Knowledge Representation

- Uptonow: State representations and search techniques
  → **How can one represent what one knows about a problem domain?**
  → **How to reason about that knowledge for answering questions for making decisions?**

- If knowledge is represented in the language of logic → ”right” reasoning
- ”Reasoning” = to determine what implicitly follows from explicitly represented facts

- ”Cool idea”:
  if initial facts are represented correctly, ”mechanistic” inference perserves correctness of inferred facts.
  Reasoning is search applied to knowledge states

Logics as Language

- **Logics are formal languages for representing information so that conclusions can be drawn**
- Syntax defines the sentences in the language → Grammar
- Semantics define the ”meaning” of sentences; define truth of a sentence in a world
- E.g. Language of arithmetrics:

  $x+2 \geq y$ is a sentence; $x2+y>$ is not a sentence

  $x+2 \geq y$ is true, iff the number $x+2$ is no less than the number y

  $x+2 \geq y$ is true in a world in which $x=7$ and $y=1$

  $x+2 \geq y$ is true in a world in which $x=0$ and $y=6$

How to represent ”Knowledge”?

- Formal Logics
  - → Propositional Logic
  - → Predicate Logic
  - – Modal Logic
  - – Fuzzy Logic
- Semantic networks
- Frames and Description Logic
- Specific languages for time, space ...

→ Big variety of knowledge representation languages

→ Tradeoff between

- Expressiveness (what kind of objects/replateions be represented)
- Allows tractable/efficient reasoning

Propositional Logic

- Language for representing and reasoning about statements
- Propositions are connected by logical operators:
  - Statements ”the street is wet” or ”it is raining” are propositions:

    If it is raining, the street is wet
  - More formally:

    It_is_raining $\Rightarrow$ street_is_wet

Let Op=$\{\neg, \wedge, \vee, \Leftrightarrow, \Rightarrow, (,)\}$ be the set of logical operators and Σ a set of symbols. The sets Op, Σ and $\{t, f\}$ are pairwise disjoint.
The set of propositional logic formula is recursively defined:

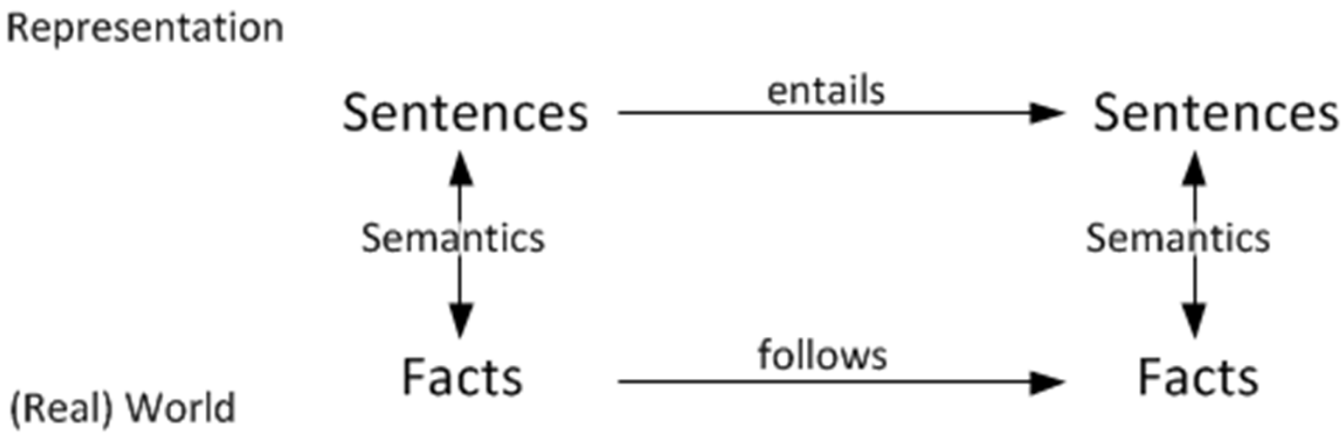
- t and f are (atomic) formula
- All elements of Σ are (atomic) formula
- If A and B are formulas then $\neg$A, A $\wedge$ B, A $\vee$ B, A$\Leftrightarrow$B, A$\Rightarrow$B, (A) are also formulas

Operators

- t: ”true”
- f: ”false”
- $\neg$A: ”not A” (Negation)
- A $\wedge$ B: ”A and B” (Conjunction)
- A $\vee$ B: ”A or B” (Disjunction)
- A$\Rightarrow$B: ”if A, then B” (Implication)
- A$\Leftrightarrow$B: ”A if and only if B” (Equivalence)

These are just **syntactical** operators for constructing formulas; what does this mean?

Representation and Reality



- ”Truth”: A sentence is true, if what it describes is actually the case in reality
- A ”Knowledge Base” (KB) B is a set of propositions (sentences) that are all connected with $\wedge$, describing a situation in which the world is in
- Entailment means that one thing follows from another: KB $\models \alpha$
  ”Knowledge base KB entails sentence α” if and only if α is true in all worlds in which KB is true

Semantics

- Propositional Logics → 2 truth values: t (”true”) and f (”false”)
- Truth value of A $\wedge$ B depends on the truth values of A and B → given the truth values of the symbols in a sentence, its truth value can be determined (”Interpretation”)
- Assigning truthvalues to propositions → Reflect the state of the world
- **Definition of logical operators by truth table:**

A	**B**	**$\neg$A**	**A$\wedge$B**	**A$\vee$B**	**A$\Rightarrow$B**	**A$\Leftrightarrow$B**
f	*f*	*t*	*f*	*f*	*t*	*t*
f	*t*	*t*	*f*	*t*	*t*	*f*
t	*f*	*f*	*f*	*t*	*f*	*f*
t	*t*	*f*	*t*	*t*	*t*	*t*

More Terminology... Formula

- Two formulas F and G are called logically **equivalent**, if they take on the same truth value for all interpretations: **F ≡ G**
  (That is a meta-level statement; F⇔B is a syntactic object of the language)

- A formula is called
  - **Satisfiable** if it is true for **at least** one interpretation;
  - **Valid**, if it is true for all interpretations (Formulas which are always true are called ”tautologies”)
  - **Unsatisfiable**, if it is not true for any interpretation
- Every interpretation that satisfies a formula is called a **model** of the formula

Helpful: Logical Equivalence

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

Inference

- For answering questions we need **"Inference"**
- **KB $\vdash_i \alpha$ means sentence α can be derived from KB by procedure i**
- Consequences of KB are a haystack; α is a needle
  Entailment = needle in haystack; inference = finding it

- **Soundness**: i is sound if
  whenever KB $\vdash_i \alpha$ it is also true that KB $\models \alpha$
- **Completeness**: i is complete if
  whenever KB $\models \alpha$ it is also true that KB $\vdash_i \alpha$
- That means, the procedure will answer any question whose answer follows from what is known by the KB

Inference/Proof Methods

Proof methods divide into (roughly) two kinds:

- **Model Checking**
  - Models are assignments of true and false to every proposition symbol (every atomic proposition)
  - Truth Table Enumeration
  - Improved backtracking
  - Heuristic search in model space (sound but incomplete)
    e.g. min-conflicts-like hill-climbing algorithms
- **Application of inference rules**
  - Legitimate (sound) generation of new sentences from old
  - Proof = a sequence of inference rule applications
    Can use inference rules as operators in a standard search algorithm
  - Typically require translation of sentences into a normal from

Truth Tables

- Use a truth table to figure out in which interpretations the following formula is true:

  $((a \Rightarrow b) \wedge a) \Rightarrow b)$

a	b	$a \Rightarrow b$	$(a \Rightarrow b) \wedge a$	$((a \Rightarrow b) \wedge a) \Rightarrow b)$
false	*false*			
false	*true*			
true	*false*			
true	*true*			

- N symbols → 2^N rows

Inference by enumeration: Example

- Use a truth table to figure out whether ”S – people are depressed” is entailed in the following KB:

  ”If it is dark and raining, people are depressed (sad).”
  ”If it is raining, then it is dark.”
  ”It is raining.”

  $D \wedge R \Rightarrow S$
  $R \Rightarrow D$
  R

  (Knowledge base sentences are coupled by $\wedge$)

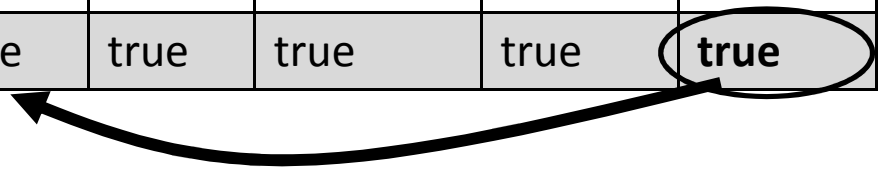
→ List all true/false settings for D, R, S and the consequences of these settings for all sentences; If all sentences in the knowledgebase are true, then check whether S is true in these settings; **if this is the case, then S is entailed in the KB**

Inference by Enumeration: Example

D	R	S	D∧R	D∧R⇒S	R⇒D	KB
false	false	false	false	true	true	false
false	false	true	false	true	true	false
false	true	false	false	true	false	false
false	true	true	true	true	false	false
true	false	false	false	true	true	false
true	false	true	false	true	true	false
true	true	false	true	false	true	false
true	true	true	true	true	true	true

Inference by Enumeration: Example

D	R	S	D∧R	D∧R⇒S	R⇒D	KB
false	false	false	false	true	true	false
false	false	true	false	true	true	false
false	true	false	false	true	false	false
false	true	true	true	true	false	false
true	false	false	false	true	true	false
true	false	true	false	true	true	false
true	true	false	true	false	true	false
true	true	true	true	true	true	**true**



In all rows in which KB is true, also S is true → KB entails S

Inference by Enumeration

- Depth-first enumeration and **check whether** α is true in all situations in which KB is true, is **sound** and **complete**

```
function TruthTable-Entails? (KB, α) returns true or false
        symbols ← list of all atomic proposition symbols in KB and α
        return TT-check-all (KB, α, symbols, {})

function TT-check-all (KB, α, symbols, model) returns true or false
        if empty?(symbols) then
                if PL-true?(KB, model) then return PL-true?(α, model)
                else return true
        else
                P ← first ( symbols)
                rest ← rest (symbols)
                return (TT-check-all (KB, α, rest, model ∪ {P=true}))
                                and
                        (TT-check-all (KB, α, rest, model ∪ {P=false}))
```

→ This is a Constraint Satisfaction problem!

