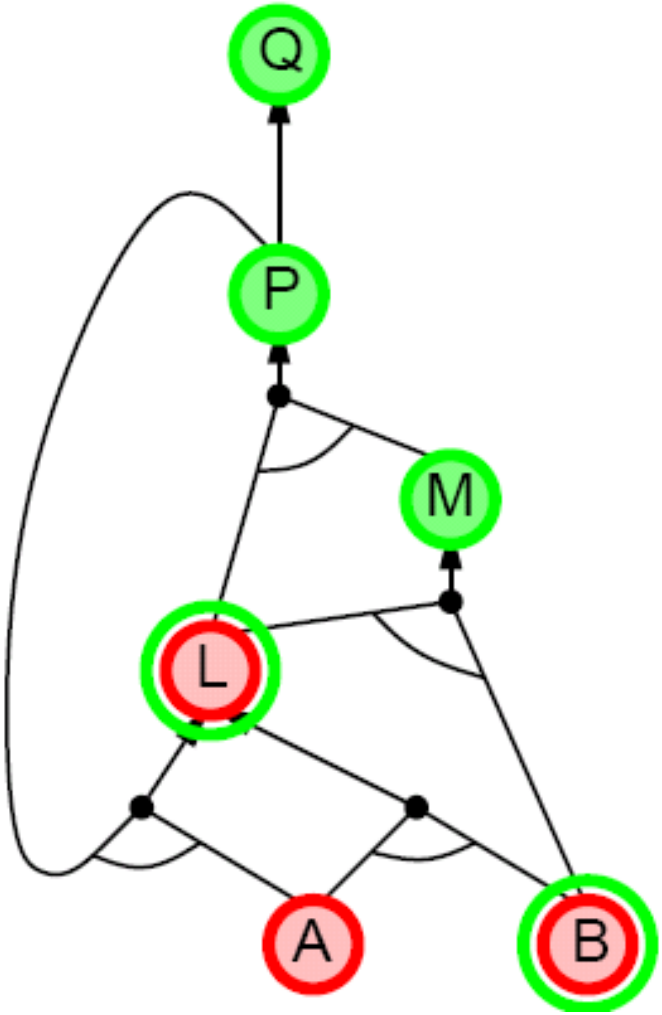
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Backward Chaining Example



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

Backward Chaining Example



$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

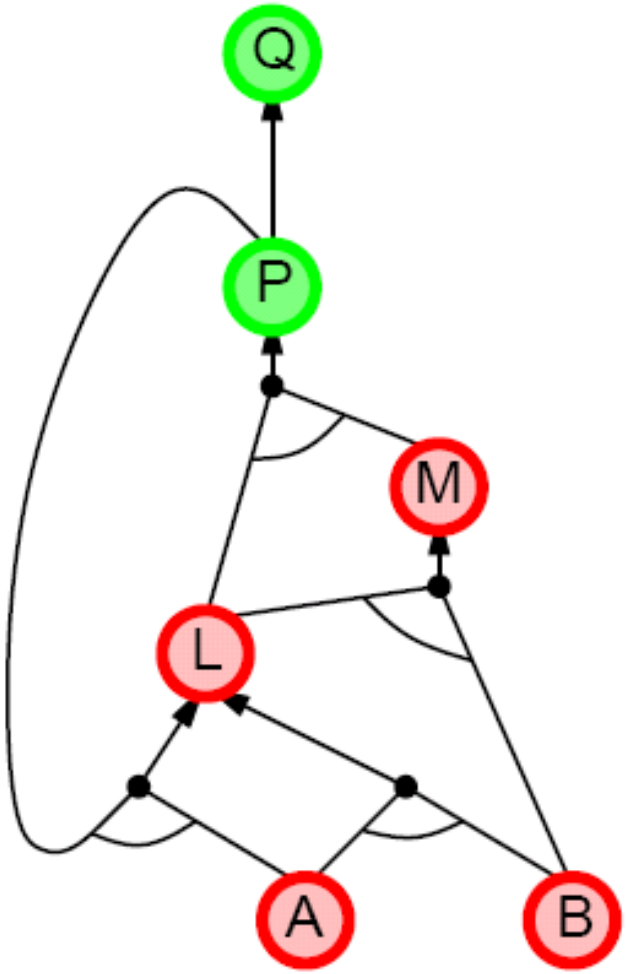
$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$

Backward Chaining Example



$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

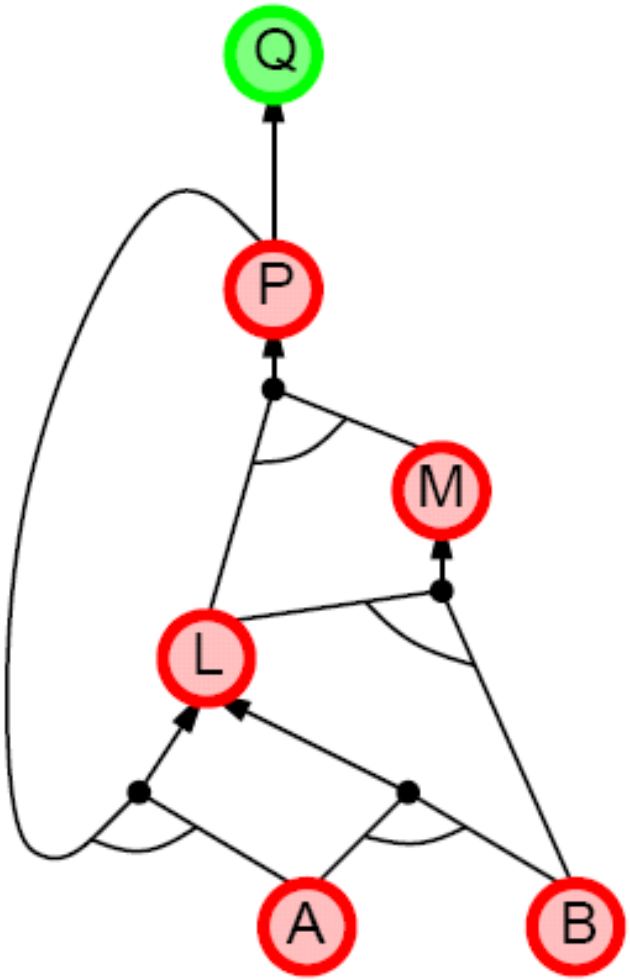
$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$

