LOGICAL AGENTS

Logical reasoning

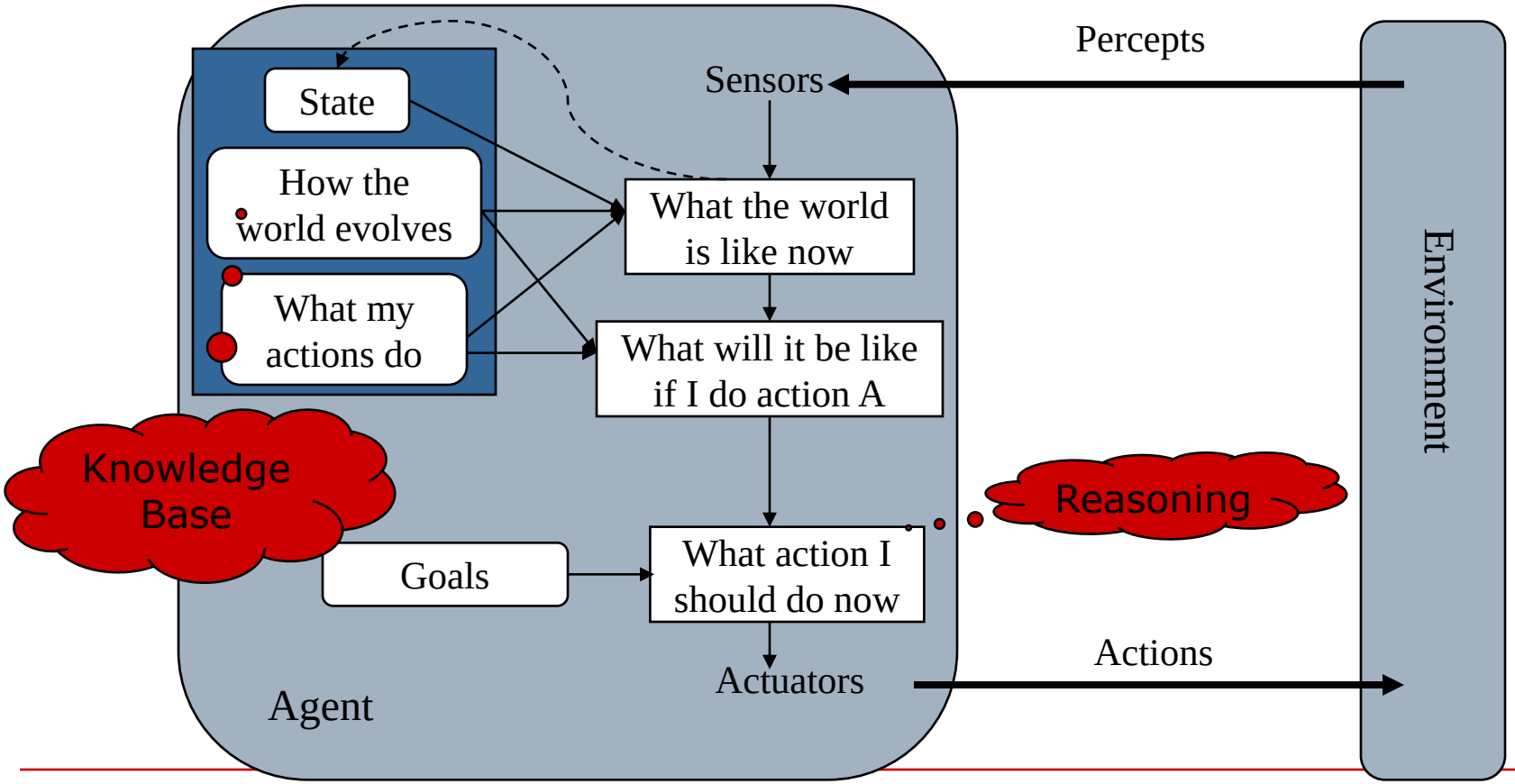
- Want a system that knows about its environment and can **reason** about possible actions.*
  - This type of system is a logical system.
  - Possibly a good choice in **partially observable** environments where we might need to **infer** about hidden information.

Two main issues in construction of a logical system:

1. Knowledge representation; how do we represent information?
2. Knowledge reasoning; how do we use information to reach decisions and/or derive new facts?

- Reasoning is also known as **inferencing**.
  - In performing inferencing, we should try to ensure that **any valid fact** can be derived **if supported** by the available information!*

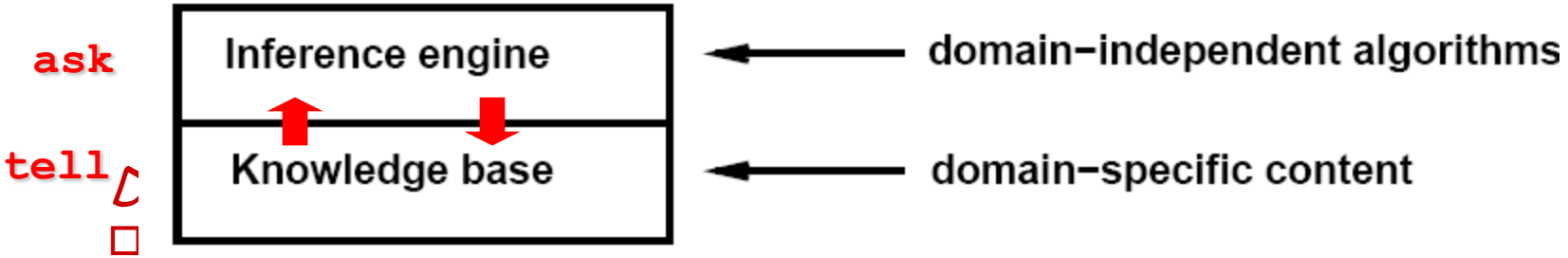
Knowledge Based Agents



Knowledge Base

Knowledge Base

- A set of *sentences*
- in a formal *knowledge representation language*
- that *encodes assertions* about the world.



- `Ask` it what to do → answers should follow by *inference rules* from the KB.

Knowledge representation

- Knowledge representation should be somewhat natural, expressive and efficient.*
- **Facts** are claims about the environment which are either **true** or **false**.
- Knowledge in the form of a set of facts about our environment are stored in a knowledge base (KB).
  - Facts are represented by **sentences**;
  - Sentences are expressed in a **representation language**.
- We should be able to **TELL** (UPDATE) the KB new information and to **ASK** (QUERY) the KB for information (presently available or deduced).
  - TELL and ASK are standard tasks for knowledge-based agents.

What is a logic?

- A formal language with associated
  - *Syntax* – what expressions are legal (well-formed)
  - *Semantics* – what legal expressions mean
    - in logic, the *truth* of each sentence with respect to a set of possible worlds.
- e.g. the language of arithmetic
  - $X+2 \geq y$ is a sentence, $x2+y$ is not a sentence
  - $X+2 \geq y$ is true in a world where x=7 and y =1
  - $X+2 \geq y$ is false in a world where x=0 and y =6

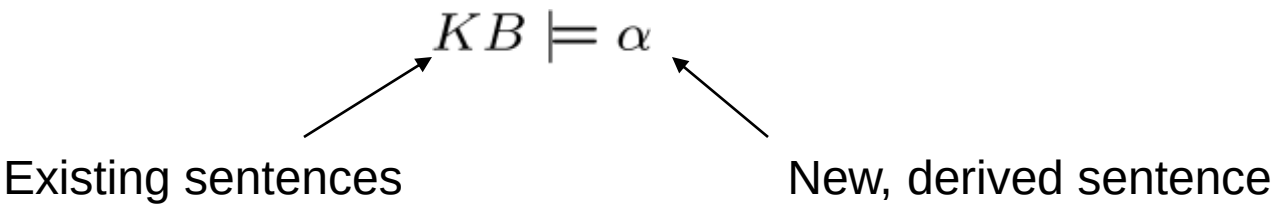
Inferencing with knowledge

Inferencing is how we derive:

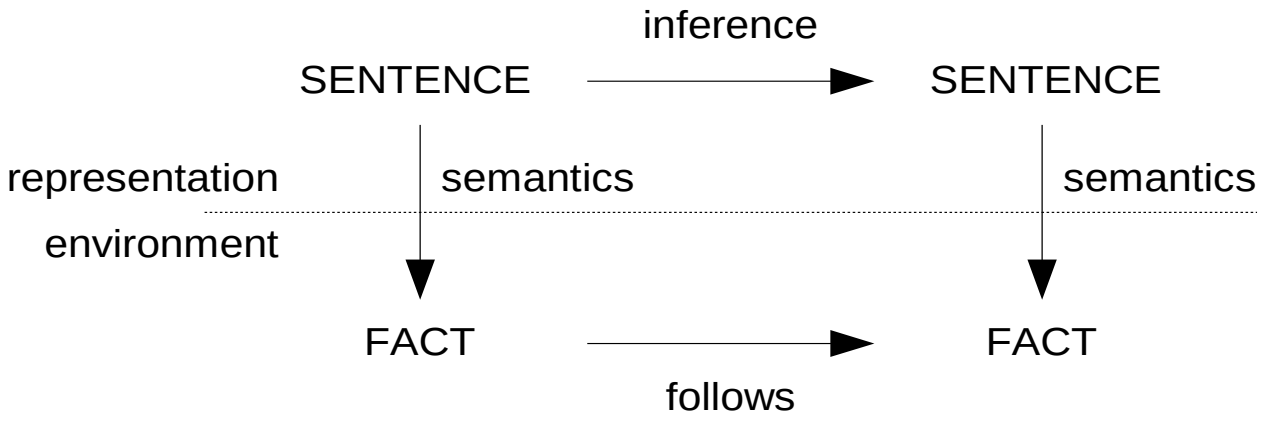
- Conclusions from existing knowledge;
- New information from existing information.

Inferencing might be used in both ASK and TELL operations.

- **Entailment** is the generation or discovery that a new sentence is TRUE given existing sentences.



Relationship between representation and inferencing



Propositional logic

- Also simply "Boolean logic."
- One method to achieve knowledge representation and logical inferencing.*
- Need to consider:
  - Symbols and connectives;
  - Sentences, syntax and semantics;
  - Means of logical inferencing.