Faster way?

- Simple enumeration is exponential: N Symbols → 2^N rows
- Faster way: **using a procedure based on sound rules of inference**: e.g.:
- Whenever R⇒D and R are true, D is true as well

R	**D**	**R⇒D**
false	false	true
false	true	true
true	false	false
true	true	true

→ This is called ”Modus Ponens”

Modus Ponens

- **Modus Ponens** (for Horn Form): **complete** for KBs that are represented in Horn clauses

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- Can be used with **forward chaining** or **backward chaining**
  These algorithms are very natural and run in linear time in the size of KB

Horn clauses

- **Conjunctive Normal Form**: Conjunction of disjunctions
- Horn Form (restricted): **KB = conjunction of Horn clauses**
  Horn Clause =
  - single proposition symbol; or
  - (conjunction of symbols) $\Rightarrow$ symbol

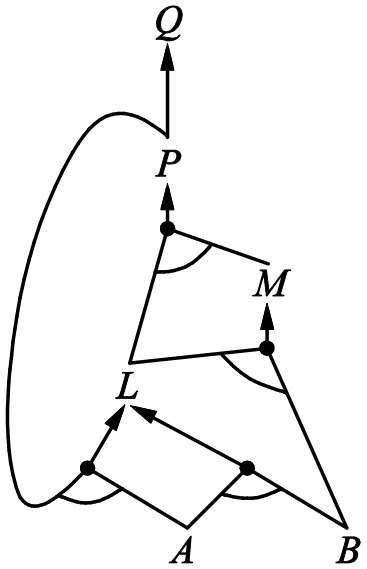
  e.g. $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

  equivalent form: $C \wedge (\neg B \vee A) \wedge (\neg C \vee \neg D \vee B)$
  (equals CNF with only one positive literal)
- **Definite Clauses** possess exactly one literal on the right side of the implication → ”Rules”
- In Horn form premises (the left side) are called ”body”, the conclusion on the right side is called ”head”
  Sentence consisting of a single positive literal is called a **fact**

Forward Chaining

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

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

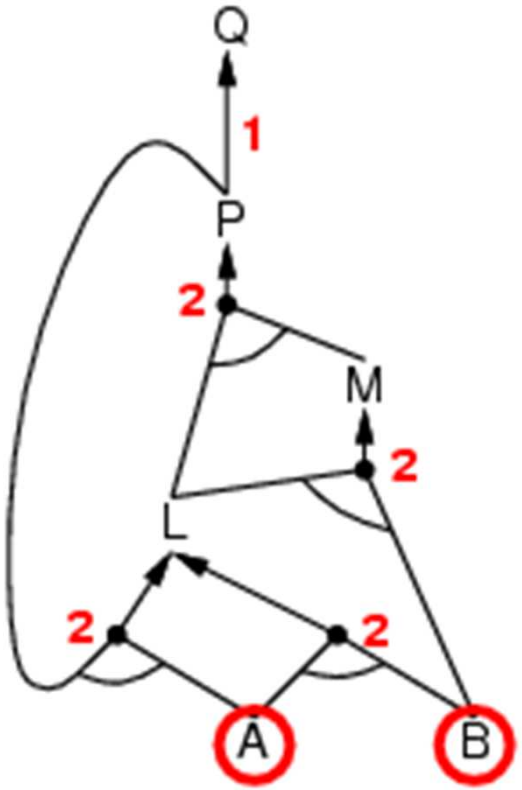


- Forward Chaining is sound and complete for Horn KB

Forward Chaining Algorithm

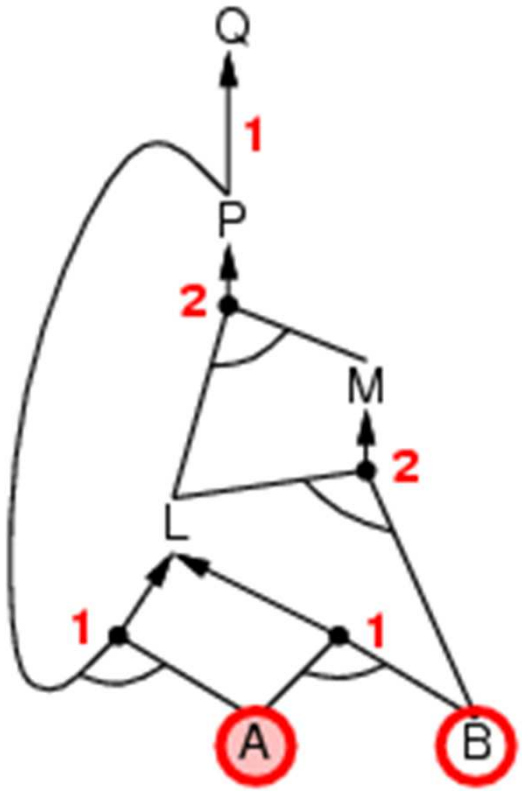
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known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

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

Forward Chaining Example



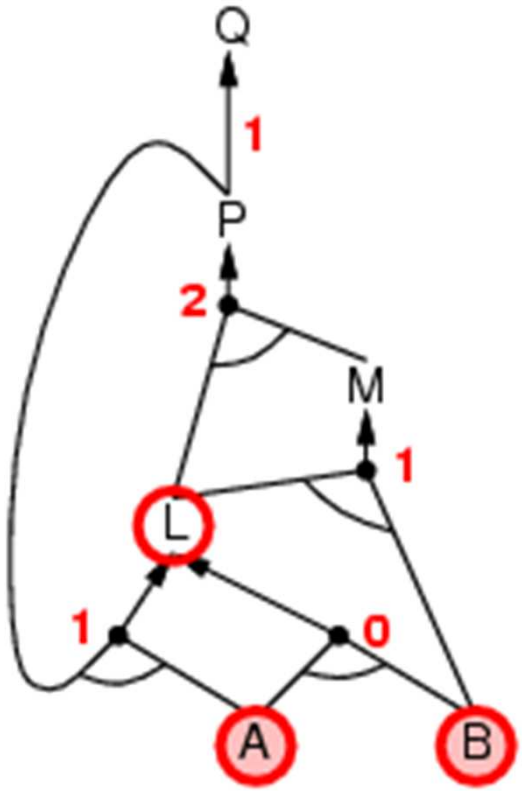
Initialization:
For each horn clause, note the number of premises to be checked;
agenda = {A, B}

Forward Chaining Example



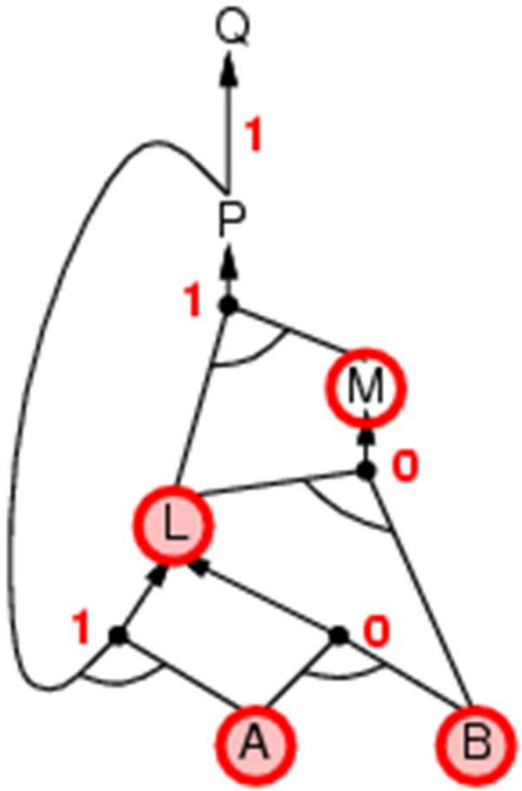
Start by tackling A:
inferred[A] ← true
decrement counter at clauses at which A appears as premise

Forward Chaining Example



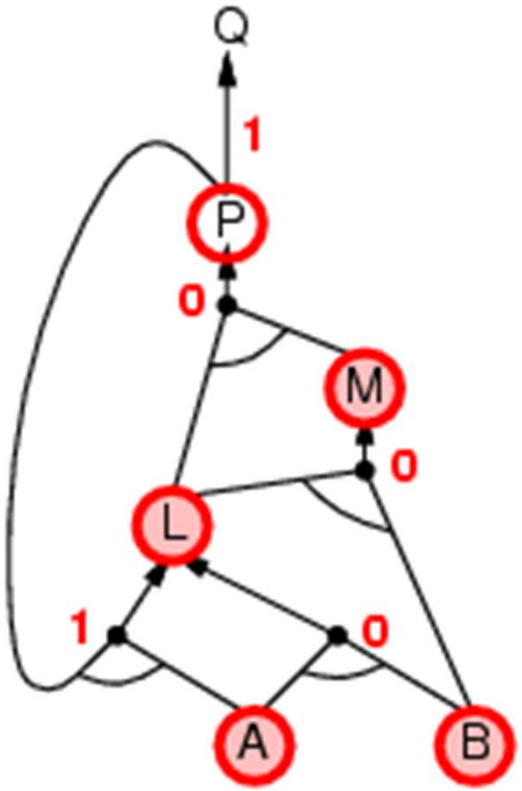
Tackling B:
inferred[B] ← true
decrement counter at clauses at which B appears as premise
→ count(A∧B)=0
→ add L to agenda

