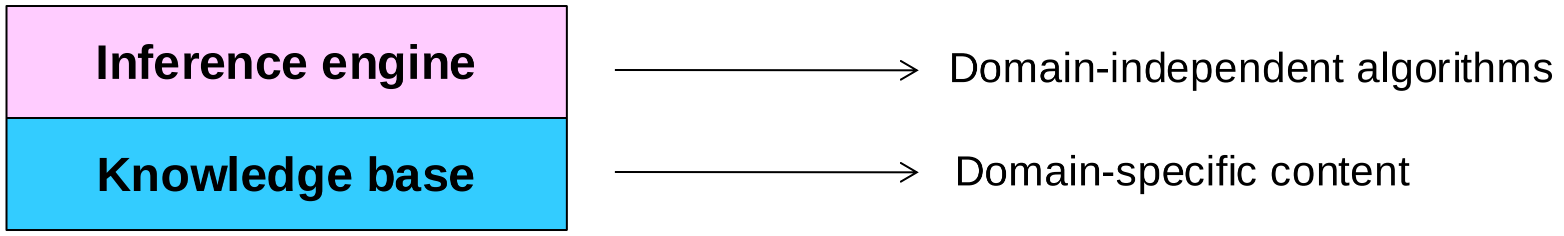
Logic and Automated Reasoning

Knowledge-based agents



- Knowledge base (KB) = set of sentences in a formal language
- Declarative approach to building an agent (or other system):
  - Tell it what it needs to know
- Then it can ask itself what to do - answers should follow from the KB
- Distinction between data and program
- Fullest realization of this philosophy was in the field of expert systems or knowledge-based systems in the 1970s and 1980s

What is logic?

- **Logic** is a formal system for manipulating facts so that true conclusions may be drawn
  - *"The tool for distinguishing between the true and the false" – Averroes (12th cen.)*
- **Syntax:** rules for constructing valid sentences
  - E.g., $x + 2 \geq y$ is a valid arithmetic sentence, $\geq x2y +$ is not
- **Semantics:** "meaning" of sentences, or relationship between logical sentences and the real world
  - Specifically, semantics defines truth of sentences
  - E.g., $x + 2 \geq y$ is true in a world where $x = 5$ and $y = 7$

Overview

- Propositional logic
- Inference rules and theorem proving
- First order logic

Propositional logic: Syntax

- **Atomic sentence:**
  - A *proposition symbol* representing a true or false statement
- **Negation:**
  - If **P** is a sentence, **¬P** is a sentence
- **Conjunction:**
  - If **P** and **Q** are sentences, **P ∧ Q** is a sentence
- **Disjunction:**
  - If **P** and **Q** are sentences, **P ∨ Q** is a sentence
- **Implication:**
  - If **P** and **Q** are sentences, **P ⇒ Q** is a sentence
- **Biconditional:**
  - If **P** and **Q** are sentences, **P ⇔ Q** is a sentence

- **¬, ∧, ∨, ⇒, ⇔** are called *logical connectives*

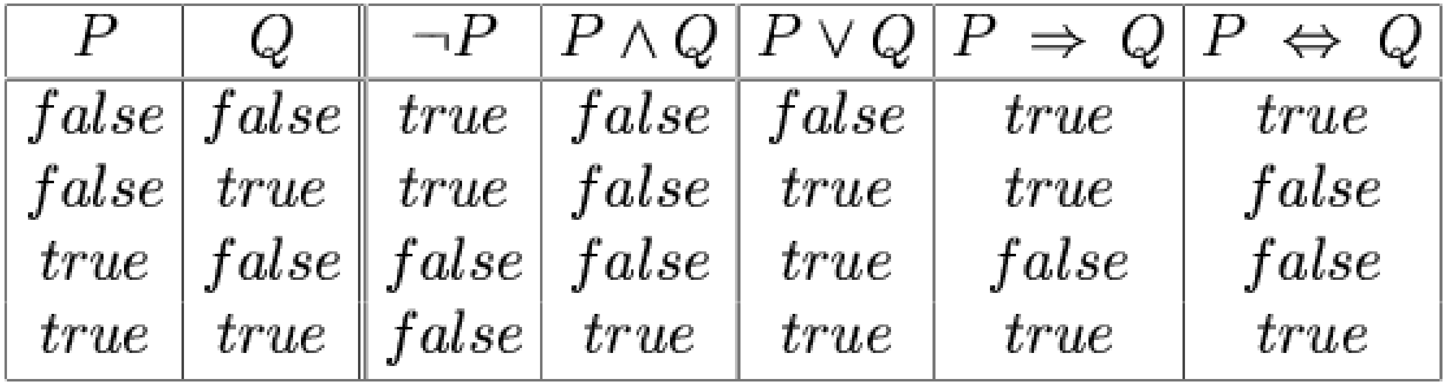
Propositional logic: Semantics

- A **interpretation *I*** specifies the true/false status of each proposition symbol in the knowledge base
- A **model** is an interpretation in which the formula of interest is **True**. Given P, Q and R propositions:
  - Could be: **P** is true, **Q** is true, **R** is false or {P, Q, R'} or {P, Q}
  - With three symbols, there are 8 possible interpretations, and they can be enumerated exhaustively: 000➔111: {P', Q', R'} ➔ {P, Q, R}
- Rules for evaluating truth with respect to a model:

$\neg P$	is true	iff	P	is false			
$P \wedge Q$	is true	iff	P	is true	and	Q	is true
$P \vee Q$	is true	iff	P	is true	or	Q	is true
$P \Rightarrow Q$	is true	iff	P	is false	or	Q	is true
$P \Leftrightarrow Q$	is true	iff	$P \Rightarrow Q$	is true	and	$Q \Rightarrow P$	is true

Truth tables

- A **truth table** specifies the truth value of a composite sentence for each possible assignments of truth values to its atoms.



P	Q	$\neg P$	$P \land Q$	$P \lor Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

- The truth value of a more complex sentence can be evaluated *recursively* or *compositionally*
- *Rain*➔ *umbrella, can carry umbrella in sun!*

Models, Interpretations, Worlds

- An interpretation I (world) is an assignment of values (T,F) to ALL variables in a formula α
- An interpretation is a model for formula α if α is True in that interpretation:
- Given formula α= A V B’: Interpretations: 11; 1,0;01;00
- Models are 10,11,00 Nonmodel: **01**
- Frequently, the interpretation and model are used interchangeably with the context determining the meaning: HERE (in our slides)
- E.g. 10,11,00 are the models in which α is TRUE

  and **01** is the model in which α is FALSE
- Also {A}, {A,B}, {}, are models of α; {B} is not a model.

Models and Interpretations

- ***May specify interpretation by listing positive literals only.***
- P $\oplus$ Q: has 4 interpretations and 2 models.

{P}, {Q} are models and interpretations,

{P,Q} is an interpretation but not a model!

However:

Some people use model to mean interpretation: and distinguish between models in which a formula is true and false!

This may be the approach in some of these slides and may be more: but ***be careful****.*

Logical equivalence

- Two sentences are logically equivalent iff they are true in same models

$(\alpha \land \beta) \equiv (\beta \land \alpha)$	commutativity of $\land$
$(\alpha \lor \beta) \equiv (\beta \lor \alpha)$	commutativity of $\lor$
$((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma))$	associativity of $\land$
$((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma))$	associativity of $\lor$
$\neg(\neg\alpha) \equiv \alpha$	double-negation elimination
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \lor \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \land \beta) \equiv (\neg\alpha \lor \neg\beta)$	de Morgan
$\neg(\alpha \lor \beta) \equiv (\neg\alpha \land \neg\beta)$	de Morgan
$(\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma))$	distributivity of $\land$ over $\lor$
$(\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))$	distributivity of $\lor$ over $\land$

Logical equivalence: Clauses

- Two sentences are logically equivalent iff they are true in same interpretations:
- α→β= α' v β [**same as T ^ α→β V F**] T=true, F=false
- α1 ^ α2 ^ α3 ^ α4→β= α1' v α2' v α3' v α4' v β
- α1 ^ α2 ^ α3 ^ α4→β1 v β2 v β3 =
  α1' v α2' v α3' v α4' v β1 v β2 v β3
- α1 ^ α2 ^ α3 ^ α4→ = α1' v α2' v α3' v α4'
- →β1 v β2 v β3 = β1 v β2 v β3

Logical equivalence: Clauses

- A **literal**: an atom or negated atom: α3, α4’, β1
- A **clause**: disjunction of Literals: α3’ v α4’ v β1
- □: is the **empty clause** –Never True-

Validity, satisfiability

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

e.g., *True*, 1 , $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

A sentence is satisfiable if it is true in some Interpretation (has a model)

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

A sentence is unsatisfiable if it is true in no Interpretation (has no models)

e.g., $A \wedge \neg A$, 0, False, $(A \vee \neg A) \Rightarrow$

Valid is also satisfiable,

Not Valid is either satisfiable or Unsatisfiable:

S1= $A \vee C$, S2= $\neg A \wedge A$

Validity, satisfiability

If sentence S1 is valid then $\neg S1$ is Unsatisfiable $S1=[A \vee \neg A]> \neg S1= \neg A \wedge A$

If sentence S2 is unstaisfiable then $\neg S2$ is Valid. $S2=[\neg A \wedge A]> \neg S2= A \vee \neg A$

If sentence S3 is satisfiable then $\neg S3$ is satisfiable or unsatisfiable: $S3= A \vee C$, $S3= A \vee \neg A$,

If sentence S4 is not valid then $\neg S4$ is Satisfiable or invalid: $S4= A \vee C$, $S4= \neg A \wedge A$,

Entailment

- **Entailment** means that a sentence follows from the premises contained in the knowledge base:

$$KB \models \alpha$$

- *KB* entails sentence *α* if and only if *α* is true in all interpretations where *KB* is true (KB Models)
  - E.g., ***KB* : {**x = 0} entails ***α:*** {x * y = 0}
  - Can *α* be true when KB is false?

  Of course: x=5, ***KB* is false,** x * y = 0 [y=0]➔ ***α is true!!***
- $KB \models \alpha$ iff $(KB \Rightarrow \alpha)$ is *valid or* $(\neg KB \vee \alpha)$ is *valid*
- $KB \models \alpha$ iff $(KB \wedge \neg\alpha)$ is *unsatisfiable: has no models*
- **Negate α and prove** $(KB \wedge \neg\alpha)$ *unsatisfiable:*
- *Refutational proof, proof by contradiction*

Entailment

- **Entailment** Example for ***KB*** $\models$ ***α***
- Let ***KB*** = P➔Q and P, Let ***α***=P, show that ***KB*** $\models$ ***α***
- 2 variable, 4 interpretations.

P	**Q**	**P➔Q**	**{P➔Q, P}**	**Q**	**{P➔Q, P, Q'}**
0	0	1	0	0	**0**
0	1	1	0	1	**0**
1	0	0	0	0	**0**
1	1	1	1	1	**0**

Entailment Example

- Let ***KB*** = R^P➔Q and P, Let ***α***=Q, show that***KB*** ╞ ***α***
- 3 variable, 8 interpretations. Enumeration:

P	Q	R	R^P	{R^P➔Q}	{R^P➔Q, P}	KB ∧ ¬ α
0	0	0	0	1	0	**0**
0	0	1	0	1	0	**0**
0	1	0	0	1	0	**0**
0	1	1	0	1	0	**0**
1	0*	0	0	1	1 ***	**1*** **
1	0	1	1	0	0	**0**
1	1	0	0	1	1	**0**
1	1	1	1	1	1	**0**

Entailment

- Let **α=A ∨ B** and
  KB =(A ∨ C) ∧(B ∨ ¬C)
- Is it the case that KB ⊨ α ?
- Check all possible models -- α must be true whenever KB is true.

A	B	C	**KB** (A ∨ C) ∧ (B ∨ ¬C)	α A ∨ 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** $(A \lor C) \land (B \lor \neg C)$	**α** $A \lor B$
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

$KB \models \alpha$

Entailment

Check that: ***(KB ∧¬α)*** *unsatisfiable*

A	B	C	KB (A ∨ C) ∧ (B ∨ ¬C)	α A ∨ 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

$KB \models \alpha$

Inference

- **Logical inference:** a procedure for generating sentences that follow from a knowledge base KB
- An inference procedure is **sound** if whenever it derives a sentence α, KB ╞ α
  - A sound inference procedure can derive only true sentences
- An inference **procedure** is **complete** if whenever *KB* ╞ α, α can be derived by the **procedure**
  - A complete inference procedure can derive every entailed sentence

Inference: Sound and Complete

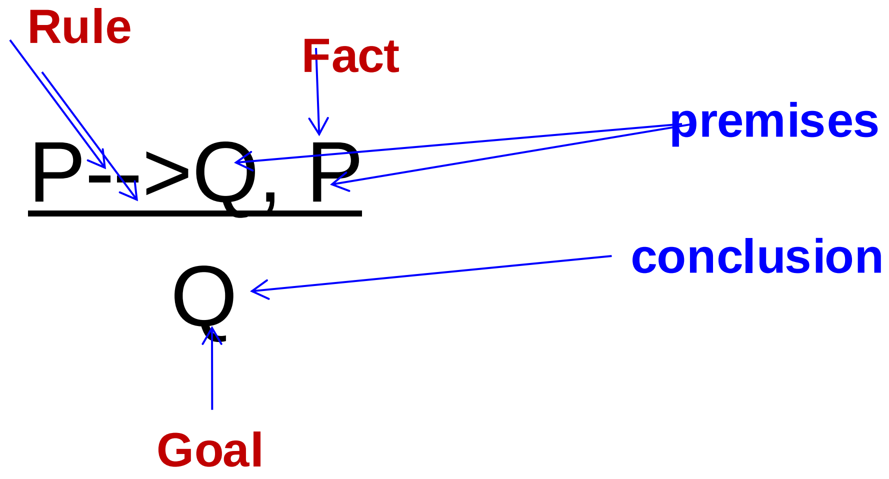
- Can be sound but not complete:
- E.g. Derive nothing from any KB.
- Or from $A \wedge B$ derive only A
- Can be complete but not sound.
- Or from $A \wedge B$ derive A and B **and C** ?
- E.g. Derive everything from any KB.
- Best if both sound and complete: **drives all and only what is derivable**: all the truth and nothing but the truth: كل الحق ولا شيء غير الحق

Inference

- How can we check whether a sentence α is entailed by KB?
- How about we enumerate all possible models of the KB (truth assignments of all its symbols), and check that α is true in every model in which KB is true?
  - Is this sound?
  - Is this complete?
- Problem: if KB contains n symbols, the truth table will be of size 2^n
- Better idea: use *inference rules*, or sound procedures to generate new sentences or *conclusions* given the *premises* in the KB

Inference rules

- Modus Ponens



- And-elimination $$\frac{\alpha \wedge \beta}{\alpha}$$

Inference rules

- And-introduction

$$\frac{\alpha,\ \beta}{\alpha \wedge \beta}$$

- Or-introduction

$$\frac{\alpha}{\alpha \vee \beta}$$

Inference rules

- Double negative elimination

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

- Unit resolution

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

Resolution

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

- Example:
  - α: "The weather is dry"
  - β: "The weather is rainy"
  - γ: "I carry an umbrella"

Resolution

- Examples:
- P v Q, ¬ P v R gives Q v R
- ¬ P v R, P gives R
- ¬ P v R, ¬R v P gives P v ¬P OR R v ¬R =1
- ¬ P, P gives   ., or empty clause or • always false =0

Resolution is complete

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

- To prove KB ╞ α, assume KB ∧ ¬ α
  and derive a contradiction: Refutation proof!
- Rewrite KB ∧ ¬ α as a conjunction of *clauses*, or disjunctions of *literals*
  - *Conjunctive normal form* (CNF) (product of sums)

    (¬P ∨ Q ∨ R) ∧(S ∨ P ∨ T ∨¬R) ∧(Q ∨ S)

  Disjuncts are clauses, sets of literals: {¬P,Q,R}, {S,P,T,¬R},{Q,S}

  Special case: the empty set (empty clause){}, •
- Keep applying resolution to clauses that contain *complementary literals* and adding resulting clauses to the list
  - If there are no new clauses to be added, then KB does not entail α
  - If two clauses resolve to form an *empty clause*, we have a contradiction and KB ╞ α

Complexity of inference

- Propositional inference is ***co-NP-complete***
  - *Complement* of the SAT problem: $\alpha \models \beta$ if and only if the sentence $\alpha \wedge \neg \beta$ is *unsatisfiable*
  - Every known inference algorithm has worst-case exponential running time
- Efficient inference possible for restricted cases

Proof, Refutation Proof

P v Q, P→R, Q→ R: can prove R? Yes:

1- P v Q -- {P, Q}

2- ¬ P v R -- {¬P, R}

3- ¬Q v R -- {¬Q, R}

Clause 9 is a proof of R. But refutation is more convenient!

