Chapter 9 Inference in First-Order Logic

CS361 Artificial Intelligence

Dr. Khaled Wassif

Spring 2017

(This is the instructor's notes and student has to read the textbook for complete material.)

Chapter Outline

- Inference Rules for Quantifiers
- Reducing First-Order Inference to Propositional Inference
- Unification Inference Rules
- Forward Chaining and Its Applications
- Backward Chaining and Logic Programming Systems
- Resolution-based Theorem Proving

Necessary Algorithms

- We already know enough to implement TELL (although maybe not efficiently)
- But how do we implement ASK?
- Recall 3 cases
  - Direct matching
  - Finding a proof (**inference**)
  - Finding a set of bindings (**unification**)

Inference with Quantifiers

- Universal Instantiation:
  - Given $\forall x$ Person(x) $\Rightarrow$ Likes(x, McDonalds)
  - Infer Person(John) $\Rightarrow$ Likes(John, McDonalds)
- Existential Instantiation:
  - Given $\exists x$, Likes(x, McDonalds)
  - Infer Likes(S_1, McDonalds)
  - S1 is a "Skolem Constant" that is not found anywhere else in the KB and refers to (one of) the individuals that likes McDonalds.

Universal Instantiation (UI)

- Every instantiation of a universally quantified sentence is inferred by:

$$\frac{\forall v \;\; \alpha}{\textsc{Subst}(\{v/g\}, \alpha)}$$

  - Infer any sentence α by substituting any variable v by a **ground term** g.
    - » Each ground term is a term with out variables
- Example:
  - $\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x) yields:
    - » King(John) $\wedge$ Greedy(John) $\Rightarrow$ Evil(John)
    - » King(Richard) $\wedge$ Greedy(Richard) $\Rightarrow$ Evil(Richard)
    - » King(Father(John)) $\wedge$ Greedy(Father(John) $\Rightarrow$ Evil(Father(John))
    - » …

Existential Instantiation (EI)

- Every instantiation of an existentially quantified sentence is inferred by:

$$\frac{\exists v \;\; \alpha}{\textsc{Subst}(\{v/k\}, \alpha)}$$

  - Infer any sentence α by substituting any variable *v* by a constant *k* that does not appear elsewhere in the KB.

- Example:
  - $\exists x$ Crown(x) $\wedge$ OnHead(x, John) yields:
    - » Crown(C_1) $\wedge$ OnHead(C_1, John)
    - » provided C_1 is a new Skolem constant

Inference with Quantifiers

- UI can be applied several times to ***add*** many new consequence sentences
  - The new KB is logically equivalent to the old
- EI can be applied once to ***replace*** the existential sentence
  - The new KB is not logically equivalent to the old
  - But the new KB is satisfyiable iff the old KB was satisfiable (**inferentially equivalent**)

Reduction to Propositional Inference

- Once inferring non-quantified sentences from quantified sentences, it becomes possible to reduce first-order inference to propositional inference.
  - Use instantiation rules to create a *relevant* KB contains propositional sentences.
  - Then use the propositional reasoning with the created KB.
- Problems:
  - When the knowledge base includes a function symbol, the set of possible ground-term substitutions is infinite!
  - May generate many irrelevant unproven propositional sentences along the way!

Reduction to Propositional Inference

- Suppose the KB had the following sentence:

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

  King(John)

  Greedy(John)

  Brother(Richard, John)

- Instantiating the universal sentence in all possible ways…

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

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

  King(John)

  Greedy(John)

  Brother(Richard, John)

- The new KB is propositionalized:
  - Propositional symbols are King(John), Greedy(John), Evil(John), King(Richard), etc…

Problems with Propositionalization

- Propositionalization tends to generate lots of irrelevant sentences.
- Example:

  $\forall x\ \mathrm{King}(x) \wedge \mathrm{Greedy}(x) \Rightarrow \mathrm{Evil}(x)$

  King(John)

  $\forall y\ \mathrm{Greedy}(y)$

  Brother(Richard, John)

  – Obvious that Evil(John) is true, but the fact Greedy(Richard) is irrelevant.
- With p k-ary predicates and n constants, there are $p \times n^k$ instantiations!!!

Unification

- **Unification** is the process of finding all legal substitutions that make different logical expressions look identical.
  - A key component of all first-order inference algorithms.
  - The UNIFY algorithm takes two sentences and returns a **unifier** for them if one exists:

    UNIFY(p, q) = θ where SUBST(θ, p) = SUBST(θ, q)
  - Example: answer the query, Knows(John, x)?

p	***q***	***θ***
Knows(John, x)	Knows(John, Jane)	{x/Jane}
Knows(John, x)	Knows(y, Bill)	{x/Bill, y/John}
Knows(John, x)	Knows(y, Mother(y))	{y/John, x/Mother(John)}
Knows(John, x)	Knows(x, Elizabeth)	fail {x/John, x/Elizabeth}

Unification

- Reason of failing the last sentence:

  UNIFY(Knows(John, *x*), Knows(*x*, Elizabeth)) = fail

  – Because *x* cannot take on two values at the same time

  – But "Everyone knows Elizabeth" and it should not fail!!

  – Must standardize apart one of the two sentences to eliminate reuse of the same variable.

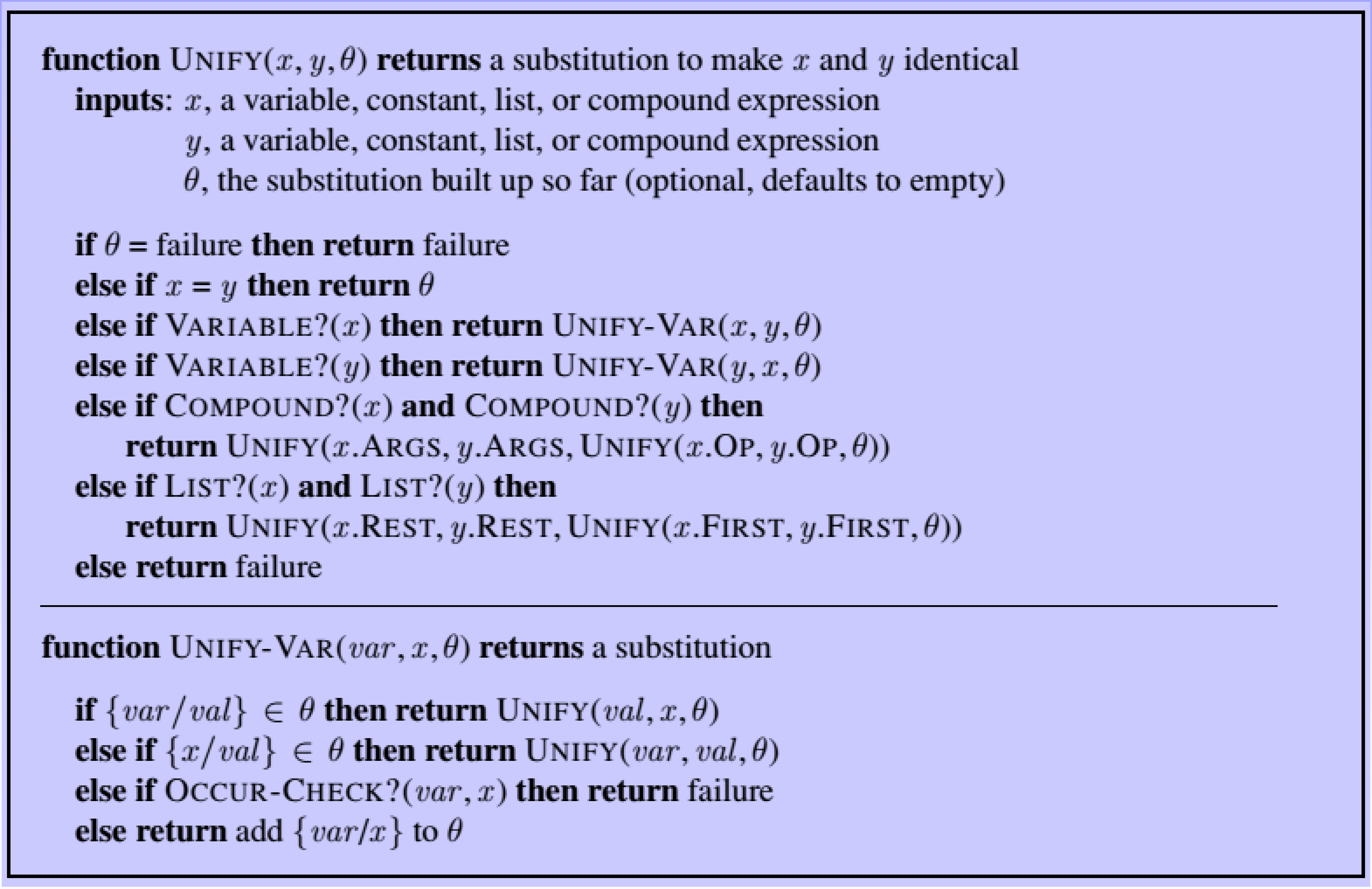
- Now, we can get the inference immediately if finding a substitution θ:

  – Such that King(*x*) and Greedy(*x*)

  – Match with King(John) and Greedy(y)

  – By the substitution $\theta = \{x/\text{John}, y/\text{John}\}$

Unification



```
function UNIFY($x, y, \theta$) returns a substitution to make $x$ and $y$ identical
  inputs: $x$, a variable, constant, list, or compound expression
          $y$, a variable, constant, list, or compound expression
          $\theta$, the substitution built up so far (optional, defaults to empty)

  if $\theta$ = failure then return failure
  else if $x = y$ then return $\theta$
  else if VARIABLE?($x$) then return UNIFY-VAR($x, y, \theta$)
  else if VARIABLE?($y$) then return UNIFY-VAR($y, x, \theta$)
  else if COMPOUND?($x$) and COMPOUND?($y$) then
      return UNIFY($x$.ARGS, $y$.ARGS, UNIFY($x$.OP, $y$.OP, $\theta$))
  else if LIST?($x$) and LIST?($y$) then
      return UNIFY($x$.REST, $y$.REST, UNIFY($x$.FIRST, $y$.FIRST, $\theta$))
  else return failure
```

```
function UNIFY-VAR($var, x, \theta$) returns a substitution

  if $\{var/val\} \in \theta$ then return UNIFY($val, x, \theta$)
  else if $\{x/val\} \in \theta$ then return UNIFY($var, val, \theta$)
  else if OCCUR-CHECK?($var, x$) then return failure
  else return add $\{var/x\}$ to $\theta$
```

Generalized Modus Ponens

- A general version of modus ponens inference rule for first-order logic that does not require instantiation:
  - For atomic sentences p_i, p_i', and q, where there is a substitution θ such that $\text{Subst}(\theta, p_i') = \text{Subst}(\theta, p_i)$, for all i:

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

    - » This rule contains $n+1$ premises: n of p_i' and one implication.
    - » The conclusion is the result of applying the substitution θ to the consequent q.
  - Example:
    - » King(John), $\forall y$ Greedy(y), $\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
    - » Conclude: Evil(John) by the substitution {x/John, y/John}

Generalized Modus Ponens

- In CS terms:
  - Given a rule containing variables.
  - If there is a consistent set of bindings for all of the variables of the left side of the rule (before the arrow).
  - Then you can derive the result of substituting all of the same variable bindings into the right side of the rule.
- Example:
  - Given:
    - » $\forall x, y, z$ Parent(x, y) $\wedge$ Parent(y, z) $\Rightarrow$ GrandParent(x, z)
    - » Parent(James, John), Parent(James, Richard), Parent(Harry, James)
  - We can derive:
    - » GrandParent(Harry, John) by bindings: {x/Harry, y/James, z/John}
    - » GrandParent(Harry, Richard) bindings: {x/Harry, y/James, z/Richard}

First-order Definite Clauses

- Closely resemble propositional definite clauses.
- Either an atomic or is an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal.
  - The following are first-order definite clauses:

    $King(x) \land Greedy(x) \Rightarrow Evil(x)$ $King(John)$ $Greedy(y)$
- Unlike propositional literals, first-order literals can include variables, in which case those variables are assumed to be universally quantified.
  - We omit universal quantifiers when writing definite clauses.
- Not every knowledge base can be converted into a set of definite clauses because of the single-positive-literal restriction, but many can.

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:**

Colonel West is a criminal

Example Knowledge Base

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

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

… ***The country Nono, an enemy of America:*** Enemy(Nono, America)

… ***An enemy of America counts as "hostile":***

Enemy(x, America) ⇒ Hostile(x)

… ***Nono … has some missiles:*** ∃x Missile(x) ∧ Owns(Nono, x) Missile(M_1) ∧ Owns(Nono, M_1)

… ***Missiles are weapons:*** Missile(x) ⇒ Weapon(x)

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

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

… ***West, who is American:*** American(West)

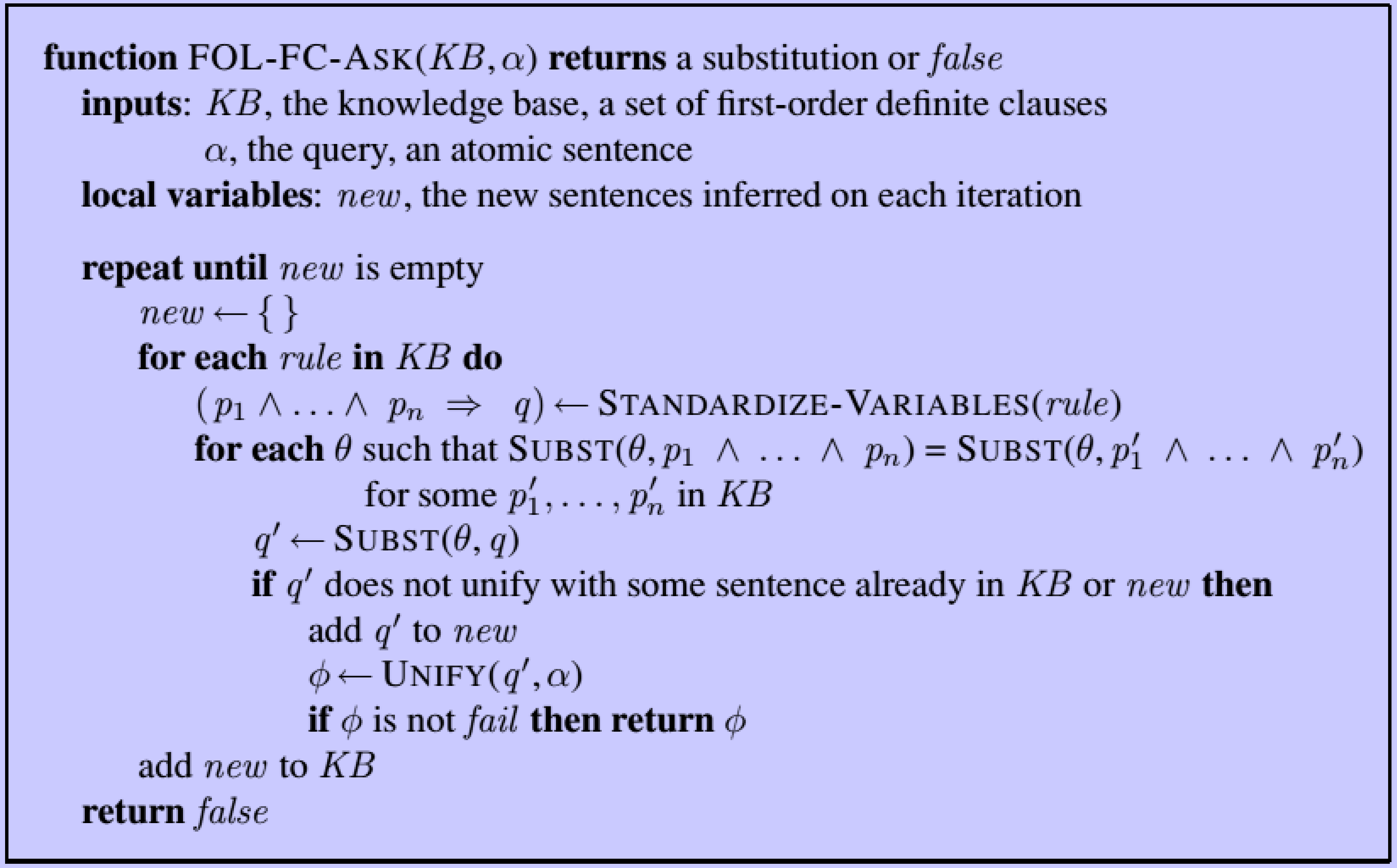
Answering Questions in FOL

- More or less anything can be stated in first-order logic.
- It is important to have algorithms that can answer any answerable question stated in first-order logic.
- Three major families of first-order inference algorithms:
  - **Forward chaining** and its applications to **deductive databases** and **production systems**
  - **Backward chaining** and **logic programming** systems
  - **Resolution**-based **theorem-proving** systems