Forward Chaining Example



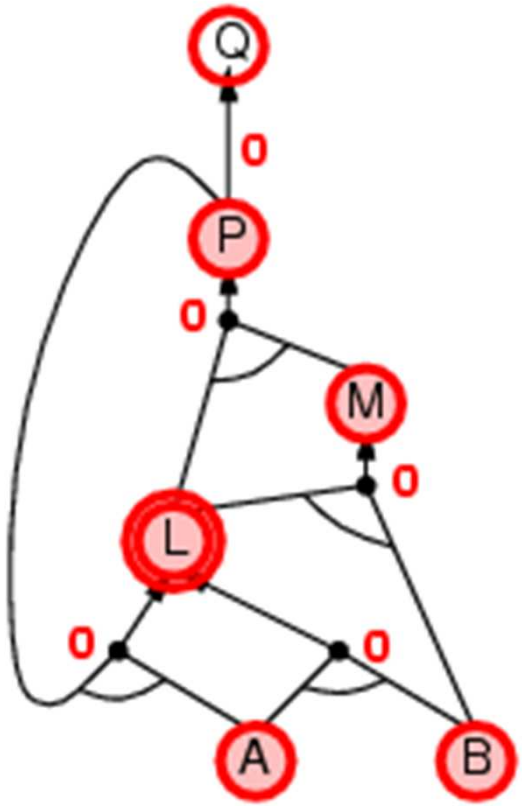
Tackling L:
inferred[L] ← true
decrement counter at clauses at which L appears as premise:
→ count(L∧B)=0
→ add M to agenda

Forward Chaining Example



Tackling M:
inferred[M] ← true
decrement counter at clauses at which M appears as premise:
→ count(L∧M)=0
→ add P to agenda

Forward Chaining Example



Tackling P:
inferred[L] ← true
decrement counter at clauses at which Lappears as premise
→ count(A∧P)=0
L already inferred; no change
→ count(P)=0
→ Q equals query ✓

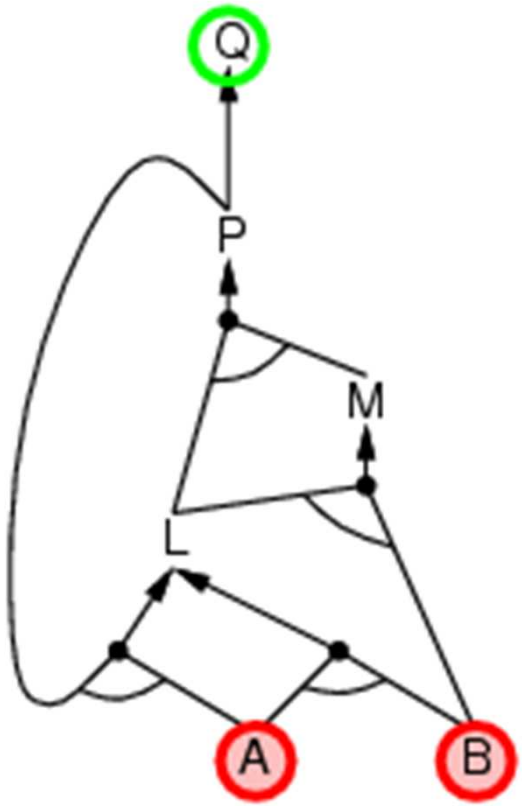
Forward Chaining derives every atomic sentence that is entailed by KB

Backward Chaining

- Idea: work backwards from the query *q*:
  to prove *q* by Backward Chaining,
  - check is *q* is known already or
  - prove all premises of some rule lead to *q*

- **Avoid loops**: check if new subgoal is already on the goal stack
- **Avoid repeated work**: check if new subgoal
  - has already been proved true, or
  - has already failed

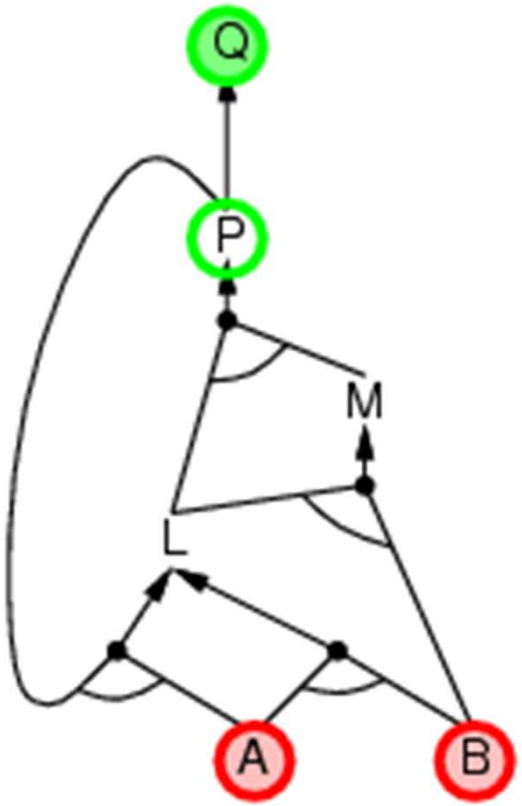
Backward Chaining Example



known are {A, B}
remaining goal stack { Q }

→ try to proof the premises of Q

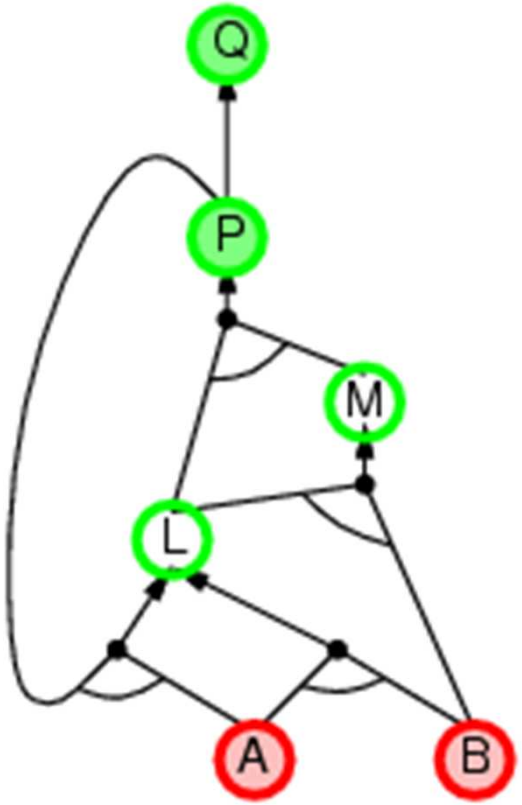
Backward Chaining Example



known are {A, B}
remaining goal stack { P, Q }

→ try to proof the premises of P

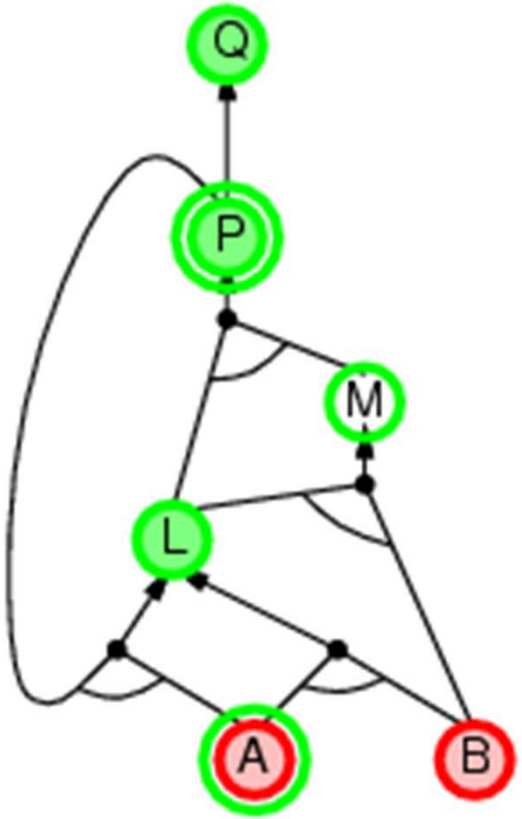
Backward Chaining Example



known are {A, B}
remaining goal stack { L, M, P, Q }

→try to proof the premises of M and L

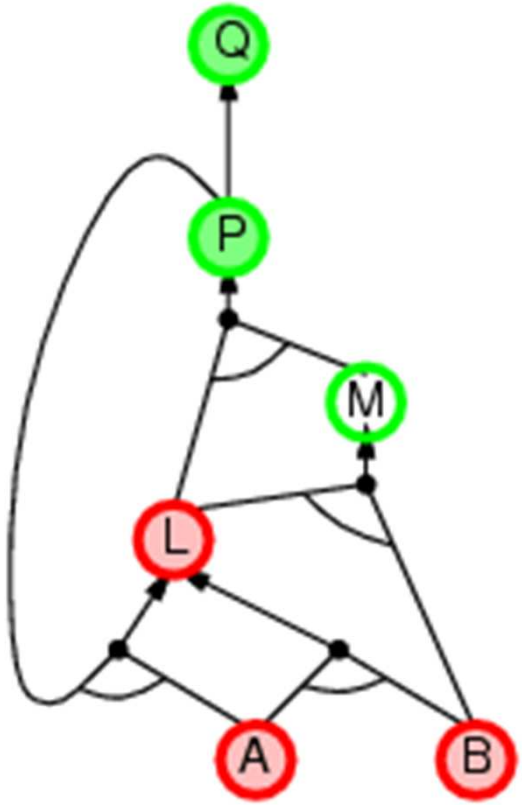
Backward Chaining Example



known are {A, B}
remaining goal stack { L, M, P, Q }

→ Try to prove P∧A →
P is already on goal stack,
A is known to be true
→ continue