4- ¬R -- {¬R}

1.	{P, Q}	Premise
2.	{~ P, R}	Premise
3.	{~ Q, R}	Premise
4.	{~ R}	Premise
5.	{Q, R}	1,2
6.	{P, R}	1,3
7.	{~ P}	2,4
8.	{~ Q}	3, 4
9.	{R}	3, 5
10.	{Q}	4, 5
11.	{R}	2,6
12.	{P}	4,6
13.	{Q}	1,7
14.	{R}	6,7
15.	{P}	1,8
16.	{R}	5,8
17.	{}	4,9

Example

If Omar visits Poland (P), then Omar visits Quebec (Q). If it is Monday (M), Omar visits Poland or Québec. Prove that, if it is Monday, then Omar visits Québec.

Query (Goal): M→ Q or ¬ M v Q: =not easy to prove this: so REFUTATION-

Goal negation: M and ¬ Q:: {M}, {¬ Q}

1.	$\{\sim P, Q\}$	Premise
2.	$\{\sim M, P, Q\}$	Premise
3.	$\{M\}$	Negated Goal
4.	$\{\sim Q\}$	Negated Goal
5.	$\{P, Q\}$	3,2
6.	$\{Q\}$	5,1
7.	$\{\}$	6,4

Definite clauses

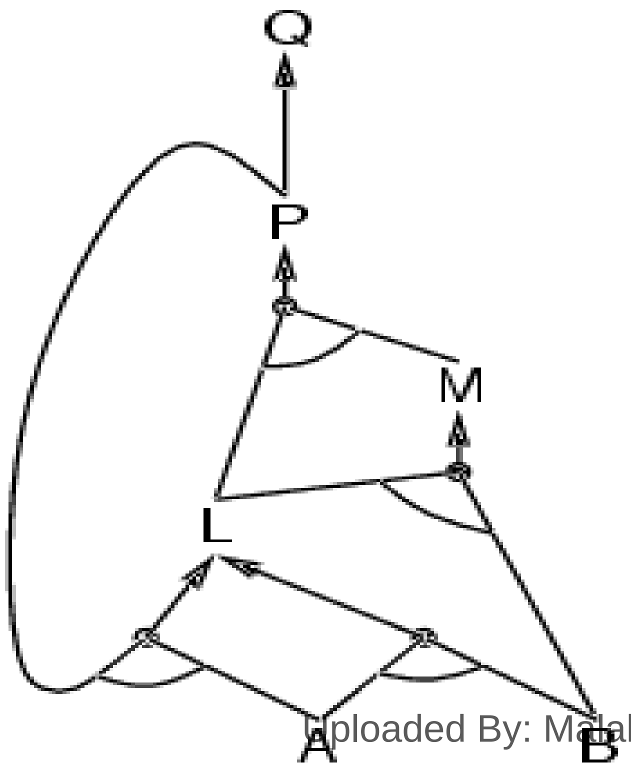
- A **definite clause** is a disjunction with exactly one positive literal
- Equivalent to $\underbrace{(P_1 \wedge \ldots \wedge P_n)}_{\textbf{premise} \text{ or } \textbf{body}} \Rightarrow \underbrace{Q}_{\substack{\textbf{conclusion} \\ \text{or } \textbf{head}}}$

- Basis of logic programming (Prolog)
- Efficient (linear-time) complete inference through *forward chaining* and *backward chaining*

Forward chaining

- Idea: find any rule whose premises are satisfied in the *KB*, add its conclusion to the *KB*, and keep going until query is found. Let Goal be Q

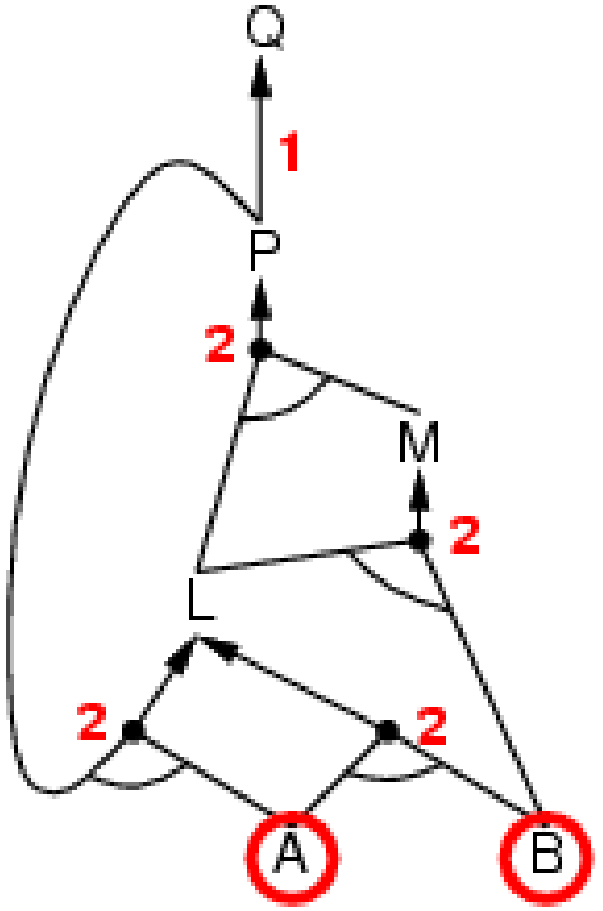
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



Forward chaining example

- AND-OR Graph
  - multiple links joined by an arc indicate conjunction – every link must be proved
  - multiple links without an arc indicate disjunction – any link can be proved

- Empty circles: symbols known to be true but not yet "processed"
- Counts: how many premises of each implication are yet unknown



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

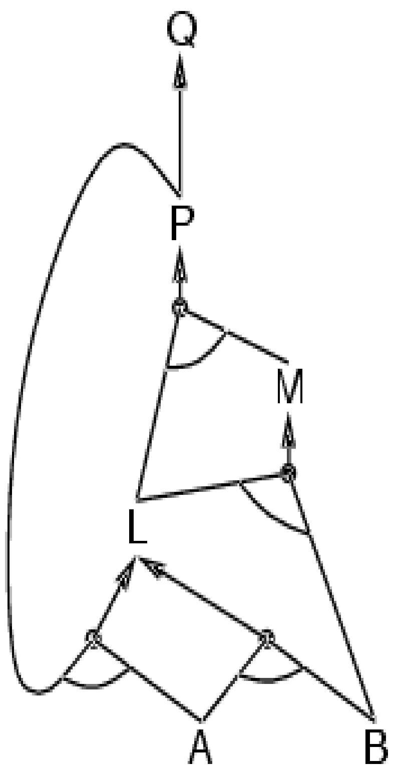
$B \wedge L \Rightarrow M$

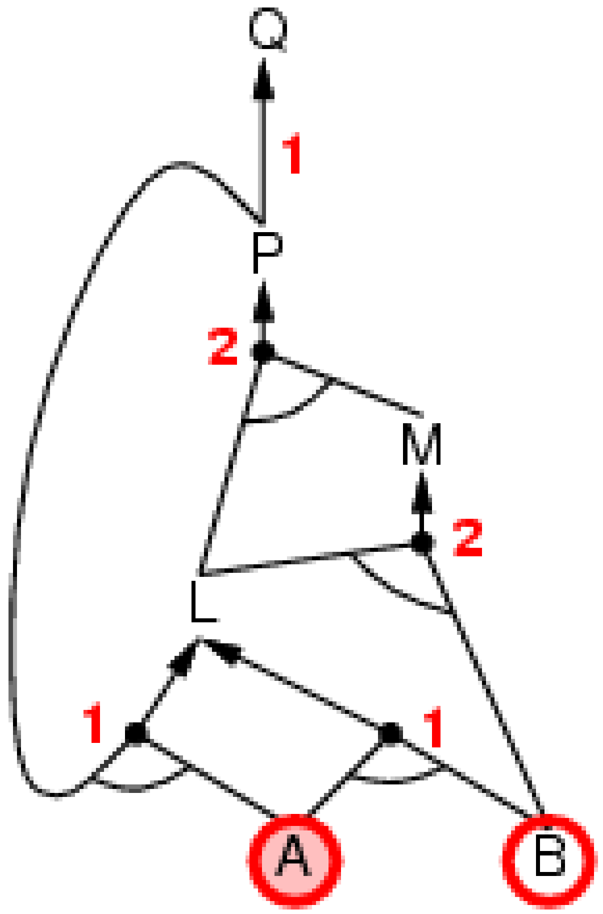
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

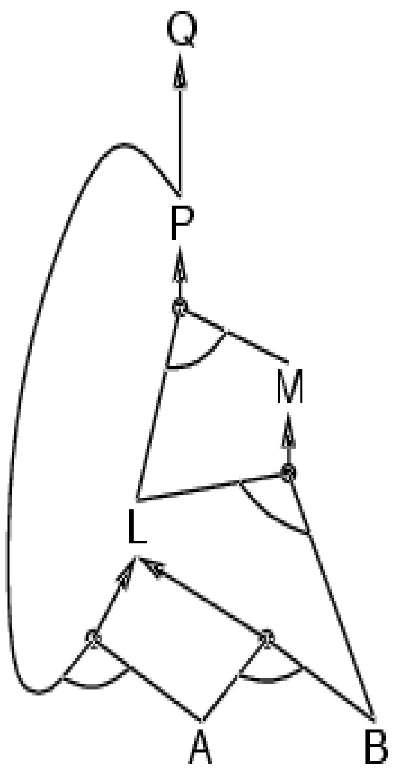
$B \wedge L \Rightarrow M$

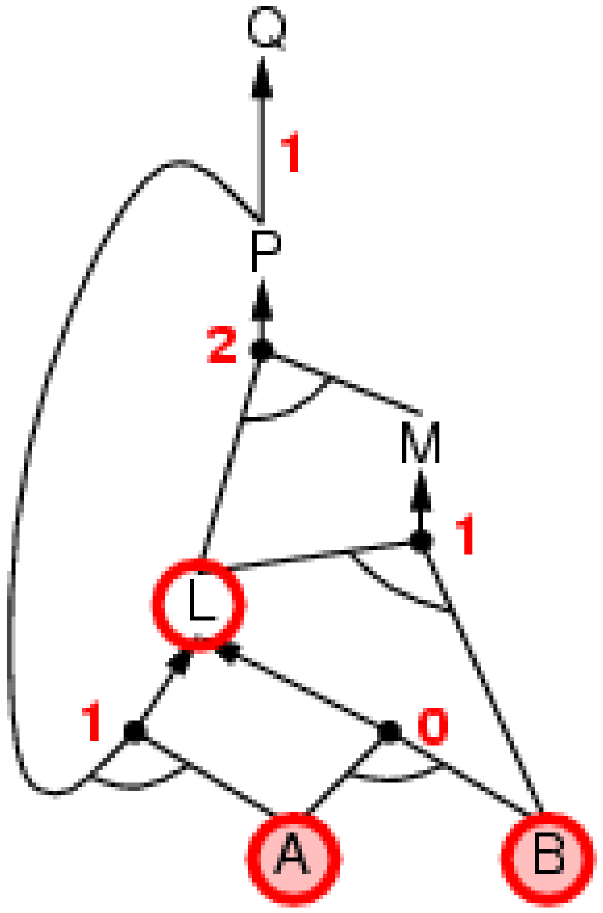
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

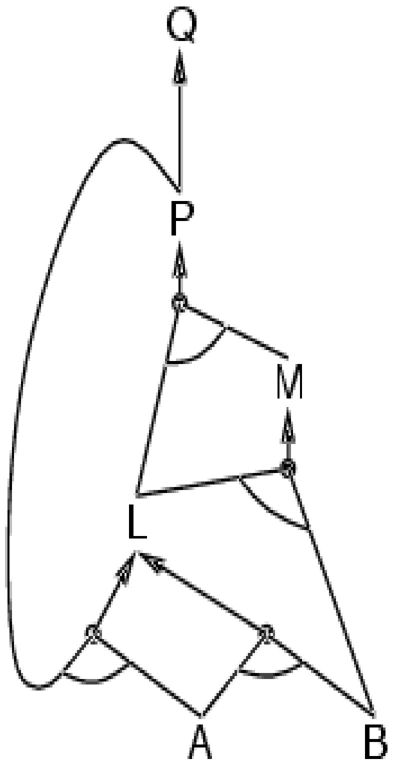
$B \wedge L \Rightarrow M$

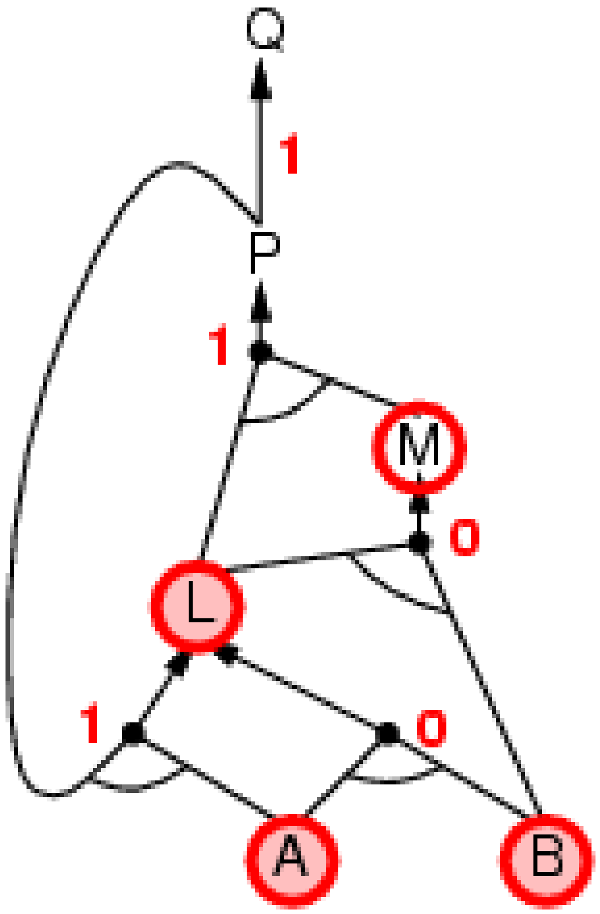
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

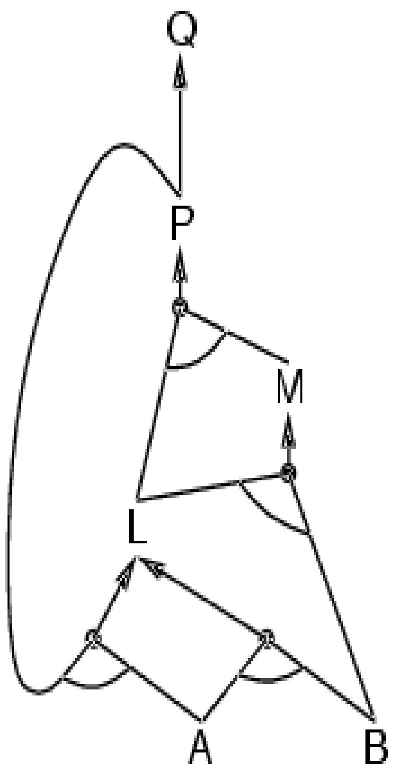
$B \wedge L \Rightarrow M$

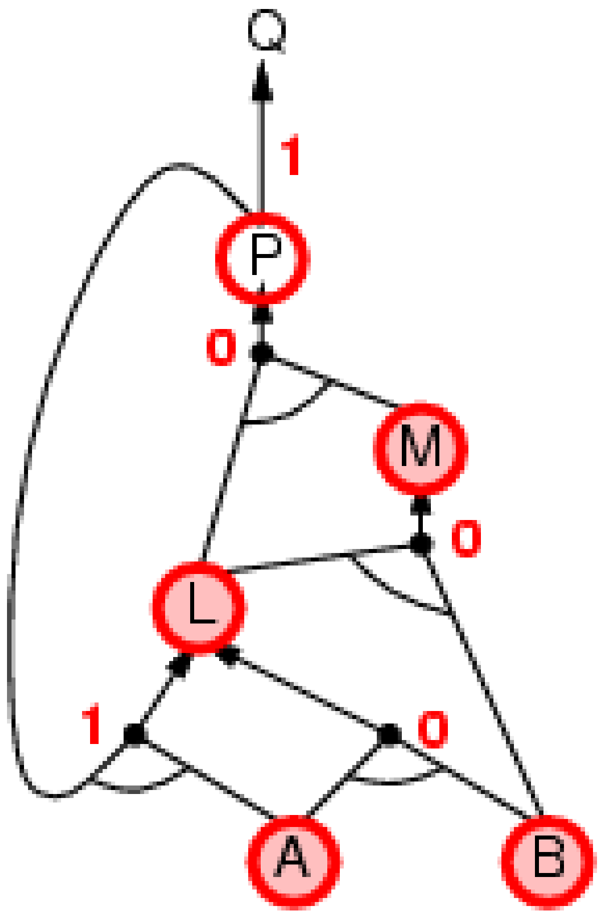
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