Forward Chaining

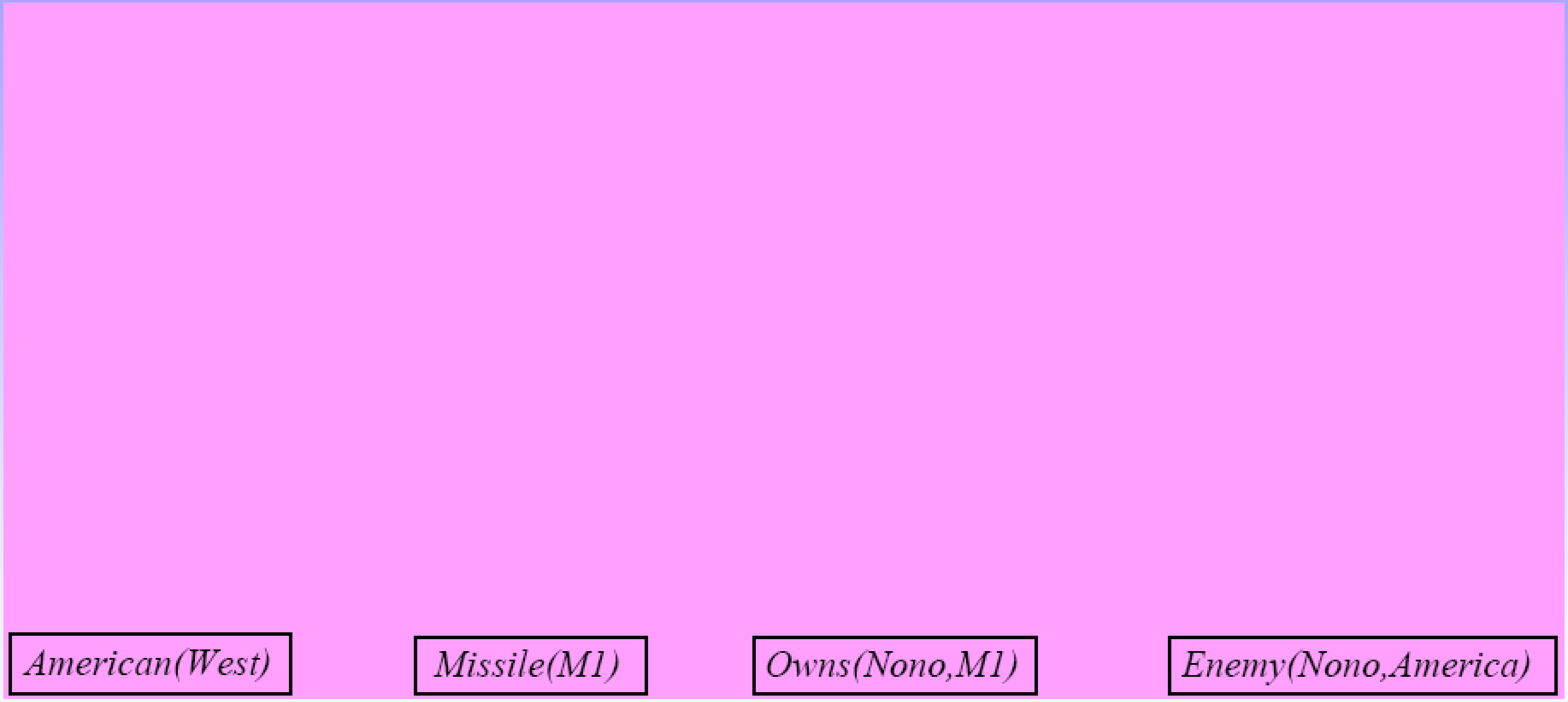
- Data driven
- Main idea:
  - – Start with atomic sentences (facts) in the knowledge base.
  - – Apply Modus Ponens in the forward direction, by triggering all rules whose premises are satisfied.
  - – Adding conclusions of the satisfied rules to the known facts.
  - – Repeat the process until the query is answered (assuming that just one answer is required) or no new facts are added.
- Applied efficiently to first-order definite clauses.
- Especially useful for systems that make inference in response to newly arrived information.
- With reasoning can be more efficient than resolution theorem proving.

Forward Chaining

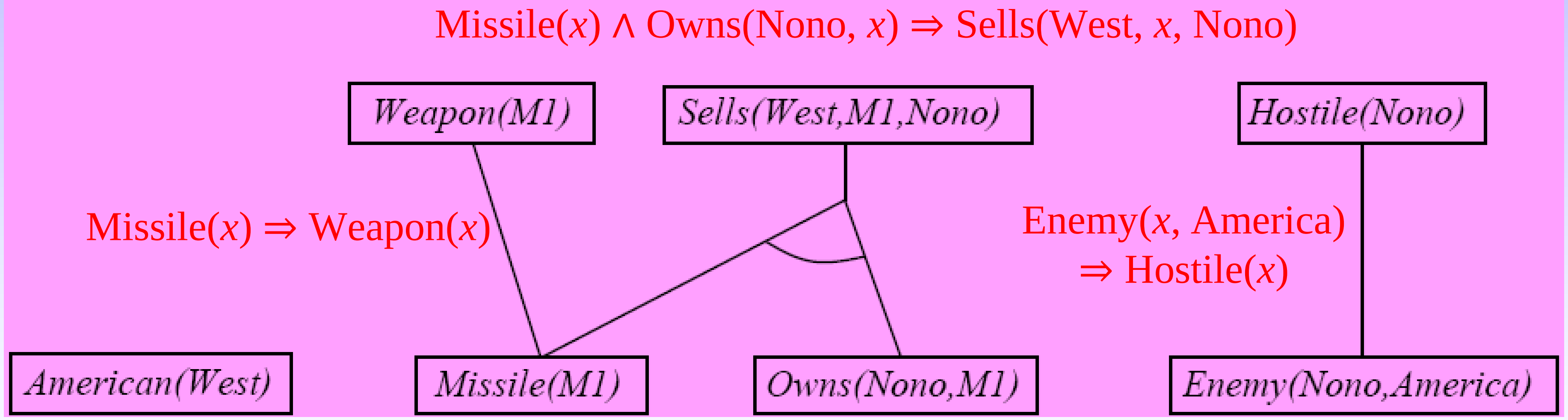


```
function FOL-FC-ASK($KB, \alpha$) returns a substitution or $false$
  inputs: $KB$, the knowledge base, a set of first-order definite clauses
          $\alpha$, the query, an atomic sentence
  local variables: $new$, the new sentences inferred on each iteration

  repeat until $new$ is empty
      $new \leftarrow \{ \}$
      for each $rule$ in $KB$ do
          $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES($rule$)
          for each $\theta$ such that SUBST($\theta, p_1 \wedge \ldots \wedge p_n$) = SUBST($\theta, p'_1 \wedge \ldots \wedge p'_n$)
                      for some $p'_1, \ldots, p'_n$ in $KB$
              $q' \leftarrow$ SUBST($\theta, q$)
              if $q'$ does not unify with some sentence already in $KB$ or $new$ then
                  add $q'$ to $new$
                  $\phi \leftarrow$ UNIFY($q', \alpha$)
                  if $\phi$ is not $fail$ then return $\phi$
      add $new$ to $KB$
  return $false$
```

FC: Example Knowledge Base

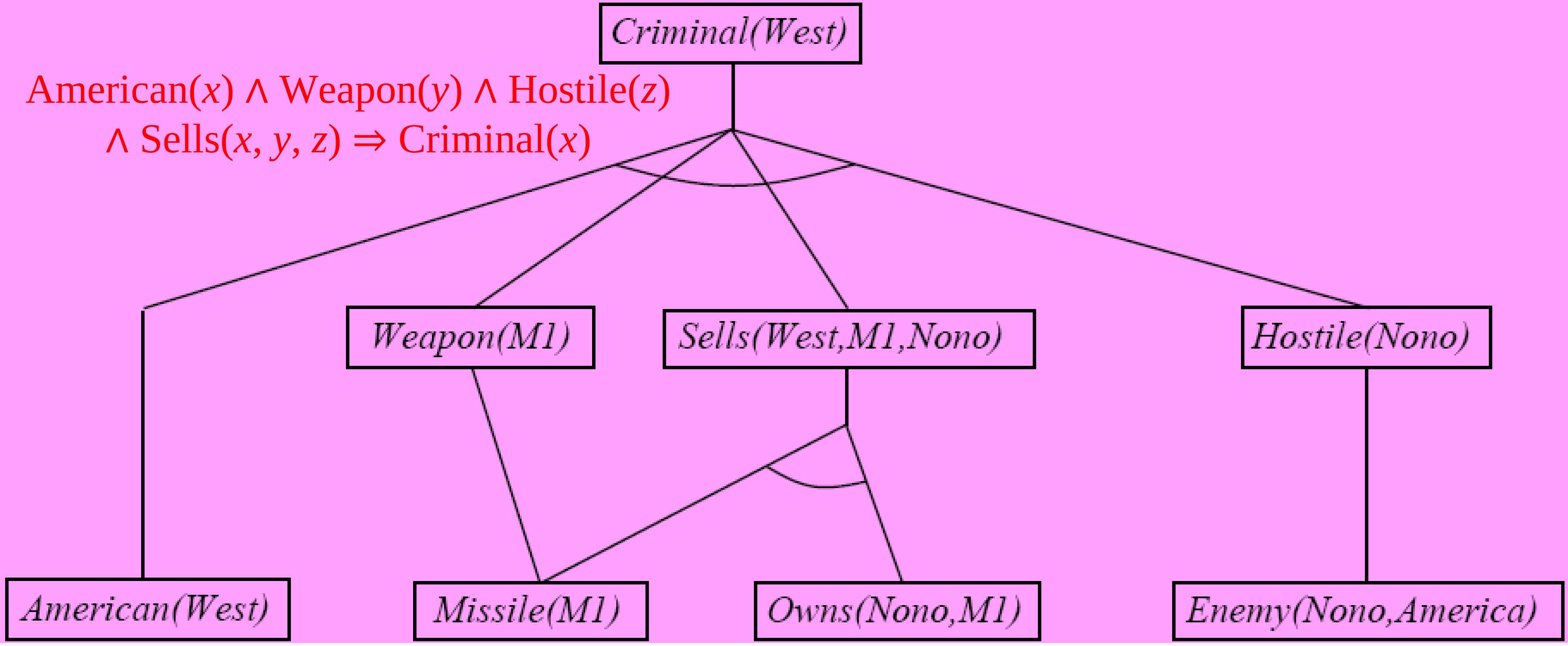


FC: Example Knowledge Base



FC: Example Knowledge Base

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



Forward Chaining

- Properties:
  - **Sound** because every inference is just done by applying Generalized Modus Ponens.
  - **Complete** for first-order definite clauses.
  - **Terminate** for Datalog KB in finite number of iterations.
    - » Datalog = first-order definite clauses + without functions
  - May not terminate in general if desired fact α is not entailed.
    - » This is unavoidable: entailment with definite clauses is semidecidable.
- There are three possible sources of inefficiency:
  - **Pattern matching** can be very expensive.
    - » Finding all possible matching between rules premises and a suitable set of facts in the knowledge base.
  - Recheck every rule on every iteration to see whether its premises are satisfied, even if very few additions are made on each iteration.
  - May generate many facts that are irrelevant to the goal.