Propositional logic symbols and connectives

- **TRUTH SYMBOLS**: T (true) and F (false) are provided by the language.
- **PROPOSITIONAL SYMBOLS**: P, Q, R, etc. mean something in the environment.
  - E.g, P means "It is hot", Q means "It is humid", R means "It is raining", etc.
- **LOGICAL CONNECTIVES**:

$\neg$	negation
$\vee$	conjunction (and)
$\wedge$	disjunction (or)
$\Rightarrow$	implication (if-then)
$\Leftrightarrow$	biconditional (if and only if)

Propositional logic sentences

- Truth and propositional symbols are considered **ATOMIC SENTENCES**.
- More complex sentences are formed using connectives.
  - Sentences formed in this way can be called Well-Formed Formula (WFF).
- Parentheses can be used to indicate precedence.
- Complex sentences evaluate to true or false.
- The **symbols** and the **connectives** together define the **syntax** of the language.
  - Again, syntax is like grammar.

Propositional Logic semantics

- Need to be able to evaluate sentences to true or false.
  - Atomic sentences must have truth assigned (i.e., be assigned T or F).
  - The evaluation of complex sentences is done using **truth tables** for the connectives.

- The **truth tables** define the **semantics** of the language. (When are statements true or false?)

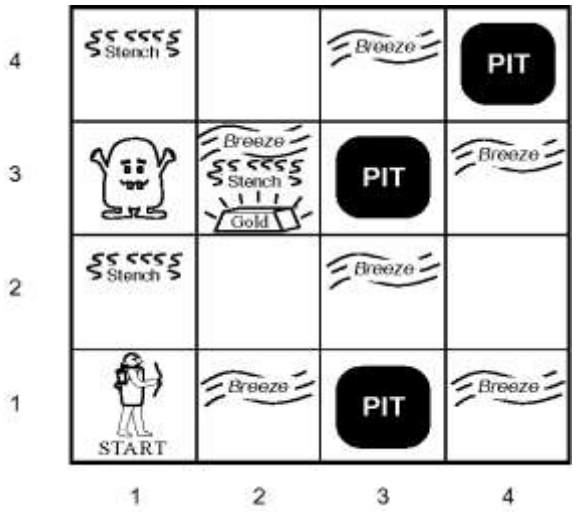
P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
F	F	T	F	F	T	T
F	T	T	F	T	T	F
T	F	F	F	T	F	F
T	T	T	T	T	T	T

- NOTE: Implication means if P is TRUE, then Q is TRUE, otherwise we make no claim.

$$P \Rightarrow Q \quad \text{is} \quad (\neg P) \vee Q$$

The wumpus world problem

- 2D cave connected by passages.
- Some rooms contain a pit into which we fall and perish.
  - We feel a breeze if near a pit.
- One room contains a Wumpus that will eat us.
  - We have one arrow that we can shoot in the direction we are facing.
  - We smell a stench if near the Wumpus.
- Somewhere, there is a pot of gold.



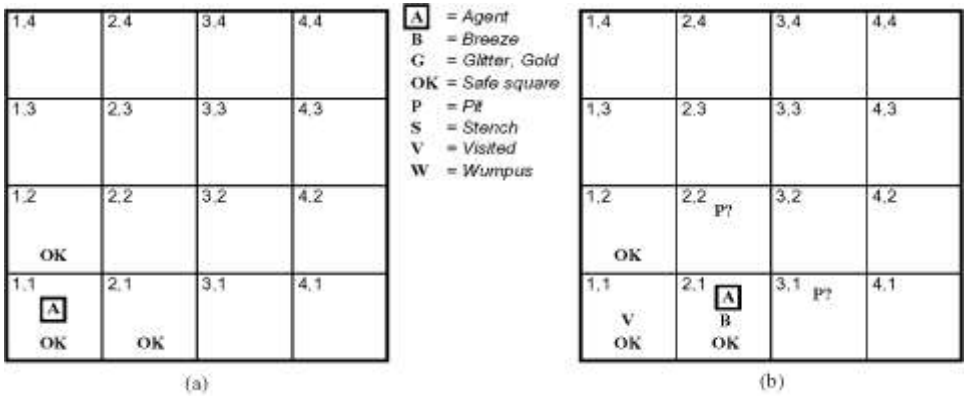
- We can move forward or backward, turn left by 90 degrees or right by 90 degrees.
- Find gold (if possible) and try and kill the Wumpus.
- We will consider some aspects of this example with respect to a logic-based agent/propositional logic.

A Hunt the Wumpus applet can be found at:
http://www.dreamcodex.com/playwumpus.php

Wumpus world characterization

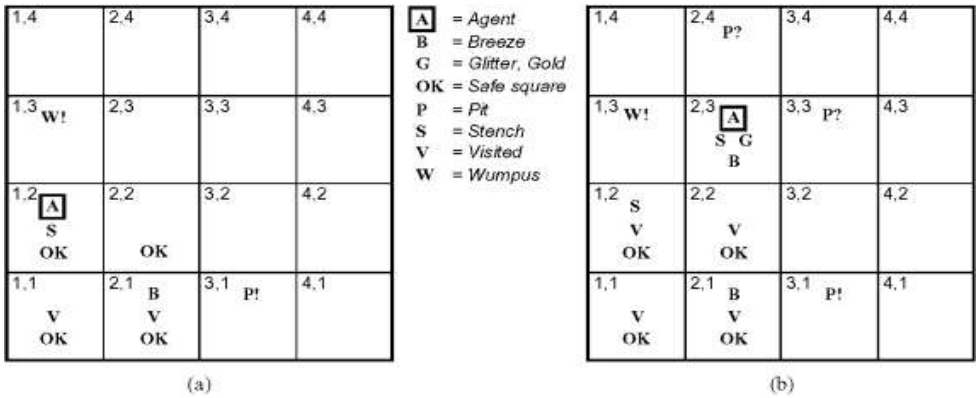
- Deterministic — Yes – outcomes exactly specified
- Static — Yes – Wumpus and Pits do not move
- Discrete — Yes
- Single-agent — Yes – Wumpus is essentially a natural feature
- Fully Observable — *No – only local perception*

Partial exploration of the wumpus world



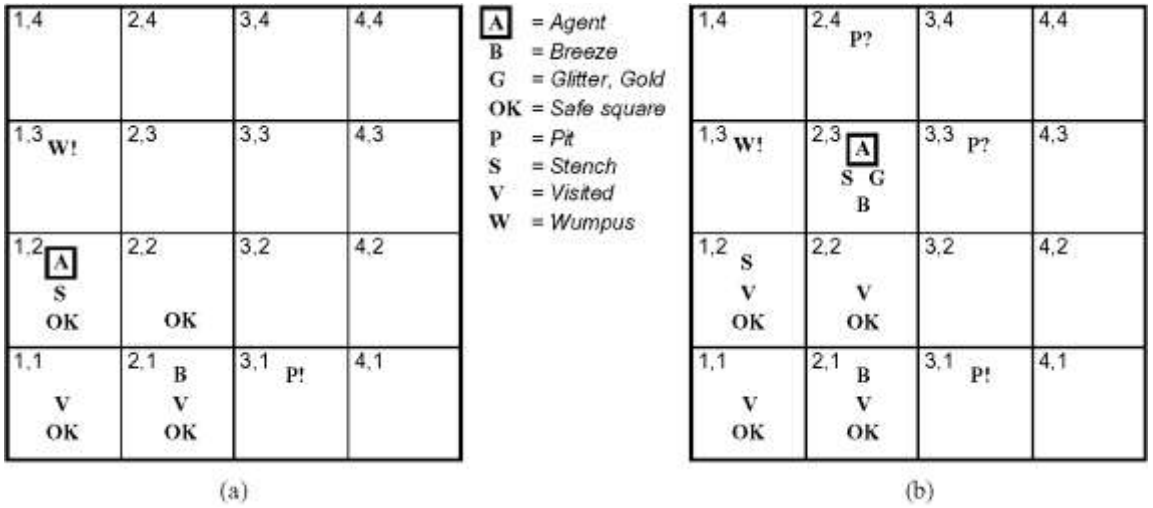
- Nothing sensed in [1,1] so [1,2] and [2,1] are safe.
- Move to [2,1].
- In [2,1] we feel a breeze, so there is a pit in either [2,2] or [3,1], but we don't know which.

Inferencing in the wumpus world



- We move [2,1]->[1,1]->[1,2] and smell a stench, but no breeze.
- We therefore know the Wumpus is in [1,3] or [2,2] but don't know which.

Conclusions via inferencing in the wumpus world



We can make some inferences:

- **We now know that the pit we sensed in [2,1] cannot be [2,2], otherwise we would have sensed a breeze in [1,2]. Therefore there must be a pit in [3,1].**
- **We also know that the Wumpus is not in [2,2], otherwise we would have smelled something in [2,1]. Therefore, the Wumpus must be in [1,3].**

How can we represent the wumpus world?

- We can represent the wumpus world (things we know and things we discover) in terms of logic as follows:

- Consider the propositional symbols (partial formulation):
  - **P(i,j)** is T if there is a pit in (I,J), otherwise F.
  - **B(i,j)** is T if there is a breeze in (I,J), otherwise F.

- We **always** know:
  - $\neg$ **P(i,j)** – no pit in starting square.
  - **B(i,j)** $\Leftrightarrow$ **(P(i+1,j)** $\vee$ **P(i-1,j)** $\vee$ **P(i,j-1)** $\vee$ **P(i,j+1))** – breeze in (i,j) if pit nearby.
  - Etc…

- We can update as we explore:
  - $\neg$ **B(1,1)** – no breeze in square (1,1).
  - **B(2,1)** – breeze in square (2,1).
  - Etc…

- KB is conjunction (AND) of all facts.

How do we perform inferencing in propositional logic?

- Want to find out if something (a sentence G) is entailed by the KB.
  - Sentences in the KB are the premises, and G is our conclusion (can we draw this conclusion?).