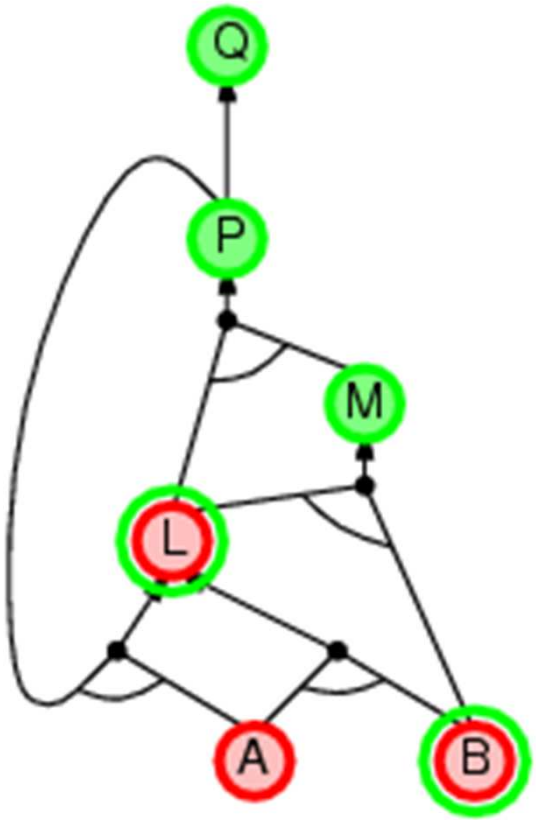
Backward Chaining Example



known are {A, B, L}
remaining goal stack: {M, P, Q}

→ try to prove A∧B for deriving L
→ goal L satisfied

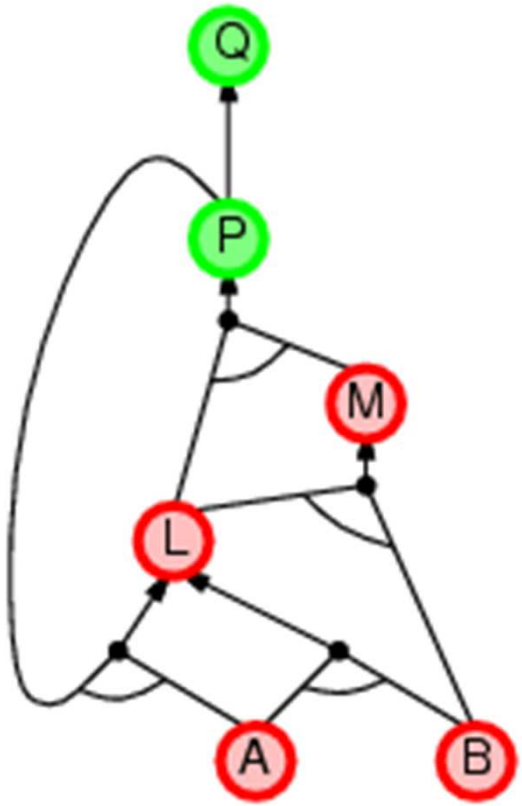
Backward Chaining Example



known are {A, B, L}
remaining goal stack: {M, P, Q}

→ try to prove L∧B for deriving M

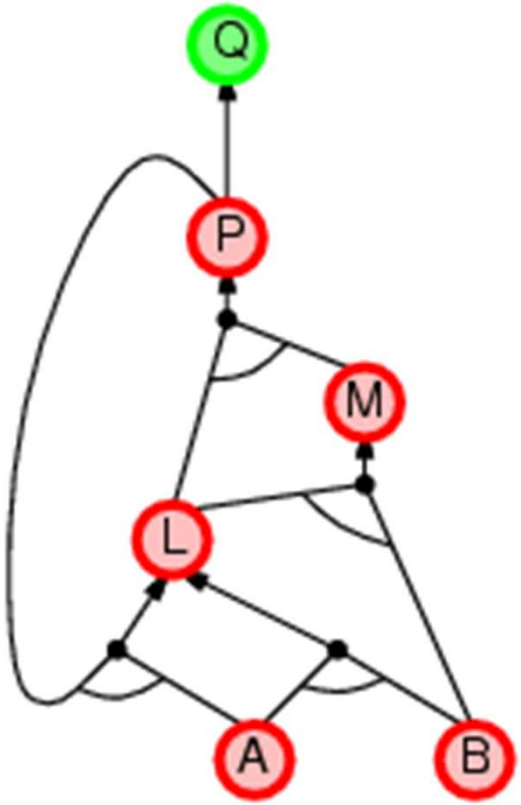
Backward Chaining Example



known are {A, B, L,M}
remaining goal stack: {P, Q}

→ try to prove L∧B for deriving M
→ goal M satisfied

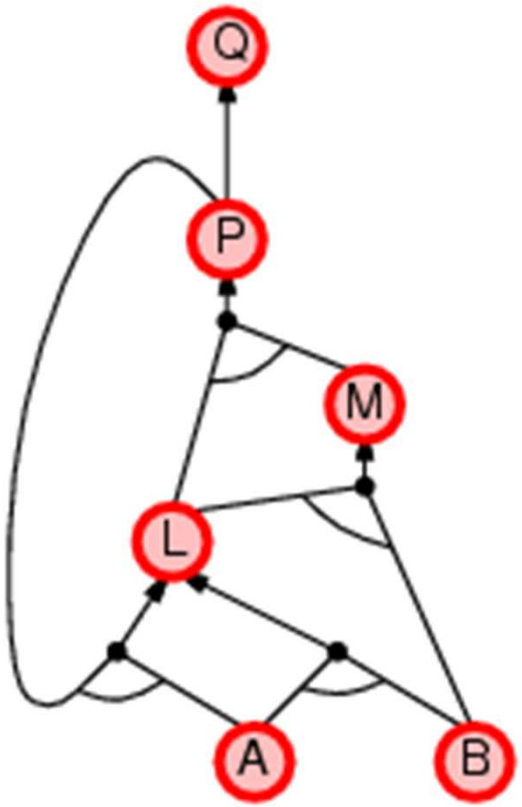
Backward Chaining Example



known are {A, B, L, M}
remaining goal stack: {Q}

→ with L∧M → goal P satisfied

Backward Chaining Example



known are {A, B, L, M, P}
remaining goal stack: {}

→ with P → goal Q satisfied ✓

Forward vs. Backward Chaining

- Forward Chaining is **data-driven**
  – like automatic, unconscious processing
  (e.g. object recogition, routine decisions)
- May do lots of work that is **irrelevant** to the goal

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

Inference Rules

- **Modus Ponens**

$$\frac{a,\ a \Rightarrow b}{b}$$

- **Modus Tollens**

$$\frac{\neg b,\ a \Rightarrow b}{\neg\, a}$$

- **Hypothetical Syllogism**

$$\frac{(a \Rightarrow b),\ (b \Rightarrow c)}{(a \Rightarrow c)}$$

- **Case Elimination**

$$\frac{(a \vee b),\ \neg b}{a}$$

- **Addition**

$$\frac{a,}{a \vee b}$$

- **Simplification**

$$\frac{a \wedge b}{a}$$

- **Conjunction**

$$\frac{a,\ b}{a \wedge b}$$

- **Resolution**

$$\frac{(a \vee b),\ (a \vee \neg b)}{a}$$

Resolution

- proof by contradiction, i.e. show that KB $\wedge \neg\alpha$ is unsatisfiable
- **Sound and complete for propositional logic**
- Based on representation in Conjunctive Normal Form (CNF-universal): conjunction of $\underbrace{\text{disjunctions of literals}}_{\text{clauses}}$

  e.g. $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- **General Resolution inference rule (for CNF)**:

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

  where l_i and m_j are **complementary** literals,

- That means: take all literals from two clauses and produce a new clause with all literals except the two complementary ones

Procedure

- To prove that α holds with respect to axioms KB
  1. Convert all propositions in KB into CNF
  2. Negate α and convert into CNF
  3. Repeat until a contradiction is found
     a) Select two clauses
     b) Resolve these two clauses (produce new clause = resolvent)
     c) If resolvent is empty, a contradiction have been found

→ $KB \wedge \neg\alpha$ is unsatisfiable

→ $KB \models \alpha$

Conversion to CNF

$A \Leftrightarrow (B \vee C)$

1. Eliminate $\Leftrightarrow$: replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$

   $(A \Rightarrow (B \vee C)) \wedge ((B \vee C) \Rightarrow A)$

2. Eliminate $\Rightarrow$: replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$

   $(\neg A \vee (B \vee C)) \wedge (\neg(B \vee C) \vee A)$

3. Move $\neg$ inwards using de Morgan's rule an double negation

   $(\neg A \vee B \vee C) \wedge ((\neg B \wedge \neg C) \vee A)$

4. Apply distributivity law ($\vee$ or $\wedge$) and flatten

   $(\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A)$

Example

- If the criminal had an accomplice, then he came in a car. The criminal had no accomplice and did not have the key, or he had the key and an accomplice. The criminal had the key.
- Proof **using resolution** whether the criminal came by car?