Efficient Forward Chaining

- **Expensiveness of matching rules against known facts**
  - Indexed knowledge base allows O(1) for retrieval of facts.
    - » e.g., query Missile(x) retrieves Missile(M_1)
  - Appropriate conjuncts ordering
    - » Find an ordering to solve the conjuncts of the rule premise so that the total cost is minimized.
    - » Starting by most constrained (minimum values) variables is a good heuristic.
    - » e.g., Missile(x) $\wedge$ Owns(Nono, x) $\Rightarrow$ Sells(West, x, Nono)
- **Redundant rule matching**
  - New fact should depend on at least one newly generated fact.
  - **Incremental forward chaining**: no need to check a rule on iteration k if its premises wasn't added on iteration k-1.
    - » Match a rule whose its premise contains a newly added +ve literal.

Efficient Forward Chaining

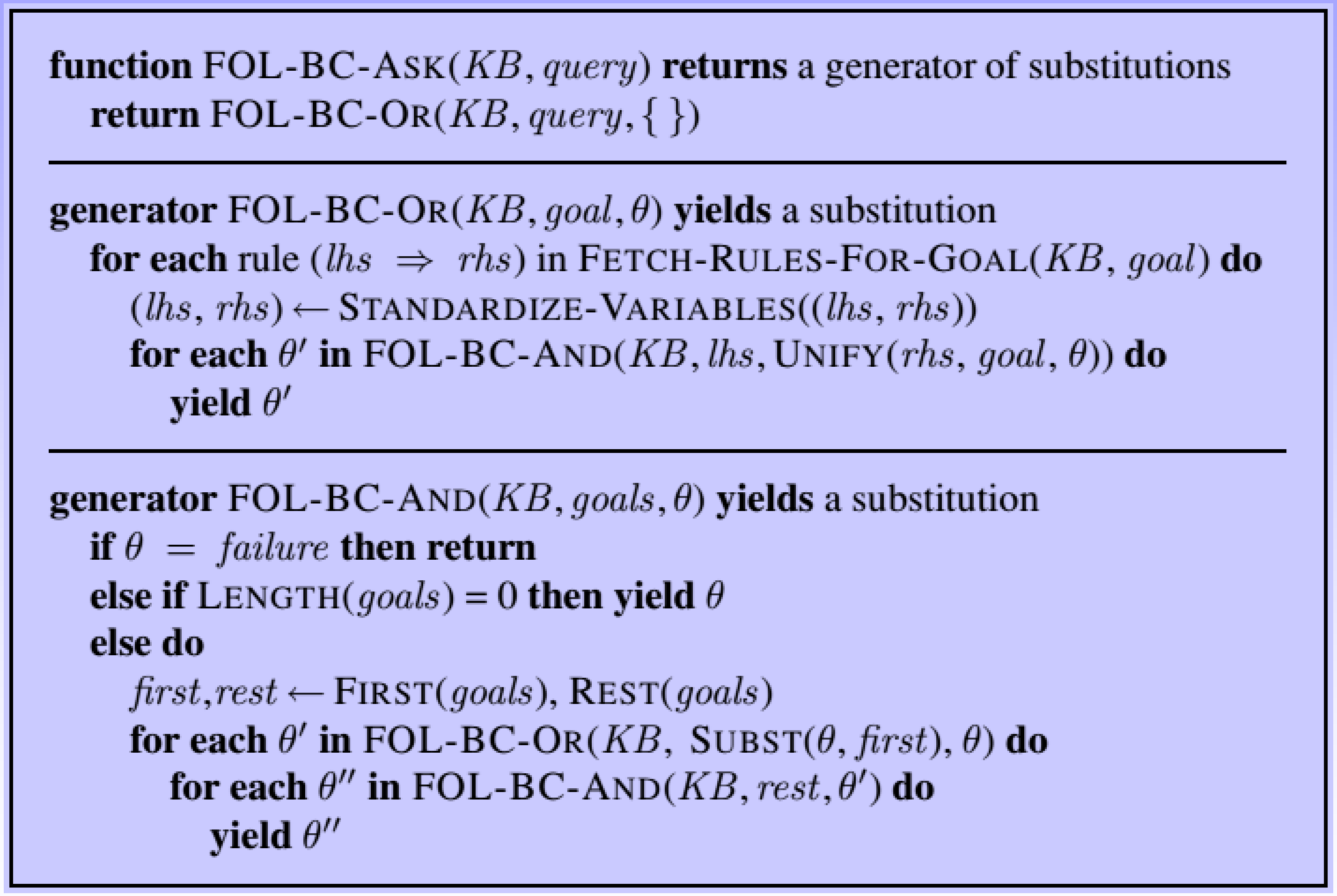
- **Irrelevant facts**
  - One way is to use backward chaining
  - Another solution is to restrict forward chaining to a selected subset of rules.
  - A third approach emerged in the field of **deductive databases**
    - » Large-scale databases, like relational databases, but use forward chaining as the standard inference tool rather than SQL queries.
    - » By rewrite the rule set, using goal information, so that only relevant variable bindings (called ***magic set***) are considered during inference.
    - » e.g., if the goal is Criminal(West):
      - The rule that concludes Criminal(x) will be rewritten to include an extra conjunct that constrains the value of x:

        $Magic(x) \wedge American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
      - The fact Magic(West) is also added to the KB.
    - » Basic idea: perform a sort of "generic" backward inference from the goal in order to work out which variable bindings need to be constrained.

Backward Chaining

- Goal driven
- Main idea:
  - Consider the item to be proven as a goal.
  - Find a rule whose its head is the goal and bindings with it.
  - Apply bindings to the body of that rule and try to prove these body as subgoals in turn.
  - If you prove all the subgoals and increasing the binding set as you go, then you will prove the goal item.
- Properties:
  - Depth-first recursive proof search: space is linear in size of proof
  - Incomplete due to infinite loops
  - Inefficient due to repeated subgoals
- Widely used with logic programming (Prolog).

Backward Chaining



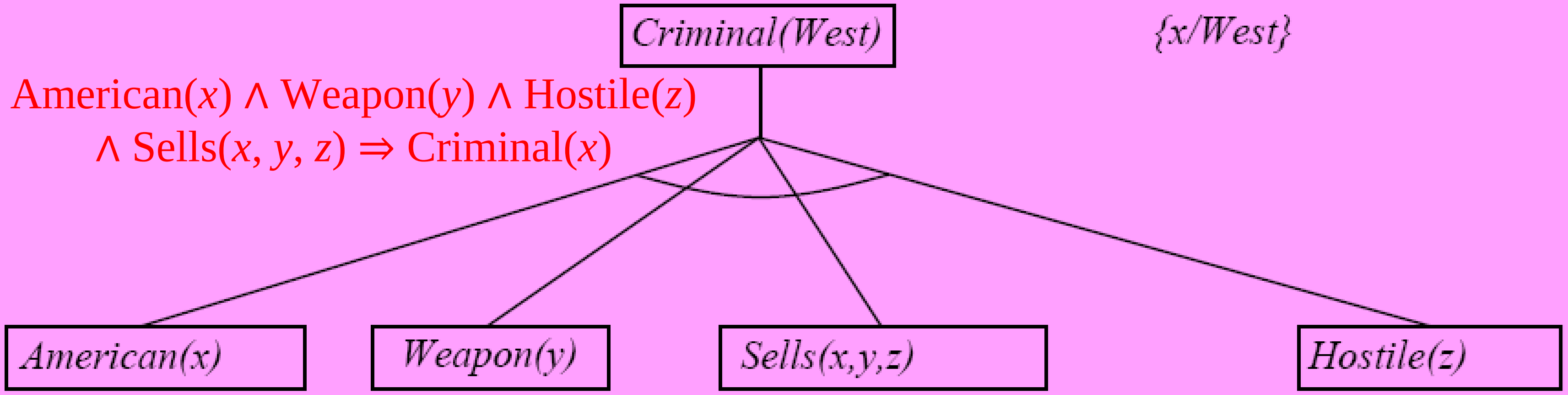
```
function FOL-BC-ASK($KB$, $query$) returns a generator of substitutions
    return FOL-BC-OR($KB$, $query$, { })

generator FOL-BC-OR($KB$, $goal$, $\theta$) yields a substitution
    for each rule ($lhs \Rightarrow rhs$) in FETCH-RULES-FOR-GOAL($KB$, $goal$) do
        ($lhs$, $rhs$) ← STANDARDIZE-VARIABLES(($lhs$, $rhs$))
        for each $\theta'$ in FOL-BC-AND($KB$, $lhs$, UNIFY($rhs$, $goal$, $\theta$)) do
            yield $\theta'$

generator FOL-BC-AND($KB$, $goals$, $\theta$) yields a substitution
    if $\theta = failure$ then return
    else if LENGTH($goals$) = 0 then yield $\theta$
    else do
        $first$,$rest$ ← FIRST($goals$), REST($goals$)
        for each $\theta'$ in FOL-BC-OR($KB$, SUBST($\theta$, $first$), $\theta$) do
            for each $\theta''$ in FOL-BC-AND($KB$, $rest$, $\theta'$) do
                yield $\theta''$
```