- Terminology:
  - **MODEL** – an assignment of truth values to symbols.
  - **VALID** – a sentence is valid (or a tautology) if true under all models.
  - **SATISFIABLE** – a sentence G (or knowledge base KB) is satisfiable if there exists a model m which makes it true. We say that m is a model of G (or KB).

- So, a KB entails a sentence G if all the models of the KB are also models of G.
  - I.e., G is T whenever the KB is T.
  - I.e., **KB** $\Rightarrow$ ***G*** is **valid**.

Inferencing with truth tables

- One way to check models is to enumerate them and form a truth table for **KB** $\Rightarrow$ ***G***

$P_1, P_2, \cdots, P_n$	$S_1, S_2, \cdots, S_m$	$KB = S_1 \wedge S_2 \wedge \cdots \wedge S_m$	G	$KB \Rightarrow G$
$F, F, \cdots F$	$\{T,F\},\{T,F\},\cdots,\{T,F\}$	$\{T,F\}$	$\{T,F\}$	$\{T,F\}$
$F, F, \cdots T$	$\cdots$	$\{T,F\}$	$\{T,F\}$	$\{T,F\}$
$T, T, \cdots T$	$\cdots$	$\{T,F\}$	$\{T,F\}$	$\{T,F\}$

Models

Sentences in KB (premise)

KB

Sentence (conclusion)

Entailment (tautology?)

- Potentially explosive since for **n** inputs, 2^n models.

Example using truth tables

- P is "It is hot", Q is "It is humid" and R is "It is raining". (SYMBOLS).
- P^Q=>R ("If it is hot and humid, then it is raining"), Q=>P ("If it is humid, then it is hot"), Q ("It is humid") – KB.
- Question: Is it raining? (i.e., is R entailed by KB?)

P, Q, R	$P \wedge Q \Rightarrow R$	$Q \Rightarrow P$	Q	KB	R	$KB \Rightarrow R$
T, T, T	T	T	T	T	T	T
T, T, F	F	T	T	F	F	T
T, F, T	T	T	F	F	T	T
T, F, F	T	T	F	F	F	T
F, T, T	T	F	T	F	T	T
F, T, F	T	F	T	F	F	T
F, F, T	T	T	F	F	T	T
F, F, F	T	T	F	F	F	T

- So, R is entailed by the KB and we can conclude it is raining.

Inferencing using rules

- Idea is to use **INFERENCING RULES** that allow us to deduce new sentences (conclusions) that are true when old sentences (premises) are true.

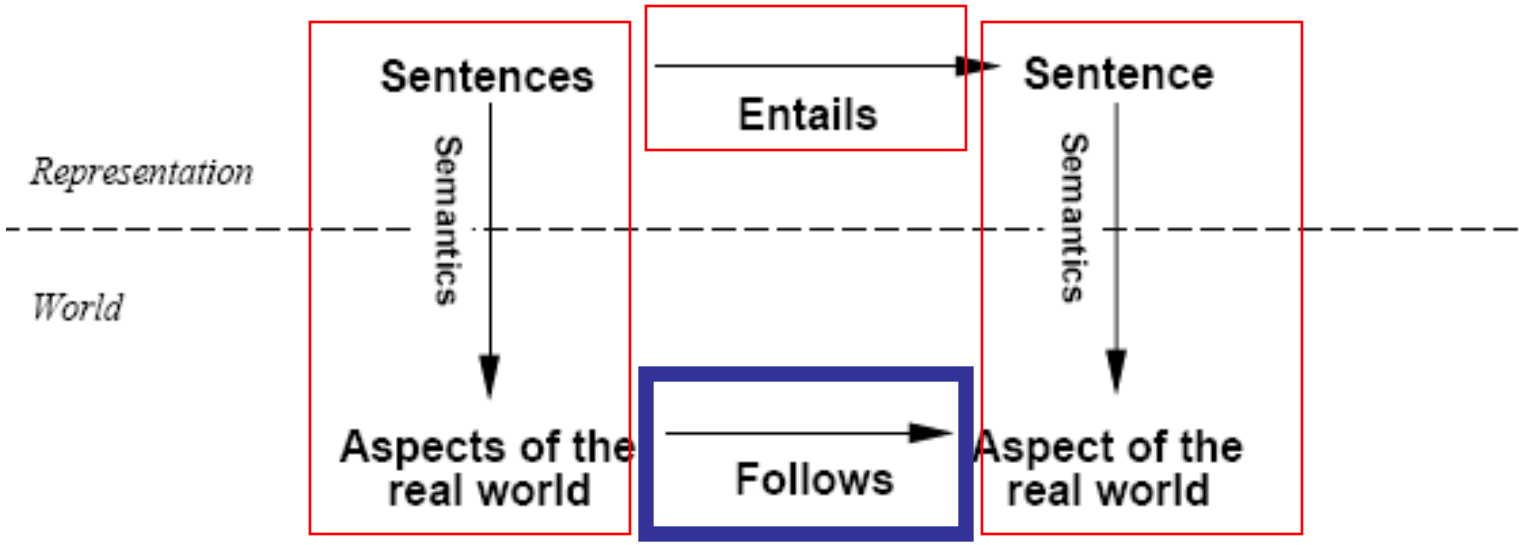
- Terminology:
  - **Soundness** – derivation of only TRUE conclusions given TRUE premises.
  - **Completeness** – if something is entailed, we will detect it.

- Truth tables are sound and complete. Should have inferencing rules and methods to use rules that are also sound and complete.

- In worst case, using rules will be as bad as truth table (i.e., effectively full enumeration).

- In best case, much faster since irrelevant knowledge will/should be ignored if rules used.

Sound inference: schematic perspective



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.

Types of rules

- We can consider the following inferencing rules:

Rule Name	Premises	Derived Conclusion
Modus Ponens	$A, A \Rightarrow B$	B
And Introduction	A, B	$A \wedge B$
And Elimination	$A \wedge B$	A
Double Negation	$\neg\neg A$	A
Unit Resolution	$A \vee B, \neg B$	A
Resolution	$A \vee B, \neg B \vee C$	$A \vee C$

modus ponendo ponens (Latin for *mode that affirms by affirming*;often abbreviated to **MP** or **modus ponens**)
If A then B, A therefore B

- Inferencing rules provide a sequence of sentences and conclusions. The last sentence is what we are trying to deduce.
  - A sequence of inferencing rules leading to our conclusion is a **PROOF**.

Example of inferencing with rules

- P is "It is hot", Q is "It is humid" and R is "It is raining". (SYMBOLS).
- P^Q=>R ("If it is hot and humid, then it is raining"), Q=>P ("If it is humid, then it is hot"), Q ("It is humid") – KB.
- Question: Is it raining? (i.e., is R entailed by KB?)

Step		Reason
1	Q	(premise)
2	$Q \Rightarrow P$	(premise)
3	P	(modus ponens) (1,2)
4	$(P \wedge Q) \Rightarrow R$	(premise)
5	$P \wedge Q$	(and-intro) (1,3)
6	R	(and-elim) (4,5)

Using resolution; Shaping the problem

- Can develop a sound and complete algorithm using **only** resolution rule.

- Need to convert KB and query to Conjunctive Normal Form (**CNF**).
  - Eliminate bi-conditional through equivalence $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$.
  - Eliminate implication through equivalence $A \Rightarrow B \equiv (\neg A) \vee B$.
  - Eliminate double negatives: $\neg (\neg A) \equiv A$, $\neg(A \vee B) \equiv \neg A \wedge \neg B$, $\neg (A \wedge B) \equiv \neg A \vee \neg B$.
  - Distribute $\vee$ over $\wedge$.

Using resolution; proof by contradiction

- So how do we conclude something is entailed by the KB?
- **PROOF by CONTRADICTION – assume our conclusion is false, and look for a contradiction. If found, the opposite of our assumption must be true!**
- In other words, to show **KB** $\models$ **S** $\equiv$ **(**$\neg$ **KB)** $\vee$ **S**, we show that **KB** $\wedge$ **(**$\neg$ **S)** is **UNSATISFIABLE**.

Resolution Algorithm

```
1. RESOLUTION(KB,S) {
2.   old_clauses = {set of clauses in CNF representing KB ∧ ¬ S}
3.   new_clauses = {}

4.   loop {
5.     for each pair of clauses (i,j) in old_clauses {
6.         resolve_clauses = RESOLVE(C_i,C_j);
7.             // all possible clauses found by resolving (i,j)
8.         if ([] ∈ resolve_clauses) return true;
9.                     // empty clause; contradicion.
10.        new_clauses = new_clauses ∪ resolve_clauses;
11.    }
12.    if (new_clauses ⊆ old_clauses) return F
13.    old_clauses = new_clauses;
14.}
```

- Without details, accept that this algorithm is both sound and complete.

Horn clauses

They are named after the logician Alfred Horn, who first pointed out the significance of such clauses in 1951, in the article "On sentences which are true of direct unions of algebras", Journal of Symbolic Logic 16, 14-21.

- Inference with resolution can be more than is required.
- Consider CNF (Conjunctive, Normal Form) knowledge base in which every sentence is composed of at **MOST ONE POSITIVE LITERAL**.

$$\neg l_1 \vee \neg l_2 \vee \cdots \neg l_k \vee l_m$$

- Can convert this into the form of an implication operation.

$$\begin{aligned} \neg l_1 \vee \neg l_2 \vee \cdots \neg l_k \vee l_m &\equiv \neg(l_1 \wedge l_2 \wedge \cdots \wedge l_k) \vee l_m \\ &\equiv (l_1 \wedge l_2 \wedge \cdots \wedge l_k) \Rightarrow l_m \end{aligned}$$

- We can then use either **forward chaining** (data-driven) or **backward-chaining** (goal driven) and (**effectively**) **Modus Ponens** to determine truth of something.
  - Can visualize using **AND-OR** graphs.