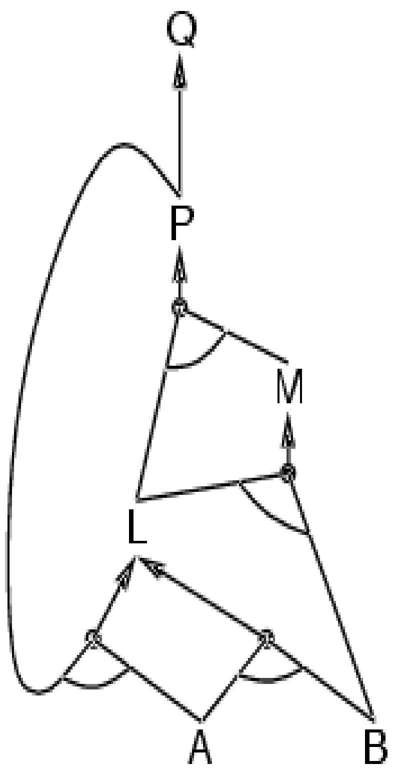
$B \wedge L \Rightarrow M$

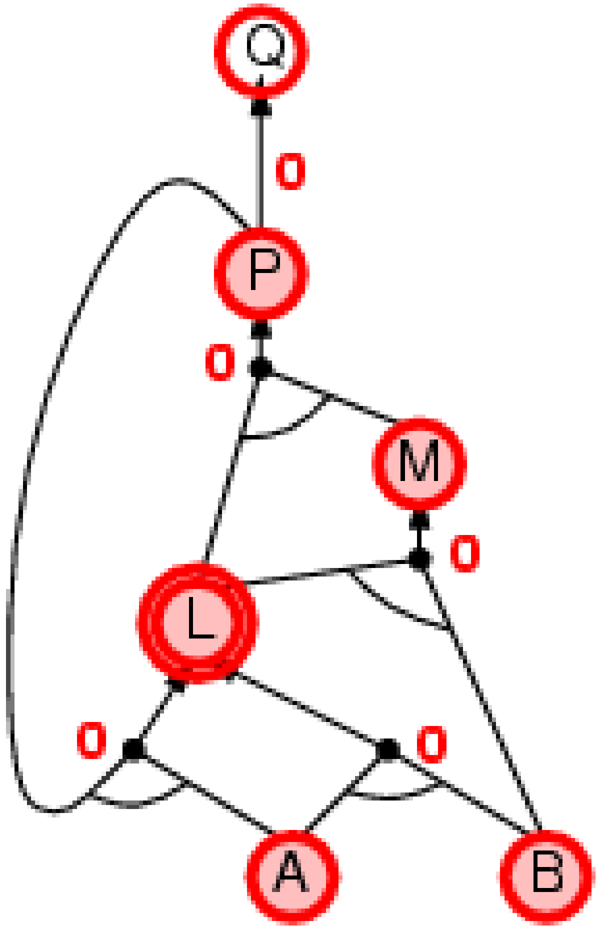
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

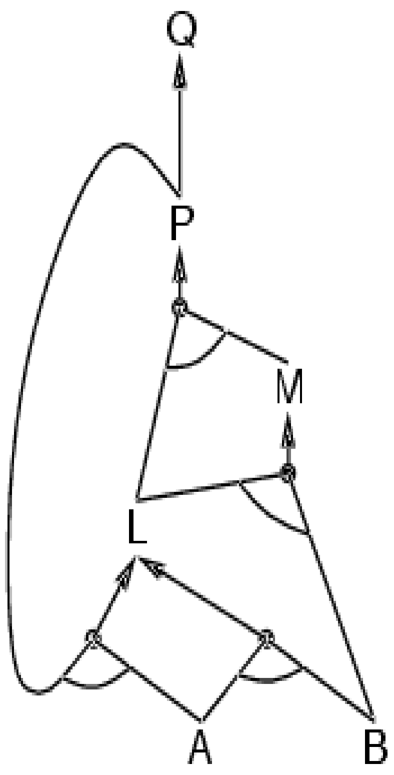
$B \wedge L \Rightarrow M$

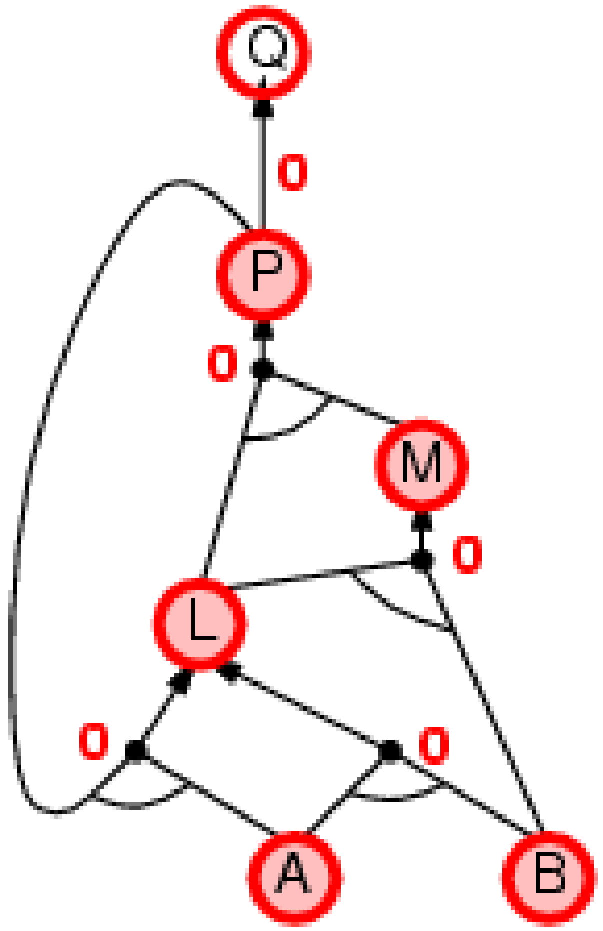
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

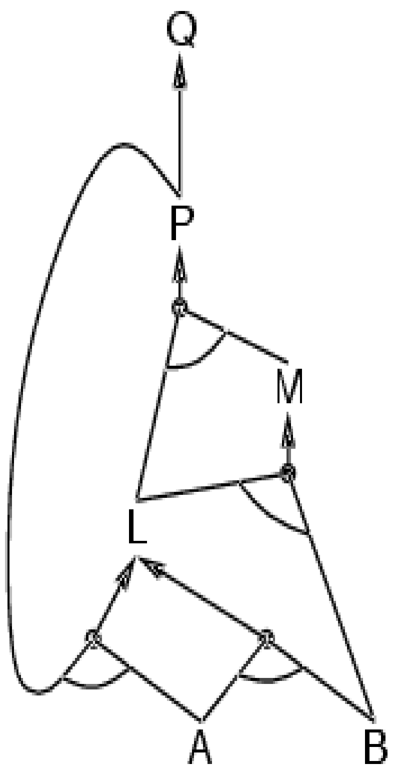
$B \wedge L \Rightarrow M$

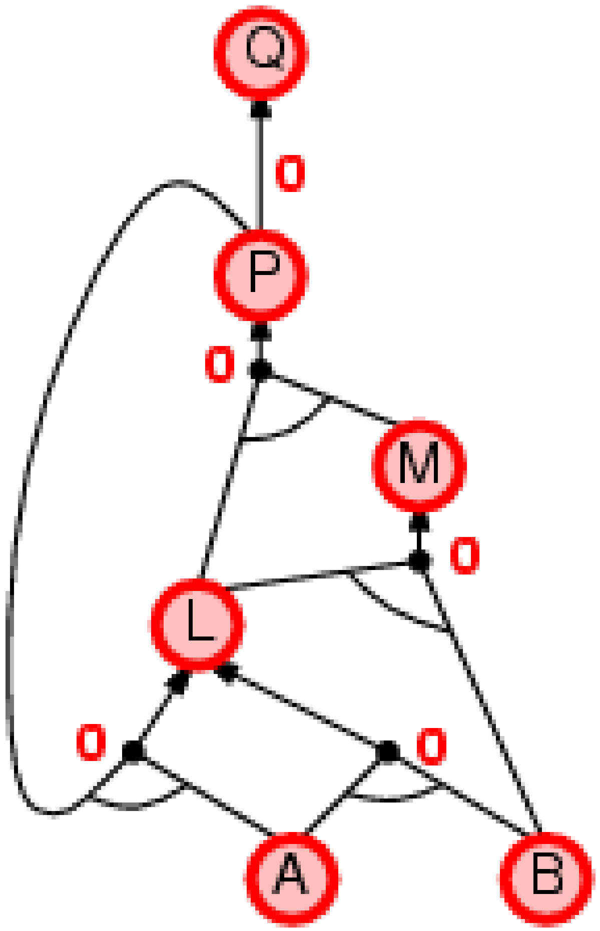
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Backward chaining

Idea: work backwards from the query *q*:

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

Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

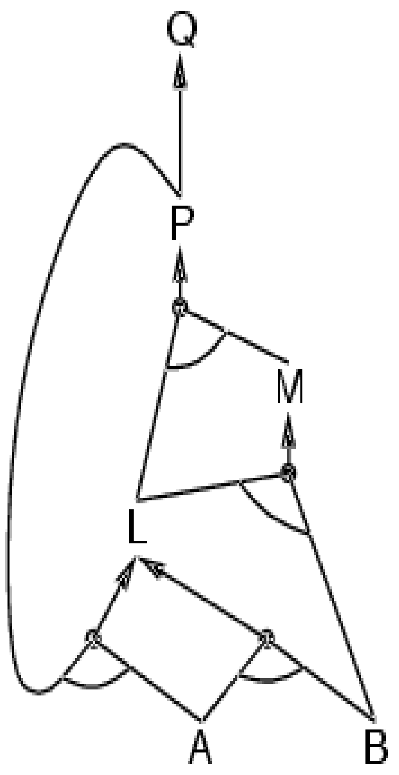
$B \wedge L \Rightarrow M$

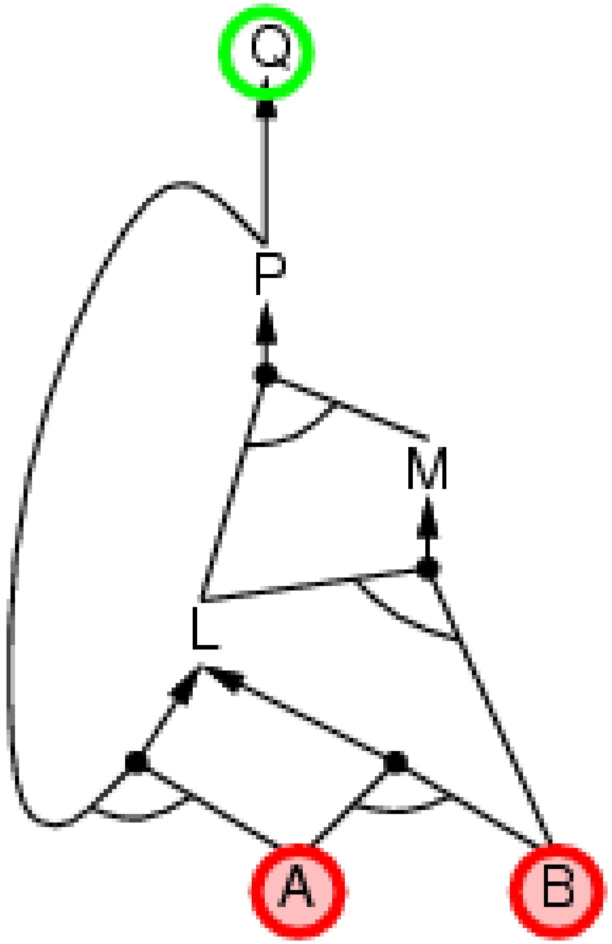
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Backward chaining example

$P \Rightarrow Q$

$L \land M \Rightarrow P$

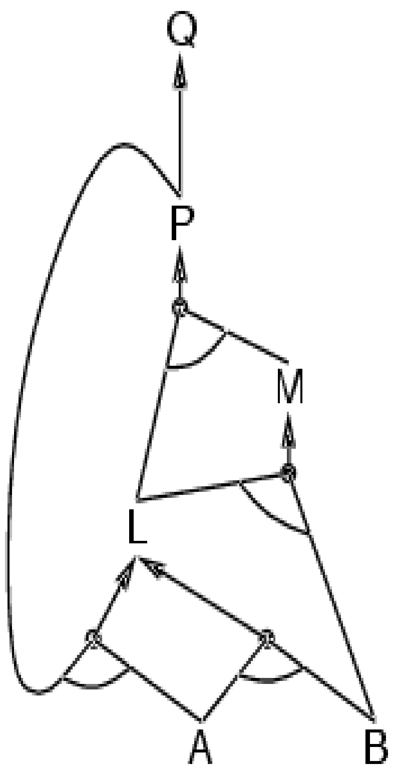
$B \land L \Rightarrow M$

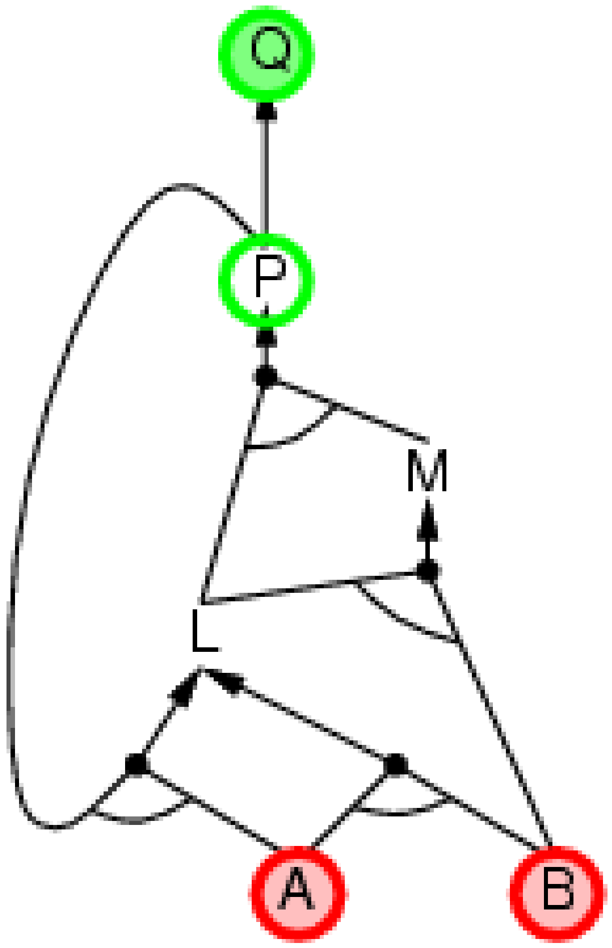
$A \land P \Rightarrow L$

$A \land B \Rightarrow L$

A

B





Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

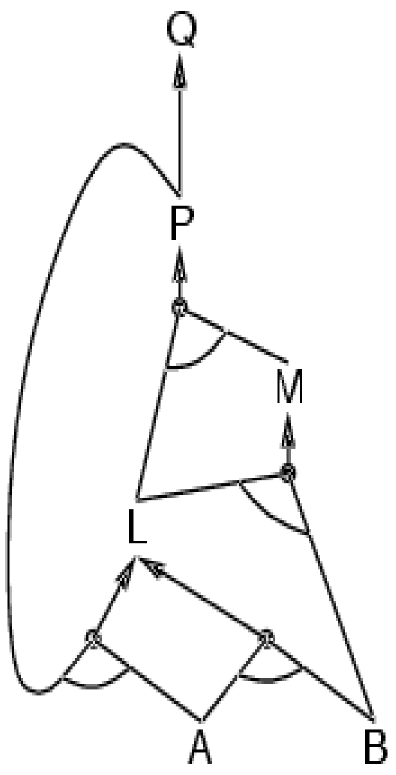
$B \wedge L \Rightarrow M$

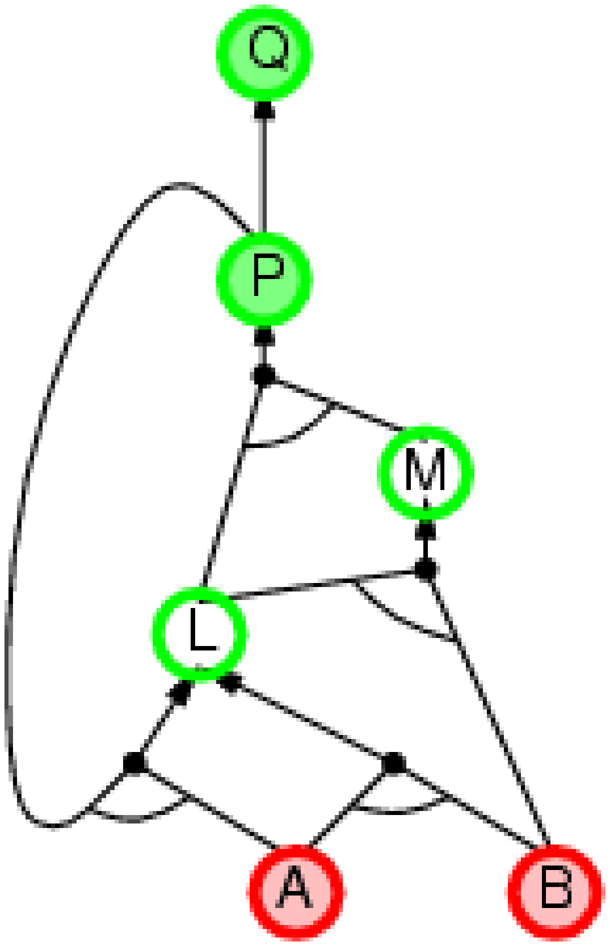
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