Propositional Logic

+ Propositional Logic is **declarative**: pieces of syntax correspond to facts – programming ”what to solve” instead of ”how to solve”
+ Propositional Logic is **compositional**: meaning of $A \wedge B$ is derived from meaning of A and B
+ Propositional Logic allows **partial/disjunctive/negated** information
+ Meaning in propostional logic is **context-independent** (unlike natural language)
– Propositional Logic has **very limited** expressive power:
e.g. cannot say ”all birds can fly”, except by writing one statement for each bird

First-order Logic

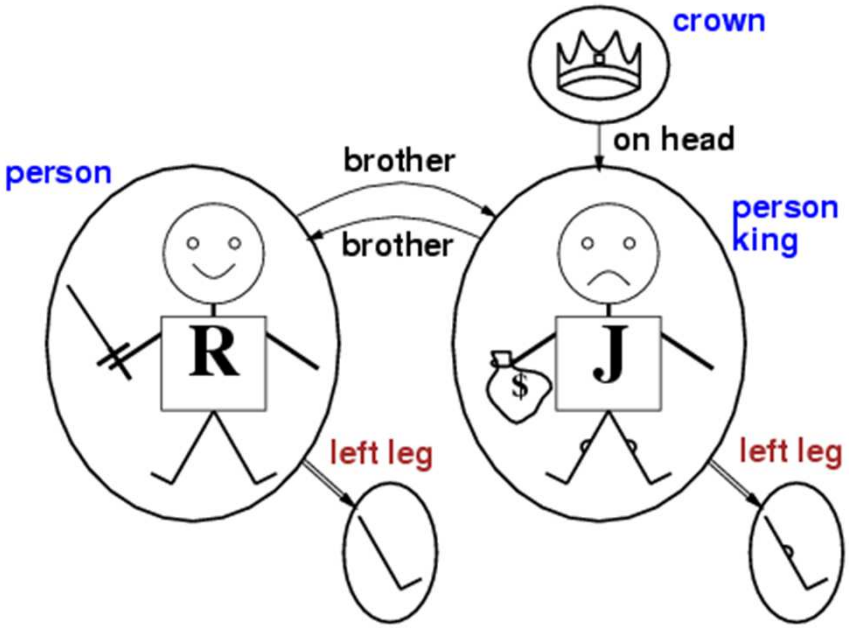
- Whereas propositional logic assumes that the world is described by facts, first order logic assumes the world contains
  - **Objects** (people, houses, numbers, ...)
  - **Relations** (red, round, bogus, prime,.. brother-of, bigger-than)
  - **Functions**: fatherof, best friend, third inning of, one more than , end of...

Truth in First-Order Logic

- Sentences are true with respect to a model and an interpretation
- Model contains ≥ 1 objects (domain elements) and relations among them
- Interpretation specifies referents for
  - constant symbols → objects
  - predicate symbolds → 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***

Example



Symbols

- Constants: R, J
- Predicates: Brother, OnHead, Person, King, Crown
- Function: LeftLeg

Terms as logical expressions referring to an object:

- John
- LeftLeg(J)

Interpretation:
R→ **Richard the Lionheart**
J → **the evil King John**
Brother → the brotherhood relation

Under this interpretation, Brother(R, J) is true just in the case **Richard the Lionheart** and **the evil King John** are in the brotherhood relation in the model

→ Computing entailment by enumerating FOL models is not easy

Syntax of FOL (with equality)

Sentence	$\rightarrow$	AtomicSentence | ComplexSentence
AtomicSentence	$\rightarrow$	Predicate | Predicate (Term, ...) | Term = Term
ComplexSentence	$\rightarrow$	(Sentence) | [Sentence]
		$\mid \neg$ Sentence | Sentence $\wedge$ Sentence
		| Sentence $\vee$ Sentence | Sentence $\Rightarrow$ Sentence
		| Sentence $\Leftrightarrow$ Sentence
		| Quantifier Variable, ... Sentence
Term	$\rightarrow$	Function (Term,...) | Constant | Variable
Quantifier	$\rightarrow$	$\forall \mid \exists$
Constant	$\rightarrow$	A | X_1 | John | ...
Variable	$\rightarrow$	a | x | s ...
Predicate	$\rightarrow$	True | False | Loves | Raining | After ...
Function	$\rightarrow$	MotherOf | LeftLegOf ...

Universal Quantification

$\forall \langle$***variables***$\rangle \langle$***sentence***$\rangle$

- Everyone at Örebro University is smart:
  $\forall x \;\; at(x, \mathit{ÖrebroUniversity}) \Rightarrow \mathit{Smart}(x)$
- $\forall x \; P$ is true in a model m iff P is true with x being **each** possible object in the model
- Roughly speaking: equivalent with conjunction of all possible instantiations of P.

- Typically, $\Rightarrow$ is the main connective with $\forall$
- **Common mistake**: using $\wedge$ as the main connective with $\forall$
  $\forall x \;\; \mathit{At}(x, \mathit{ÖrebroUniversity}) \wedge \mathit{Smart}(x)$
  means ”Everyone is at Örebro University and everyone is smart”

Existential Quantification

$\exists \langle$***variables***$\rangle$ $\langle$***sentence***$\rangle$

- Someone at Örebro University is smart:
  $\exists x$ *At(x, ÖrebroUniversity)* $\wedge$ *Smart(x)*
- $\exists x\ P$ is true in a model *m* iff *P* is true with *x* being **some possible** object in the model
- Roughly speaking: equivalent with disjunction of all possible instantiations of *P*.

- Typically, $\wedge$ is the main connective with $\exists$
- **Common mistake**: using $\Rightarrow$ as the main connective with $\exists$
  $\exists x$ *At*(*x,ÖrebroUniversity*) $\Rightarrow$ *Smart(x)*

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

- Quantifier Duality:
  $\forall x\ Likes(x, IceCream)$ $\neg\exists x\ \neg Likes(x, IceCream)$
  $\exists x\ Likes(x, Broccoli)$ $\neg\forall x\ \neg Likes(x, Broccoli)$

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

Example Knowledge Base: Kinship Domain

- Objects in the domain are **people**
- Unary predicates: Male and Female;
  Kinship relations are represented using binary predicates:
  Parent, Brother, Child, Dauther, Wife, Grandchild, Cousin, Aunt...
- Functions for mother and father, as every person has exactly one
- What we know ... **Axioms** of the Kinship Domain (definitions based on a set of basic predicates)

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

  $\forall$w,h $Husband(h,w) \Leftrightarrow (Male(m) \wedge Spouse(h,w))$

  $\forall$x $Male(x) \Leftrightarrow \neg Female(x)$

  $\forall$p,c $Parent(p,c) \Leftrightarrow Child(c,p)$

  $\forall$x,y $Sibling(x,y) \Leftrightarrow x \neq y \wedge \exists p\ Parent(p,x) \wedge Parent(p,y)$

  .....
- "Theorems" are entailed by axioms

Reduction to Propositional Logic

Suppose the KB contains just the following:

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

$King(John)$

$Greedy(John)$

$Brother(Richard,John)$

Instantiating the universal sentence in **all possible** ways, we have:

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

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

$King(John)$

$Greedy(John)$

$Brother(Richard, John)$

The new KB is "**propositionalized**": proposition symbols are King(John), Greedy(John), Evil(John), King(Richard), etc

Every FOL KB can be **propositionalized so as to preserve entailment**

Unification

- Without propositionalizing, how can we do inference?
  For example, we know:
  - $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
  - *King(John)*
  - *Greedy(John)*
- For deriving Evil(John), we need to "match" x with John:
  We need to find a substitution for the variable x, so that the expressions King(x) and King(John), Greedy(y) and Greedy(John) become **equal.**
- **If we find such a substitution, all the rules from propositional logic can be applied!**

→ **Unification is the process of finding a substitution for variables in formulas, so that two formulas are made equal**

Unification

- Substitution θ is some form of table relating variables to their counterparts
- Unify(α,β) = θ if αθ = βθ
  that means: θ is a unifier for α and β, if the two sentences are equal after applying the substitution

p	q	θ
Knows(John, x)	Knows(John, Jane)	
Knows(John, x)	Knows(v, OJ)	
Knows(John, x)	Knows(y, Mother(y))	
Knows(John, x)	Knows(x, OJ)	

- **Standardizing apart** eliminates overlap of variables (see last sentence), e.g., Knows(z_{17},OJ)

Unification

- Substitution θ is some form of table relating variables to their counterparts
- Unify(α,β) = θ if αθ = βθ
  that means: θ is a unifier for α and β, if the two sentences are equal after applying the substitution

p	q	θ
Knows(John, x)	Knows(John, Jane)	{x/Jane}
Knows(John, x)	Knows(v, OJ)	{x/OJ, y/John}
Knows(John, x)	Knows(y, Mother(y))	{y/John, x/Mother(John)}
Knows(John, x)	Knows(x, OJ)	{fail}

- **Standardizing apart** eliminates overlap of variables (see last sentence), e.g., Knows(z_{17},OJ)

Most General Unifier

- There is often more than one possibility to unify two formulas:

  Unify(*Knows(John,x)*, *Knows(y,z)*,)
  θ = {y/John, x/z }
  θ = {y/John, x/John, z/John}

- The first unifier is **more general** than the second.

- We do not just unify two formulas, but apply unifcation in most of the reasoning steps → so we do not want to make more restrictive assignments than necessary → always use the **most general unifier**
- There is a single **most general unifier** (MGU) that is unique up to renaming of variables: MGU = { y/John, x/z }

→ **Unification algorithm finds it**

Unification Algorithm

function UNIFY(x, y, θ) **returns** a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound
    y, a variable, constant, list, or compound
    θ, the substitution built up so far

  if θ = failure **then return** failure
  else if $x = y$ **then return** θ
  else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) **and** COMPOUND?(y) **then**
    return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
  else return failure

Unification Algorithm

```
function UNIFY-VAR(var, x, θ) returns a substitution
    inputs: var, a variable
            x, any expression
            θ, the substitution built up so far

    if {var/val} ∈ θ then return UNIFY(val, x, θ)
    else if {x/val} ∈ θ then return UNIFY(var, val, θ)
    else if OCCUR-CHECK?(var, x) then return failure
    else return add {var/x} to θ
```

With unification we have the tool for extending the inference mechanisms from Propositional Logic to First-Order Predicate Logic:

Generalized Modus Ponens

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$

where $p_i'\theta = p_i\,\theta$ for all *i*

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

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

Prerequisites:

- Generalized Modus Ponens must have KB of **definite clauses** (exactly one positive literal,
  = Horn clauses with one literal on right of ⇒)
- All variables assumed universally quantified

Example Problem

- 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: Colonel West is a criminal

Example Knowledge Base

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

$$\textit{American}(x) \land \textit{Weapon}(y) \land \textit{Sells}(x,y,z) \land \textit{Hostile}(z) \Rightarrow \textit{Criminal}(x)$$

Nono … has some missiles, i.e., $\exists x\ \text{Owns}(\text{Nono},x) \land \text{Missile}(x)$:

$$\textit{Owns}(\text{Nono},M_1) \text{ and } \textit{Missile}(M_1)$$

… all of its missiles were sold to it by Colonel West

$$\textit{Missile}(a) \land \textit{Owns}(\text{Nono},a) \Rightarrow \textit{Sells}(\text{West},a,\text{Nono})$$