Forward chaining

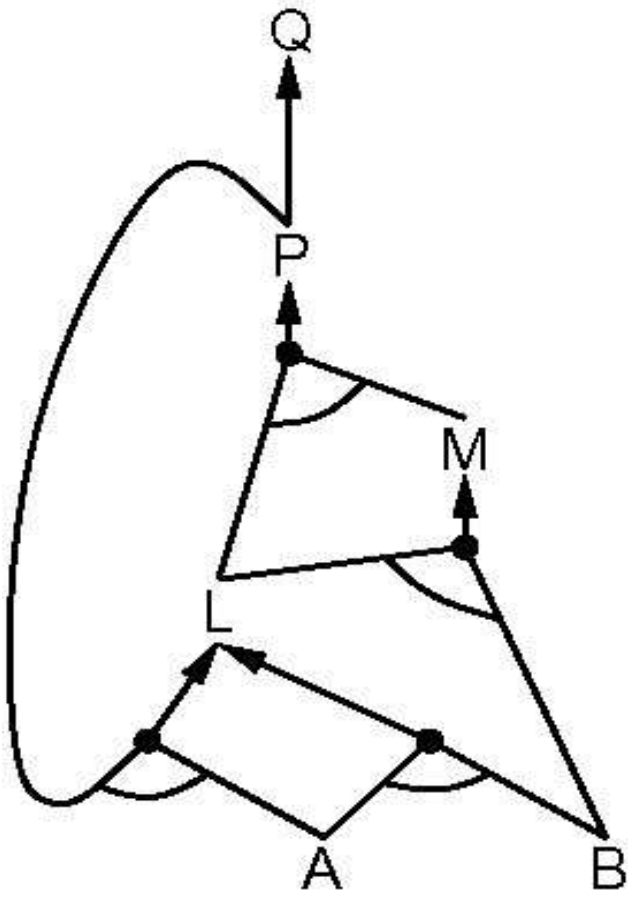
- Forward-chaining:
  - Take unit literals/symbols in the KB and add to queue.
  - Use them to evaluate the premises of implications. When an implication becomes true, its conclusion (a literal/symbol) is true, and it is added to the queue.
  - Repeat until question answered, or nothing else to do.
  - **Data-driven** since we start from what we know and go forward towards goal.

Backward chaining

- Backward-chaining:
  - Take question and add any implications that can prove it true to queue.
  - Examine premises of queued implications and try to prove them by adding implications for the premises into the queue.
  - Repeat until all premises are known to be true or until can't find a way to prove a premise.
  - **Goal-driven** since we start from goal and works backwards to prove what is needed.

And-Or Graphs Visualized

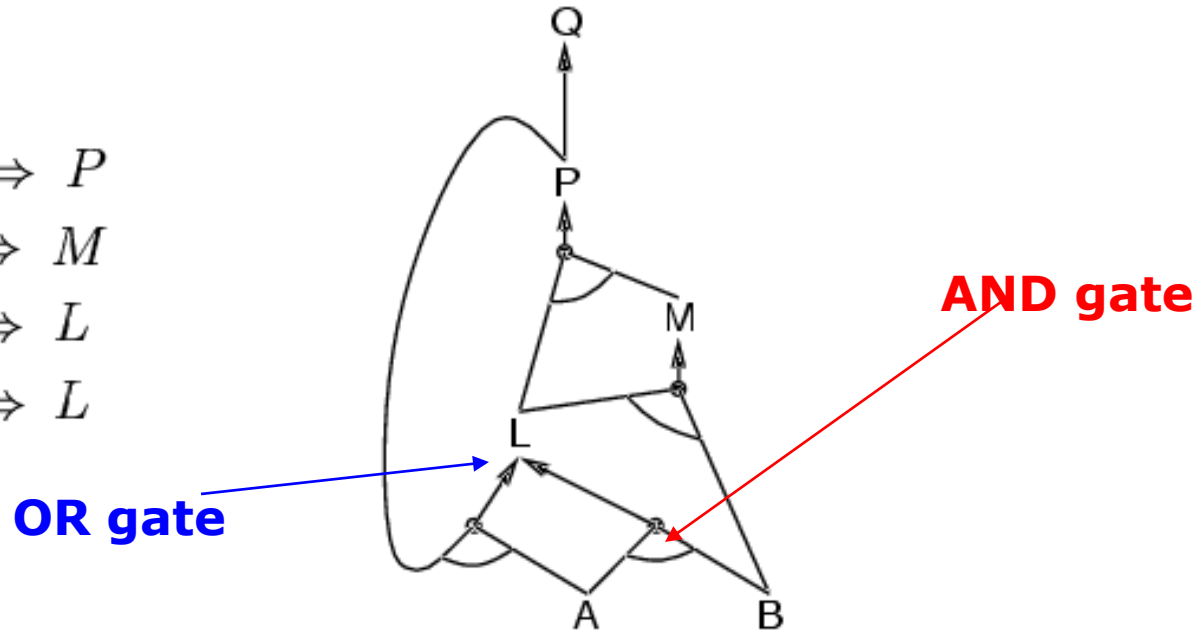
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



Forward chaining

- 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 is sound and complete for Horn KB

Backward chaining

Idea: work backwards from the query *q*

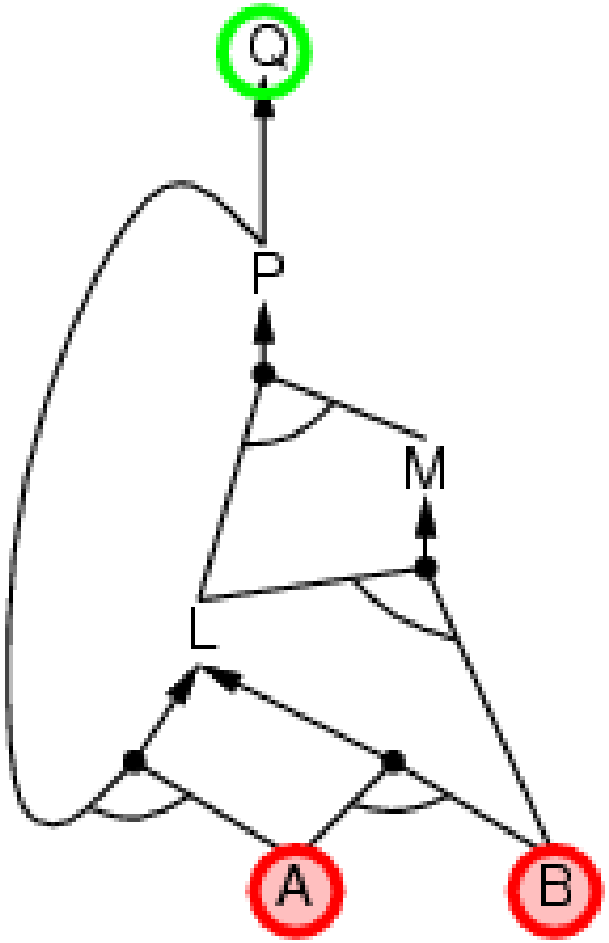
- check if *q* is known already, or
- prove by BC all premises of some rule concluding *q*
- Hence BC maintains a stack of sub-goals that need to be proved to get to q.

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

Avoid repeated work: check if new sub-goal

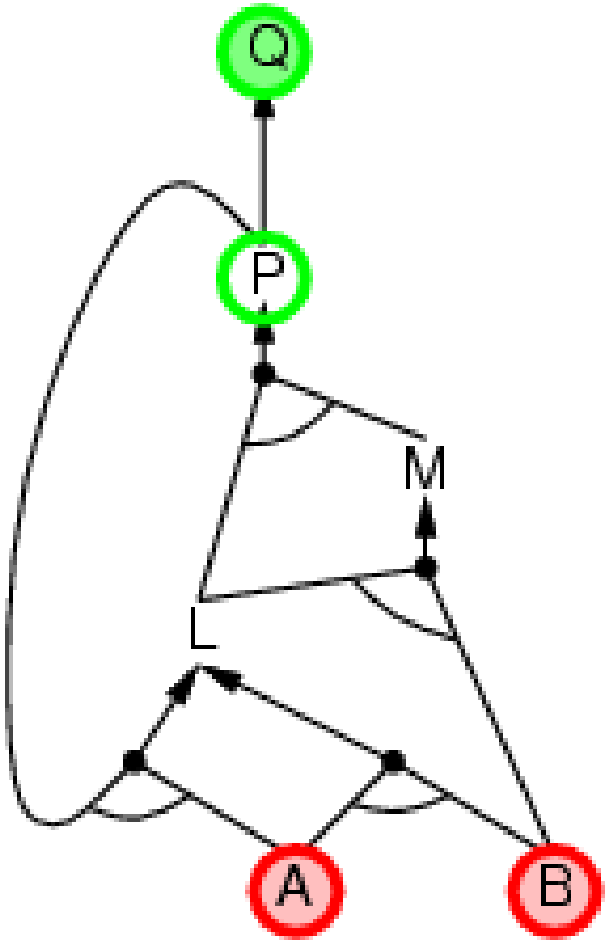
1. has already been proved true, or
2. has already failed

Backward chaining example



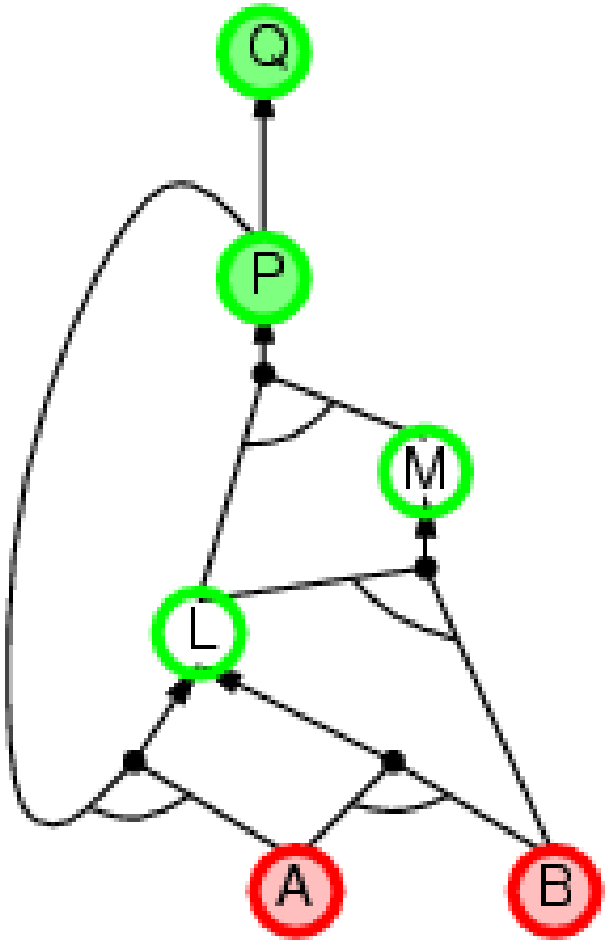
A
B
Q?