Backward Chaining Example



$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

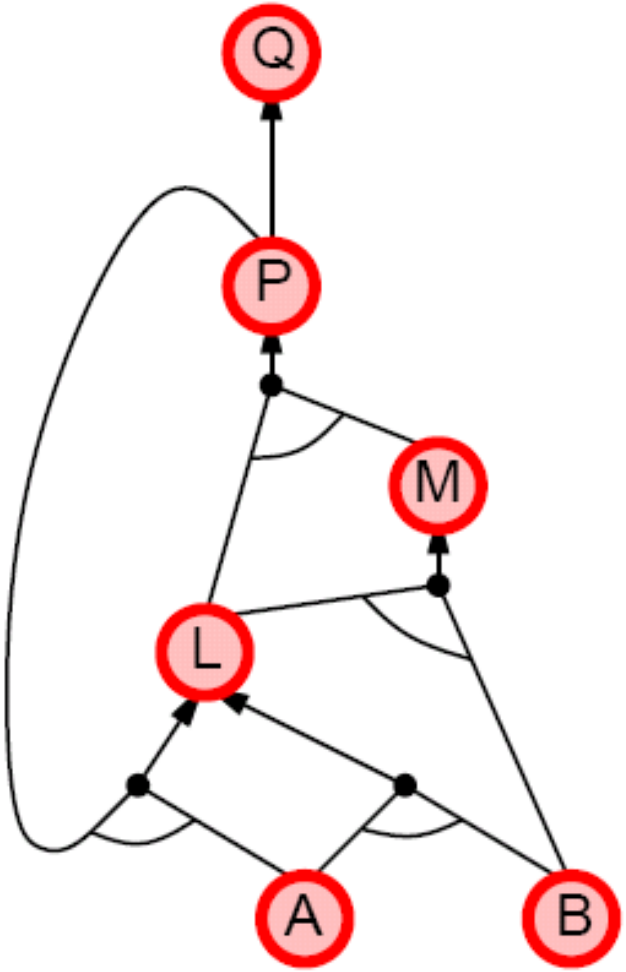
$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$

Backward Chaining Example



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

Forward vs. Backward Chaining

- Forward chaining Advantage
  - Asks for more data but utilizes this data to eliminate possibilities and therefore processes fewer rules.
- Forward Chaining Disadvantage
  - Many rules may be applicable at each stage: so how should we choose which one to apply next at each stage?
  - The whole process is not directed towards a goal: so how do we know when to stop applying the rules?
- Backward Chaining Advantage
  - The search is goal directed, so we only apply the rules that are necessary to achieve the goal.
  - Asks for less data than forward-chaining but examines more rules
- Backward Chaining Disadvantage
  - The goal has to be known.
  - Fortunately, many AI systems can be formulated in a goal based fashion.
- Complexity of BC can be much less than linear in size of KB.

Limitations of Propositional Logic

- number of propositions
  - since everything has to be spelled out explicitly, the number of rules is immense
- dealing with change (monotonicity)
  - even in very simple worlds, there is change
  - the agent's position changes
  - time-dependent propositions and rules can be used
    - even more propositions and rules
- propositional logic has only one representational device, the proposition
  - difficult to represent objects and relations, properties, functions, variables, …

First Order Logic

Problem of Propositional Logic

- Propositional logic has very limited expressive power
  - Can't have one proposition to represent a group of objects
  - e.g.: if we want to say "Every BZU 's student is happy"
    - In Propositional Logic ➔ "Ali is happy", "Ahmed is happy" , "Mohamed is happy" …
  - e.g., cannot say "pits cause breezes in adjacent squares" except by writing one sentence for each square.
  - We want to be able to say this in one single sentence: "for all squares and pits, pits cause breezes in adjacent squares.
  - First order logic will provide this flexibility by using ∀, ∃.
    - **i.e ∀x At(x, BZU) ⇒ Happy(x)**
- **So, FOL fixes the problems of PL:**
  - PL doesn't have variables BUT FOL does.
  - PL can't directly express properties of individuals or relations between individuals, But FOL can do that.
- Inferencing in PL is fairly easy But In FOL it is more complex

First-Order logic

- P-Logic assumes the world contains facts that are either true or false.
- First-Order Logic **models the world in terms of**:
  - Objects: which are things with individual identities
    - e.g. people, houses, numbers, colors, baseball games, wars, computers, …
  - Variable: Represents object, X,Y,Z…
  - Predicate/relation: Gives relation between objects, variables, i.e. brother of, bigger than, part of, comes between, …
    - Person (Ahmed), Likes(Ali, Yasser), EquilateralTriangle(X,Y,Z)
  - Function: a relation where there is only one "value" for any given "input"
    - MotherOf(Yasser), OldestSonOf(Walid, Aymen), …
  - Properties
    - Describe specific aspects of objects, used to distinguish between objects
      - Green, round, heavy, visible,

Syntax of FOL: Basic elements

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

Atomic & Complex Sentences

- A sentence represents a fact in the world that is assigned a truth value.
- Atomic sentence = predicate ($term_1$,...,$term_n$) or $term_1$ = $term_2$
  - **Father(Ahmed, Mohamed), Mother(Asmaa, Ahmed), Sister(Dena, Ahmed)**
  - **Married(Ahmed, Asmaa)**
  - **Married(Father-Of(Ahmed), Mother-Of(Ahmed))**

- Complex sentences are made from atomic sentences using connectives:

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

  - **Sibling(Ahmed,Ali) $\Rightarrow$ Sibling(Ali,Ahmed)**
  - **Friend(Ali,Ahmed) $\Rightarrow$ Friend(Ahmed,Ali)**
  - **Father(Ahmed, Yasser) $\wedge$ Mother(Asmaa, Yasser) $\wedge$ Sister(Yara, Yasser) $\neg$ Sister(Yasser, Yara)**
  - **Parents(Ahmed, Dena, Ali, Yara) $\wedge$ Married(Ahmed, Dena)**
  - **Parents(Yasser, Yaraa) $\Rightarrow$ Married(Yasser, Yara)**

Universal Quantifiers (∀)

- Used to express properties of collections of objects and eliminates the need to explicitly enumerate all objects as in PL.
- Universal quantifier: ∀<variables> <sentence>
- Means the sentence holds true for all values in the domain of variable x.
- Everyone at BZU is smart: ∀x At(x,BZU) ⇒ Smart(x)
- ∀x P is true in a model m iff P is true with x being each possible object in the model.
  - All humans are mammals.
    - ∀x Human(x) ⇒ Mammal(x): for all x if x is a human then x is a mammal
  - All birds can fly
    - ∀ x Bird(x) ⇒ Can-Fly(x)
- Main connective typically ⇒ forming if-then rules
  - Mammals must have hair
    - ∀x Mammal(x) ⇒ HasHair(x): for all x if x is a mammal then x has hair
- Equivalent to the conjunction of P instantiations: At(Ahmed,BZU) ⇒ Smart(Ahmed) ∧ At(Aymen,BZU) ⇒ Smart(Aymen) ∧ …

A common mistake to avoid

- Common mistakes with the use of $\wedge$ as main connective with $\forall$:
- Examples
- $\forall x$ At(x, BZU) $\wedge$ Smart(x): means "Everyone is at BZU and everyone is smart".
- $\forall x$ Human(x) $\wedge$ Mammal(x) means? Everything is human and a mammal

  (Human(Ali) $\wedge$ Mammal(Ali)) $\wedge$

  (Human(Yasser) $\wedge$ Mammal(Yasser)) $\wedge$

  (Human(Y) $\wedge$ Mammal(Y)) $\wedge$ …

  - $\forall x$ student(x)^smart(x) means "Everyone in the world is a student and is smart"
- Universal quantification should be rarely used to make blanket statements about every individual in the world.