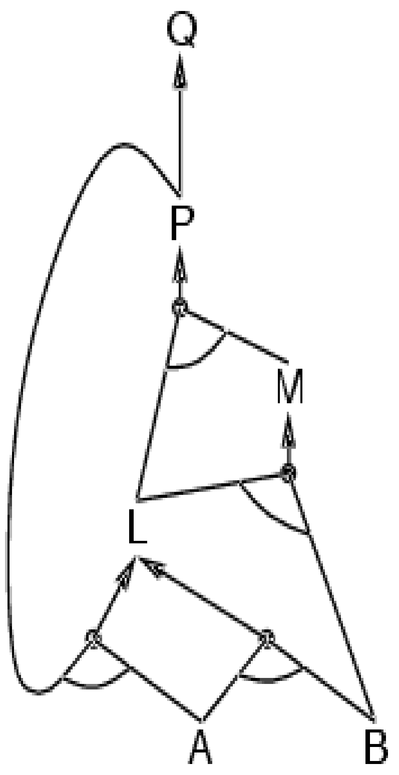
$B \wedge L \Rightarrow M$

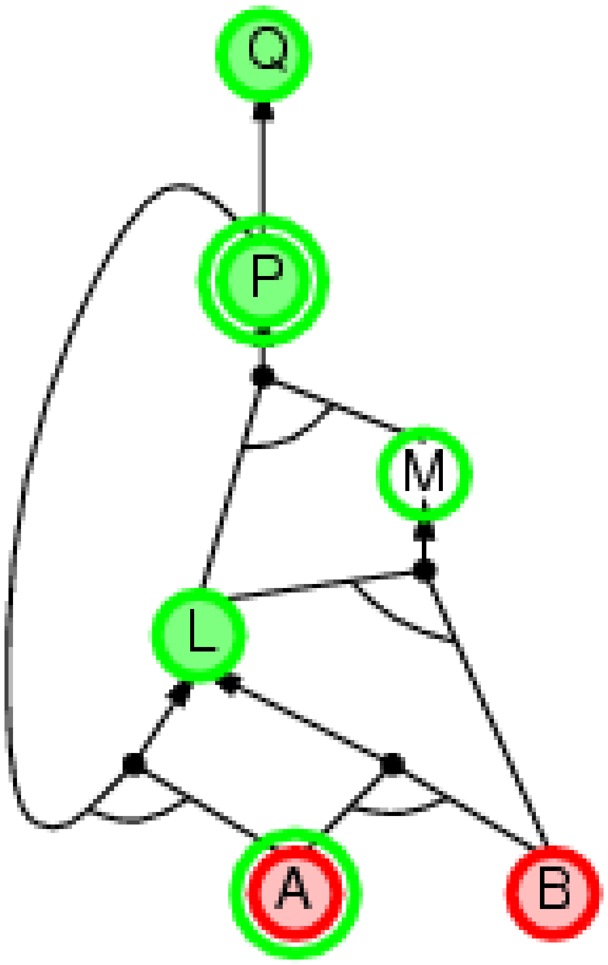
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

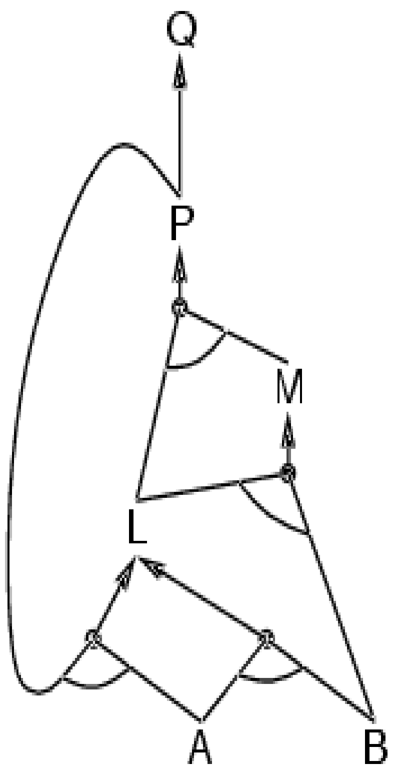
$B \wedge L \Rightarrow M$

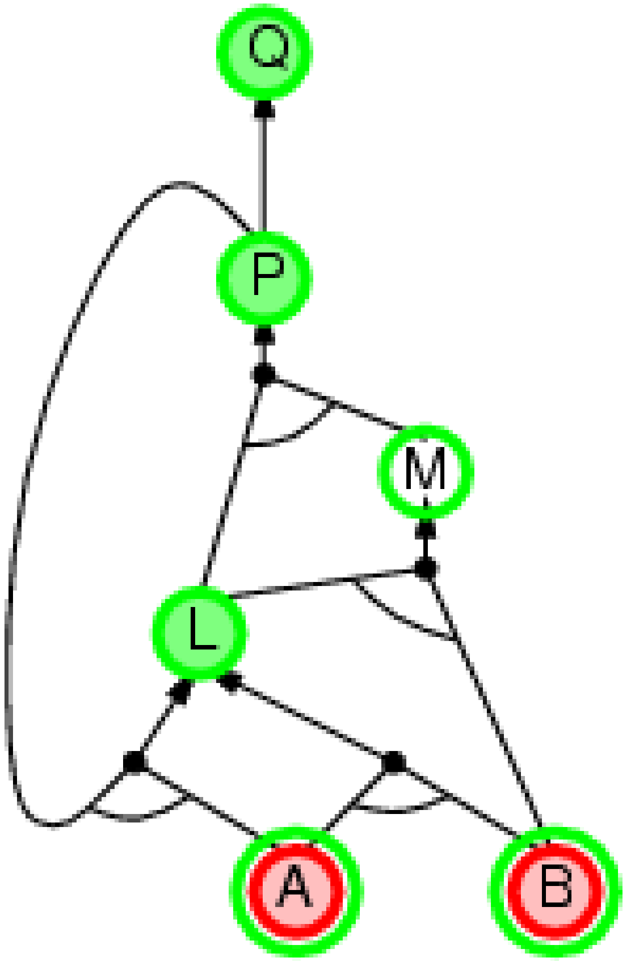
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

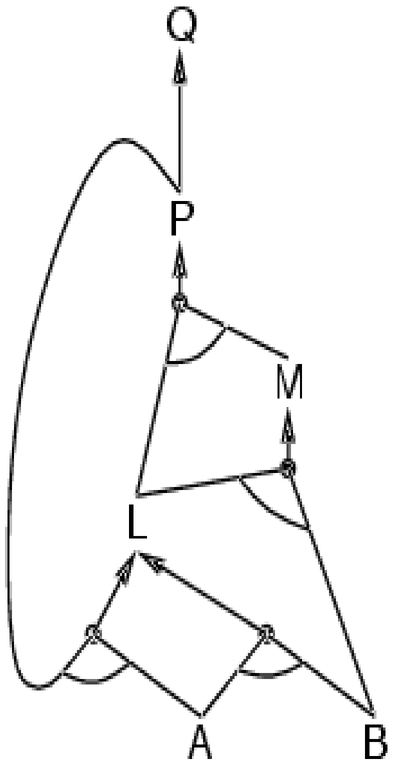
$B \wedge L \Rightarrow M$

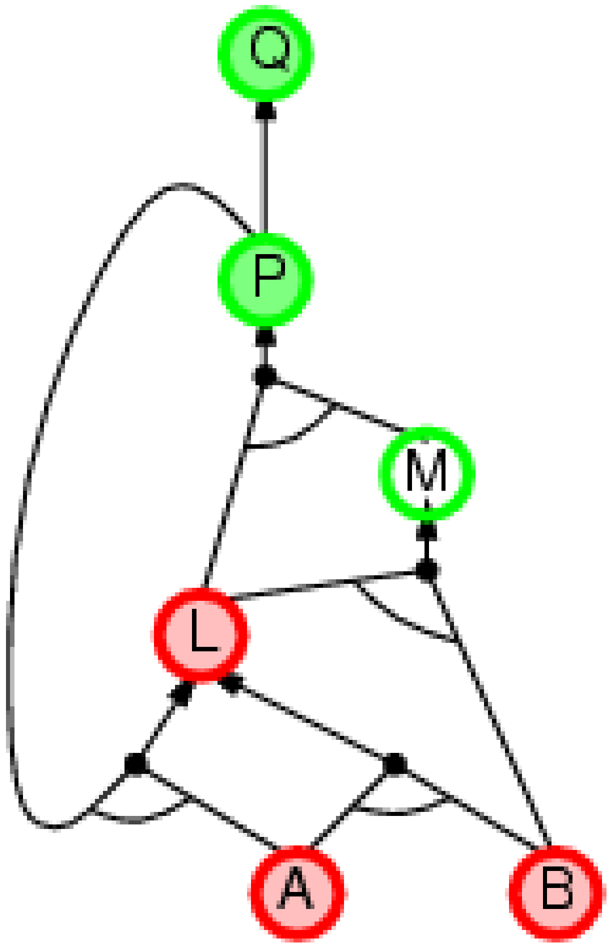
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

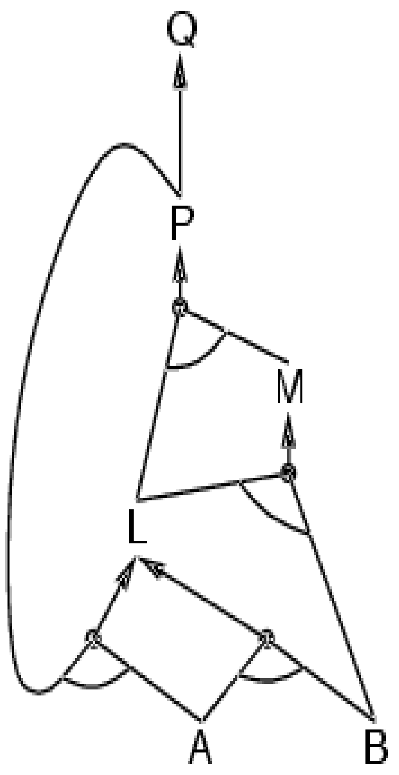
$B \wedge L \Rightarrow M$

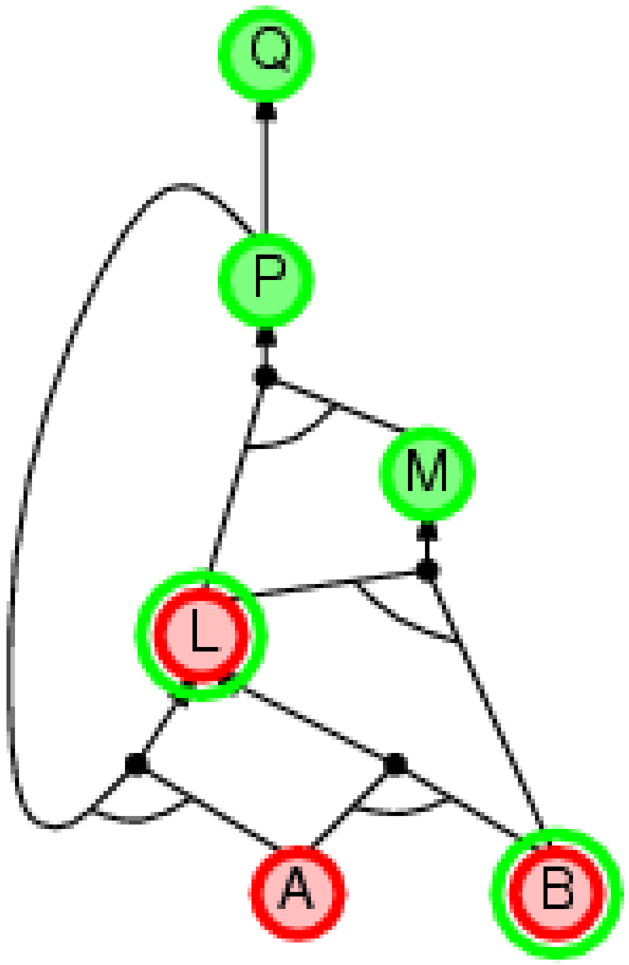
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

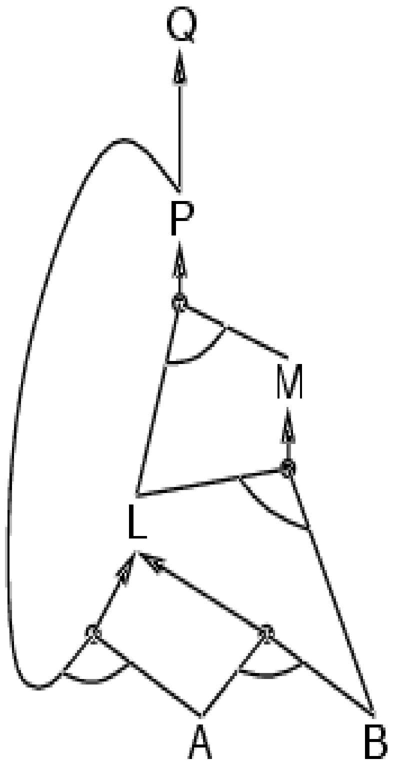
$B \wedge L \Rightarrow M$

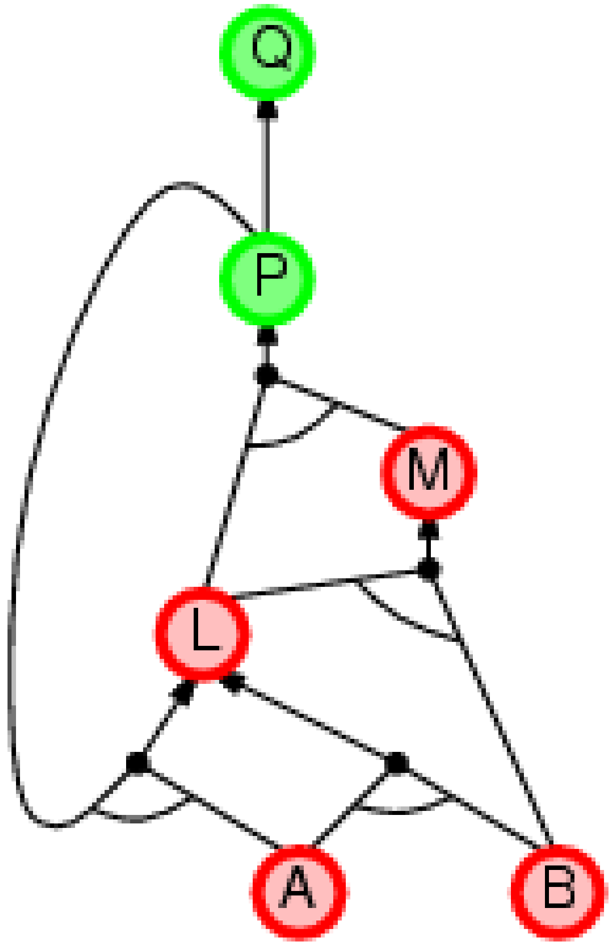
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

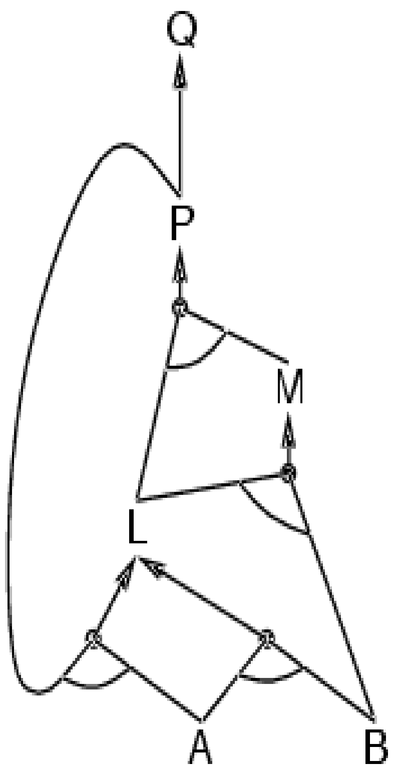
$B \wedge L \Rightarrow M$

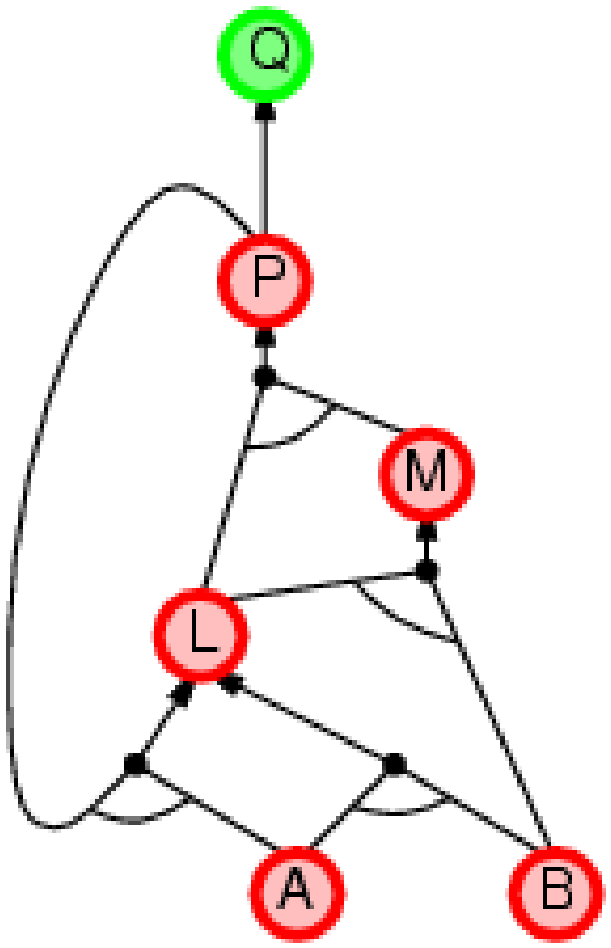
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Backward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