Backward chaining example



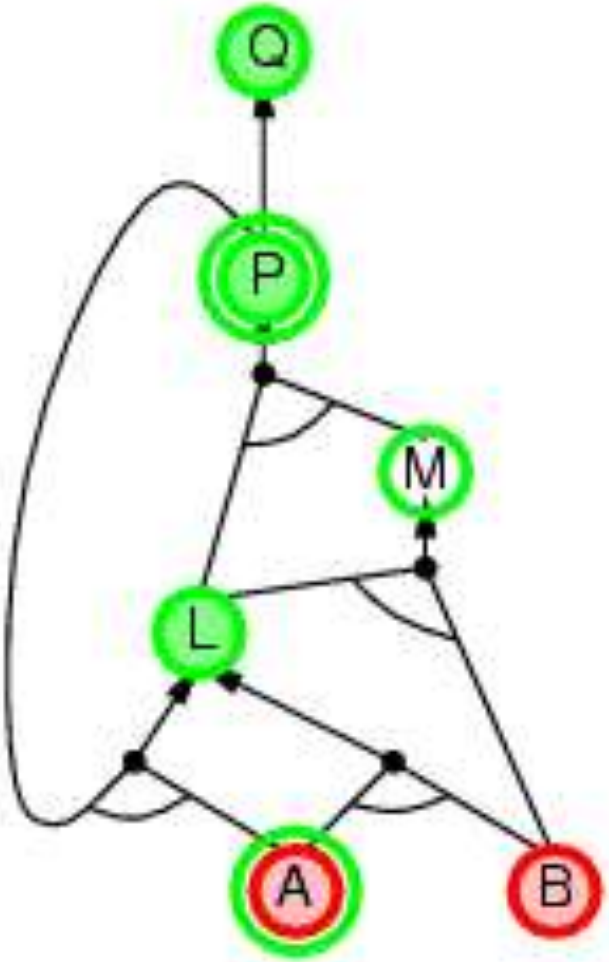
A
B
Q?<=P?

Backward chaining example



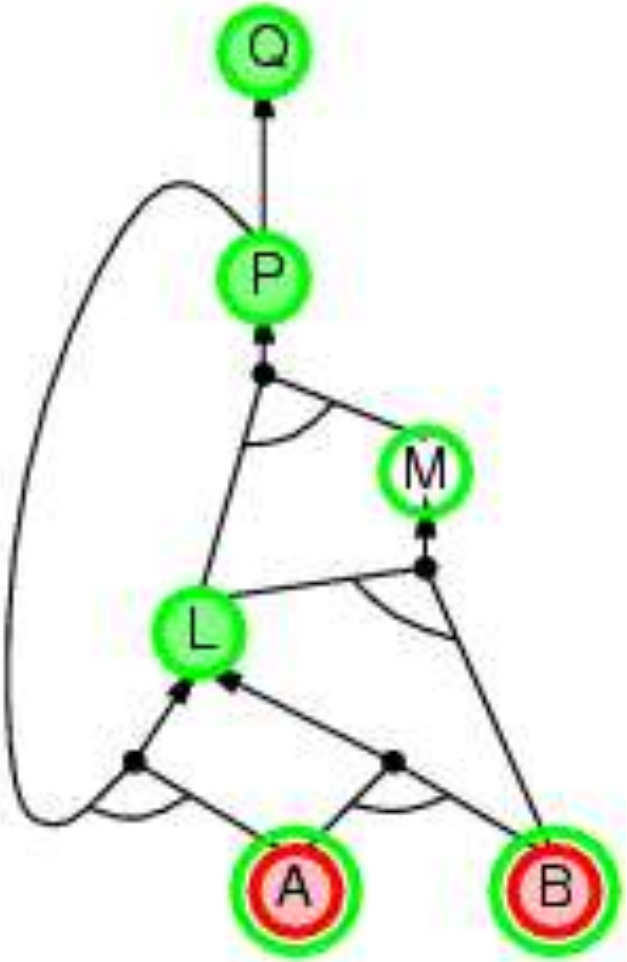
A
B
Q?<=P?
L? ∧ M? => P?

Backward chaining example



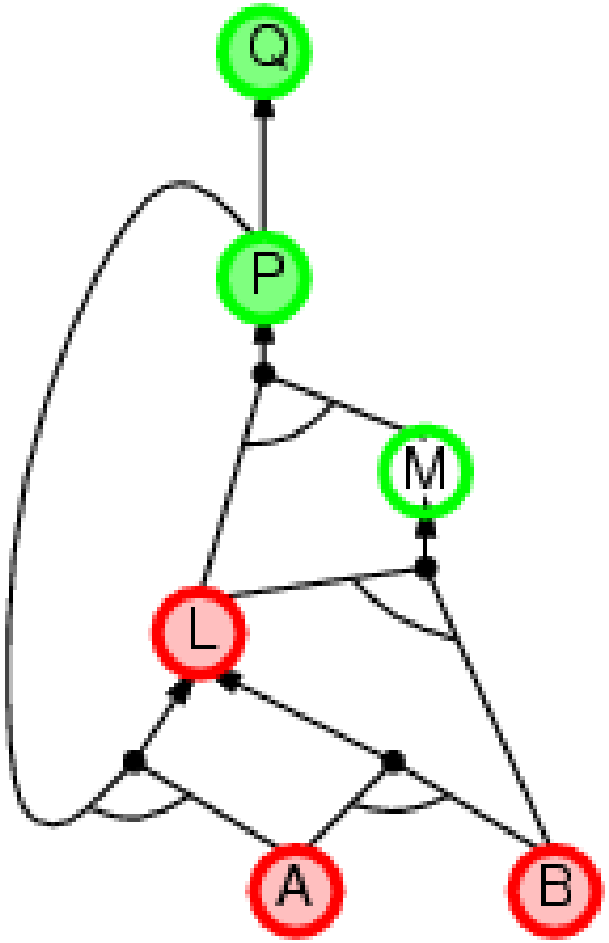
A
B
Q?<=P?
L? ∧ M? => P?
P? ∧ A => L?

Backward chaining example



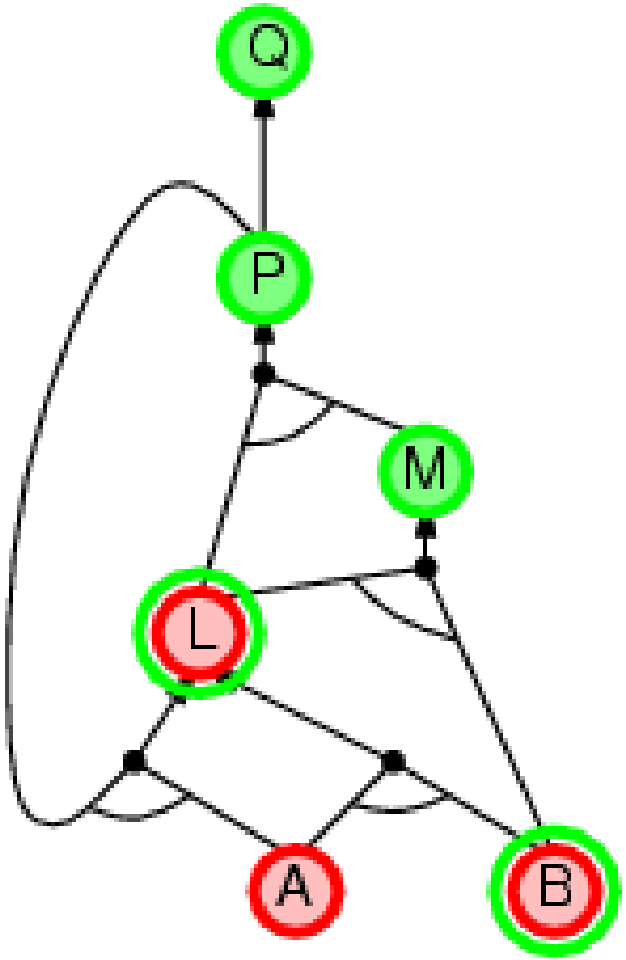
A
B
Q?<=P?
L? ∧ M? => P?
P? ∧ A => L?
A ∧ B => L

Backward chaining example



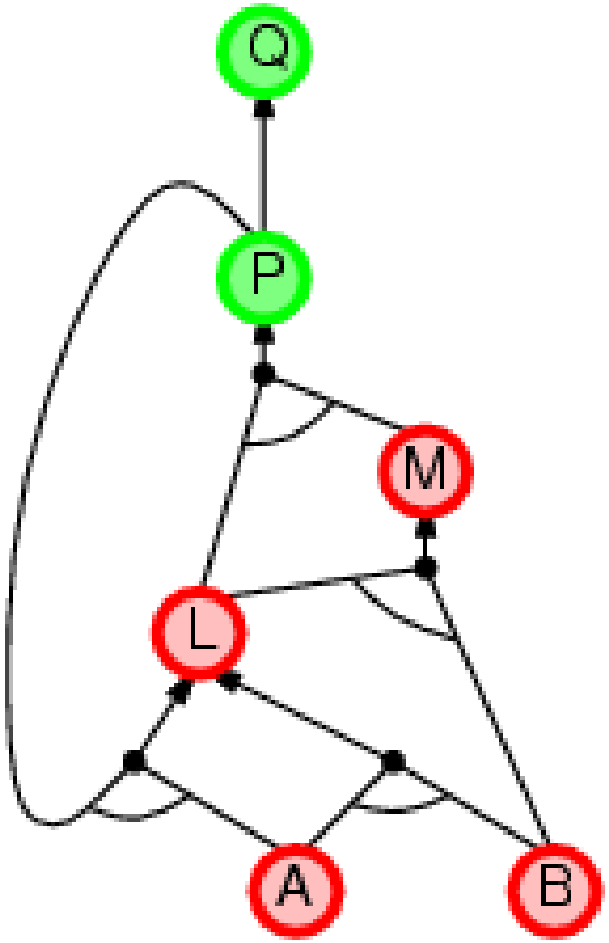
A
B
Q?<=P?
L? ∧ M? => P?
P? ∧ A => L?
A ∧ B => L