Existential quantification

Existential quantifier: ∃<variables> <sentence>

- Means the sentence holds true for some value/s of x in the domain of x.
- Existential quantifiers (∃) are usually used with "∧" to specify a list of properties about an individual:
  - ∃x student(x) ^ smart(x) means "There is a student who is smart"
  - Mammals may have arms. ∃ x Mammal(x) ∧ HasArms(x), this interpreted as there exist an x such that x is a mammal and x has arms
  - "Some humans are computer scientists": ∃ x Human(x) ∧ Computer-Scientist(x)
  - "Ahmed has a sister who is a computer scientist": ∃ x Sister(x, Ahmed) ∧ Computer-Scientist(x).
  - "Some birds can't fly": ∃ x Bird(x) ∧ ¬ Can-Fly(x)
- Common mistake: using ⇒ as the main connective with ∃ Results in a weak statement: examples:
- ∃ x At(x, BZU) ⇒ Smart(x): is true if there is anyone who is not at BZU !

Properties of quantifiers

- $\forall x\ \forall y$ is the same as $\forall y\ \forall x$
- $\exists x\ \exists y$ is the same as $\exists y\ \exists x$
- $\exists x\ \forall y$ is not the same as $\forall y\ \exists x$
- $\exists x\ \forall y$ Loves(x,y)
  - "There is a person who loves everyone in the world"
- $\forall y\ \exists x$ Loves(x,y)
  - "Everyone in the world is loved by at least one person"
- Quantifier duality: each can be expressed using the other
- $\forall x$ Likes(x,IceCream) $\quad \neg\exists x\ \neg$Likes(x,IceCream)
- $\exists x$ Likes(x,Broccoli) $\quad \neg\forall x$ Likes(x,Broccoli)

Equality

- $term_1 = term_2$ is true under a given interpretation if and only if $term_1$ and $term_2$ refer to the same object

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

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

Example: Family Relationships

- Objects: People
- Properties: Gender, …
  - Expressed as unary predicates Male(x), Female(y)
- Relations: parenthood, brotherhood, marriage
  - Expressed through binary predicates Brother(x,y), …
- Functions: motherhood, fatherhood
  - Mother(x), Father(y)
  - Because every person has exactly one mother and one father
  - There may also be a relation Mother-of(x,y), Father-of(x,y)

- **Family Relationships**
  - $\forall w,h\ \text{Husband}(h,w) \Leftrightarrow \text{Male}(h) \wedge \text{Spouse}(h,w)$
  - $\forall x\ \text{Male}(x) \Leftrightarrow \neg\text{Female}(x)$
  - $\forall g,c\ \text{Grandparent}(g,c) \Leftrightarrow \exists\ p\ \text{Parent}(g,p) \wedge \text{Parent}(p,c)$
  - $\forall x,y\ \text{Sibling}(x,y) \Leftrightarrow \neg(x=y) \wedge \exists\ p\ \text{Parent}(p,x) \wedge \text{Parent}(p,y)$

Thinking in FOL

☞ **Convert the following English sentences to FOL.**

- **Bob is a fish.**
  - What are the objects?
    Bob look for nouns and noun phrases
  - What are the relations?
    is a fish look for verbs and verb phrases

  Answer: Fish(Bob) a unary relation or **property**

☞ **Convert the following English sentences to FOL.**

- **America bought Alaska from Russia.**
  - What are the objects?
    America, Alaska, Russia
  - What are the relations?
    bought(who, what, from) an **n-ary relation** where n is 3

  Answer: Bought(America,Alaska,Russia)

Thinking in FOL

☞ Now lets think about quantifying variables.

- Ahmed collects everything.
  - What are the objects? Ahmed
  - What are the variables and how are they quantified?
    everything x, all - universal

  Answer: $\forall x$ Collects(Ahmed,x)

  Collects(Ahmed,Pencil) $\wedge$ Collects(Ahmed,Fish) $\wedge$ …
- When to restrict the domain, e.g. people:
- All: $\forall x$ Person(x) $\wedge \ldots \Rightarrow \ldots$
  - Things: anything, everything, whatever
  - People: anybody, anyone, everybody, everyone, whoever
- Some (at least one): $\exists x$ Person(x) $\wedge \ldots \wedge \ldots$
  - Things: something
  - People: somebody, someone
- None: $\neg\exists x$ Person(x) $\wedge \ldots \wedge \ldots$
  - Things: nothing
  - People: nobody, no one

Thinking in FOL

☞ How about sentences with multiple variables?

- Somebody collects something.
  - What are the objects? none!
  - What are the variables and how are they quantified? somebody x and something y, some - existential

  Answer: $\exists$ x,y Person(x) $\wedge$ Collects(x,y)
- Everybody collects everything.
- Everybody collects something.
- Something is collected by everybody.

☞ Convert the following English sentences to FOL.

- Nothing collects anything.
  - What are the variables? nothing x and anything y
  - How are they quantified? not one (i.e. not existential) and all (universal)

  Answer: $\neg\exists$x $\forall$y Collects(x,y)

☞ What's the "double-negative" equivalent?

Everything does not collect anything.

Answer: $\forall$x,y $\neg$Collects(x,y)

- Everything collects nothing.

Thinking in FOL

☞ More complex quantified sentences:

1. All hoarders" المكتنزون" collect everything.
   - How are ideas connected? being a hoarder implies collecting everything

   Answer: $\forall x,y\ Horder(x) \Rightarrow Collects(x,y)$
   - Hoarders collect valuable things. Is ambiguous!
     - Some hoarders collect all valuable things.
     - Some hoarders collect some valuable things.
     - All hoarders collect some valuable things.
     - All hoarders collect all valuable things.
2. All stinky shoes are allowed.
3. How are ideas connected? being any shoe and stinky implies it is allowed

   Answer: $\forall x\ Shoe(x) \wedge Stinky(x) \Rightarrow Allowed(x)$.
4. No stinky shoes are allowed.

   Answer: $\neg\exists\ x\ Shoe(x) \wedge Stinky(x) \wedge Allowed(x)$.
5. (All) Stinky shoes are not allowed.

   Answer: $\forall x\ Shoe(x) \wedge Stinky(x) \Rightarrow \neg Allowed(x)$

Thinking in FOL

☞ And now for functional relations and equalities:

- John's income is 20K.
  - Are functional relations specified?
  - Are equalities specified?

  Answer: Income(John) = 20K
- There are exactly two shoes.
  - Are quantities specified?
  - Are equalities implied?

  Ans.: $\exists x,y\ Shoe(x) \wedge Shoe(y) \wedge \neg(x=y) \wedge \forall z\ (Shoe(z) \Rightarrow (x=z) \vee (y=z))$
- Interesting words: always, sometimes, never
  - Good people always have friends = All good people have friends.