Missiles are weapons:

$$\textit{Missile}(b) \Rightarrow \textit{Weapon}(b)$$

An enemy of America counts as "hostile“:

$$\textit{Enemy}(c,\text{America}) \Rightarrow \textit{Hostile}(c)$$

West, who is American …

$$\textit{American}(\text{West})$$

The country Nono, an enemy of America …

$$\textit{Enemy}(\text{Nono},\text{America})$$

Example Knowledge Base

- American(x) $\wedge$ Weapon(y) $\wedge$ Sells(x,y,z) $\wedge$ Hostile(z) $\Rightarrow$ Criminal(x)
- Missile(a) $\wedge$ Owns(Nono,a) $\Rightarrow$ Sells(West,a,Nono)
- Missile(b) $\Rightarrow$ Weapon(b)
- Enemy(c,America) $\Rightarrow$ Hostile(c)

General Knowledge

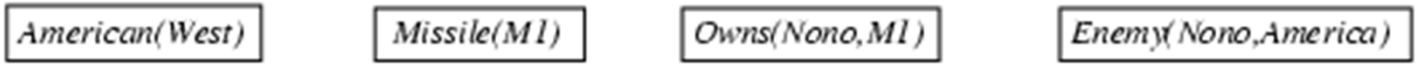
- Owns(Nono,M1) and Missile(M1)
- American(West)
- Enemy(Nono,America)

Specific Problem

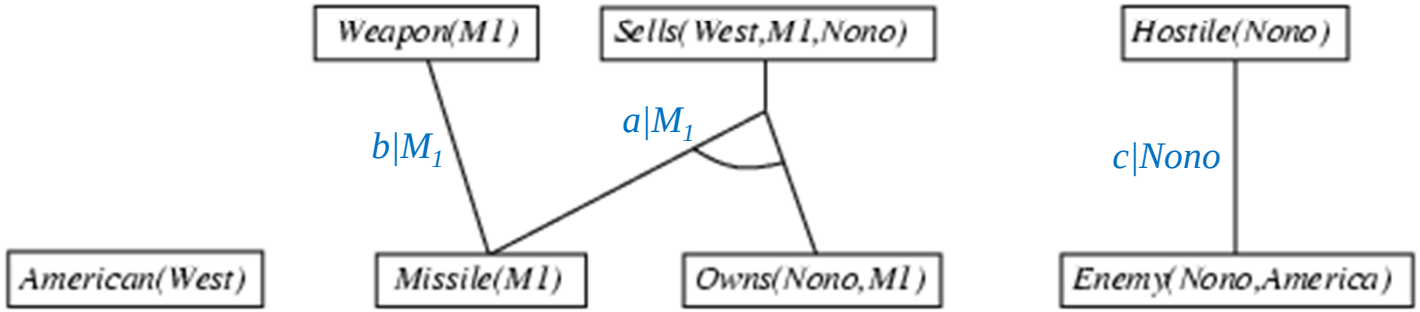
- Criminal(West)?

Query

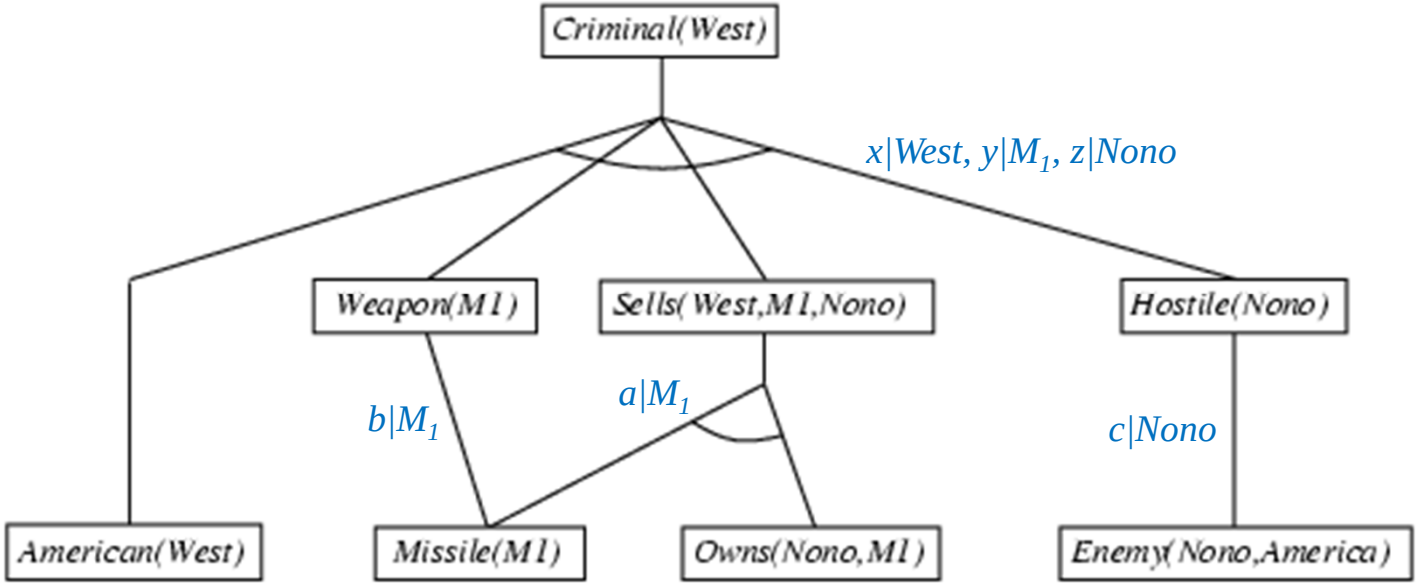
Forward Chaining



Forward Chaining



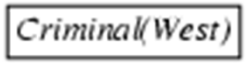
Forward Chaining



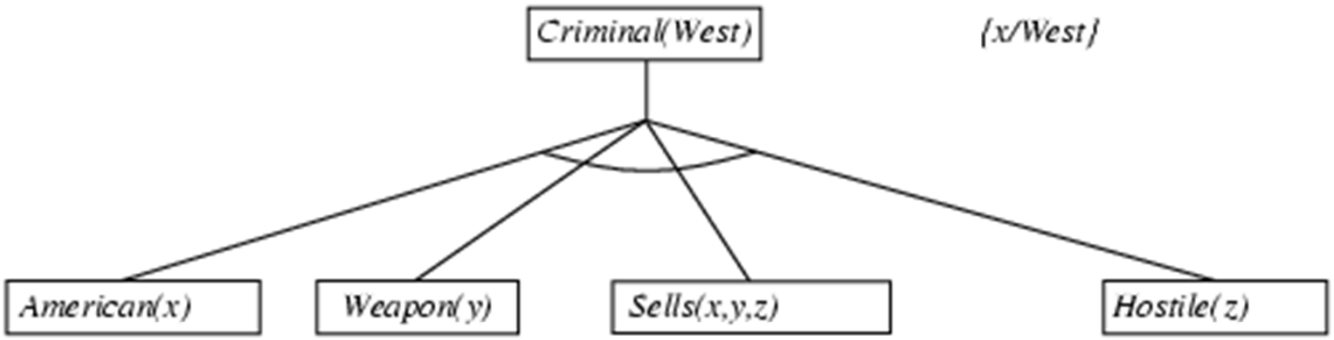
Properties of Forward Chaining

- Sound and complete **only** for first-order definite clauses (Horn clauses)
- **Datalog** (Programming Language for deductive data bases)
  = first-order definite clauses + no functions
  Forward Chaining terminates for Datalog in finite number of iterations
- May not terminate in general if α is not entailed
- **Efficiency**
  - Incremental forward chaining: no need to match a rule on iteration *k* if a premise wasn't added on iteration *k-1* → match each rule whose premise contains a newly added positive literal
  - Matching itself can be expensive:
    Database indexing allows O(1) retrieval of known facts
    e.g., query *Missile(x)* retrieves $Missile(M_1)$
  - Forward chaining is widely used in deductive databases