Backward chaining example



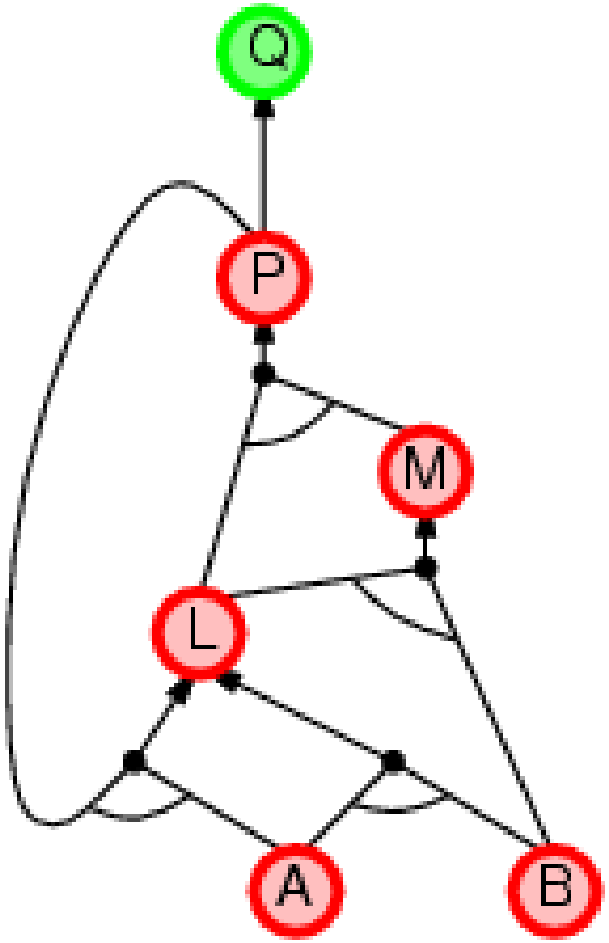
A
B
Q?<=P?
L? ∧ M? => P?
P? ∧ A => L?
A ∧ B => L
L ∧ B => M

Backward chaining example



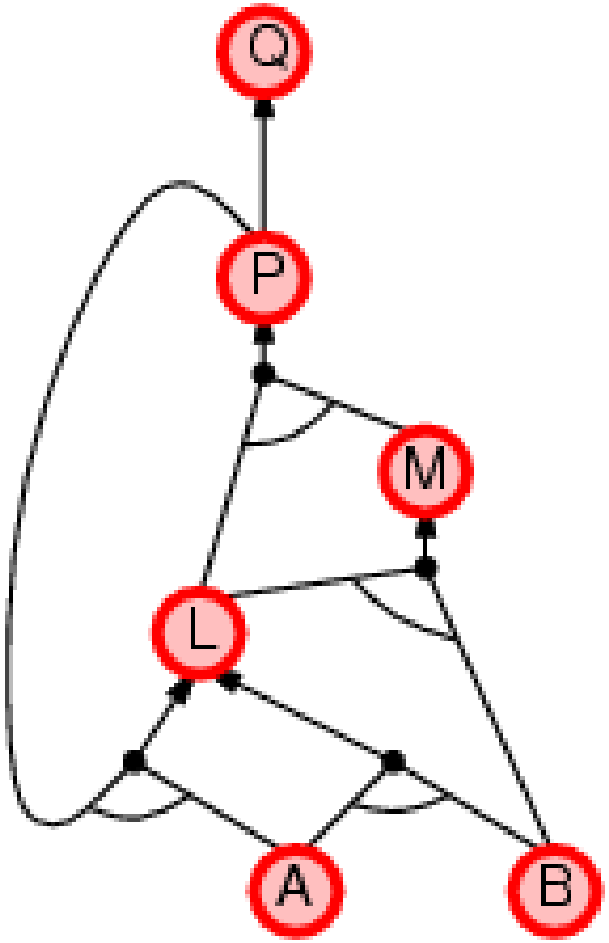
A
B
Q?<=P?
L ∧ M => P?
P? ∧ A => L?
A ∧ B => L
L ∧ B => M

Backward chaining example



A
B
Q?<=P
L ∧ M => P
P? ∧ A => L?
A ∧ B => L
L ∧ B => M

Backward chaining example



A
B
Q <= *P*
L ∧ *M* => *P*
P ∧ *A* => *L*
A ∧ *B* => *L*
L ∧ *B* => *M*

Effective inferencing in practice

- We actually already talked about this when discussing CSP.
- Since propositional logic can be converted into CNF expressions, a query becomes a question of determining if a sentence in CNF is **satisfiable**.
- So, we can use **SAT solvers** to do this:
  - Complete Tree-based (DPLL) algorithms (zChaff, BerkMin, GRASP, etc).
  - Incomplete repair/improvement methods (WalkSAT).

First order logic

- Propositional logic is limited in several ways.
  - Hard to represent information concisely.
  - Must deal with facts that are either TRUE or FALSE.

- Idea is to build a **more powerful logic** (use foundation of propositional logic) by **adding more expressive concepts**.

- **First Order Logic** allows for the representation of **Objects, Functions** on objects and **Relations** between objects.

- The concept of Objects and Relations will allow us to represent almost all grammatically correct English sentences in First Order Logic.

User defined symbols in first order logic

- **CONSTANT SYMBOLS** – that represent individual objects in the world.
  - E.g., Mary, Bill, Book, Three, etc.

- **FUNCTION SYMBOLS** – that represent functions used to map objects to objects.
  - E.g., Brother(Mary), SonOf(Bill,Mary), Cosine(10), etc.

$$f : D^m \rightarrow D$$

- **PREDICATE SYMBOLS** – that represent relationships between objects in the sense that objects are mapped to TRUE or FALSE.
  - E.g., Brother(Rick,Father(John)) – Rick and John are constant symbols (objects), Fasther is a function symbol and Brother is a predicate symbol returning TRUE or FALSE.

$$p : D^m \rightarrow \{T, F\}$$

Logic define symbols in first order logic

- First order logic provides some symbols on its own:
  - **TRUTH SYMBOLS** – that are TRUE and FALSE.
  - **VARIABLE SYMBOLS** – e.g., A, B, X, etc.

Sentences in first order logic

- Sentences are built from **TERMS** and **ATOMS** (or atomic sentences).
- **TERM** – is an expression referring to an object; constant symbols, function symbols and variable symbols are terms.
- **ATOM** (**ATOMIC SENTENCE**) – an n-ary predicate symbols that has value TRUE or FALSE. Each of the n-ary arguments are terms.
- **SENTENCE** (**COMPLEX SENTENCE**) – an atom, or a bunch of atoms joined together using logical connectives (like AND, OR, IMPLICATION, BICONDITIONAL, NOT, …).

Quantifiers in first order logic

- Logic also provides **variable quantifiers** that allow the expression of properties for entire collections of objects.
- **UNIVERSAL QUANTIFIER (∀)** – means "for all".
- **EXISTENTIAL QUANTIFIER (∃)** – means "there exists".
- Quantifiers are used with variable symbols.
- Quantifiers can be used in sentences too:
  - Let X be a variable symbol and P be a sentence.
  - Then **(∀ X, P(X))** is a sentence and **(∃ X, P(X))** is a sentence.

Universal quantifier (∀) explained

- Like an "AND" in the sense that the sentence **(∀ X, P(X))** means **P** is **true** for all values in the domain of X.
- Commonly used with the implication operator (⇒) to for rules.
- E.g., "All cats are mammals" becomes **(∀ X CAT(X) ⇒ MAMMAL(X))**.
  - **NOTE:** we can also read this as "if X is a CAT, then X is also a MAMMAL.

Existential quantifier (∃) explained

- Like an "OR" in the sense that the sentence **(∃ X, P(X))** means **P** is **true** for some (at least one) value in the domain of X.
- Commonly used with the and operator (∧) to list properties about an object.
- E.g., "Some sister of spot is a cat" becomes **(∃ X SISTER(X,SPOT)) ∧ CAT(X))**.

Nesting and mixing quantifiers

- Switching the order of multiple universal quantifiers does not change the meaning;
  - $\forall X, \forall Y\ P(X,Y) \Leftrightarrow \forall Y, \forall X, P(X,Y)$
- Switching the order of multiple existential quantifiers does not change the meaning.
  - $\exists X, \exists Y\ P(X,Y) \Leftrightarrow \exists Y, \exists X, P(X,Y)$

Nesting and mixing quantifiers

- Switching the order of a universal quantifier and an existential quantifier **DOES** change meaning.
  - E.g., is LOVES(X,Y) means "X loves Y" then
    - $\forall$ **X,** $\exists$ **Y P(X,Y)** means "everyone loves someone"
    - $\exists$ **Y,** $\forall$ **X P(X,Y)** means "someone is loved by everyone"

- Negation of quantifiers:
  - $\neg \exists$ **X, P(X)** is equivalent to $\forall$ **X,** $\neg$ **P(X)**
  - $\neg \forall$ **X, P(X)** is equivalent to $\exists$ **X,** $\neg$ **P(X)**

Examples of building sentences

- Almost all grammatically correct sentences can be represented in first order logic.
  - "You can fool some of the people all of the time".
  - "You can fool all of the people some of the time".
  - "no one likes taxes".
  - "basketball players are tall".
  - "some people like anchovies".
  - "purple mushrooms are poisonous".
  - "some purple mushrooms are poisonous".
  - "no purple mushrooms are poisonous".

Inferencing in first order logic

- Use same rules as in propositional logic:

Rule Name	Premises	Derived Conclusion
Modus Ponens	$A, A \Rightarrow B$	B
And Introduction	A, B	$A \wedge B$
And Elimination	$A \wedge B$	A
Double Negation	$\neg\neg A$	A
Unit Resolution	$A \vee B, \neg B$	A
Resolution	$A \vee B, \neg B \vee C$	$A \vee C$

- Additional rules to handle universal quantifier and existential quantifier.