    $\forall x\ Person(x) \wedge Good(x) \Rightarrow \exists y(Friend(x,y))$
  - Busy people sometimes have friends = Some busy people have friends.

    $\exists x\ Person(x) \wedge Busy(x) \wedge \exists y(Friend(x,y))$
  - Bad people never have friends = Bad people have no friends.

    $\forall x\ Person(x) \wedge Bad(x) \Rightarrow \neg\exists y(Friend(x,y))$

    or equivalently: No bad people have friends.

    $\neg\exists x\ Person(x) \wedge Bad(x) \wedge \exists y(Friend(x,y))$

Other Examples

All students are smart.

$\forall x (Student(x) \Rightarrow Smart(x))$

There exists a student.

$\exists x \; Student(x).$

There exists a smart student.

$\exists x (Student(x) \wedge Smart(x))$

Every student loves some student.

$\forall x (Student(x) \Rightarrow \exists y (Student(y) \wedge Loves(x,y)))$

Every student loves some other student.

$\forall x (Student(x) \Rightarrow \exists y (Student(y) \wedge \neg (x = y) \wedge Loves(x,y)))$

There is a student who is loved by every other student.

$\exists x (Student(x) \wedge \forall y (Student(y) \wedge \neg(x = y) \Rightarrow Loves(y,x)))$

Bill is a student.

$Student(Bill)$

Other Examples

Bill takes either Analysis or Geometry (but not both)

$\text{Takes(Bill, Analysis)} \Leftrightarrow \neg \text{Takes(Bill, Geometry)}$

Bill takes Analysis or Geometry (or both).

$\text{Takes(Bill, Analysis)} \vee \text{Takes(Bill, Geometry)}$

Bill takes Analysis and Geometry.

$\text{Takes(Bill, Analysis)} \wedge \text{Takes(Bill, Geometry)}$

Bill does not take Analysis.

$\neg \text{Takes(Bill, Analysis)}.$

No student loves Bill.

$\neg \exists x (\text{Student}(x) \wedge \text{Loves}(x, \text{Bill}))$

Bill has at least one sister.

$\exists x \text{ SisterOf}(x,\text{Bill})$

Bill has no sister.

$\neg \exists x \text{ SisterOf}(x,\text{Bill})$

Other Examples

Bill has at most one sister.

$\forall x, y (SisterOf(x, Bill) \wedge SisterOf(y, Bill) \Rightarrow x = y)$

Bill has exactly one sister.

$\exists x (SisterOf(x, Bill) \wedge \forall y (SisterOf(y, Bill) \Rightarrow x = y))$

Bill has at least two sisters.

$\exists x, y (SisterOf(x, Bill) \wedge SisterOf(y, Bill) \wedge \neg (x = y))$

Every student takes at least one course.

$\forall x (Student(x) \Rightarrow \exists y (Course(y) \wedge Takes(x,y)))$

Only one student failed History.

$\exists x (Student(x) \wedge Failed(x, History) \wedge \forall y (Student(y) \wedge Failed(y, History) \Rightarrow x = y))$

No student failed Chemistry but at least one student failed History.

$\neg \exists x (Student(x) \wedge Failed(x, Chemistry)) \wedge \exists x (Student(x) \wedge Failed(x, History))$

Every student who takes Analysis also takes Geometry.

$\forall x (Student(x) \wedge Takes(x, Analysis) \Rightarrow Takes(x, Geometry))$

No student can fool all the other students.

$\neg \exists x (Student(x) \wedge \forall y (Student(y) \wedge \neg (x = y) \Rightarrow Fools(x,y)))$

Inference Rules for FOL

- Inference rules for PL apply to FOL as well (Modus Ponens, And-Introduction, And-Elimination, etc.)
- New (sound) inference rules for use with **quantifiers**:
  - Universal Elimination
  - Existential Introduction
  - Existential Elimination
  - Generalized Modus Ponens (GMP)
- **Resolution**
  - Clause form (CNF in FOL)
  - Unification (consistent variable substitution)
  - Refutation resolution (proof by contradiction)

Inference Rules for FOL

- **Universal Elimination (∀x) P(x) |-- P(c).**
  - If (∀x) P(x) is true, then P(c) is true for **any** constant c in the domain of x, i.e.,, (∀x) P(x) |= P(c).
  - Replace all occurrences of x in the scope of ∀x by the **same** ground term (a constant or a ground function).
  - Example: (∀x) eats(Ziggy, x) |-- eats(Ziggy, IceCream)
- **Existential Introduction P(c) |-- (∃x) P(x)**
  - If P(c) is true, so is (∃x) P(x), i.e., P(c) |= (∃x) P(x)
  - Replace all instances of the given constant symbol by the same **new** variable symbol.
  - Example eats(Ziggy, IceCream) |-- (∃x) eats(Ziggy, x)
- **Existential Elimination**
  - From (∃x) P(x) infer P(c), i.e., (∃x) P(x) |= P(c), where c is a new constant symbol,
    - All we know is there must be some constant that makes this true, so we can introduce a brand new one to stand in for that constant, *even though we don't know exactly what that constant refer to.*
    - Example: (∃x) eats(Ziggy, x) |= eats(Ziggy, Stuff)

Generalized Modus Ponens (GMP)

- Combines And-Introduction, Universal-Elimination, and Modus Ponens

- Ex: $P(c), Q(c), (\forall x)(P(x) \wedge Q(x)) \Rightarrow R(x) \vdash R(c)$

  $P(c), Q(c) \vdash P(c) \wedge Q(c)$ (*by and-introduction*)

  $(\forall x)(P(x) \wedge Q(x)) \Rightarrow R(x)$

  $\vdash (P(c) \wedge Q(c)) \Rightarrow R(c)$ (*by universal-elimination*)

  $P(c) \wedge Q(c), (P(c) \wedge Q(c)) \Rightarrow R(c) \vdash R(c)$ (*by modus ponens*)

- All occurrences of a quantified variable must be instantiated to the same constant.

  $P(a), Q(c), (\forall x)(P(x) \wedge Q(x)) \Rightarrow R(x) \vdash R(c)$

  because all occurrences of x must either instantiated to a or c which makes the modus ponens rule not applicable.