BC: Example Knowledge Base

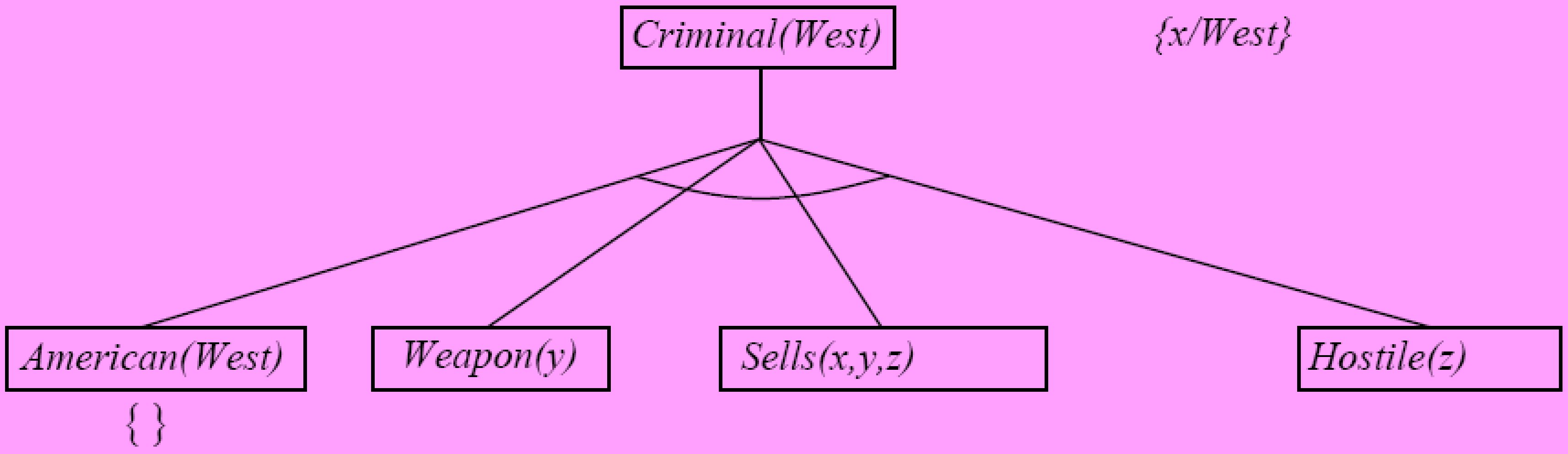
$\boxed{\mathit{Criminal}(\mathit{West})}$

BC: Example Knowledge Base

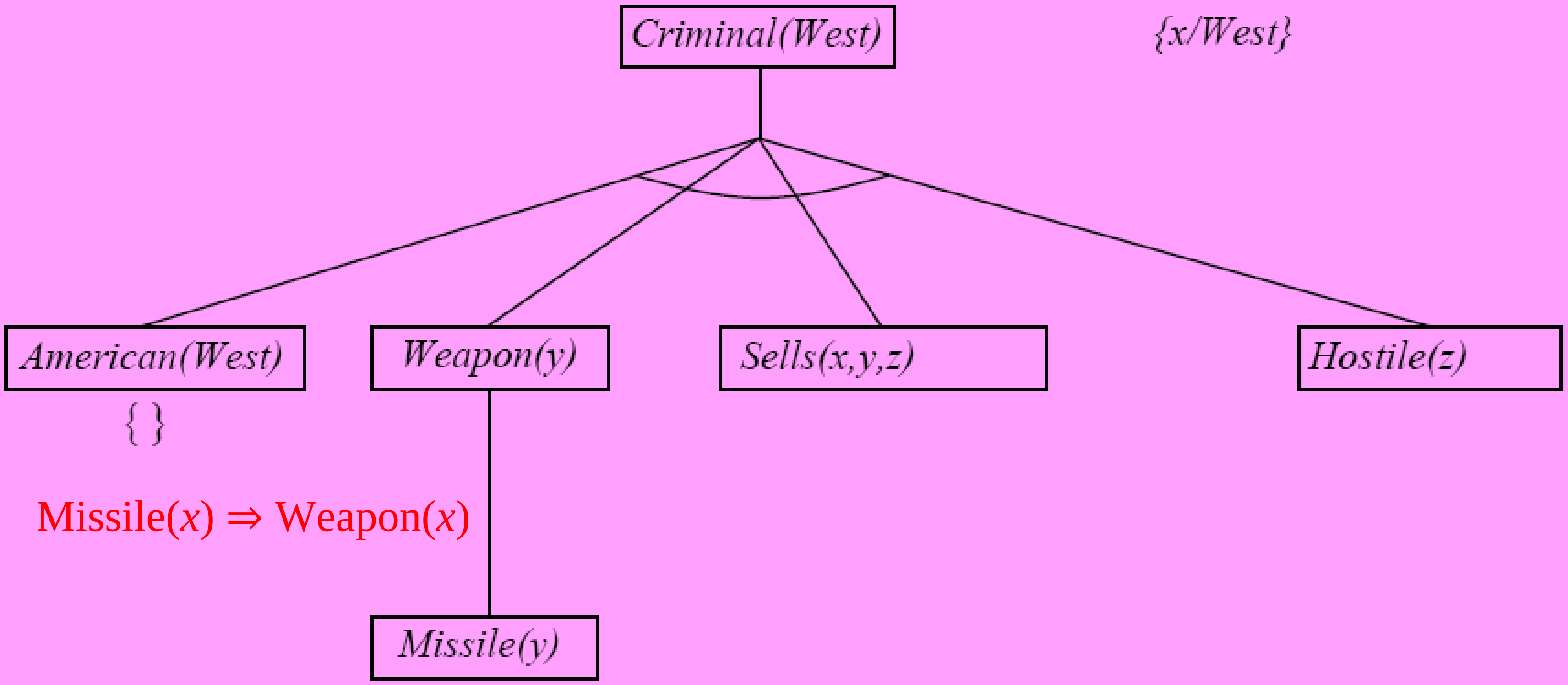
American(x) ∧ Weapon(y) ∧ Hostile(z)
∧ Sells(x, y, z) ⇒ Criminal(x)



