Introduction to AI

Knowledge Representation and Reasoning

Francesca Toni

(thanks to Marek Sergot and Kostas Stathis)

Chapters 7–9, Russell and Norvig book

Outline

- Logic for Knowledge Representation and Reasoning
  - Unification
  - Resolution
- Forward chaining (bottom-up computations)
- Backward chaining (top-down computations)
- Logic Programming for Knowledge Representation and Reasoning

Knowledge Representation and Reasoning: problem

$$\begin{tikzcd}
\text{Problem} \arrow[r, "?"'] \arrow[d, "1"] & \text{Solution} & \underline{\text{Knowledge level}} \\
\text{Representation} \arrow[r, "2"] & \text{Reasoning output} \arrow[u, "3"] & \underline{\text{Symbol level}}
\end{tikzcd}$$

In order to find a solution to a problem:

1. find a suitable representation for the problem, equipped with a reasoning mechanism (for automatic computation of outputs)
2. compute output
3. the output can be mapped directly into a solution to the original problem

Knowledge Representation and Reasoning: approach

- How to represent knowledge underlying a (solution to a) problem in a form that a machine can automatically reason with?
- Logic-based mechanisms, as logic is equipped with
  - formal language (representation)
  - automated theorem proving (reasoning)

A brief history of reasoning

450b.c. Stoics	propositional logic, inference (maybe)
322b.c. Aristotle	“syllogisms” (inference rules), quantifiers
1565 Cardano	probability theory (propositional logic + uncertainty)
1847 Boole	propositional logic (again)
1879 Frege	first-order logic (FOL)
1922 Wittgenstein	proof by truth tables
1930 Gödel	$\exists$ complete algorithm for FOL
1930 Herbrand	complete algorithm for FOL (reduce to propositional)
1931 Gödel	$\neg\exists$ complete algorithm for arithmetic
1960 Davis/Putnam	“practical” algorithm for propositional logic
1965 Robinson	“practical” algorithm for FOL—resolution

Knowledge Representation for Resolution

- Resolution works with clauses:

$$\neg p_1 \lor \ldots \lor \neg p_m \lor q_1 \lor \ldots \lor q_n$$

where each p_i and each q_j is an atom, $m \geq 0, n \geq 0$

(m=n=0: empty clause/false/contradiction, often written $\square$)

- Every clause can be written equivalently as an implication:

$$p_1 \land \ldots \land p_m \rightarrow q_1 \lor \ldots \lor q_n$$

often written like this:

$$q_1 \lor \ldots \lor q_n \leftarrow p_1 \land \ldots \land p_m$$

Note: every formula in *propositional logic* is logically equivalent to a conjunction of clauses (conjunctive normal form). This property also holds for *first-order logic*, i.e.
a set of sentences can be written equivalently in clausal form
(universal quantification + conjunctive normal form+Skolemization).

Resolution inference rule: propositional case

The resolution inference rule combines two clauses to make a new one.

Basic propositional version:

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

This rule is applied repeatedly until the empty clause is derived.

e.g. given $\neg A \vee B$, $\neg B$, $\neg C \vee A$, C:

- resolution with $\neg A \vee B$, $\neg C \vee A$ gives $B \vee \neg C$
- resolution with C and $B \vee \neg C$ gives B
- resolution with B and $\neg B$ gives $\square$

Completeness of resolution

- If a set of propositional clauses is unsatisfiable, then resolution will eventually return the empty clause.
- Thus, to prove that P (the *query/goal*) is entailed by a set of sentences S (S $\models$ P):
  1. compute the conjunctive normal form S′ of S
  2. compute the conjunctive normal form NP′ of ¬P
  3. apply resolution to S′ and NP′
- *Issues:*
  - *First-order case?*
  - *Search for "good" sequence of resolution steps?*

First-order case: Universal instantiation

Every instantiation of a universally quantified sentence is entailed by it:

$$\frac{\forall v\, \alpha}{\alpha\{v/g\}}$$

for any variable v and term g

{v/g} is a substitution

- If g is a ground term (that contains no variables): ground instantiation
- for a substitution σ , the expression $\alpha\, \sigma$ is the formula obtained from α by applying σ