Resolution for FOL

- Resolution rule operates on two *clauses*
  - A clause is a **disjunction** of literals (without explicit quantifiers)
  - Relationship between clauses in KB is **conjunction**
- Resolution Rule for FOL:
  - clause C1: $(l_1, l_2, \ldots \boldsymbol{l_i}, \ldots l_n)$ and

    clause C2: $(l'_1, l'_2, \ldots \boldsymbol{l'_j}, \ldots l'_m)$
  - if $\boldsymbol{l_i}$ and $\boldsymbol{l'_j}$ are two **opposite literals** (e.g., P and ~P) and their argument lists can be be made the same (**unified)** by a set of variable bindings θ = {x1/y1, … Xk/yk} where x1, … Xk are variables and y1, …Yk are terms, then derive a new clause (called resolvent)

    subst($(l_1, l_2, \ldots l_n, l'_1, l'_2, \ldots l'_m)$, θ)

    where function subst(expression, θ) returns a new expression by applying all variable bindings in θ to the original expression

We need answers to the following questions

- How to convert FOL sentences to clause form (especially how to remove quantifiers)
- How to unify two argument lists, i.e., how to find their most general unifier (**mgu**) θ
- How to determine which two clauses in KB should be resolved next (among all resolvable pairs of clauses) and how to determine a proof is completed

Conversion procedure

***step* 1:** remove all "=>" and "<=>" operators

(using P => Q $\equiv$ ~P v Q and P <=> Q $\equiv$ P => Q ^ Q => P)

***step* 2:** move all negation signs to individual predicates

(using de Morgan's law)

***step* 3:** remove all existential quantifiers ∃y

case 1: y is not in the scope of any universally quantified variable, then replace all occurrences of y by a skolem constant

case 2: if y is in scope of universally quantified variables x1, … xi, then replace all occurrences of y by a skolem function that takes x1, … Xi as its arguments

***step* 4:** remove all universal quantifiers ∀x (with the understanding that all remaining variables are universally quantified)

***step* 5:** convert the sentence into CNF (using distribution law, etc)

***step* 6:** use parenthesis to separate all disjunctions, then drop all v's and ^'s

Conversion examples

$\forall x\ (P(x) \wedge Q(x) \Rightarrow R(x))$

$\forall x\ \sim(P(x) \wedge Q(x)) \vee R(x)$ (by step 1)

$\forall x\ \sim P(x) \vee \sim Q(x) \vee R(x)$ (by step 2)

$\sim P(x) \vee \sim Q(x) \vee R(x)$ (by step 4)

$(\sim P(x),\ \sim Q(x),\ R(x))$ (by step 6)

$\exists y\ rose(y) \wedge yellow(y)$

$rose(c) \wedge yellow(c)$

(where c is a skolem constant)

$(rose(c)),\ (yellow(c))$

Conversion examples

∀x [person(x) => ∃y (person(y) ^ father(y, x))]

∀x [~person(x) v ∃y (person(y) ^ father(y, x))] (by step 1)

∀x [~person(x) v (person(f_sk(x)) ^ father(f_sk(x), x))] (by step 3)

~person(x) v (person(f_sk(x)) ^ father(f_sk(x), x)) (by step 4)

(~person(x) v person(f_sk(x)) ^ (~person(x) v father(f_sk(x), x))

(by step 5)

(~person(x), person(f_sk(x)), (~person(x), father(f_sk(x), x))

(by step 6)

(where f_sk(.) is a skolem function)

Unification

- The goal is to find a set of variable bindings so that the argument lists of two opposite literals (in two clauses) can be made the same.
  - Unify(P,Q) takes two atomic (i.e. single predicates) sentences P and Q and returns a substitution that makes P and Q identical.

    - Knows(John,x), Knows(John, Jane)
      - {x/Jane)
    - Knows(John,x), Knows(y, Bill)
      - {x/Bill,y/John)

Unification Rules

- Only variables can be bound to other things.
  - Constants a and b cannot be unified (different constants in general refer to different objects)
  - Constant a and function f(x) cannot be unified (unless the inverse function of f is known, which is not the case for general functions in FOL)
  - f(x) and g(y) cannot be unified (function symbols f and g in general refer to different functions and their exact definitions are different in different interpretations)
- Cannot bind variable x to y if x appears anywhere in y
  - Try to unify x and f(x). If we bind x to f(x) and apply the binding to both x and f(x), we get f(x) and f(f(x)) which are still not the same (and will never be made the same no matter how many times the binding is applied)
- Otherwise, bind variable x to y, written as x/y (this guarantees to find the most general unifier, or **mgu**)
  - Suppose both x and y are variables, then they can be made the same by binding both of them to any constant c or any function f(.). Such bindings are less general and impose unnecessary restriction on x and y.
- To unify two terms of the same function symbol, unify their argument lists (**unification is recursive**)

  Ex: to unify f(x) and f(g(b)), we need to unify x and g(b)

Unification Rules

- When the argument lists contain multiple terms, unify each pair of terms

  Ex. To unify (x, f(x), …) (a, y, …)

  1. unify x and a ($\theta = \{x/a\}$).
  2. apply θ to the remaining terms in both lists, resulting (f(a), …) and (y, …)
  3. unify f(a) and y with binding y/f(a)
  4. add y/f(a) to new θ
  5. goto step 2

  …

Unification

- Which unification solves

  Knows(John,x), Knows(y, Mother(y))

- Choose
  - A: {y/Bill,x/Mother(John))
  - B: {y/John,x/Bill)
  - C: {y/John,x/Mother(John))
  - D: {y/Mother(John),x/John)
  - E: {y/John,x/Mother(y))

Unification Examples

- parents(x, father(x), mother(Bill)) and parents(Bill, father(Bill), y)
  - unify x and Bill: $\theta = \{x/Bill\}$
  - unify father(Bill) and father(Bill): $\theta = \{x/Bill\}$
  - unify mother(Bill) and y: $\theta = \{x/Bill\}, y/mother(Bill)\}$
- parents(x, father(x), mother(Bill)) and parents(Bill, father(y), z)
  - unify x and Bill: $\theta = \{x/Bill\}$
  - unify father(Bill) and father(y): $\theta = \{x/Bill, y/Bill\}$
  - unify mother(Bill) and z: $\theta = \{x/Bill, y/Bill, z/mother(Bill)\}$
- parents(x, father(x), mother(Jane)) and parents(Bill, father(y), mother(y))
  - unify x and Bill: $\theta = \{x/Bill\}$
  - unify father(Bill) and father(y): $\theta = \{x/Bill, y/Bill\}$
  - unify mother(Jane) and mother(Bill): Failure because Jane and Bill are different constants

More Unification Examples

- P(x, g(x), h(b)) and P(f(u, a), v, u))
  - unify x and f(u, a): $\theta = \{x/ f(u, a)\}$;
    remaining lists: (g(f(u, a)), h(b)) and (v, u)
  - unify g(f(u, a)) and v: $\theta = \{x/f(u, a), v/g(f(u, a))\}$;
    remaining lists: (h(b)) and (u)
  - unify h(b) and u: $\theta = \{x/f(u, a), v/g(f(h(b), a)), u/h(b)\}$;
- P(f(x, a), g(x, b)) and P(y, g(y, b))
  - unify f(x, a) and y: $\theta = \{y/f(x, a)\}$
    remaining lists: (g(x, b)) and (g(f(x, a), b))
  - unify x and f(x, a): failure because x is in f(x, a)

Unification - Purpose

Given:

$\neg Knows\ (John, x) \vee Hates\ (John, x)$

$Knows\ (John, Jim)$

Derive:

$Hates\ (John, Jim)$

Unification:

$$unify(Knows(John,x), Knows(John,Jim)) = \{x/Jim\}$$

Need unifier {x/Jim} for resolution to work.