BC: Example Knowledge Base

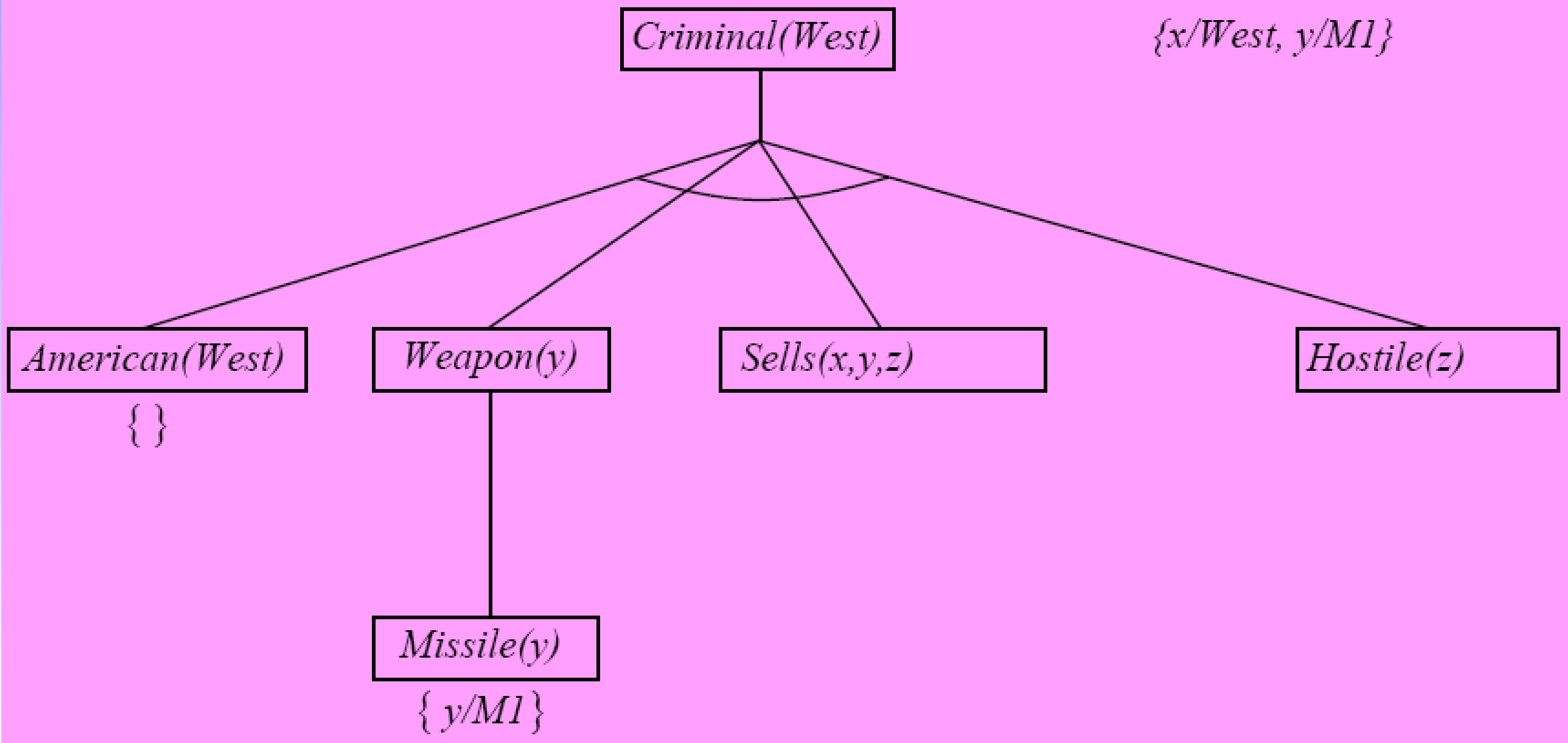


BC: Example Knowledge Base

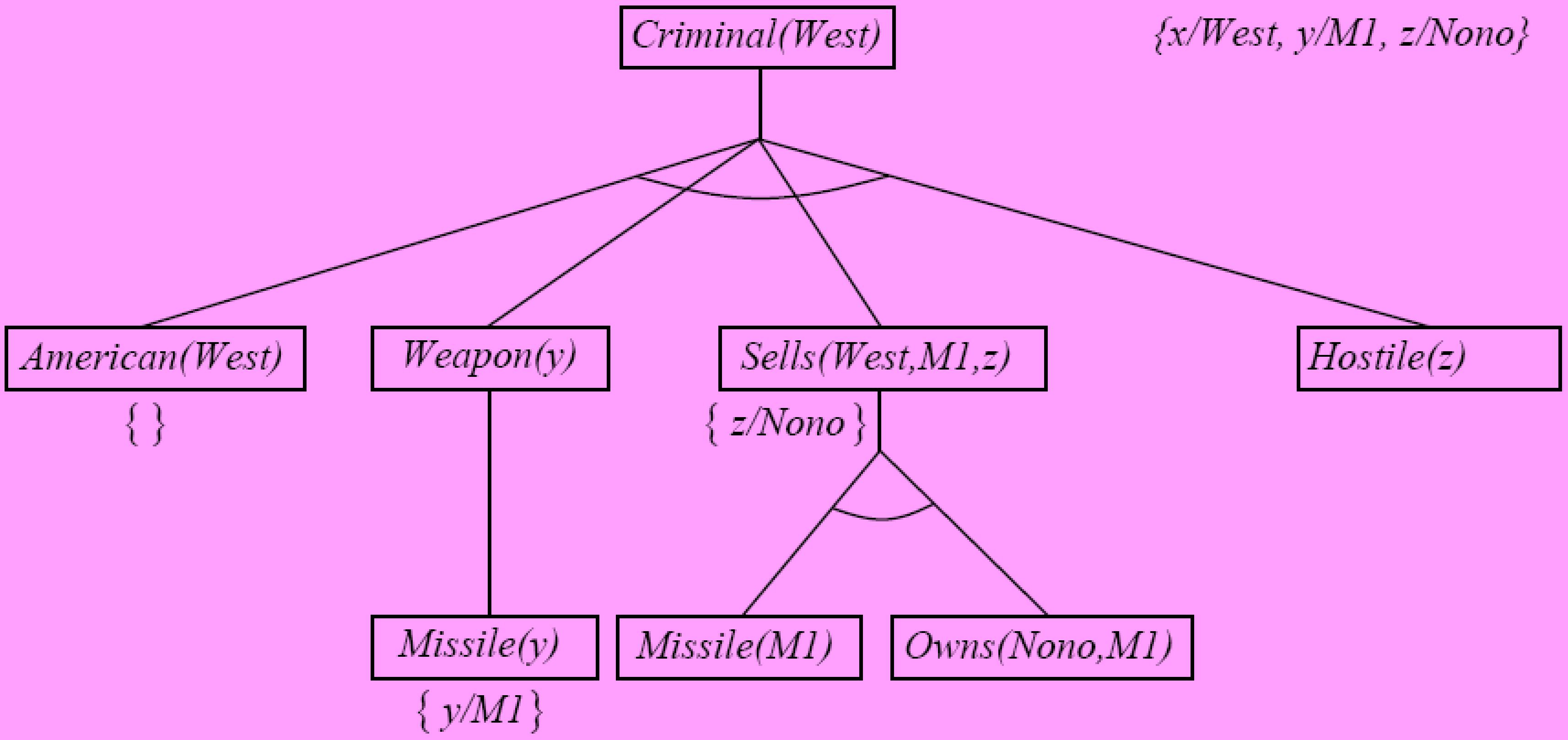


Missile(x) ⇒ Weapon(x)

BC: Example Knowledge Base

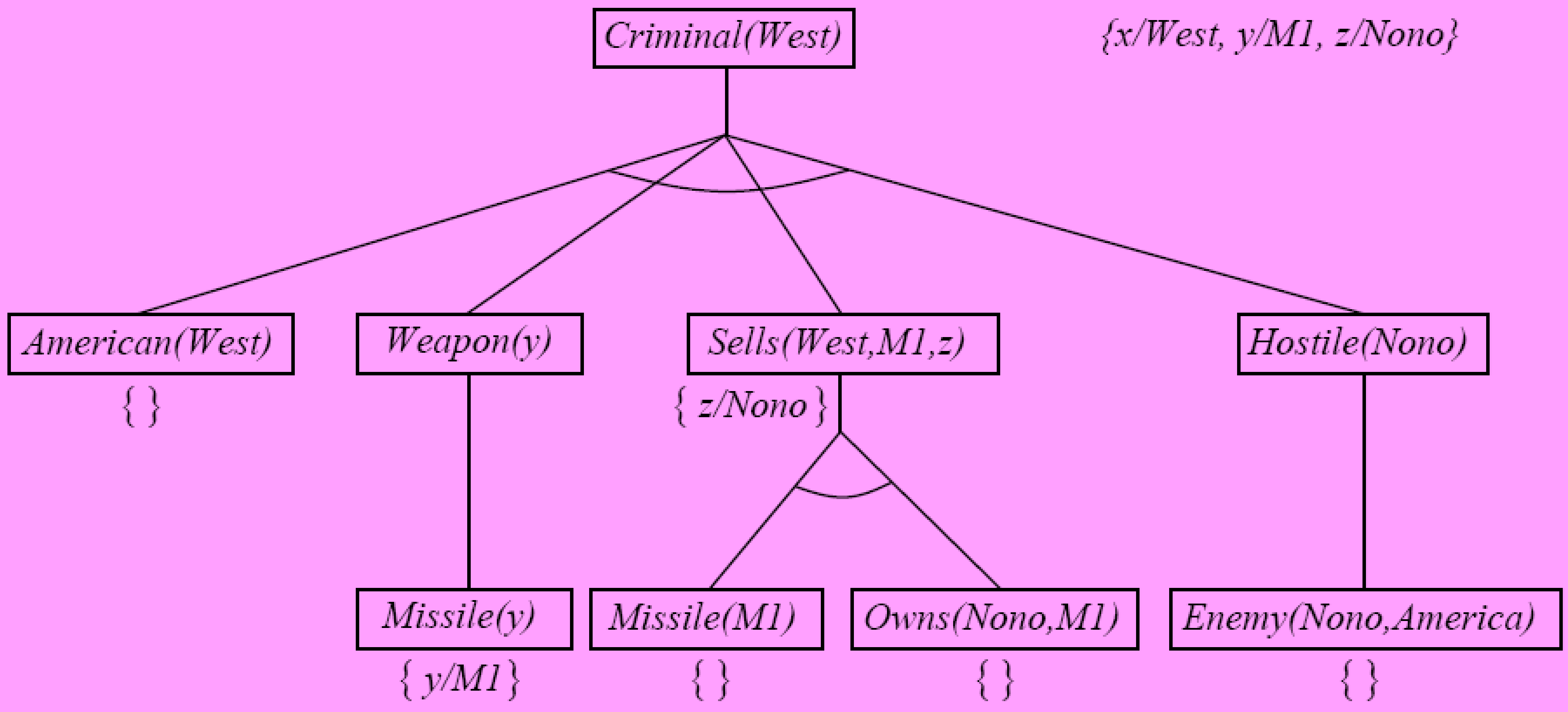


BC: Example Knowledge Base



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

BC: Example Knowledge Base



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

Logic Programming

- Declarative: computation as inference on logical KBs.
- **Logic Programming**
  - Identify problem
  - Assemble information
  - Tea Break
  - Encode information in KB
  - Encode problem instance as facts
  - Ask queries
  - Find false facts
- **Ordinary Programming**
  - Identify problem
  - Assemble information
  - Figure out solution
  - Program Solution
  - Encode problem instance as data
  - Apply program to data
  - Debug procedural errors