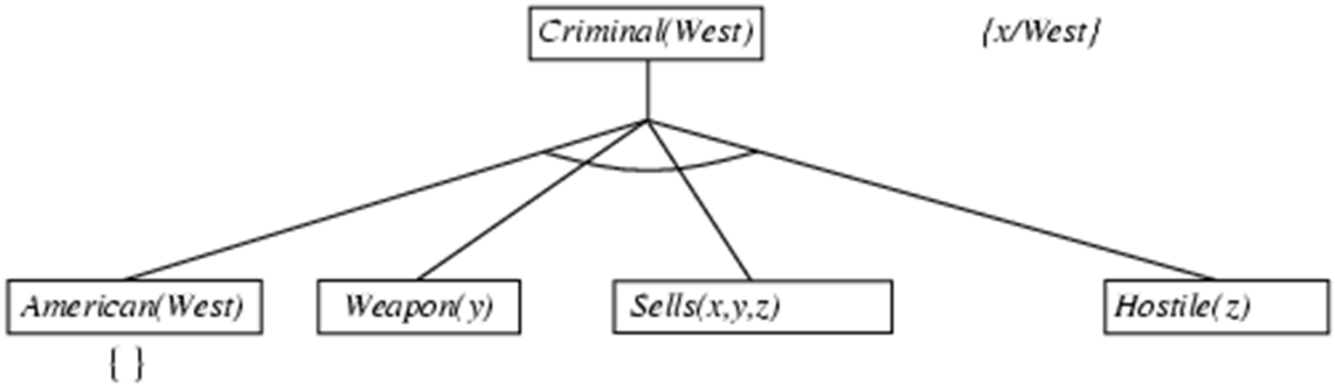
Backward Chaining



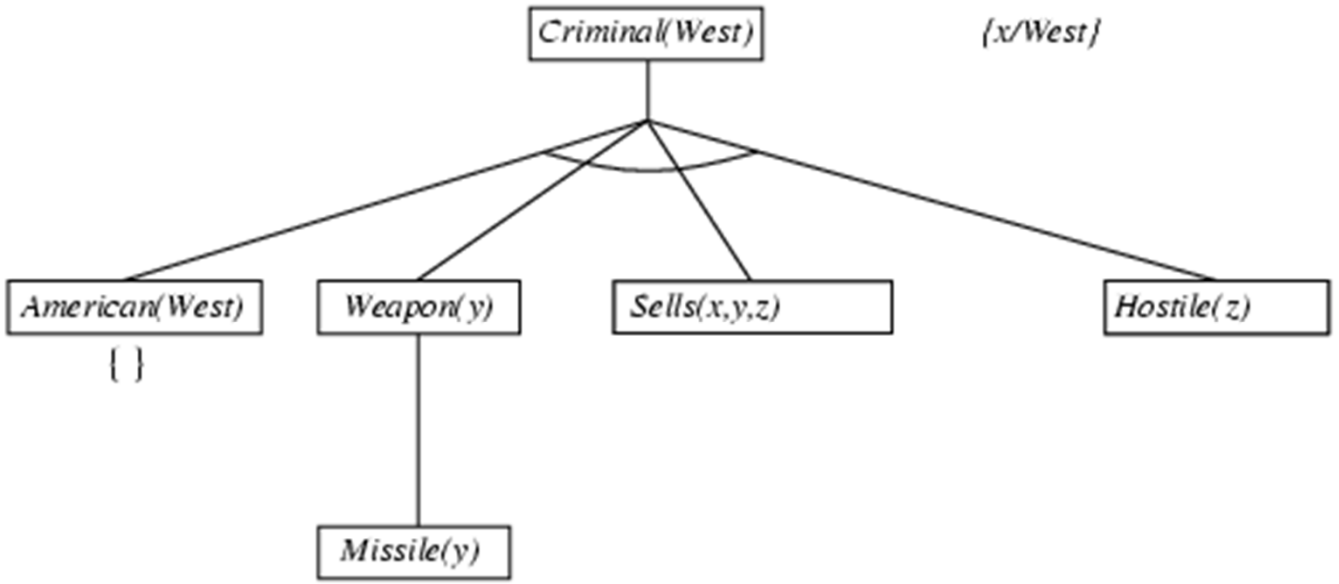
Backward Chaining



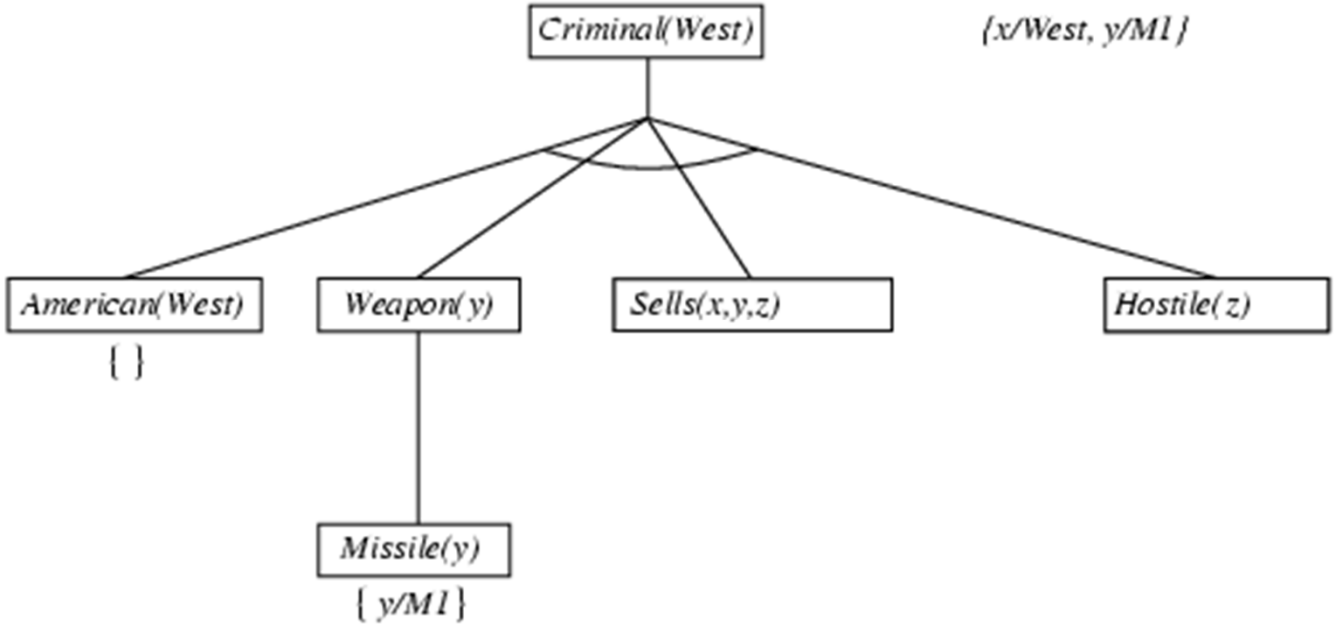
Backward Chaining



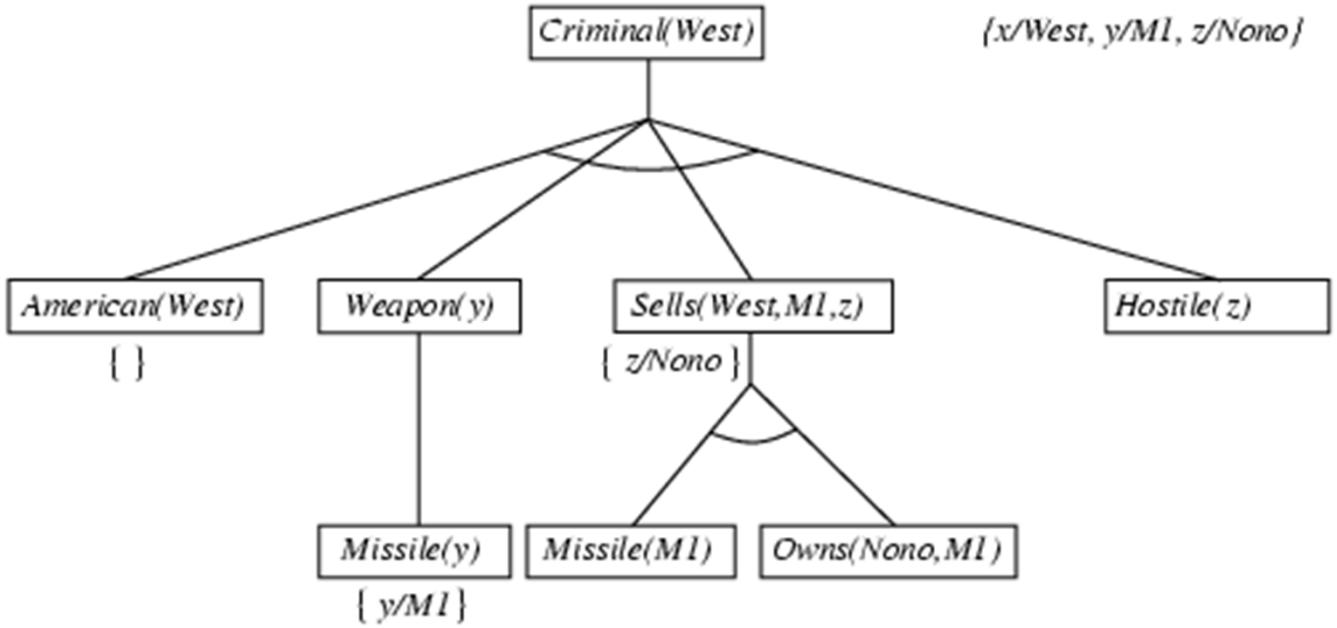
Backward Chaining



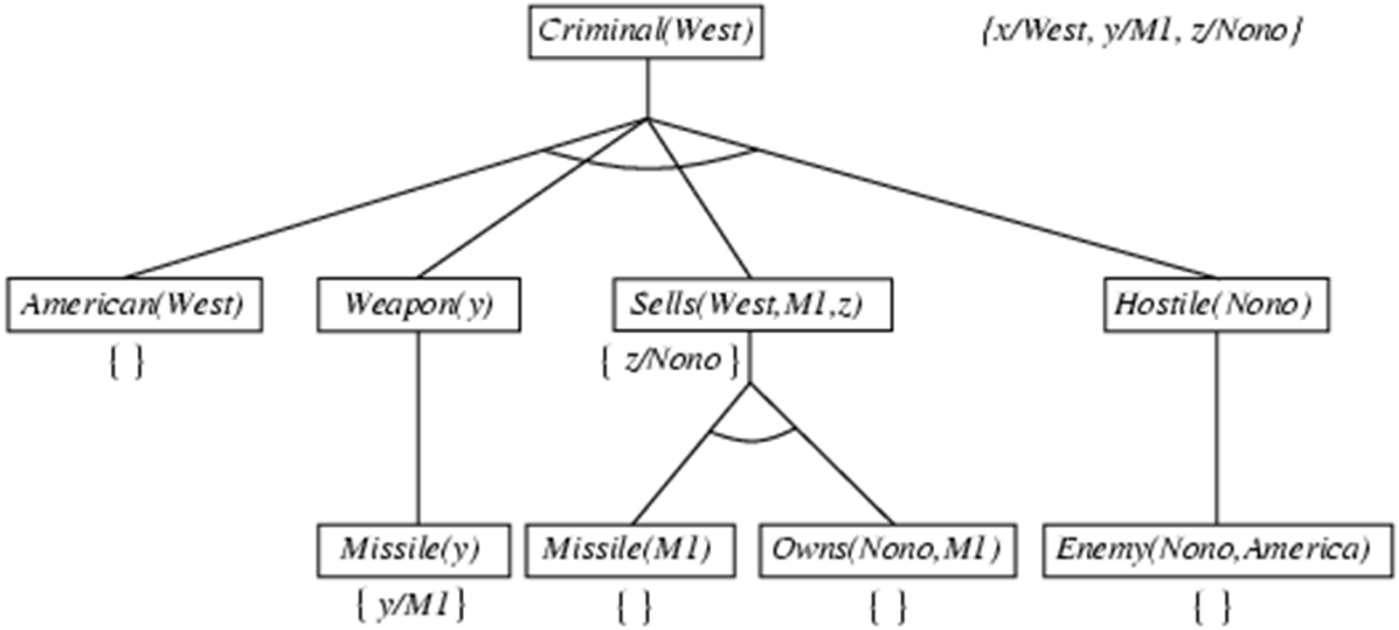
Backward Chaining



Backward Chaining



Backward Chaining



Properties of Backward Chaining

- Depth-first recursive proof search: space is linear in size of proof
- **Incomplete** due to infinite loops