Add to knowledge base:

$$\neg Knows(John, Jim) \vee Hates(John, Jim)$$

Unification – Purpose (example)

Who does John hate?

$\exists x$: *Hates* (*John*, *x*)

Knowledge base (in clause form):

1. $\neg$*Knows* (*John*, *v*) $\vee$ *Hates* (*John*, *v*)
2. *Knows* (*John*, *Jim*)
3. *Knows* (*y*, *Leo*)
4. *Knows* (*z*, *Mother*(*z*))
5. $\neg$*Hates* (*John*, *x*) (since $\neg \exists x$: *Hates* (*John*, *x*) $\Leftrightarrow \forall$ x: $\neg$Hates(John,x))

   Resolution with 5 and 1:

   unify(*Hates*(*John*, *x*), *Hates*(*John*, *v*)) = {x/v}
6. $\neg$*Knows* (*John*, v)

   Resolution with 6 and 2:

   unify(*Knows*(*John*, v), *Knows*(*John*, *Jim*))= {v/Jim}

   or resolution with 6 and 3:

   unify(*Knows* (*John*, v), *Knows* (y, Leo)) = {y/John, v/Leo}

   or Resolution with 6 and 4:

   unify(*Knows* (*John*, v), *Knows* (z, Mother(z))) = {z/John, v/Mother(z)}

Answers:

1. Hates(John,x) with {x/v, v/Jim} (i.e. John hates Jim)
2. Hates(John,x) with {x/v, y/John, v/Leo} (i.e. John hates Leo)
3. Hates(John,x) with {x/v, v/Mother(z), z/John} (i.e. John hates his mother)

Unification Algorithm

```
procedure unify(p, q, θ)        /* p and q are two lists of terms and |p| = |q| */
    if p = empty then return θ;  /* success */
    let r = first(p) and s = first(q);
    if r = s then return unify(rest(p), rest(q), θ);
    if r is a variable then temp = unify-var(r, s);
    else if s is a variable then temp = unify-var(s, r);
        else if both r and s are functions of the same function name then
            temp = unify(arglist(r), arglist(s), empty);
            else return "failure";
    if temp = "failure" then return "failure";  /* p and q are not unifiable */
    else θ = θ     temp;   /* apply tmp to old θ then insert it into θ */
        return unify(subst(rest(p), tmp), subst(rest(q), tmp), θ);
end{unify}
procedure unify-var(x, y)
    if x appears anywhere in y then return "failure";
    else return (x/y)
end{unify-var}
```

Most General Unifier

In cases where there is more than one substitution choose the one that makes the least commitment (most general) about the bindings.

UNIFY **(*Knows* (*John,x*), *Knows* (*y,z*))**

= {*y* / *John*, *x* / *z*}

not {*y* / *John*, *x* / *John*, *z* / *John*}

not {*y* / *John*, *x* / *z*, *z* / *Freda*}

...

See R&N for general unification algorithm. O(n^2) with Refutation

Resolution in FOL

- Convert all sentences in KB (axioms, definitions, and known facts) and the goal sentence (the theorem to be proved) to clause form
- Two clauses C1 and C2 can be resolved if and only if **r** in C1 and **s** in C2 are two opposite literals, and their argument list arglist_r and arglist_s are unifiable with mgu = θ.
- Then derive the resolvent sentence: subst((C1 – {r}, C2 – {s}), θ)

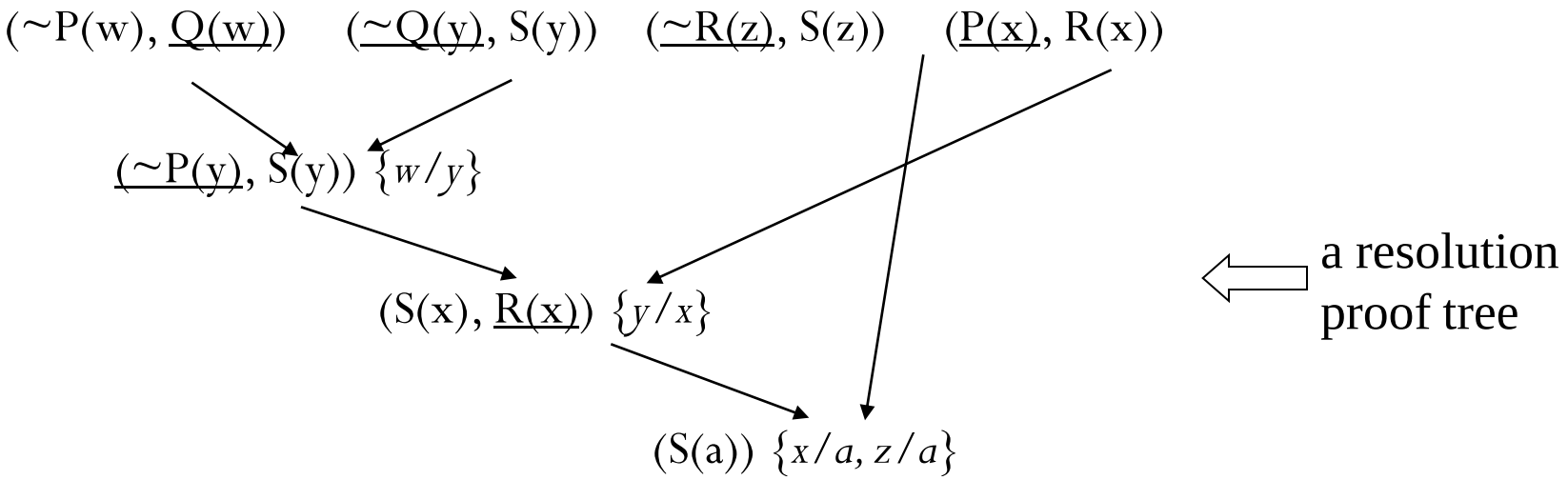
  (substitution is applied to all literals in C1 and C2, but not to any other clauses)

Resolution example

- Prove that

$\forall w\ P(w) \Rightarrow Q(w),\ \forall y\ Q(y) \Rightarrow S(y),\ \forall z\ R(z) \Rightarrow S(z),\ \forall x\ P(x) \vee R(x) \models \exists u\ S(u)$

- Convert these sentences to clauses ($\exists u\ S(u)$ skolemized to $S(a)$)
- Apply resolution

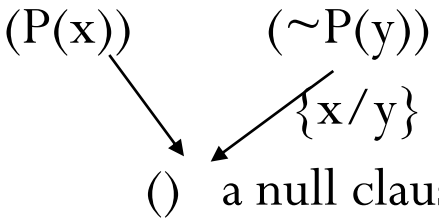


- Problems
  - The theorem S(a) does not actively participate in the proof
  - The last resolution is more than a mechanical step
  - Hard to determine if a proof (with consistent variable bindings) is completed if the theorem consists of more than one clause