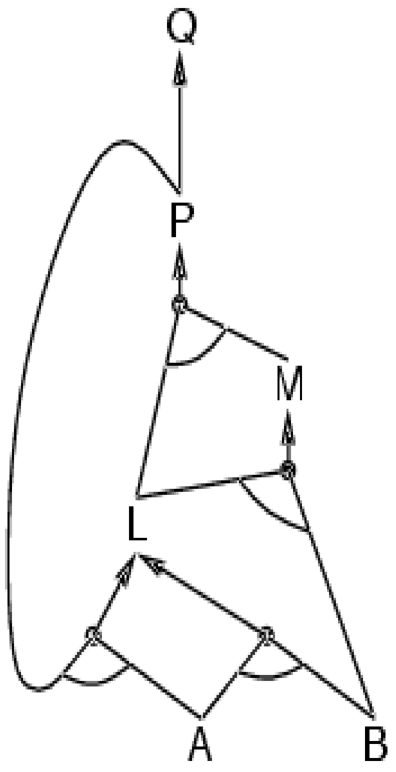
$B \wedge L \Rightarrow M$

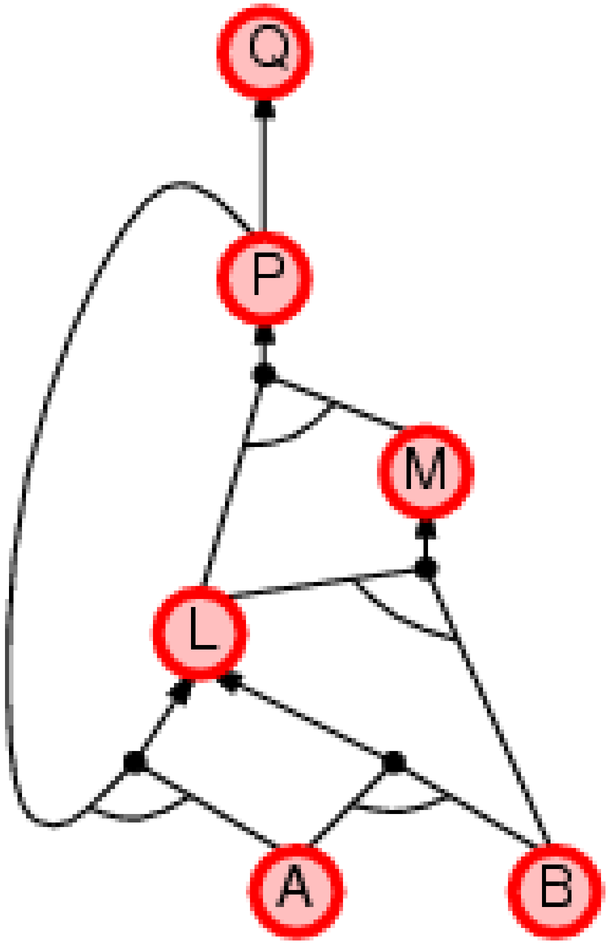
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B





Forward vs. backward chaining

- Forward chaining is data-driven, automatic processing
  - May do lots of work that is irrelevant to the goal

- Backward chaining is goal-driven, appropriate for problem-solving
  - Complexity can be much less than linear in size of KB

Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Basic concepts of logic:
  - syntax: formal structure of sentences
  - semantics: truth of sentences wrt models
  - entailment: necessary truth of one sentence given another
  - inference: deriving sentences from other sentences
  - soundness: derivations produce only entailed sentences
  - completeness: derivations can produce all entailed sentences
- Resolution is complete for propositional logic
- Forward, backward chaining are linear-time, complete for definite clauses

First-Order Logic

Limitations of propositional logic

- Suppose you want to say “All humans are mortal”
  - In propositional logic, you would need ~10 billion statements
- Suppose you want to say “Some people can run a marathon”
  - You would need a disjunction of 10 billion statements

First-order logic

- Propositional logic assumes the world consists of atomic facts
- First-order logic assumes the world contains objects, relations, and functions

Syntax of FOL

• **Constants:**	**John, Sally, 2, ...**
• **Variables:**	**x, y, a, b,...**
• **Predicates:**	**Person(John), Siblings(John, Sally), IsOdd(2), ...**
• **Functions:**	**MotherOf(John), Sqrt(x), ...**
• **Connectives:**	$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
• **Equality:**	**=**
• **Quantifiers:**	$\forall, \exists$

• **Term:**	**Constant** or **Variable** or **Function(Term$_1$, ... , Term$_n$)**
• **Atomic sentence:**	**Predicate(Term$_1$, ... , Term$_n$)** or **Term$_1$ = Term$_2$**
• **Complex sentence:**	made from atomic sentences using connectives and quantifiers

- Possible overloading between Functions and **Predicates**:
- **FatherOf(Ali,Omar)**: True or False; FatherOf(Hasan): possibly Issam

Semantics of FOL

- Sentences are true with respect to a model and an interpretation
- Model contains objects (domain elements) and relations among them
- Interpretation specifies referents for

constant symbols	→	objects
predicate symbols	→	relations
function symbols	→	functional relations

- An atomic sentence **Predicate(Term$_1$, ... , Term$_n$)** is true iff the objects referred to by **Term$_1$, ... , Term$_n$** are in the relation referred to by **Predicate**

Universal quantification

- **$\forall x\ P(x)$**

- Example: “Everyone at BZU is smart”

  $\forall x\ At(x,BZU) \Rightarrow Smart(x)$

  Why not **$\forall x\ At(x,BZU) \wedge Smart(x)$**?

- Roughly speaking, equivalent to the conjunction of all possible instantiations of the variable:

  $[At(Mariam, BZU) \Rightarrow Smart(Mariam)] \wedge \ldots$

  $[At(Hasan, BZU) \Rightarrow Smart(Hasan)] \wedge \ldots$

- **$\forall x\ P(x)$** is true in a model m iff **$P(x)$** is true with **x** being each possible object in the model

Existential quantification

- **$\exists x\ P(x)$**

- Example: “Someone at BZU is smart”

  $\exists x\ At(x,BZU) \wedge Smart(x)$

  Why not **$\exists x\ At(x, BZU) \Rightarrow Smart(x)$**?

- Roughly speaking, equivalent to the disjunction of all possible instantiations:

  $[At(Mariam, BZU) \wedge Smart(Mariam)] \vee$

  $[At(Hasan, BZU) \wedge Smart(Hasan)] \vee \ldots$

- **$\exists x\ P(x)$** is true in a model m iff **$P(x)$** is true with **x** being some possible object in the model

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"

  $\forall y\ \exists x$ Loves(x,y)

  "Everyone is loved by at least one person"

- **Quantifier duality:** each quantifier can be expressed using the other with the help of negation

  $\forall x$ Likes(x,IceCream)

  $\exists x$ Likes(x,Broccoli)

Equality

- **$Term_1$ = $Term_2$** is true under a given model if and only if **$Term_1$** and **$Term_2$** refer to the same object

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

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

Using FOL: The Kinship Domain

- Brothers are siblings

  $\forall x,y$ Brother(x,y) $\Rightarrow$ Sibling(x,y)
- “Sibling” is symmetric

  $\forall x,y$ Sibling(x,y) $\Leftrightarrow$ Sibling(y,x)
- One's mother is one's female parent

  $\forall m,c$ (Mother(c) = m) $\Leftrightarrow$ (Female(m) $\wedge$ Parent(m,c))

Using FOL: The Set Domain

- $\forall s\ Set(s) \Leftrightarrow (s = \{\}) \vee (\exists x,s_2\ Set(s_2) \wedge s = \{x|s_2\})$
- $\neg\exists x,s\ \{x|s\} = \{\}$
- $\forall x,s\ x \in s \Leftrightarrow s = \{x|s\}$
- $\forall x,s\ x \in s \Leftrightarrow [\ \exists y,s_2\ (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))]$
- $\forall s_1,s_2\ s_1 \subseteq s_2 \Leftrightarrow (\forall x\ x \in s_1 \Rightarrow x \in s_2)$
- $\forall s_1,s_2\ (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$
- $\forall x,s_1,s_2\ x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$
- $\forall x,s_1,s_2\ x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$

Translating English to FOL

Every gardener likes the sun.

$\forall x\ gardener(x) \rightarrow likes(x,Sun)$

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

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

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

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

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

Equivalent

All purple mushrooms are poisonous.

$\forall x\ (mushroom(x) \wedge purple(x)) \rightarrow poisonous(x)$

No purple mushroom is poisonous.

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

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

Equivalent

There are exactly two purple mushrooms.

$\exists x\ \exists y\ mushroom(x) \wedge purple(x) \wedge mushroom(y) \wedge purple(y)\ \wedge\ \neg(x=y) \wedge \forall z\ (mushroom(z) \wedge purple(z)) \rightarrow ((x=z) \vee (y=z))$

Clinton is not tall.

$\neg tall(Clinton)$

Every person who commits a crime must be punished.

$\forall x\ \ commitscrime(x) \rightarrow punished(x)$

What is : $\forall x\ \ commitscrime(x) \wedge punished(x)$ XX

Translating English to FOL

- Everybody is loved by all people:
  ∀ z ∀x Loved1By2(x,z)
- Somebody is loved by all people:
  ∃z ∀x Loved1By2(z,x) **…** ∀**x Loved1By2(C1,x)**
- Somebody is loved by somebody:
  ∃z ∃x Loved1By2(x,z) **… Loved1By2(C1,C2)**
- Everybody is loved by somebody:
  ∀z ∃x Loved1By2(z,x) **…** ∀**z Loved1By2(z,M(z))**; M(z)=mother/fiancé(z)
- Some people love all animals:
  ∃z ∀x Person(z) ∧ Animal(x) ∧ Loved1By2(x,z)
- Any two real numbers have a number between them:
  ∀ x ∀x ∃z Between(x,y,z) **…** ∀ **x** ∀**x Between(x,y,f(x,y));** f(x,y) =x+y/2

- **X is above Y iff X is directly on top of Y or** there is a pile of one or more other objects **directly on top** of one another starting with X and ending with Y**.**
  ∀x ∀y above(x,y) ↔ (on(x,y) ∨ ∃z (**on(x,z)** ∧ above(z,y)))

Example: A simple genealogy KB by FOL

- **Build a small genealogy knowledge base using FOL that**
  - contains facts of immediate family relations (spouses, parents, etc.)
  - contains definitions of more complex relations (ancestors, relatives)
  - is able to answer queries about relationships between people
- **Predicates:**
  - parent(x, y), child(x, y), father(x, y), daughter(x, y), etc.
  - spouse(x, y), husband(x, y), wife(x,y)
  - ancestor(x, y), descendant(x, y)
  - male(x), female(y)
  - relative(x, y)
- **Facts:**
  - husband(Joe, Mary), son(Fred, Joe)
  - spouse(John, Nancy), male(John), son(Mark, Nancy)
  - father(Jack, Nancy), daughter(Linda, Jack)
  - daughter(Liz, Linda)

- **Rules for genealogical relations**
  - $(\forall x,y)$ parent(x, y) $\leftrightarrow$ child (y, x)
    $(\forall x,y)$ father(x, y) $\leftrightarrow$ parent(x, y) $\wedge$ male(x) (similarly for mother(x, y))
    $(\forall x,y)$ daughter(x, y) $\leftrightarrow$ child(x, y) $\wedge$ female(x) (similarly for son(x, y))
  - $(\forall x,y)$ husband(x, y) $\leftrightarrow$ spouse(x, y) $\wedge$ male(x) (similarly for wife(x, y))
    $(\forall x,y)$ spouse(x, y) $\leftrightarrow$ spouse(y, x) (**spouse relation is symmetric**)
  - $(\forall x,y)$ parent(x, y) $\rightarrow$ ancestor(x, y)
    $(\forall x,y)(\exists z)$ parent(x, z) $\wedge$ ancestor(z, y) $\rightarrow$ ancestor(x, y)
  - $(\forall x,y)$ descendant(x, y) $\leftrightarrow$ ancestor(y, x)
  - $(\forall x,y)(\exists z)$ ancestor(z, x) $\wedge$ ancestor(z, y) $\rightarrow$ relative(x, y)
  -
  - $(\forall x,y)$ parent(x, y) $\rightarrow$ mother (y, x) or father (y, x) !!!!!
  - $(\forall x,y)$ sibling(x, y) $\rightarrow$ brother (y, x) or sister (y, x) !!!!!
  - $(\forall x,y)$ uncle(x, y) $\rightarrow$ خال (y, x) or عم (y, x) !!!!!

- **Rules for genealogical relations**

  (related by common ancestry)

  $(\forall x,y)$ spouse(x, y) $\rightarrow$ relative(x, y) (related by marriage)

  $(\forall x,y)(\exists z)$ relative(z, x) $\wedge$ relative(z, y) $\rightarrow$ relative(x, y) (**transitive**)

  $(\forall x,y)$ relative(x, y) $\leftrightarrow$ relative(y, x) **(symmetric**)

- **Queries**
  - ancestor(Jack, Fred)   /* the answer is yes */
  - relative(Liz, Joe)   /* the answer is yes */
  - relative(Nancy, Matthew)

    /* no answer in general, no if under closed world assumption */
  - $(\exists z)$ ancestor(z, Fred) $\wedge$ ancestor(z, Liz)

Why “First order”?

- FOL permits quantification over variables
- Higher order logics permit quantification over functions and predicates:

$$\forall P,x\ [P(x) \lor \neg P(x)]$$

$$\forall x,y\ (x=y) \Leftrightarrow [\forall P\ (P(x) \Leftrightarrow P(y))]$$

Inference in FOL

- All rules of inference for propositional logic apply to first-order logic
- We just need to reduce FOL sentences to PL sentences by instantiating variables and removing quantifiers

Reduction of FOL to PL

- Suppose the KB contains the following:

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

  King(John), Greedy(John), Brother(Richard,John)
- How can we reduce this to PL?
- Let's instantiate the universal sentence in all possible ways:

  $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$

  $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$

  King(John) Greedy(John) Brother(Richard,John)
- The KB is *propositionalized*
  - Proposition symbols are King(John), Greedy(John), Evil(John), King(Richard), etc.

Reduction of FOL to PL

- What about existential quantification, e.g.,

  $\exists x\ Crown(x) \land OnHead(x, John)$?

- Let's instantiate the sentence with a new constant that doesn't appear anywhere in the KB:

  $Crown(C_1) \land OnHead(C_1, John)$

Propositionalization

- Every FOL KB can be *propositionalized* so as to preserve entailment
  - A ground sentence is entailed by the new KB iff it is entailed by the original KB

- **Idea:** propositionalize KB and query, apply resolution, return result

- **Problem:** with function symbols, there are infinitely many ground terms
  - For example, Father(X) yields Father(John), Father(Father(John)), Father(Father(Father(John))), etc.

Propositionalization

- **Theorem** (Herbrand 1930):
  - If a sentence α is entailed by an FOL KB, it is entailed by a *finite* subset of the propositionalized KB
- **Idea:** For *n* = 0 to Infinity do
  - Create a propositional KB by instantiating with depth-n terms
  - See if α is entailed by this KB
- **Problem:** works if α is entailed, loops if α is not entailed
- **Theorem** (Turing 1936, Church 1936):
  - Entailment for FOL is **semidecidable**: algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence

Inference in FOL

- *“All men are mortal. Socrates is a man; therefore, Socrates is mortal.”*

- $\forall x$ man(x) $\rightarrow$ mortal(x)
- man(Socrates)
- Mortal(Socrates)??
- It seems to work if we replace **x** by **Scorates**
- Can we prove this without full propositionalization as an intermediate step?
- Can we do that with the least propositionalization?