  → fix by checking current goal against every goal on stack
- **Inefficient** due to repeated subgoals (both success and failure)

  → fix using caching of previous results (extra space)

- Widely used for **logic programming** (even without improvements)

Prolog Systems

- Basis: **Backward Chaining with Horn Clauses**, no checks for infinite recursion
- Widely used in Europe, Japan (basis of 5th Generation Project), mainly for symbol-manipulation tasks (writing compilers) or parsing natural language, many expert systems
- Program = set of clauses : `head :- literal`$_1$`, ..., literal`$_n$.

  `criminal(X):-`
  `american(X),weapon(Y),sells(X,Y,Z),hostile(Z).`
- Efficient unification, efficient retrieval of matching clauses, compilation
- Depth-first, left-to-right backward chaining
- Built-in predicates for arithmetic, etc. (e.g. X is Y*Z+3)
- **Closed-world assumption** (”negation as failure”)

  e.g. given `alive(X) :- not dead(X)`
  `alive(joe)` succeeds if `dead(joe)` fails

Prolog Examples

- Depth-first search (in trees!) from a start state X:

```
dfs(X) :- goal(X).
dfs(X) :- successor(X,S),dfs(S).
```

- Database for `successor`, returns true, if there is a path.

- 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=[]
```

Resolution revisited

- Full first-order version:

$$\frac{l_1 \vee \cdots \vee l_k, \qquad m_1 \vee \cdots \vee m_n}{(l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where **UNIFY(l_i, $\neg m_j$) = θ**

- For example

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

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

- Apply resolutions steps to CNF(KB $\wedge \neg \alpha$)
- **Remember: Knowledge base must be in CNF!**

Conversion to CNF

Everyone who loves all animals, is loved by someone:

$\forall x\ [\forall y\ \mathit{Animal}(y) \Rightarrow \mathit{Loves}(x,y)] \Rightarrow [\exists y\ \mathit{Loves}(y,x)]$

1. **Eliminate Biconditionals and Implications**

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

2. **Move $\neg$ inwards** ($\neg\forall x\ p \equiv \exists x\ \neg p$, $\neg\ \exists x\ p \equiv \forall x\ \neg p$)

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

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

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

3. **Standardize Variables:** each Quantifier should use a different one

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

Conversion to CNF

4. **Skolemize**: a more general form of existential instantiation. Each existential variable is replaced by a skolem function of the enclosing universally quantified variables

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

5. **Drop Universal Quantifiers**

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

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

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

Example Knowledge Base

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

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

Nono … has some missiles, i.e., $\exists$x Owns(Nono,x) $\wedge$ Missile(x):

$\exists$ **m** ***Owns(Nono, m) and Missile(m)***

… all of its missiles were sold to it by Colonel West

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

Missiles are weapons:

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

An enemy of America counts as "hostile“:

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

West, who is American …

American(West)

The country Nono, an enemy of America …

Enemy(Nono,America)

Crime Example Sentences in CNF

$\neg$ *American(x)* $\vee$ $\neg$*Weapon(y)* $\vee$ $\neg$*Sells(x,y,z)* $\vee$ $\neg$*Hostile(z)* $\vee$ *Criminal(x)*

$\neg$ *Missile(x)* $\vee$ $\neg$*Owns(Nono,x)* $\vee$ *Sells(West,x,Nono)*

$\neg$ *Enimy(x,America)* $\vee$ *Hostile(x)*

$\neg$ *Missile(x)* $\vee$ *Weapon(x)*

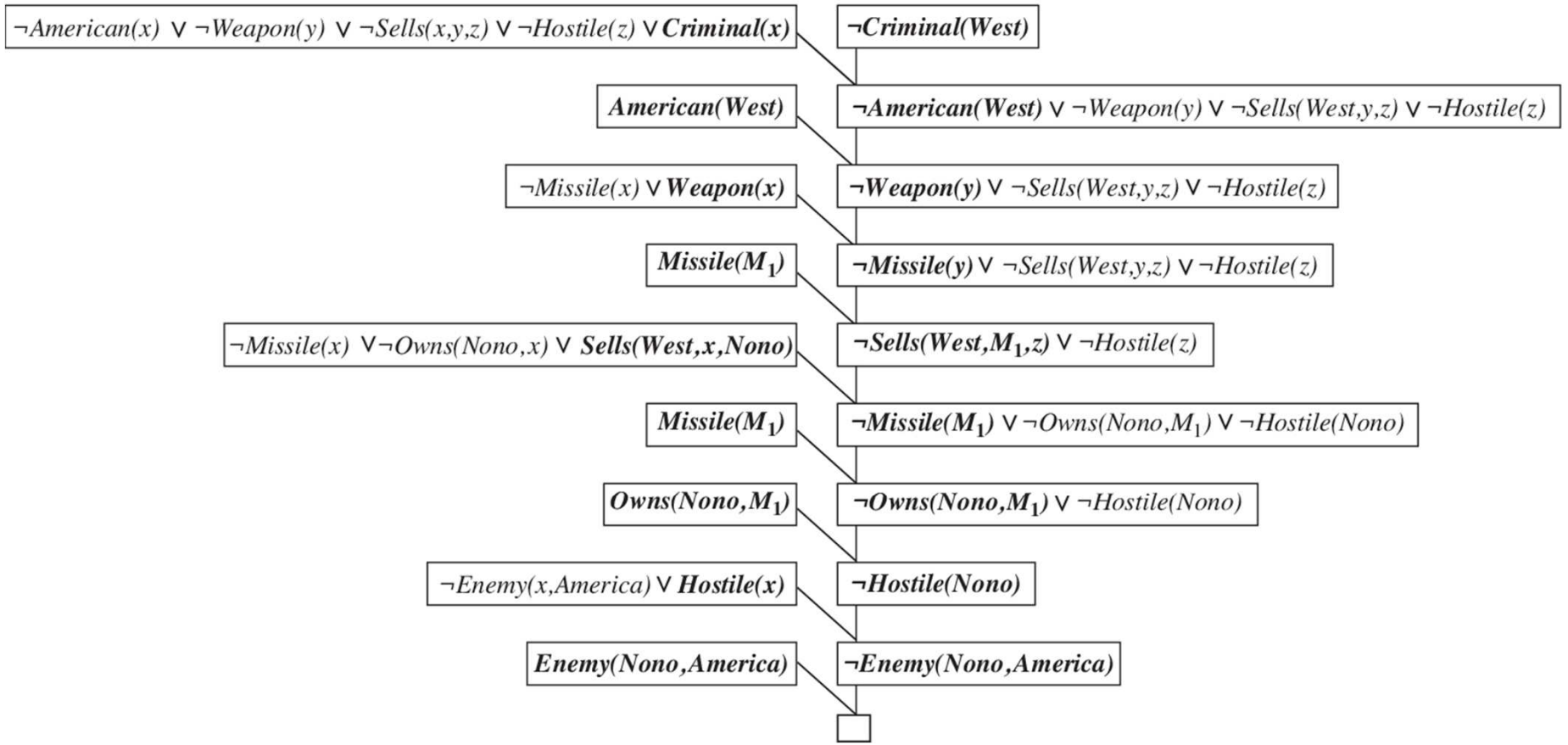
Owns(Nono,M1)

Missile(M1)

American(West)

Enimy(Nono,America)

Resolution Proof



Examples for Resolution Heuristics

- Repeated applications of resolution inference rule will eventually find a proof if one exists; yet unnecessary resolutions just make the KB larger, yet we want to reduce it to a contradiction.
- Streamline Knowledge Base with:
  - Subsumption Elimination: **Subsumption**: eliminates all sentences that are are more specific than existing sentences in the KB: if P(x) is in KB, then no sense adding P(A) or P(A)$\vee$P(B)
  - Tautology Elimination: Eliminate clauses that are always true, contain e.g. P(A)$\vee$P($\neg$A)
  - Pure Literale Elimination: Delete all causes that contain a literal without instance in the in the facts (P(A)$\vee$P(S)) without any S or $\neg$S in the facts. We want to derive the empty resolvent, cannot be done with pure literals.

Examples for Resolution Heuristics

- Strategies for efficient proofs guide which clauses to select for resolution.
  - **Unit Preference**: Prefer resolutions where one of the sentences is a single literal → inferences produce shorter clauses
  - **Set of Support**: Every resolution involve at least one element of the ”set of support”, e.g. clauses derived from negated goal (goal-directed proofs)
  - **Input resolution**: any resolution combines one of the initial entries of the KB with some other sentence

Knowledge-Based System/Problem Solving

1. **Identify the task**: What kind of questions will the KB support / what facts will be available
2. **Assemble the relevant knowledge**: “Knowledge Acquisition”, scope of the KB, not yet formally represented
3. **Decide on a vocabulary of predicates, functions, and constants** → Ontology of the domain
4. **Encode general knowledge about the domain** (go back to 3, if necessary)
5. **Encode a description of the specific problem instance**
6. **Pose queries to the inference procedure and get answers**
7. **Debug the knowledge base**