Resolution Refutation: a better proof strategy

- Given a consistent set of axioms KB and goal sentence Q, show that KB |= Q.
- **Proof by contradiction**: Add ~Q to KB and try to prove false.

  because (KB |= Q) <=> (KB ^ ~Q |= False, or KB ^ ~Q is inconsistent)
- How to represent "**false**" in clause form
  - P(x) ^ ~P(y) is inconsistent
  - Convert them to clause form then apply resolution



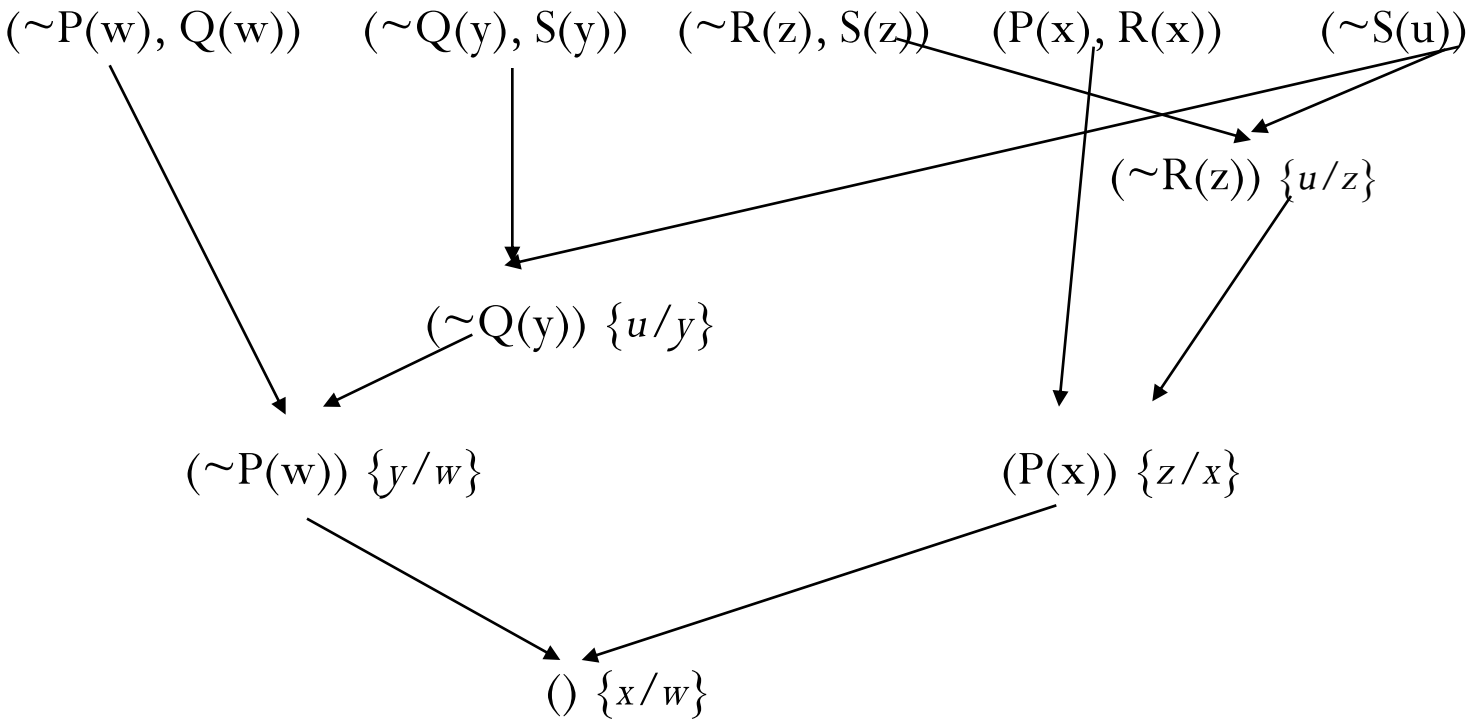
  - A null clause represents false (inconsistence/contradiction)
  - **KB |= Q if we can derive a null clause from KB ^ ~Q by resolution**

Resolution Refutation Example

- Prove by resolution refutation that

  $\forall w\ P(w) \Rightarrow Q(w),\ \forall y\ Q(y) \Rightarrow S(y),\ \forall z\ R(z) \Rightarrow S(z),\ \forall x\ P(x) \vee R(x) \models \exists u\ S(u)$

- Convert these sentences to clauses ($\sim \exists u\ S(u)$ becomes $\sim S(u)$)



Refutation Resolution Procedure

```
procedure resolution(KB, Q)
    /* KB is a set of consistent, true FOL sentences, Q is a goal sentence.
       It  returns success if KB |-- Q, and failure otherwise */
    KB = clause(union(KB, {~Q}))   /* convert KB and ~Q to clause form */
    while null clause is not in KB do
        pick 2 sentences, S1 and S2, in KB that contain a pair of opposite
              literals whose argument lists are unifiable
        if none can be found then return "failure"
        resolvent = resolution-rule(S1, S2)
        KB = union(KB, {resolvent})
    return "success "
end{resolution}
```

Example of Automatic Theorem Proof:

Did Curiosity kill the cat

- Jack owns a dog. Every dog owner is an animal lover. No animal lover kills an animal. Either Jack or Curiosity killed the cat, who is named Tuna. Did Curiosity kill the cat?
- These can be represented as follows:

A. $(\exists x)\ Dog(x) \wedge Owns(Jack,x)$

B. $(\forall x)\ ((\exists y)\ Dog(y) \wedge Owns(x, y)) \Rightarrow AnimalLover(x)$

C. $(\forall x)\ AnimalLover(x) \Rightarrow (\forall y)\ Animal(y) \Rightarrow \sim Kills(x,y)$

D. $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$

E. $Cat(Tuna)$

F. $(\forall x)\ Cat(x) \Rightarrow Animal(x)$

Q. $Kills(Curiosity, Tuna)$

Example of Automatic Theorem Proof:

Did Curiosity kill the cat

- **Convert to clause form**

  A1. (Dog(D)) /* D is a skolem constant */

  A2. (Owns(Jack,D))

  B. (~Dog(y), ~Owns(x, y), AnimalLover(x))

  C. (~AnimalLover(x), ~Animal(y), ~Kills(x,y))

  D. (Kills(Jack,Tuna), Kills(Curiosity,Tuna))

  E. (Cat(Tuna))

  F. (~Cat(x), Animal(x))

- **Add the negation of query:**

  ¬Q: (~Kills(Curiosity,Tuna))

Example of Automatic Theorem Proof:

Did Curiosity kill the cat

- **The resolution refutation proof**

R1: ¬Q, D, {}, (Kills(Jack, Tuna))

R2: R1, C, {x/Jack, y/Tuna}, (~AnimalLover(Jack), ~Animal(Tuna))

R3: R2, B, {x/Jack}, (~Dog(y), ~Owns(Jack, y), ~Animal(Tuna))