Logic Programming: Prolog

- Basis: backward chaining with Horn clauses + lots of bells and whistles
  - – Widely used in Europe and Japan
  - – Basis of 5th Generation Languages and Projects
- Program = set of clauses = head :- literal$_1$, literal$_2$, … , literal$_n$

  criminal(X) :- american(X), weapon(Y), sells(X, Y, Z), hostile(Z).
- Efficient unification by open coding
- Efficient retrieval of matching clauses by direct linking
- 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

Logic Programming: Prolog

- Example: Appending two lists to produce a third one.

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

  - query: append([1, 2], [3, 4, 5], NL)?.
  - answer:
    - » NL = [1, 2, 3, 4, 5]
  - query: append([1, 2], L, [1, 2, 3, 4, 5])?.
  - answer:
    - » L = [3, 4, 5]
  - query: append(A, B, [1,2])?.
  - answers:

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

Inference is Expensive!

- You start with:
  - A large collection of facts (predicates) in the knowledge base.
  - A large collection of possible transformations (rules).
- Some of these rules apply to:
  - A single fact to yield a new fact.
  - A pair of facts to yield a new fact.
- So at every step you must:
  - Choose some rule to apply
  - Choose one or two facts to which you may apply the rule
  - If there are *n* facts in the knowledge base
    - » There are *n* potential ways to apply a single-operand rule
    - » There are *n* * (*n* - 1) potential ways to apply a two-operand rule
  - Add the new fact that ever-expanding the knowledge base.
- The search space is huge!

The Magic of Resolution

- Here's how resolution works:
  - Transform each of your facts into a particular form, called a clause.
  - Apply a ***single rule***, the resolution principle, to a pair of clauses.
    - » Clauses are closed with respect to resolution – that is, when resolve two clauses, get a new clause.
    - » Add the new clause to the knowledge base.
- So the number of facts you have grows linearly:
  - You still have to choose a pair of facts to resolve.
  - You never have to choose a rule, because there's only one.

Propositional Resolution: Review

- Resolution allows a complete inference mechanism (search-based) using only one rule of inference.
- Resolution rule:
  - Given: $P_1 \lor P_2 \lor P_3 \ldots \lor P_n$, and $\neg P_1 \lor Q_1 \ldots \lor Q_m$
  - Conclude: $P_2 \lor P_3 \ldots \lor P_n \lor Q_1 \ldots \lor Q_m$

  Complementary literals P_1 and $\neg P_1$ “cancel out”
- To prove a proposition S by resolution,
  - Start with $\neg S$
  - Resolve with a fact from the knowledge base (that contains S)
  - Repeat until all propositions have been eliminated
  - If this can be done, a contradiction has been derived and the original proposition S must be true.

Propositional Resolution Example

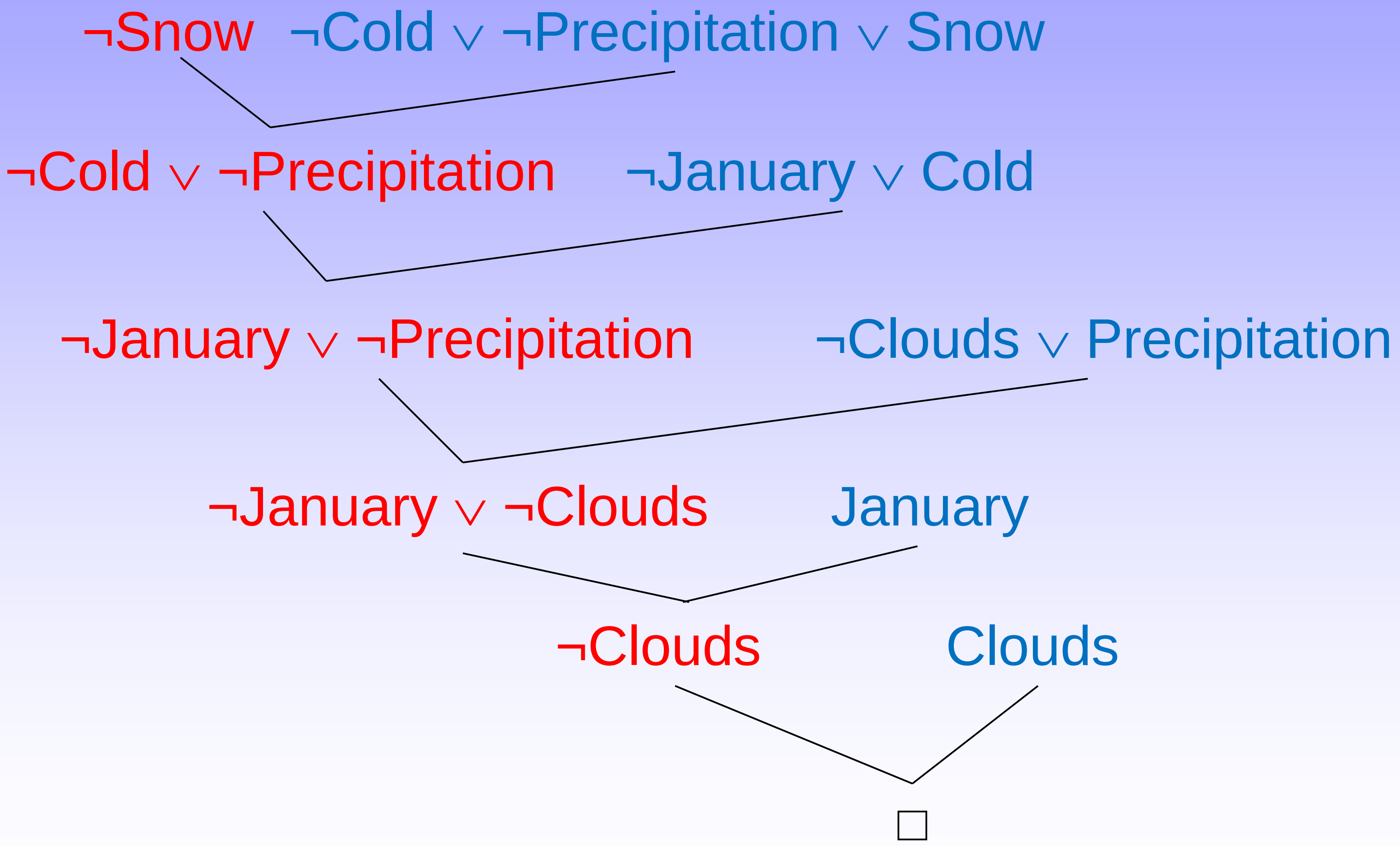
- Rules:
  - Cold ∧ Precipitation ⇒ Snow

    ¬Cold ∨ ¬Precipitation ∨ Snow
  - January ⇒ Cold

    ¬January ∨ Cold
  - Clouds ⇒ Precipitation

    ¬Clouds ∨ Precipitation
- Facts:
  - January, Clouds
- Prove:
  - Snow

Propositional Resolution Example



FOL Resolution Theorem Proving

- Convert everything in the FOL knowledge base to Conjunctive Normal Form (CNF).
  - As in the propositional case, first-order resolution requires that sentences be in CNF.
- Resolve, with unification of variables.
  - Save bindings as you go!
- If resolution is successful, proof succeeds.
- If there was a variable in the item to prove, return variable’s value from the unification bindings.