Quantifier rules – Universal Elimination

- **UNIVERSAL ELIMINATION**
  - Given a sentence **P(X)** containing a universally quantified variable **X**:
  - any (and all) terms ***g*** without variables in the domain of **X** can be substituted for **X** and the result **P(*g*)** is TRUE.

- E.g., ∀ **X, EATS(BOB,X)**
  - we can infer **EATS(BOB,*CHOCOLATE*)** if *CHOCOLATE* is in the domain of X.

Quantifier rules – Existential Elimination

- **EXISTENTIAL ELIMINATION**
  - Any existentially quantified variable in a true sentence can be replaced by a constant symbol not already in the knowledge base.
  - The resulting symbol (sentence) is true.

- E.g., ∃ **X, EATS(BOB,X)**
  - we can infer **EATS(BOB,*C*)** if *C* is not already in the knowledge base.
  - May not know what Bob eats, but we can assign a name to it, in this case *C*.

Quantifier rules – Existential Introduction

- **EXISTENTIAL INTRODUCTION**
  - Given a sentence **P**, a variable **X** not in **P**, and a term **g** in **P** (without variables):
  - we can infer $\exists$ **X, P(X)** by replacing **g** with **X**
- E.g., **DRINKS(ANDREW,BEER)** we can infer $\exists$ **X, DRINKS(ANDREW,X)**
- Why can't we do universal introduction as well?
  - E.g., from **DRINKS(ANDREW,BEER)** infer $\forall$ **X, DRINKS(ANDREW,X)**?

Inferencing example

- Consider the following:
  - "All bill's dogs are brown. Bill owns fred. Fred is a dog".
  - Can we infer "fred is brown"?

- **TRANSLATE INTO FIRST ORDER LOGIC:**
  - "All bill's dogs are brown" -
    $$\forall \mathbf{X}\ \mathbf{DOG(X)} \wedge \mathbf{OWNS(bill,X)} \Rightarrow \mathbf{BROWN(X)}$$
  - "Bill owns fred" – **OWNS(bill,fred)**
  - "Fred is a dog" – **DOG(fred)**

- **APPLY INFERENCING RULES:**
  - AND-INTRODUCTION - **DOG(fred)** $\wedge$ **OWNS(bill,fred)**
  - UNIVERSAL ELIMINATION -
    DOG(fred) $\wedge$ **OWNS(bill,fred)** $\Rightarrow$ **BROWN(fred)**
  - MODUS PONENS - **BROWN(fred)**

Generalized Modus Ponens

- Combines the steps of And-Introduction, Universal-Elimination and Modus Ponens into one step.
- Can build an inferencing mechanism using Generalized Modus Ponens (much like using Modus Ponens in propositional logic).
- Generalized Modus Ponens is not complete. There might be sentences entailed by a knowledge base we can not infer.
  - Has to do with structure of clauses.
  - Is complete for Horn clauses (positive literals, conjunctions only).
    - E.g., $\forall$ **X DOG(X)** $\wedge$ **OWNS(bill,X)** $\Rightarrow$ **BROWN(X)**

Forward Chaining

- Talked about this with Modus Ponens and propositional logic.
- Generalized Modus Ponens can be used with **forward chaining**.
- **Forward chaining**:
  - Start with a set of facts (KB) and derive new information.
  - Continue until goal is reached, always updating the KB with new conclusions.
  - **Data-driven** since no particular path to goal.
    - Cf. Uninformed search strategies.

Backward Chaining

- Talked about this with Modus Ponens and propositional logic.
- Generalized Modus Ponens can be used with **backward chaining**.
- **Backward chaining**:
  - Start with a set of facts (KB) and a goal to be proven.
  - Look for implication sentences that would allow our goal to be proven.
  - Consider these sentences to be sub-goals, and try to prove them.
  - **Goal-driven** since we always strive to prove something goal related.
- E.g.,"Is fred brown?" which we represent as **BROWN(fred)?**
  - We match the implication $\forall$ **X DOG(X)** $\wedge$ **OWNS(bill,X)** $\Rightarrow$ **BROWN(X) if X=fred.**
  - We know we have two sub-goals, namely **DOG(fred)?** And **OWNS(bill,fred)?**

Unification

- Generalized Modus Ponens requires **pattern matching**.
  - E.g., in our previous example, we needed to make **DOG(fred)** and **DOG(X)** match by using **variable instantiation**.
  - **Variable instantiation** means finding a value for **X** to make expressions equivalent.

- **Unification** is the algorithm to match two expressions.
  - Returns the **variable substitutions** to make two sentences match, or **failure** if no match possible.
  - **UNIFY(p,q)** = θ are matched where θ is the list of substitutions in **p** and **q**.

- UNIFY(p,q) should return the matching that places the **least** restrictions on variable values.
  - Resulting substitution list is called the **most general unifier (MGU)**.

Unification rules

Rules to unify p and q:

- Function symbols and predicate symbols must have identical names and number of arguments.
- Constant symbols unify with only identical constant symbols.
- Variables unify with other variable symbols, constant symbols or function symbols.
- Variable symbols may not be unified with other terms in which the variable occurs. E.g., X cannot unify with G(X) since this will lead to G(G(G(G(…G(X))))). This is an **occur check error**..

Simplified pseudocode for unification

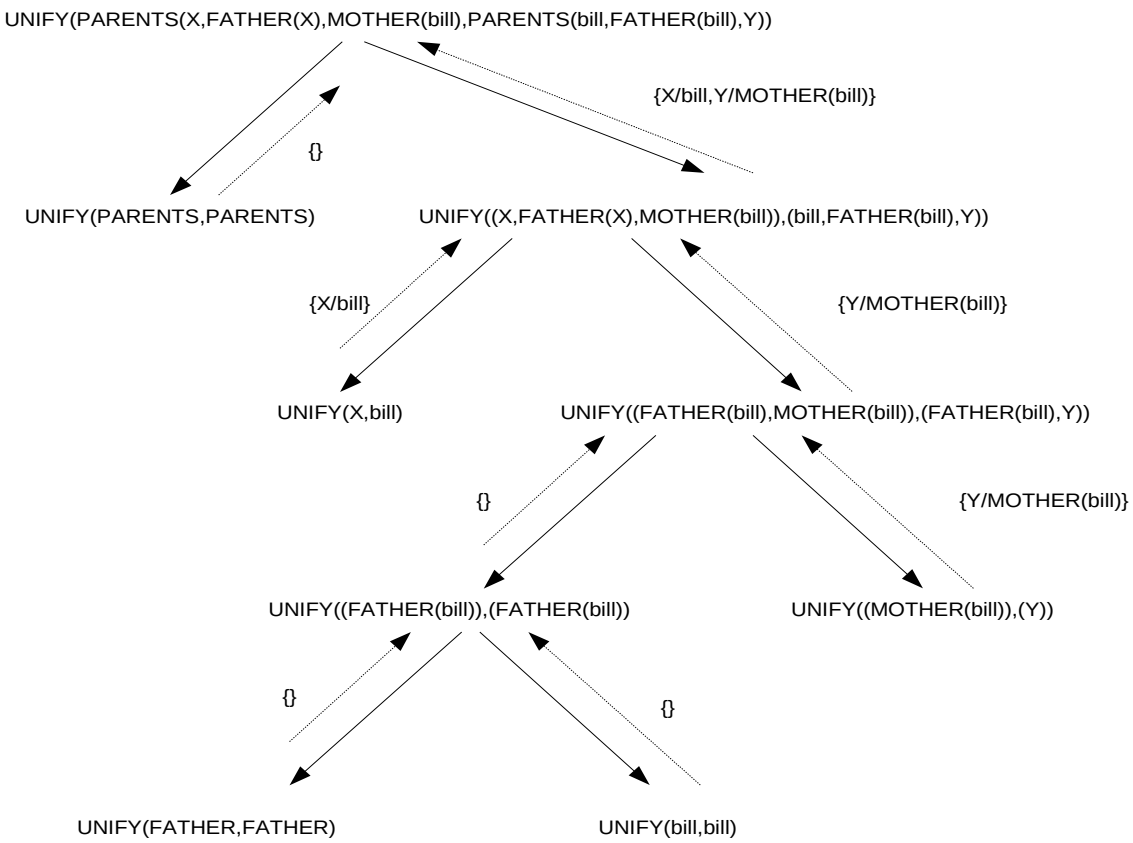
```
1. UNIFY(p,q,θ) {
2.    /* scan p and q left to right. */
3.    /* let r and s be terms where there is disagreement between p and q. */

4.    /* examine each r/s pair.
5.    if (IsVariable(r)) {
6.       if (r ∈ s) return failure; // occur check error.
7.       else {
8.                θ = θ ∧ r/s.
9.                /* apply substitutions to p and q */
10.               UNIFY(p,q,θ);
11.   }
12.   if (IsVariable(s)) {
13.      if (s ∈ r) return failure; // occur check error.
14.      else {
15.               θ = θ ∧ s/r.
16.               /* apply substitutions to p and q */
17.               UNIFY(p,q,θ);
18.   }
19.}
```

- Additional checks for matching constant, function and predicate symbols.

Example of unification

- Unify: **PARENTS(X,FATHER(X),MOTHER(bill))** and **PARENTS(bill,father(bill),Y)**.



Resolution

- Sound and complete for first order logic.
- Idea is to prove something using **refutation** (proof by contradiction).
  - We want to prove something (clause/sentence).
  - We add its negation into the KB.
  - We attempt to reach a contradiction as a result of the addition of our negated goal.
  - If contradiction reached, our goal must be true.
- Note: same idea as in propositional logic, but need to handle **quantifiers** and **variables**.

Resolution

Resolution is step-by-step:

1. Convert problem into first order logic expressions.
2. Convert logic expressions into clause form.*
3. Add negation of what we intend to prove (in clause form) to our logic expressions.
4. Use resolution rule to produce new clauses that follow from what we know.
5. Produce a contradiction that proves our goal.