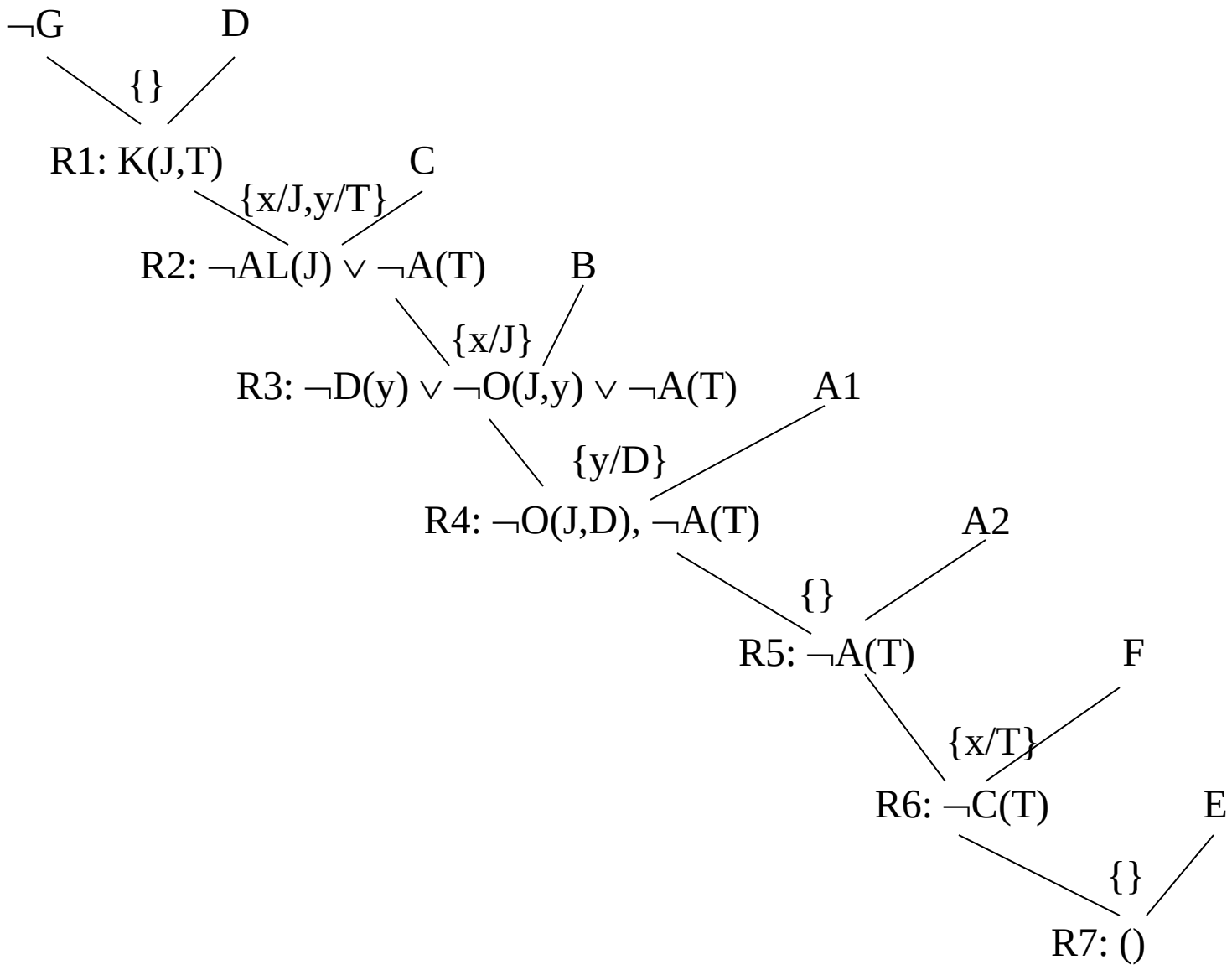
R4: R3, A1, {y/D}, (~Owns(Jack, D), ~Animal(Tuna))

R5: R4, A2, {}, (~Animal(Tuna))

R6: R5, F, {x/Tuna}, (~Cat(Tuna))

R7: R6, E, {} ()

The proof tree



Horn Clauses

- A Horn clause is a clause with at most one positive literal:

  (~P1(x), ~P2(x), …, ~Pn(x) v Q(x)), equivalent to

  ∀x P1(x) ^ P2(x) … ^ Pn(x) => Q(x) or

  Q(x) <= P1(x), P2(x), … , Pn(x) (in prolog format)

  - if contains no negated literals (i.e., Q(a) <=): facts
  - if contains no positive literals (<= P1(x), P2(x), … , Pn(x)): query
  - if contain no literal at all (<=): null clause
- Most knowledge can be represented by Horn clauses
- Easier to understand (keeps the implication form)
- Easier to process than FOL
- Horn clauses represent a subset of the set of sentences representable in FOL. For example, it cannot represent uncertain conclusions, e.g., Q(x) v R(x) <= P(x)).

Example of forward chaining

- Example: KB = All cats like fish, cats eat everything they like, and Ziggy is a cat. In FOL, KB =

1. **(Ax) cat(x) => likes(x, Fish)**
2. **(Ax)(Ay) (cat(x) ^ likes(x,y)) => eats(x,y)**
3. **cat(Ziggy)**

- Goal query: Does Ziggy eat fish?

Data-driven

Proof:

1. Use GMP with (1) and (3) to derive: 4. **likes(Ziggy, Fish)**
2. Use GMP with (3), (4) and (2) to derive **eats(Ziggy, Fish)**
3. So, Yes, Ziggy eats fish.

Backward chaining

- Example: Does Ziggy eat fish?
- To prove **`eats(Ziggy, Fish)`**, first see if this is known from one of the axioms directly. Here it is not known, so see if there is a Horn clause that has the consequent (i.e., right-hand side) of the implication matching the goal.

Proof: Goal Driven

1. Goal matches RHS of Horn clause (2), so try and prove new sub-goals **`cat(Ziggy)`** and **`likes(Ziggy, Fish)`** that correspond to the LHS of (2)
2. **`cat(Ziggy)`** matches axiom (3), so we've "solved" that sub-goal
3. **`likes(Ziggy, Fish)`** matches the RHS of (1), so try and prove **`cat(Ziggy)`**
4. **`cat(Ziggy)`** matches (as it did earlier) axiom (3), so we've solved this sub-goal
5. There are no unsolved sub-goals, so we're done. Yes, Ziggy eats fish.

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
  - Efficient when you want to compute all conclusions
- BC is goal-driven, better for problem-solving
  - Where are my keys? How do I get to my next class?
  - Complexity of BC can be much less than linear in the size of the KB
  - Efficient when you want one or a few decisions

Chapter Summary

- some problems require more sophisticated techniques than searching for a solution
- reasoning utilizes existing knowledge to generate new knowledge
  - requires appropriate representation and reasoning methods
- logic provides a flexible and powerful framework for representation and reasoning
  - used for the formulation of abstract models that reflect essential aspects of the problem and environment
  - propositional logic is relatively simple, but also limited