Converting FOL to CNF

1. Eliminate implication:
   - $\forall x\ P(x) \Rightarrow Q(x)$ is equivalent to $\forall x\ \neg P(x) \vee Q(x)$
2. Move $\neg$ "inwards":
   - $\neg\forall x\ P(x)$ is equivalent to $\exists x\ \neg P(x)$
   - $\neg\exists x\ P(x)$ is equivalent to $\forall x\ \neg P(x)$
3. Standardize variables:
   - $\forall x\ P(x) \vee \forall x\ Q(x)$ becomes $\forall x\ P(x) \vee \forall y\ Q(y)$
4. Skolemize:
   - $\exists x\ P(x)$ becomes P(A) or P(F(x)) using **Skolem functions**
5. Drop universal quantifiers:
   - Since all quantifiers are now $\forall$, we don't need them
6. Apply distribution law ($\vee$ over $\wedge$)

Converting FOL to CNF

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

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

1. Eliminate implication
   - $\forall x\ [\forall y\ \neg Animal(y) \lor Loves(x, y)] \Rightarrow [\exists y\ Loves(y, x)]$
   - $\forall x\ \neg[\forall y\ \neg Animal(y) \lor Loves(x, y)] \lor [\exists y\ Loves(y, x)]$
2. Move $\neg$ "inwards"
   - $\forall x\ [\neg\forall y\ \neg\neg Animal(y) \land \neg Loves(x, y)] \lor [\exists y\ Loves(y, x)]$
   - $\forall x\ [\exists y\ Animal(y) \land \neg Loves(x, y)] \lor [\exists y\ Loves(y, x)]$
3. Standardize variables
   - $\forall x\ [\exists y\ Animal(y) \land \neg Loves(x, y)] \lor [\exists z\ Loves(z, x)]$

Converting FOL to CNF

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

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

4. Skolemize
   - $\forall x\ [\mathrm{Animal}(A) \land \neg\mathrm{Loves}(x, A)] \lor \mathrm{Loves}(B, x)$
   - $\forall x\ [\mathrm{Animal}(F(x)) \land \neg\mathrm{Loves}(x, F(x))] \lor \mathrm{Loves}(G(x), x)$
5. Drop universal quantifiers
   - $[\mathrm{Animal}(F(x)) \land \neg\mathrm{Loves}(x, F(x))] \lor \mathrm{Loves}(G(x), x)$
6. Apply distribution law($\lor$ over $\land$)
   - $[\mathrm{Animal}(F(x)) \lor \mathrm{Loves}(G(x), x)] \land$
     $[\neg\mathrm{Loves}(x, F(x)) \lor \mathrm{Loves}(G(x), x)]$

Resolution with Unification

- The resolution rule for first-order logic is simply a lifted version of the propositional resolution rule:
  - Two clauses, which are assumed to share no variables, can be resolved if they contain complementary literals.
  - First-order literals are complementary if one *unifies with* the negation of the other.
  - We have:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\textsc{Subst}(\theta, \ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)}$$

where $\textsc{Unify}(\ell_i, \neg m_j) = \theta$

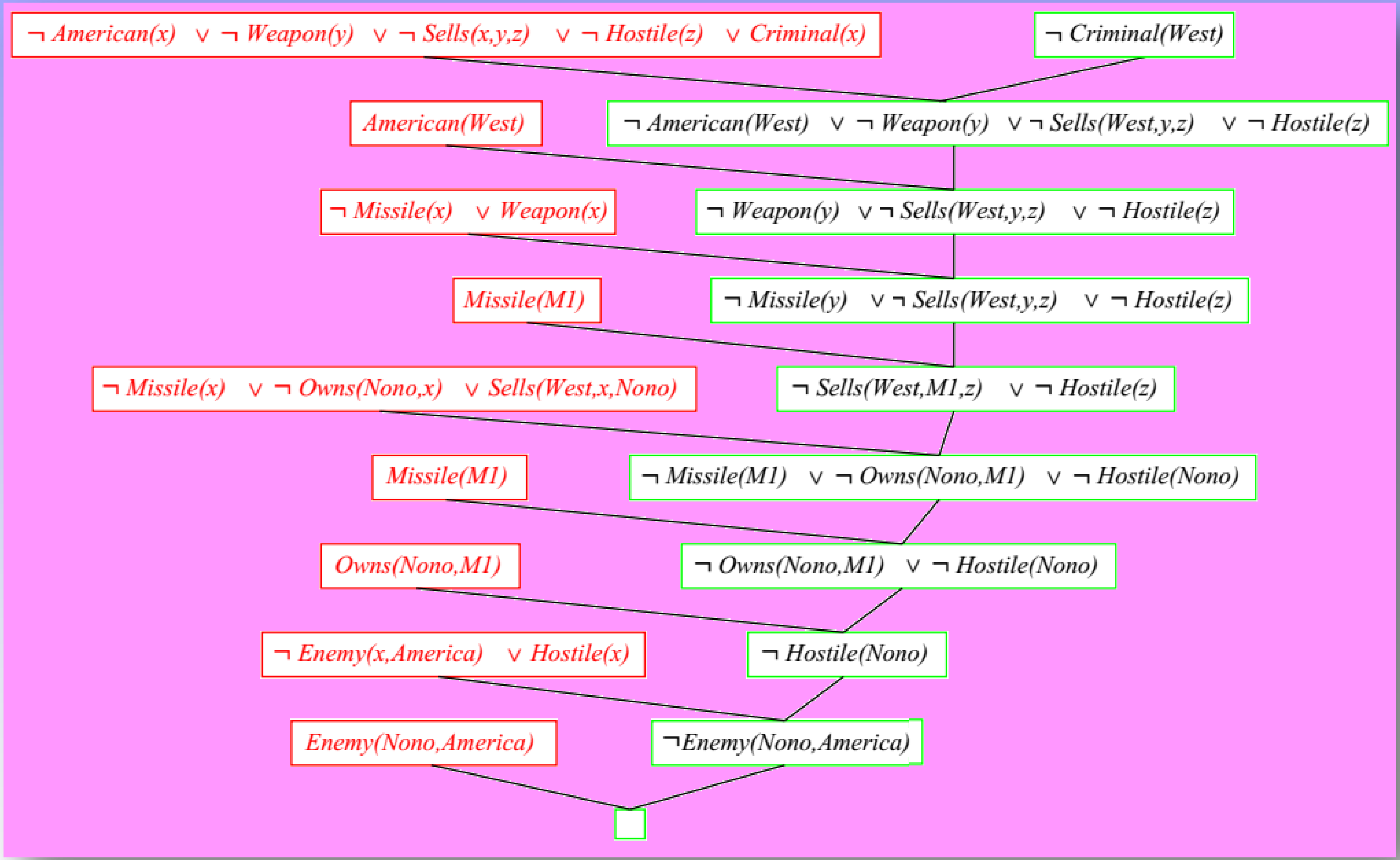
Resolution with Unification

- Example:
  - We can resolve the following two clauses:

    $[\text{Animal}(\text{F}(x)) \lor \text{Loves}(\text{G}(x), x)]$ and $[\neg\text{Loves}(u, v) \lor \neg\text{Kills}(u, v)]$
  - By eliminating the complementary literals:

    $\text{Loves}(\text{G}(x), x)$ and $\neg\text{Loves}(u, v)$
  - With unifier: $\theta = \{u/\text{G}(x), v/x\}$
  - To produce the **resolvent** clause: $[\text{Animal}(\text{F}(x)) \lor \neg\text{Kills}(\text{G}(x), x)]$
- This rule is called the **binary resolution** rule because it resolves exactly two literals.
  - The full resolution rule resolves subsets of literals in each clause that are unifiable.

Resolution: Example Knowledge Base


¬ 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)

Convert to FOL, Then to CNF

1. John likes all kinds of food
2. Apples are food.
3. Chicken is food.
4. Anything that anyone eats and isn't killed by is food.
5. Bill eats peanuts and is still alive.
6. Sue eats everything Bill eats.

Prove Using Resolution

1. John likes peanuts.
2. Sue eats peanuts.
3. Sue eats apples.
4. What does Sue eat?
   - Translate to Sue eats X
   - Result is a valid binding for X in the proof

Another Example

- Steve only likes easy courses
- Science courses are hard
- All the courses in the basket weaving department are easy
- BK301 is a basket weaving course
- What course would Steve like?

Thoughts on Resolution

- Resolution is sound and complete.
- Strategies (heuristics) for efficient resolution include:
  - **Unit preference:** may be incomplete
    - » Prefer to do resolutions where one of the sentences is a **unit clause**.
    - » Favor inferences that produce shorter clauses.
  - **Set of support:** may be incomplete
    - » Identify "useful" resolutions and ignore the rest.
    - » Every resolution involve at least an element of special set of clauses.
  - **Input resolution:** complete
    - » Every resolution combines one of the input sentences (from KB or the query) with some other sentence – used in last example.
    - » In Horn KBs, Modus Ponens is a kind of this strategy.
  - **Subsumption:** complete
    - » Eliminate all sentences that are subsumed by (more specific than) an existing sentence in KB.
    - » Keep KB small and thus helps keep the search space small.

Resolution Summary

- Resolution is a single inference role which is both sound and complete.
- Every sentence in KB is represented in clause form;
  - Any sentence in FOL can be reduced to this clause form.
- Two sentences can be resolved if one contains a +ve literal and the other contains a matching –ve literal:
  - The result is a new sentence which is the disjunction of the remaining literals in both.
- Resolution can be used as a relatively efficient theorem-prover by adding to the KB the negative of the sentence to be proved and attempting to derive a contradiction.

SUMMARY

- Inference is the process of adding information to the knowledge base.
- We want inference to be:
  - Sound: what we add is true if the KB is true.
  - Complete: if the KB entails a sentence we can derive it.
- Unification identify appropriate substitutions for variables in first-order proofs, making the process more efficient in many cases.
- Forward chaining, backward chaining, and resolution are typical inference mechanisms for first order logic.