- **Any variable substitutions used during the above steps are those assignments for which the opposite of the negated goal is true.**
  - **Has to do with Unification, more later.**

Conversion to clause form

- Recall resolution for clauses $\alpha \vee \beta$, $\neg \beta \vee \gamma$ resolves to $\alpha \vee \gamma$.
- So resolution works with pairs (or groups) of disjuncts to produce new disjuncts.
- Therefore, need to convert first order logic into disjuncts (Conjunctive Normal Form, or clause form).
  - Individual clauses *conjuncted* together, each clause made up of *disjunctions* of literals.

Conversion procedure

1. Eliminate all $\Rightarrow$ using the fact that $A \Rightarrow B \equiv \neg A \vee B$.

2. Reduce scope of negation to the predicate level using:

   a) $\neg (\neg A) \equiv A$
   b) $\neg (\exists X), A(X) \equiv (\forall X), \neg A(X)$
   c) $\neg (\forall X), A(X) \equiv (\exists X), \neg A(X)$
   d) $\neg (A \wedge B) \equiv \neg A \vee \neg B$
   e) $\neg (A \vee B) \equiv \neg A \wedge \neg B$

3. Rename variables so that they are different if bound by different quantifiers.

Conversion procedure

4. Eliminate existential quantifiers. This is known as **Skolemization**.

   a) E.g., $\exists$ **X, DOG(X)** may be replaced by **DOG(fido)** where **fido** is a **Skolem constant**.

   b) E.g., $\forall$ **X** ($\exists$ **Y MOTHER(X,Y)**) must be replaced by $\forall$ **X MOTHER(X,m(X))** where **m(X)** is a **Skolem function of X**.

5. Drop all $\forall$ universal quantifiers. Assume all variables to be universally quantified.

6. Convert each expression to a conjunction of disjuncts using $\boldsymbol{A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)}$.

7. Split conjuncts into separate clauses and rename variables in different clauses.

Example of conversion

- Convert: "Everyone who loves all animals is loved by someone"

  $\forall X [\forall Y Animal(Y) \Rightarrow Loves(X,Y)] \Rightarrow [\exists Y Loves(Y,X)]$

1. Remove $\Rightarrow$ from expression:

   $\forall X [\neg\forall Y \neg Animal(Y) \vee Loves(X,Y)] \vee [\exists Y Loves(Y,X)]$

2. Reduce scope of negation:

   $\forall X [\exists Y \ Animal(Y) \wedge \neg Loves(X,Y)] \vee [\exists Y Loves(Y,X)]$

3. Rename variables:

   $\forall X [\exists Y \ Animal(Y) \wedge \neg Loves(X,Y)] \vee [\exists Z Loves(Z,X)]$

4. Eliminate $\exists$ using Skolemization:

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

Example of conversion

5. Drop universal quantifiers:
   [Animal(F(X)) ∧ ¬Loves(X, F(X))] ∨ [Loves(G(X), X)]

6. Convert to conjuction of disjuncts:
   [Animal(F(X)) ∨ Loves(G(X), X)] ∧ [¬Loves(X, F(X)) ∨ Loves(G(X), X)]

7. Separate into clauses:
   [Animal(F(X)) ∨ Loves(G(X), X)]
   [¬Loves(X, F(X)) ∨ Loves(G(X), X)]

8. Rename variables (again) in different clauses:
   [Animal(F(X)) ∨ Loves(G(X) ,X)]
   [¬Loves(W, F(W)) ∨ Loves(G(W), W)]

Example of resolution refutation

- Consider the following story:

  "Anyone passing his or her artificial intelligence exam and winning the lottery is happy. But anyone who studies or is lucky can pass all his exams. Pete did not study but is lucky. Anyone who is lucky wins the lottery. Is Pete happy?".

Example of resolution refutation

Step 1: Change sentences to first order logic:

1. "Anyone passing his or her artificial intelligence exam and winning the lottery is happy"

   $(\forall X)\ (PASS(X,ai) \wedge WIN(X,lottery) \Rightarrow HAPPY(X))$

2. "Anyone who studies or is lucky can pass all his exams"

   $(\forall X\ \forall Y)\ (STUDIES(X) \vee LUCKY(X) \Rightarrow PASS(X,Y))$

3. "Pete did not study but is lucky"

   $\neg\ STUDY(pete) \wedge LUCKY(pete)$

4. "Anyone who is lucky wins the lottery"

   $(\forall X)\ (LUCKY(X) \Rightarrow WINS(X,lottery))$

Example of Resolution Refutation

Step 2: Convert all sentences into clause form:

1. $(\forall X)\ (PASS(X,ai) \wedge WIN(X,lottery) \Rightarrow HAPPY(X))$ *gives:*

   $\neg PASS(X,ai) \vee \neg WIN(X,lottery) \vee HAPPY(X)$

2. $(\forall X \forall Y)\ (\neg STUDIES(X) \vee LUCKY(X) \Rightarrow PASS(X,Y))$ *gives:*

   $\neg STUDIES(Y) \vee PASS(Y,Z)$

   $\neg LUCKY(W) \vee PASS(W,V)$

3. $\neg STUDY(pete) \wedge LUCKY(pete)$ *gives:*

   $\neg STUDIES(pete)$

   $LUCKY(pete)$

4. $(\forall X)\ (LUCKY(X) \Rightarrow WINS(X,lottery))$ *gives:*

   $\neg LUCKY(U) \vee WINS(U,lottery)$

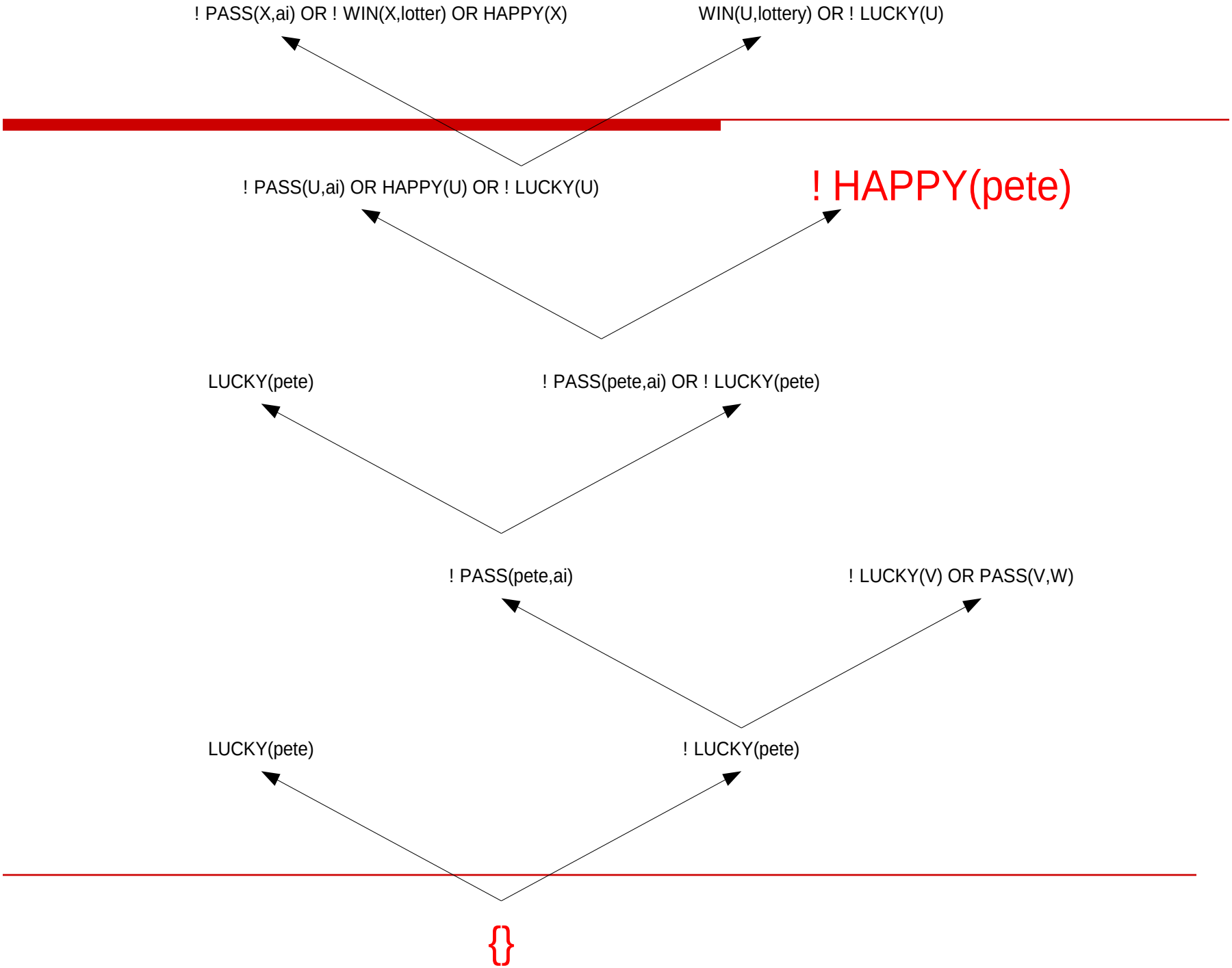
Example of resolution refutation

Step 3: Add negation (in clause form) of what we want to know:

$\neg$ HAPPY(pete)

Step 4: Use resolution (and our negated goal) to build a resolution refutation graph to prove a contradiction.

- When building the graph to prove the contradiction, it should be the case that the negative of the goal gets used somewhere in the proof.
- Contradiction occurs when resolve clauses like A, $\neg$ A
-
- result = {} (NULL set) since A cannot be true and false at the same time!


! PASS(X,ai) OR ! WIN(X,lotter) OR HAPPY(X)
WIN(U,lottery) OR ! LUCKY(U)
! PASS(U,ai) OR HAPPY(U) OR ! LUCKY(U)
! HAPPY(pete)
LUCKY(pete)
! PASS(pete,ai) OR ! LUCKY(pete)
! PASS(pete,ai)
! LUCKY(V) OR PASS(V,W)
LUCKY(pete)
! LUCKY(pete)
{}