Generalized Modus Ponens (GMP)

$$\frac{(p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q),\ p_1',\ p_2',\ \ldots,\ p_n' \quad \text{such that } \mathbf{SUBST}(\theta, p_i) = \mathbf{SUBST}(\theta, p_i') \text{ for all } i}{\mathbf{SUBST}(\theta, q)}$$

- All variables assumed universally quantified
- **Example:**

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

  King(John)   Greedy(John)   Brother(Richard,John)

  p_1 is King(x), p_2 is Greedy(x), q is Evil(x)

  p_1' is King(John), p_2' is Greedy(y), θ is {x/John,y/John}

  SUBST(θ,q) is Evil(John)

Predicate Logic and CNF

$[Q \Rightarrow P] \Rightarrow S$; To CNF: $[\neg Q \vee P] \Rightarrow S$; $\neg[\neg Q \vee P] \vee S$; $[Q \wedge \neg P] \vee S$; $[Q \vee S] \wedge [\neg P \vee S]$; $\{Q,S\}$; $\{P, S\}$

- Converting FOL to CNF is harder - we need to worry about variables and quantifiers.

1. Eliminate all implications $\Rightarrow$ **$P \Rightarrow Q = \neg P \vee Q$**
2. Reduce the scope of all $\neg$ to single term.
3. Make all variable names **unique** (set apart)
4. Move Quantifiers Left [leftmost]
5. Eliminate *Existential* Quantifiers [No $\exists$**]**
6. Eliminate Universal Quantifiers [all are $\forall$**]**
7. Convert to conjunction of disjuncts CNF
8. Create separate clause for each conjunct CNF.

Eliminate Existential Quantifiers

- To eliminate the quantifier, we can replace the variable with a function.
- We don’t know what the function is, we just know it exists.
- So we invent one!
- It is a function of n variables where n is the number of universally quantifies variables preceding the existential quantifier

Skolem functions

$\exists$ *y* President*(y)*

We replace y with a new function func:

President*(func())*

func is called a skolem function.

In general the function must have the same number of arguments as the number of universal quantifiers in the current scope.

Skolemization Example

$\forall x \exists y$ Father(y,x)

create a new function named foo and replace y with the function (f1(x) is the function.

$\forall x$ Father$(f1(x),x)$

$\forall$ **z**$\forall x\ \forall t\ \exists y$ Father(y,x,z,t)

$\forall$ **z**$\forall x\ \forall t$ Father$(f2(z,x,t),x,z,t)$

$\forall$ **z**$\forall x\ \exists y\ \forall t$ Father(y,x,z,t)

$\forall$ **z**$\forall x\ \forall t$ Father$(f3(z,x),x,z,t)$

$\exists y$ Father(y,x,z,t)

Father*(f3(),x,z,t) OR Father(C,x,z,t), C f of 0 var*

Conversion to CNF

- Everyone who loves all animals is loved by someone:

  $\forall x\ [[\forall y\ [Animal(y) \Rightarrow Loves(x,y)]] \Rightarrow [\exists y\ Loves(y,x)]]$ original

- 1. Eliminate biconditionals and implications

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

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

- 2. Move $\neg$ inwards: $\neg\forall x\ p \equiv \exists x\ \neg p,\ \ \neg\ \exists x\ p \equiv \forall x\ \neg p$

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

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

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

Conversion to CNF contd.

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

$\forall x\ [\exists y\ Animal(y) \land \neg Loves(x,y)] \lor [\exists z\ Loves(z,x)]$

4. Skolemize: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:F(x)/y, G(x)/z

$\forall x\ [Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(x),x)$

5. Drop universal quantifiers: No need, all variables are Universally quantified

$[Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(x),x)$

6. Distribute $\lor$ over $\land$:

$[Animal(F(x)) \lor Loves(G(x),x)] \land [\neg Loves(x,F(x)) \lor Loves(G(x),x)]$

Unification

Now that we have FOL Clauses (CNF) with variables (all universally quantified) we want to apply inference rules

Suppose we have the rule

If x is a working automobile, then x has an engine.

P(x) $\rightarrow$ Q(x) OR the clause $\neg$ P(x) $\vee$ Q(x)

and we have the fact

Tim's KIA is a working automobile

P(a)

These cannot immediately be resolved because P(x) and P(a) don't quite match. They must be "unified."

Modus Ponens in the Predicate Calculus

$P(a) \rightarrow Q(a)$
$P(a)$

$Q(a)$

But we have
$P(x) \rightarrow Q(x)$
So we create a *substitution instance* of it:
$P(a) \rightarrow Q(a)$

using the substitution $\{ a/x \}$

Substitutions

A ***substitution*** **is a set of term/variable pairs.**
e.g., { a/x, f(a)/y, w/z }
Each element of the set is an ***elementary substitution*****.**
A particular variable occurs at most once on the the right-hand side of any elementary subst. in a substitution.
{ a/x, b/x } is not an acceptable substitution.

The empty set is an acceptable substitution.

If E is a term or a formula of the predicate calculus, and S is a substitution, then S(E) is the result of ***applying*** **S to E, i.e., replacing each variable of S that occurs in E by the the corresponding term in S.**

Unifiers

Given a pair of literals L1 and L2, if S is a substitution such that
$S(L1) = S(L2)$ or $S(L1) = \neg S(L2)$,
then S is a *unifier* for L1 and L2.

Example:
$L1 = \neg P(x, f(a))$
$L2 = P(b, y)$

$S = \{ b/x, f(a)/y \}$

$S(L1) = \neg P(b, f(a)) = \neg S(L2)$
Therefore S is a unifier for L1 and L2.

The Occurs Check

A unifier may not contain an elementary substitution of the form f(x)/x and may not cause an indefinite recursion. In general the term in a term/variable pair may not include that variable.

P(x) and P(f(x)) cannot be unified, since f(x)/x is illegal.
P(y, f(y)) and P(f(x), y) cannot be unified, since
S = { f(x)/y, f(y)/x } leads to an indefinite recursion.

Testing whether a variable appears in its corresponding term is called the "occurs check".

Some automated reasoning systems do not perform the occurs check in order to save time.

Generality of Unifiers

Some pairs of literals may have more than one possible unifier.

P(x), P(y) is unified by each of { x/y }, { y/x }, { z/x, z/y }, { a/x, a/y}, { f(a)/x, f(a)/y } etc.

Suppose S1 and S2 are unifiers for P1 and P2.

Then S1(P1) = S1(P2) or S1(P1) = ~S1(P2)

S2(P1) = S2(P2) or S2(P1) = ~S2(P2)

S1 is more general than S2 if there exists S3 such that S3(S1(P1)) = S2(P1).

S3(S1(P1)) means apply S3 to the result of applying S1 to P1.

{ a/x, a/y } is less general than { y/x }. { y/x } is more general than { a/x, a/y }.

Most General Unifiers

A unifier S1 for L1 and L2 is a *most general unifier* (MGU) for L1 and L2 provided that for any other unifier S2 of L1 and L2 there exists a substitution S3 such that
S3(S1(L1)) = S2(L1).

S1 = { y/x } is a most general unifier for P(x), P(y).

S2 = { f(a)/x, f(a)/y } is not a MGU for P(x), P(y).

S3 = { f(a)/y }

S3(S1(P(x))) = P(f(a)) = S2(P(x)).

Finding a Most-General Unifier

1. Given literals L1 and L2, place a cursor at the left end of each. Skip any negation sign. If the predicate symbols do not match, return NOT-UNIFIABLE.

2. Let S = { }.

3. Move the cursors right to the next term (argument) in each literal.
If there are no terms left, return S as a MGU else
Let t1 and t2 be the terms.

Finding an MGU (Cont.)

4a. If t1 = t2, go to Step 3.
4b. Otherwise, if either t1 or t2 is a variable (call it v and call the other term t), then attempt to add t/v to S (see below).
4c. Otherwise if t1 and t2 both begin with function symbols, and these symbols are the same, move the cursors to the first argument of these symbols and go to step 4a.
4d. Otherwise, return NOT-UNIFIABLE.

Finding an MGU (Cont.)

When trying to add t/v to S, apply { t/v } to each term in S. If this results in any elementary substitution whose term includes its variable, return NOT-UNIFIABLE.
Otherwise, replace each elementary substitution in S by the result of applying { t/v } and add t/v itself to S.

Example

L1: P(a, f(x, a))
L2: P(a, f(g(y), y))

S: { }
S: { g(y)/x }
S: { g(a)/x, a/y }

Unification

More Examples of Unification:

If we apply the same substitution to both: they become the same.

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,Mary)	{x/Mary, y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John, x/Mother(John)}
Knows(John,x)	Knows(x,Mary)	{x_1/John, x_2/Mary}
Knows(John,x)	Knows(y,z)	{y/John, x/z}
P(x, x)	P(f(x),x)	No. Fails OC
P(x, x)	P(f(y),y)	{f(y)/x}
P(f(y), f(y))	P(f(y),y)	{f(y)/x}o f(y)/y= XXXX

- Standardizing apart eliminates overlap of variables
- Most general unifier: others special cases: {y/John, x/z} more general than {y/John, x/John, z/John}

Inference with GMP

$$\frac{(p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q),\ p_1',\ p_2',\ \ldots,\ p_n' \quad \text{such that } \mathbf{SUBST}(\theta, p_i) = \mathbf{SUBST}(\theta, p_i') \text{ for all } i}{\mathbf{SUBST}(\theta, q)}$$

- **Forward chaining**
  - Like search: keep proving new things and adding them to the KB until we can prove q
- **Backward chaining**
  - Find $p_1, \ldots, p_n$ such that knowing them would prove q
  - Recursively try to prove $p_1, \ldots, p_n$

Example knowledge base

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.
- Prove that Col. West is a criminal

Example knowledge base

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

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono has some missiles

$\exists x\ Owns(Nono,x) \wedge Missile(x)$

$Owns(Nono,M_1) \wedge Missile(M_1)$ **FACTS**

All of Nono's missiles were sold to it by Colonel West

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

Missiles are weapons:

$Missile(x) \Rightarrow Weapon(x)$

An enemy of America counts as "hostile":

$Enemy(x,America) \Rightarrow Hostile(x)$

Captain West is American

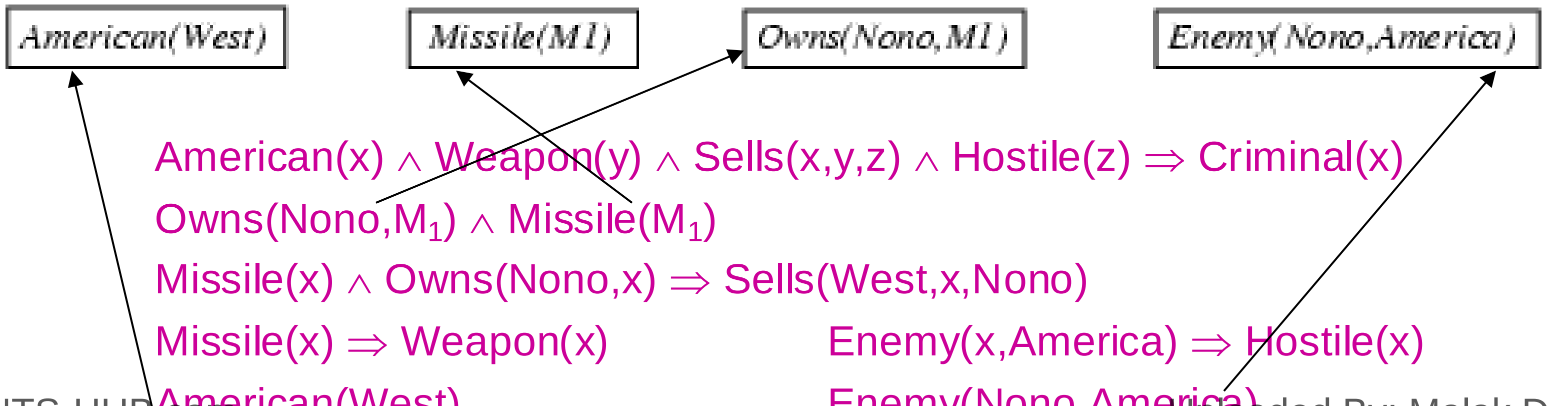
$American(West)$ **FACTS**

The country Nono is an enemy of America

$Enemy(Nono,America)$

Goal: Criminal(West) ?

Forward chaining proof



American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Owns(Nono,M_1) ∧ Missile(M_1)

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

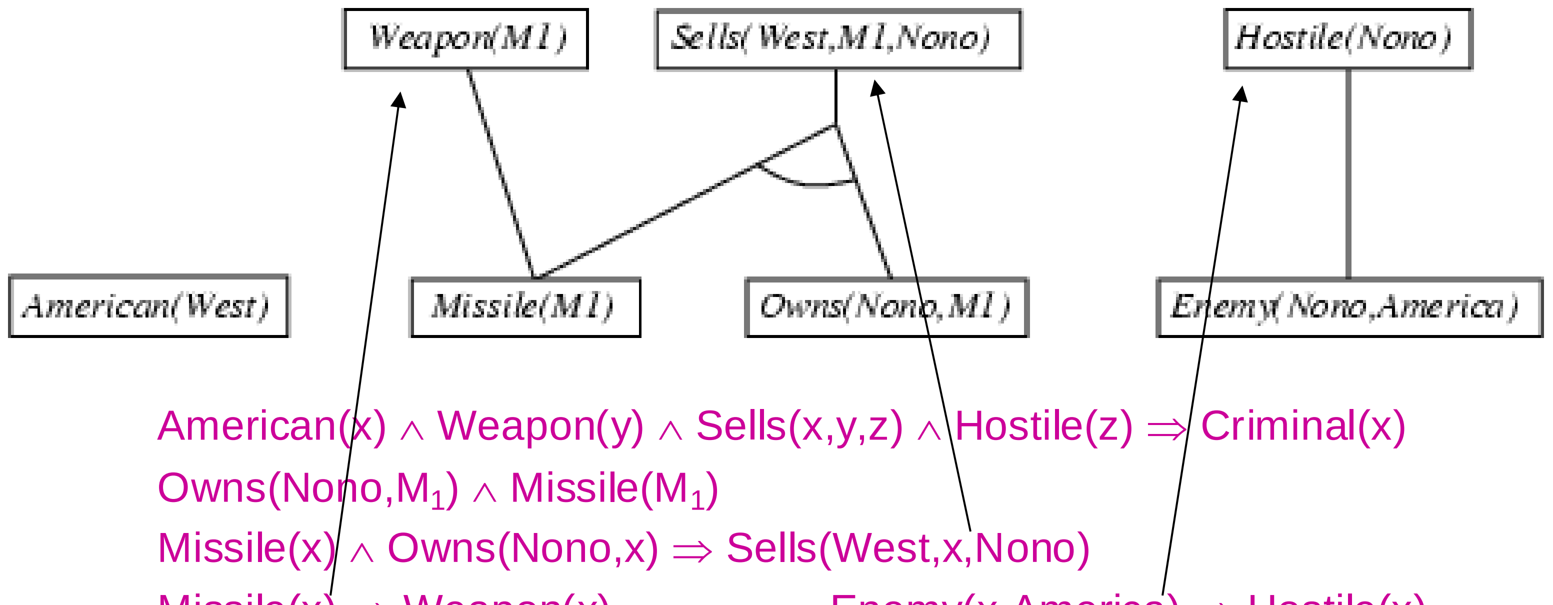
Missile(x) ⇒ Weapon(x)

Enemy(x,America) ⇒ Hostile(x)

American(West)

Enemy(Nono,America)

Forward chaining proof



American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Owns(Nono,M_1) ∧ Missile(M_1)

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

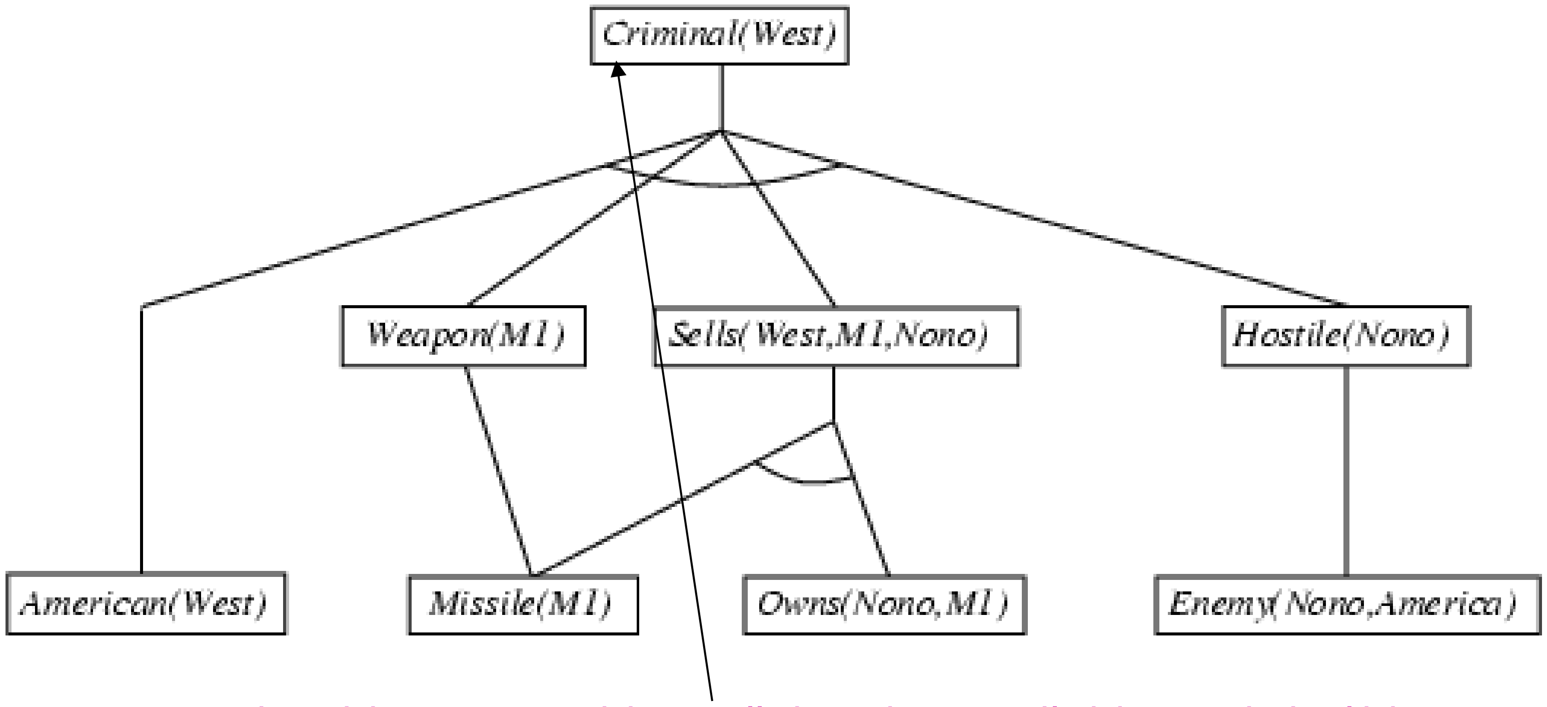
Missile(x) ⇒ Weapon(x)