E.g.: $\forall x\, \forall y\, (Father(x,y) \land Happy(y) \rightarrow Happy(x))$ yields instantiations:

$Father(Joe,Joe) \land Happy(Joe) \rightarrow Happy(Joe)$	substitution {x/Joe, y/Joe}
$Father(Joe,Ann) \land Happy(Ann) \rightarrow Happy(Joe)$	substitution {x/Joe, y/Ann}
$Father(Joe,Bob) \land Happy(Bob) \rightarrow Happy(Joe)$	substitution {x/Joe, y/Bob}
$\forall x Father(x,Ann) \land Happy(Ann) \rightarrow Happy(x)$	substitution {y/Ann}
$\forall z Father(Mary, z) \land Happy(z) \rightarrow Happy(Mary)$	substitution {x/Mary, y/z}
$\forall x' \forall y' Father(x', y') \land Happy(y') \rightarrow Happy(x')$	substitution {x/x', y/y'}

…

Note: we use classical logic conventions here

Reduction to propositional inference

For a small set of sentences S, one way to proceed:

1. replace all sentences in S by their ground instantiations
2. now just use inference methods for propositional logic

But ...

With *p* predicates of arity *k* and n constants, there are $p * n^k$ instantiations!

Worse still, if there are function symbols in S, there are infinitely many instantiations:

- e.g. *A, F(A), F(F(A)), F(F(F(A))),...* are all ground terms.

First-order case: Unification

A substitution σ unifies atomic sentences p and q if $p\sigma = q\sigma$

p	q	σ
Knows(John, x)	Knows(John, Jane)	{x/Jane}
Knows(John, x)	Knows(y,OJ)	{x/OJ, y/John}
Knows(John, x)	Knows(y,Mother(y))	{y/John, x/Mother(John)}

Unification+resolution

Idea: Unify rule premises with known facts, apply unifier to conclusion.

E.g. from Knows(John, Jane)
Knows(John,OJ)
Knows(John,Mother(John))
and $\forall$ x(Knows(John, x) $\rightarrow$ Likes(John, x))
we can conclude Likes(John, Jane)
Likes(John,OJ)
Likes(John,Mother(John))

Most general unifier

Example: Knows(John, x) and Knows(John, y)

The following are all unifiers:

{x/John, y/John}

{x/Jane, y/Jane}

{x/Mother(John), y/Mother(John)}

{x/Mother(z), y/Mother(z)}

{x/z, y/z}

Only the last one is a ***most general unifier (mgu)***.

θ is a most general unifier of formulas α and β if and if

1. θ is a unifier of formulas α and β, i.e. $\alpha\,\theta = \beta\,\theta$, and
2. if σ is any other unifier of α and β ($\alpha\,\sigma = \beta\,\sigma$) then $\alpha\,\sigma$ is an instance of $\alpha\,\theta$, i.e. $\alpha\,\sigma = (\alpha\,\theta)\,\sigma'$ for some substitution σ'.

Unification algorithm

Given any two formulas, there is a (very efficient) unification algorithm which checks whether and can be unified, and produces a most general unifier if they can. (Details omitted)

Unification is very powerful. Some examples:

$$\left.\begin{array}{l} p(x, y, F(z)) \\ p(F(y), A, x) \end{array}\right\}$$ unifier {x/F(A),y/A,z/A}

$$\left.\begin{array}{l} p(x, x, F(F(A))) \\ p(y, F(z), F(y)) \end{array}\right\}$$ unifier {x/F(A),y/F(A),z/A}

$$\left.\begin{array}{l} p(x, F(A), y) \\ p(F(y), x, B) \end{array}\right\}$$ cannot unify: $A \neq B$ for constants A and B

$$\left.\begin{array}{l} p(x, x, F(A)) \\ p(F(y), y, z) \end{array}\right\}$$ cannot unify: would require y = F(y) – `occurs check’

Resolution inference rule: first-order case

Basic first-order version:

$$\frac{\alpha \vee \beta\,,\ \neg\beta' \vee \gamma}{(\alpha \vee \gamma)\theta}$$ where θ is a mgu of β, β'