Enemy(x,America) ⇒ Hostile(x)

American(West)

Enemy(Nono,America)

Forward chaining proof



American(x) $\wedge$ Weapon(y) $\wedge$ Sells(x,y,z) $\wedge$ Hostile(z) $\Rightarrow$ Criminal(x)

Owns(Nono,M_1) $\wedge$ Missile(M_1)

Missile(x) $\wedge$ Owns(Nono,x) $\Rightarrow$ Sells(West,x,Nono)

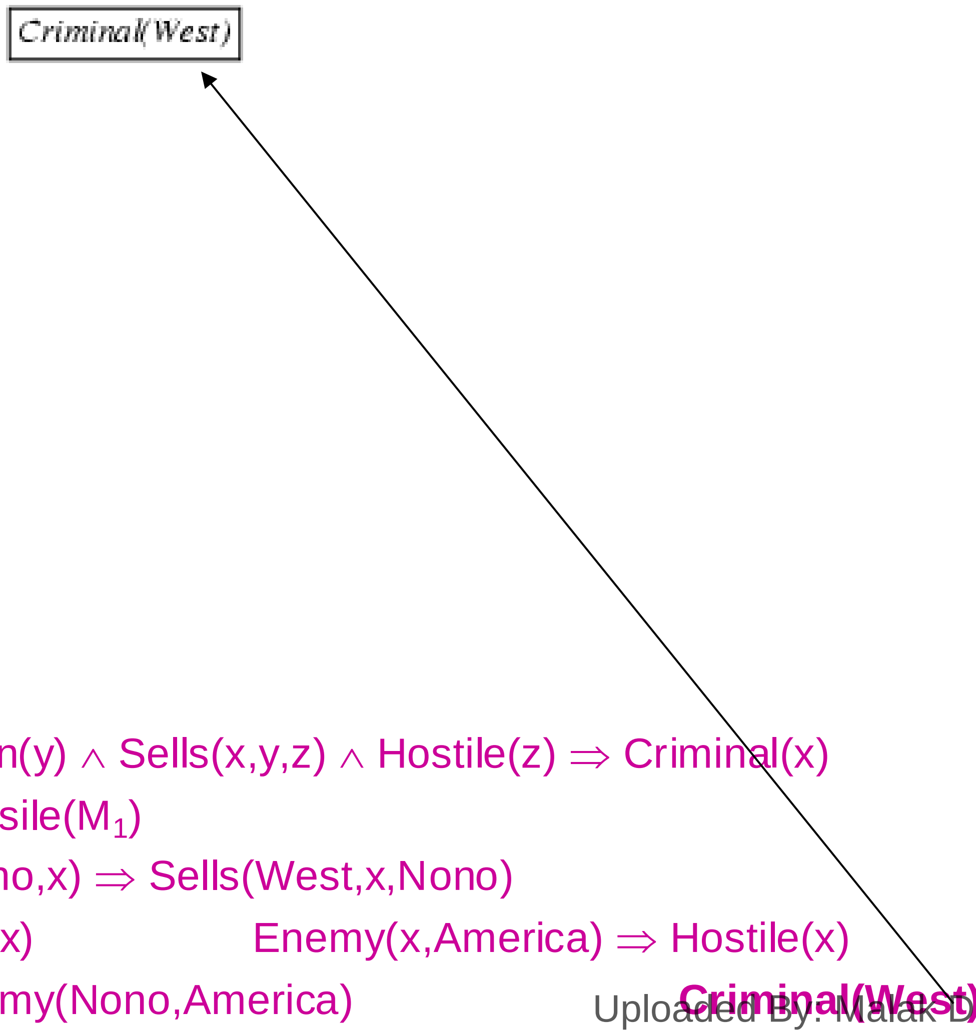
Missile(x) $\Rightarrow$ Weapon(x)

Enemy(x,America) $\Rightarrow$ Hostile(x)

American(West)

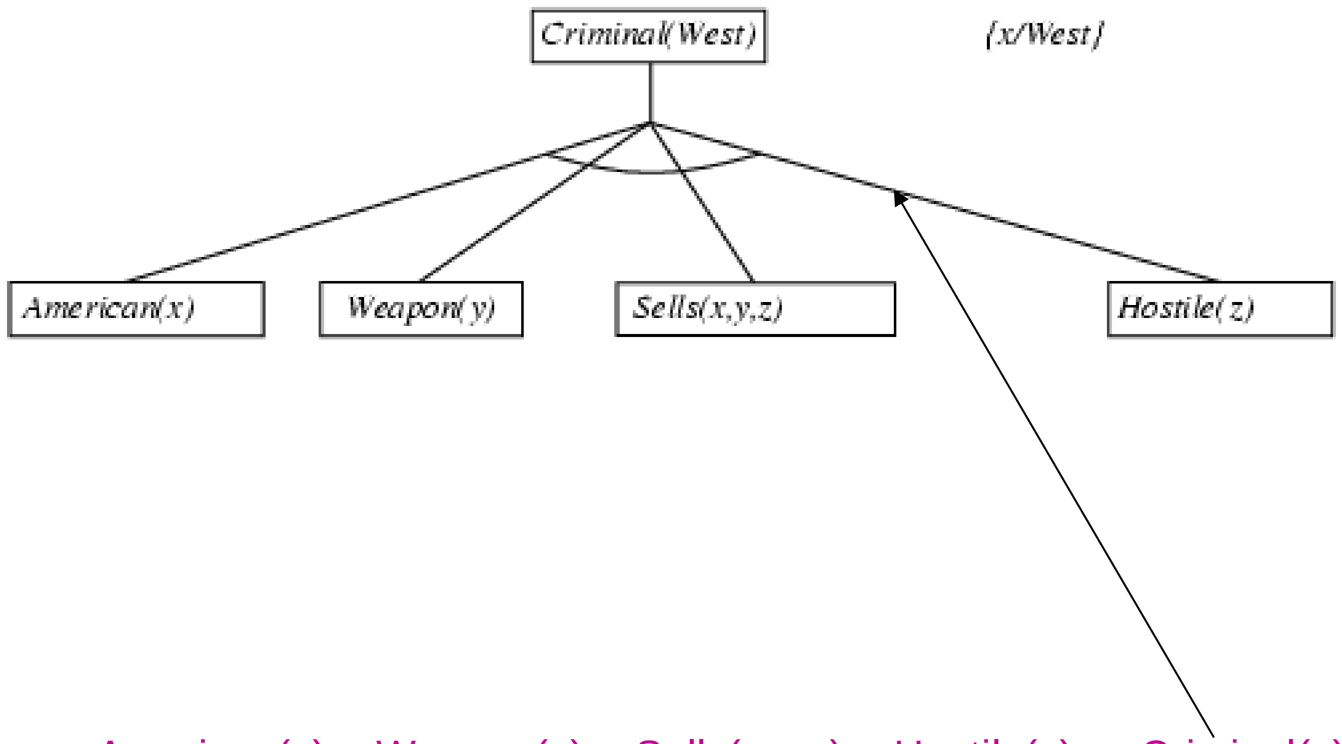
Enemy(Nono,America)

Backward chaining example



American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
Owns(Nono,M_1) ∧ Missile(M_1)
Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)
Missile(x) ⇒ Weapon(x) Enemy(x,America) ⇒ Hostile(x)
American(West) Enemy(Nono,America) **Criminal(West)**

Backward chaining example



American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Owns(Nono,M_1) ∧ Missile(M_1)

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

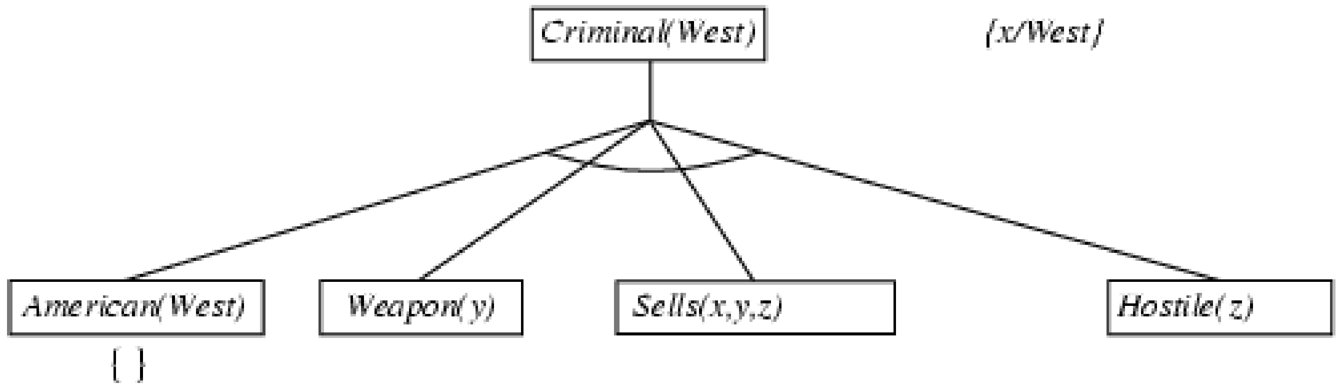
Missile(x) ⇒ Weapon(x)

Enemy(x,America) ⇒ Hostile(x)

American(West)

Enemy(Nono,America)

Backward chaining example



American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Owns(Nono,M_1) ∧ Missile(M_1)

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

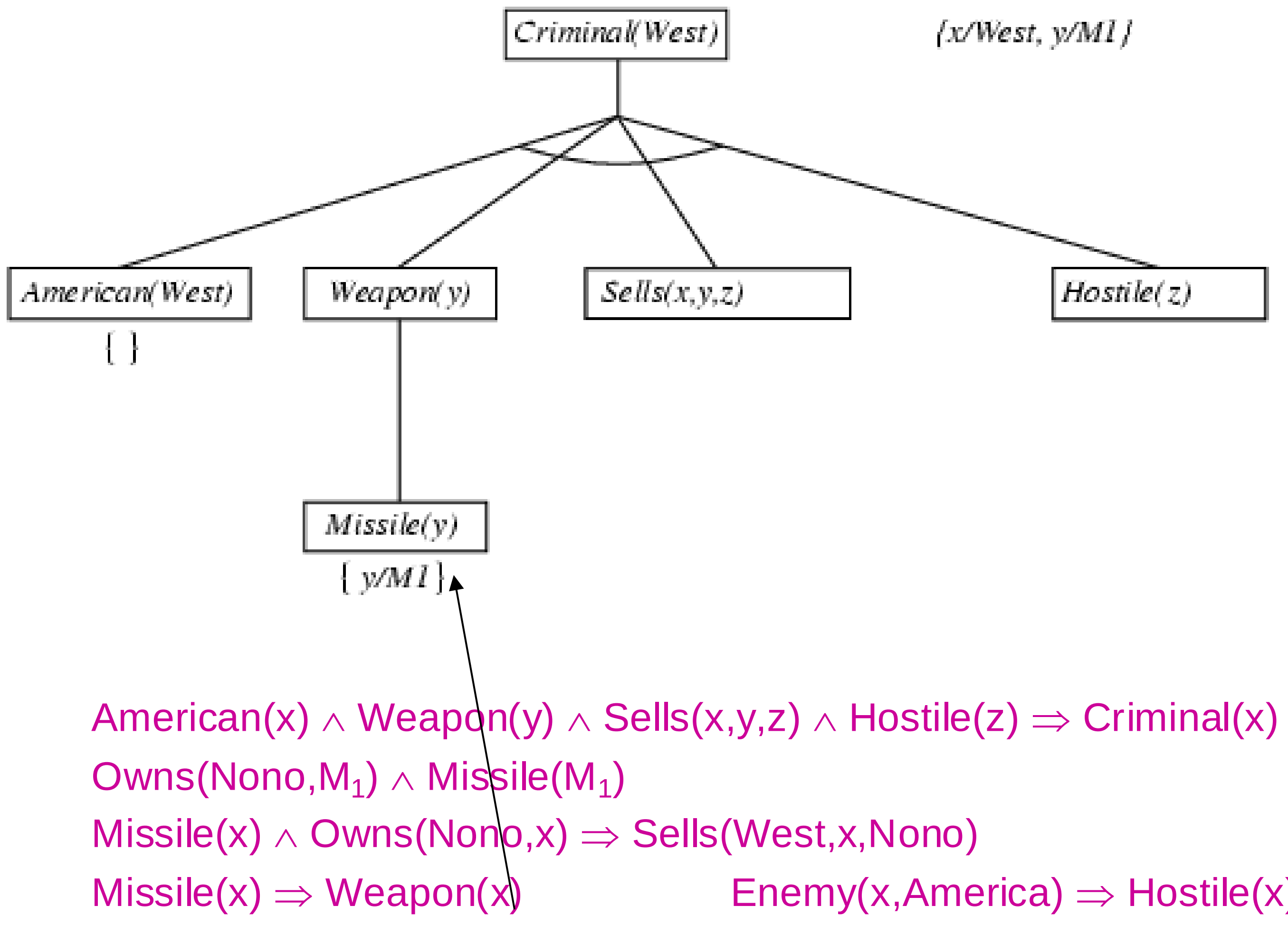
Missile(x) ⇒ Weapon(x)

Enemy(x,America) ⇒ Hostile(x)

American(West)

Enemy(Nono,America)

Backward chaining example



American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Owns(Nono,M_1) ∧ Missile(M_1)

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

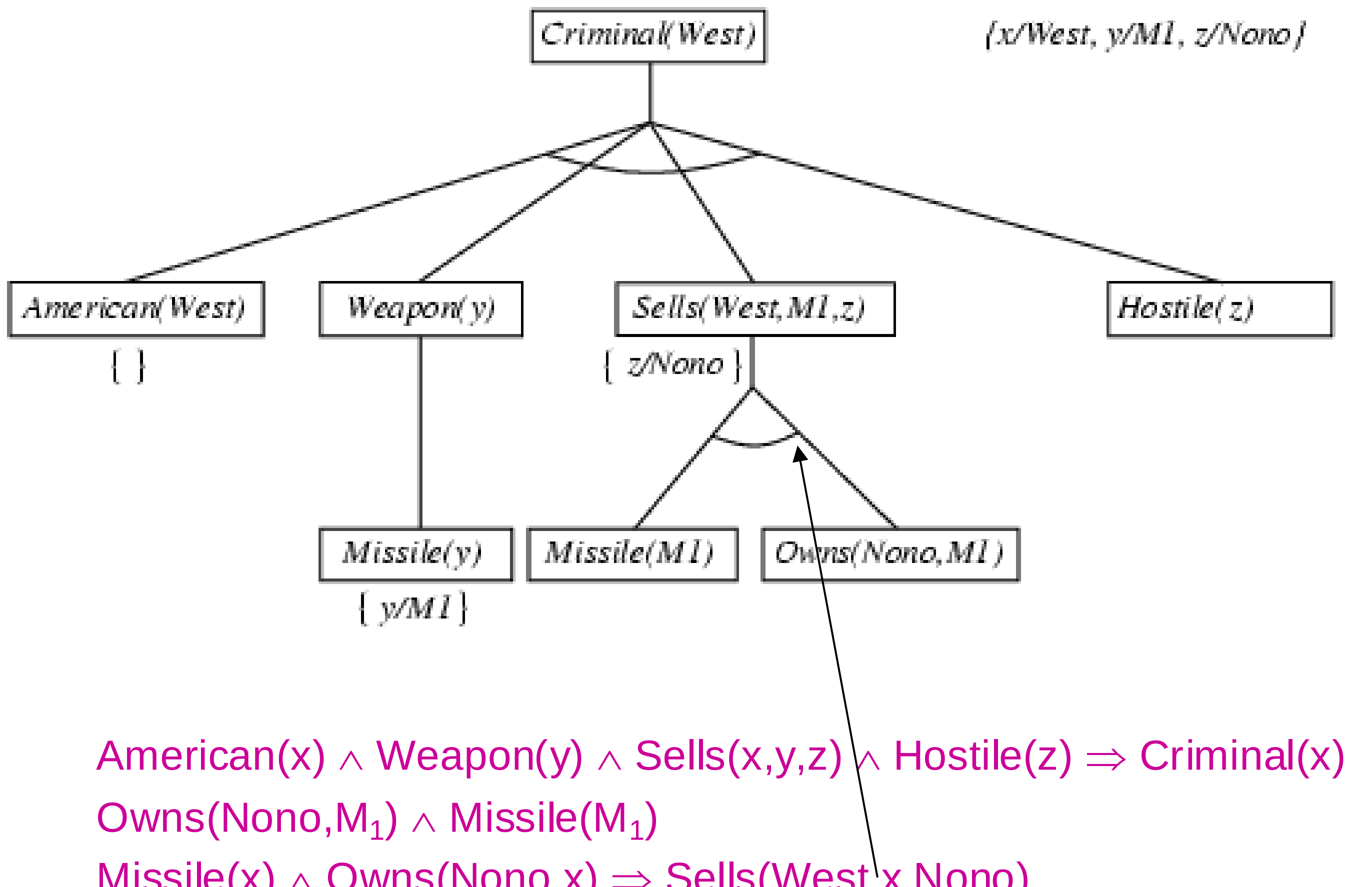
Missile(x) ⇒ Weapon(x)

Enemy(x,America) ⇒ Hostile(x)

American(West)

Enemy(Nono,America)

Backward chaining example



American(x) $\wedge$ Weapon(y) $\wedge$ Sells(x,y,z) $\wedge$ Hostile(z) $\Rightarrow$ Criminal(x)

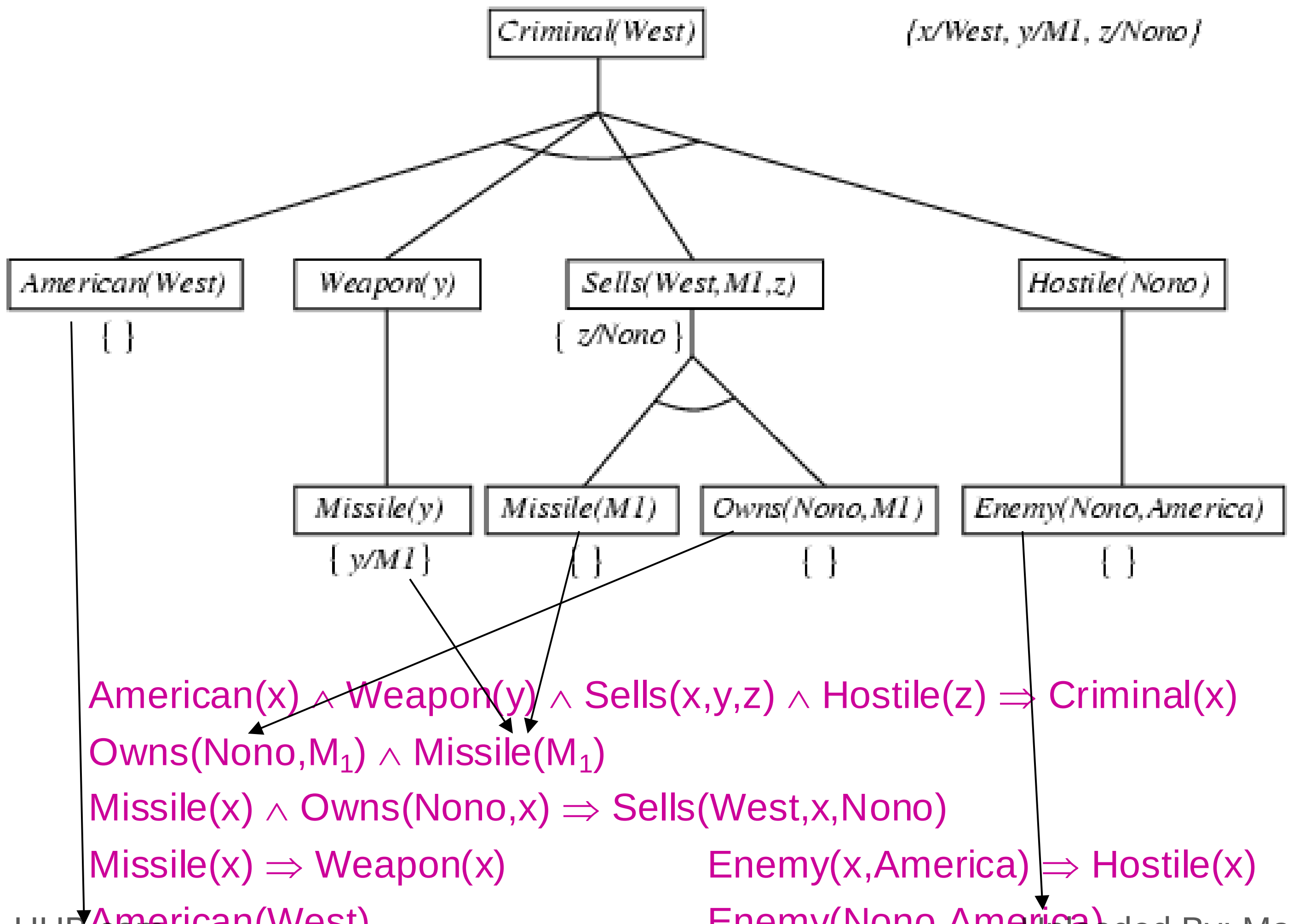
Owns(Nono,M_1) $\wedge$ Missile(M_1)

Missile(x) $\wedge$ Owns(Nono,x) $\Rightarrow$ Sells(West,x,Nono)

Missile(x) $\Rightarrow$ Weapon(x) Enemy(x,America) $\Rightarrow$ Hostile(x)

American(West) Enemy(Nono,America)

Backward chaining example



American(x) $\wedge$ Weapon(y) $\wedge$ Sells(x,y,z) $\wedge$ Hostile(z) $\Rightarrow$ Criminal(x)

Owns(Nono,M_1) $\wedge$ Missile(M_1)

Missile(x) $\wedge$ Owns(Nono,x) $\Rightarrow$ Sells(West,x,Nono)

Missile(x) $\Rightarrow$ Weapon(x)   Enemy(x,America) $\Rightarrow$ Hostile(x)

American(West)   Enemy(Nono,America)

Resolution: FOL version

$$\frac{p_1 \vee \cdots \vee p_k, \qquad q_1 \vee \cdots \vee q_n \quad \text{such that } \mathrm{UNIFY}(p_i, \neg q_j) = \theta}{\mathrm{SUBST}(\theta, p_1 \vee \cdots \vee p_{i-1} \vee p_{i+1} \vee \cdots \vee p_k \vee q_1 \vee \cdots \vee q_{j-1} \vee q_{j+1} \vee \cdots \vee q_n)}$$

- For example,

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

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

- Refutation Proof: Apply resolution steps to $CNF(KB \wedge \neg\alpha)$; complete for FOL

Example: Resolution

1- $\neg$ Smart(x) $\vee$ $\neg$ LikesHockey(x) $\vee$ NorthSchool(x)

2- $\neg$ Canadian(y) $\vee$ LikesHockey(y)

3- $\neg$ Skates(z) $\vee$ LikesHockey(z)

4- Smart(Joe)

5- Skates(Joe)

Goal is to find out if NorthSchool(Joe) is true NorthSchool(Joe) ?

3+5= LikesHockey(Joe)….(6)

4+1= $\neg$ LikesHockey(Joe) $\vee$ NorthSchool(Joe) ….(7)

6+7= NorthSchool(Joe) Goal

Next Refutational Proof

Example:Resolution Refutation

1- ¬ Smart(x) ∨ ¬ LikesHockey(x) ∨ NorthSchool(x)

2- ¬ Canadian(y) ∨ LikesHockey(y)

3- ¬ Skates(z) ∨ LikesHockey(z)

4- Smart(Joe)

5- Skates(Joe)

0- ¬ NorthSchool (Joe) (Goal: NorthSchool (Joe))

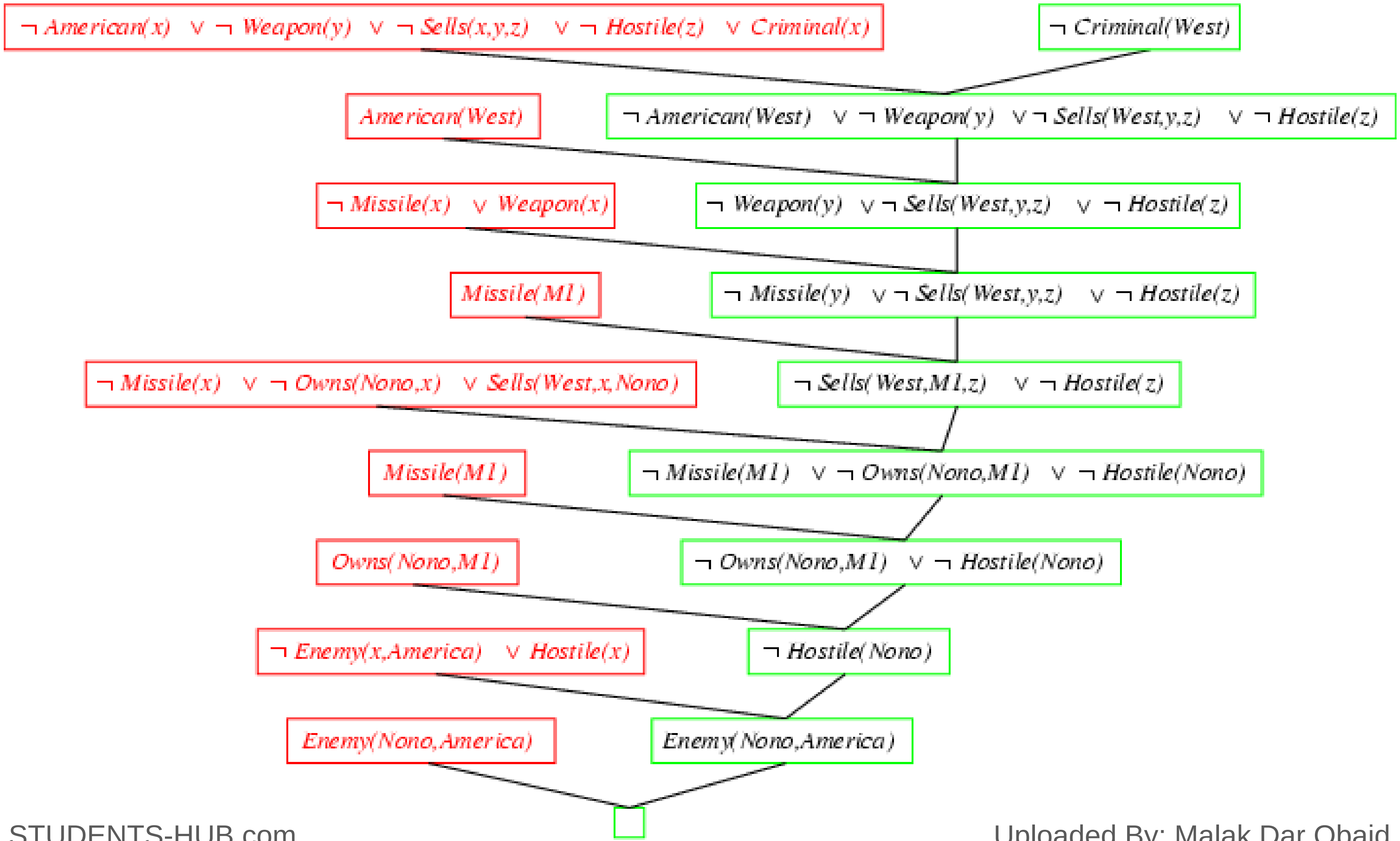
0+1= ¬ Smart(Joe) ∨ ¬ LikesHockey(Joe) ….(6)

3+5= LikesHockey(Joe)….(7)

4+7= ¬ Skates(Joe)….(8)

0+8=•

Resolution proof: definite clauses


¬ American(x) ∨ ¬ Weapon(y) ∨ ¬ Sells(x,y,z) ∨ ¬ Hostile(z) ∨ Criminal(x)
¬ Criminal(West)
American(West)
¬ American(West) ∨ ¬ Weapon(y) ∨ ¬ Sells(West,y,z) ∨ ¬ Hostile(z)
¬ Missile(x) ∨ Weapon(x)
¬ Weapon(y) ∨ ¬ Sells(West,y,z) ∨ ¬ Hostile(z)
Missile(M1)
¬ Missile(y) ∨ ¬ Sells(West,y,z) ∨ ¬ Hostile(z)
¬ Missile(x) ∨ ¬ Owns(Nono,x) ∨ Sells(West,x,Nono)
¬ Sells(West,M1,z) ∨ ¬ Hostile(z)
Missile(M1)
¬ Missile(M1) ∨ ¬ Owns(Nono,M1) ∨ ¬ Hostile(Nono)
Owns(Nono,M1)
¬ Owns(Nono,M1) ∨ ¬ Hostile(Nono)
¬ Enemy(x,America) ∨ Hostile(x)
¬ Hostile(Nono)
Enemy(Nono,America)
Enemy(Nono,America)

Logic programming: Prolog

- **FOL:**

  $King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

  $Greedy(y)$

  $King(John)$

- **Prolog:**

```
evil(X) :- king(X), greedy(X).
greedy(Y).
king(john).
```

- Closed-world assumption:
  - Every constant refers to a unique object
  - Atomic sentences not in the database are assumed to be false
- Inference by backward chaining, clauses are tried in the order in which they are listed in the program, and literals (predicates) are tried from left to right

Prolog example

```
parent(abraham,ishmael).
parent(abraham,isaac).
parent(isaac,esau).
parent(isaac,jacob).

grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
descendant(X,Y) :- parent(Y,X).
descendant(X,Y) :- parent(Z,X), descendant(Z,Y).

? parent(david,solomon).
? parent(abraham,X).
? grandparent(X,Y).
? descendant(X,abraham).
```

Prolog example

```
parent(abraham,ishmael).
parent(abraham,isaac).
parent(isaac,esau).
parent(isaac,jacob).
```

- What if we wrote the definition of `descendant` like this:

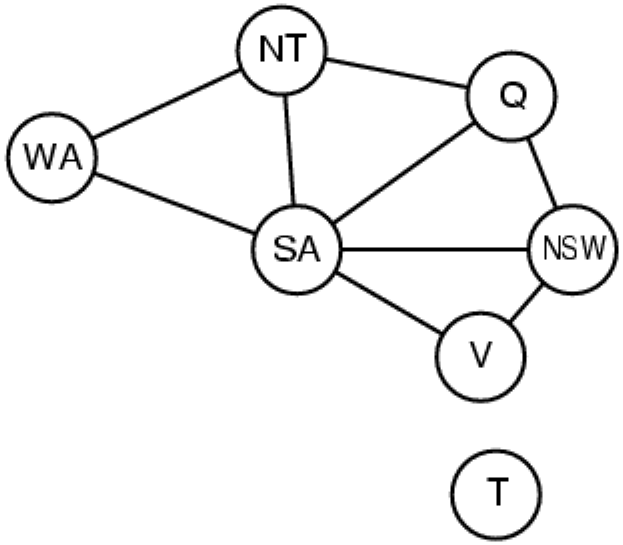
```
descendant(X,Y) :- descendant(Z,Y), parent(Z,X).
descendant(X,Y) :- parent(Y,X).

? descendant(W,abraham).
```

- Backward chaining would go into an infinite loop!
  - Prolog inference is **not complete**, so the ordering of the clauses and the literals is really important

Applications of Automated Reasoning

- Diagnosis of Digital Circuits
- Software/hardware verification
- Rule Based Systems (customer Support)
- Mathematics: proving theorems
- Deductive databases: rules instead of facts: can save lots of space: database constraints (salaries of managers >supervised_employee)
- Ways to treat negation! Can’t prove something

Graph coloring



```
colorable(Wa,Nt,Sa,Q,Nsw,V) :-
diff(Wa,Nt), diff(Wa,Sa), diff(Nt,Q), diff(Nt,Sa), diff(Q,Nsw),
diff(Q,Sa), diff(Nsw,V), diff(Nsw,Sa), diff(V,Sa).

diff(red,blue).        diff(red,green).       diff(green,red).
diff(green,blue).      diff(blue,red).        diff(blue,green).
```

Prolog lists

- Appending two lists to produce a third:

```
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

- query: `append(A,B,[1,2])`

- answers:

```
A=[]    B=[1,2]
A=[1]   B=[2]
A=[1,2] B=[]
```