Full first-order version:

$$\frac{\alpha^1 \vee \ldots \vee \alpha^{j-1} \vee \alpha^j \vee \alpha^{j+1} \ldots \vee \alpha^m, \quad \beta^1 \vee \ldots \beta^{k-1} \vee \beta^k \vee \beta^{k+1} \vee \ldots \vee \beta^n}{(\alpha^1 \vee \ldots \vee \alpha^{j-1} \vee \alpha^{j+1} \ldots \vee \alpha^m, \beta^1 \vee \ldots \beta^{k-1} \vee \beta^{k+1} \vee \ldots \vee \beta^n)\theta}$$

where θ is a mgu of α^j , $\neg\beta^k$

Special case: definite clauses

For many practical purposes it is sufficient to restrict attention to the special case of definite clauses :

$p \leftarrow q_1, q_2, ..., q_n$ [equivalently $\neg q_1 \lor ... \lor \neg q_n \lor p$]

where $p, q_1, ..., q_n$ are all atoms, ($n \geq 0$).

- We call p the *head* and $q_1, ..., q_n$ the *body* of the clause.
- When $n = 0$, the clause $p \leftarrow$ can be written as p - called a *fact*.

Note: Definite clauses are sometimes called 'Horn clauses'. Strictly speaking though the term 'Horn clauses' for definite clauses is not correct, since Horn clauses also include $\leftarrow q_1, ..., q_n$.

$\leftarrow q_1, ..., q_n$ is logically equivalent to $\neg q_1 \lor ... \lor \neg q_n$, which is logically equivalent to $\neg(q_1 \land ... \land q_n)$.

Generalized Modus Ponens (GMP)

For S a set of definite clauses, the following is what Russell and Norvig call Generalized Modus Ponens:

$$\frac{p'_1, p'_2, \ldots, p'_n, (q \leftarrow p_1, p_2, \ldots, p_n)}{q\theta}$$

where $p'_i\theta = p_i\theta$ forall i (θ is the composition of the mgus for all p'_I, p_i)

This is a special case of (several steps of) resolution.

- Example:

$$\frac{\text{Faster(Bob,Pat), Faster(Pat,Steve), } \forall\, x,y \text{ Faster(x, y)} \leftarrow \text{Faster(x, z), Faster(z,y)}}{\text{Faster(x,y) } \theta}$$

where θ = {x/Bob, y/Steve}

From propositional to first-order resolution: summary

- To prove that S $\models$ P:
    1. compute the conjunctive normal form S′ of S
    2. compute the conjunctive normal form NP′ of $\neg$P
    3. apply **first-order resolution** to S′ and NP′
        - If S′ is a set of definite clause and NP′ is a Horn clause $\leftarrow q_1, \ldots, q_n$ then apply **GMP** to derive $q_1, \ldots, q_n$ from S′
- *Issue: Search for "good" sequence of resolution/GMP steps?*

Knowledge Representation and Reasoning: Example

The US law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, and enemy of America, has some missiles of type M1, and all of its missiles were sold to it by Colonel West, an American.

Show that Colonel West is a criminal.

Knowledge Representation for the Example

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

  Criminal(x) ← American(x), Weapon(y), Sells(x, y, z), Hostile(z)

- Nono has missiles of type M1:

  Owns(Nono,M1)

  Missile(M1)

- All of Nono's missiles were sold to it by Colonel West, who is an American:

  Sells(West, x,Nono) ← Owns(Nono, x),Missile(x)

  American(West)

- Missiles are weapons, while an enemy of America counts as "hostile":

  Weapon(x) ← Missile(x)

  Hostile(x) ← Enemy(x,America)

- Nono is an enemy of America:

  Enemy(Nono,America)

Note: this is a set of definite clauses

Reasoning using Resolution: forward chaining (bottom-up computation)

- To prove that S $\models$ P:
  1. compute the conjunctive normal form S' of S
  2. compute the conjunctive normal form NP' of $\neg$P
  3. If S' is a set of definite clause and NP' is a Horn clause $\leftarrow q_1, ..., q_n$ then apply **GMP** to derive $q_1, ..., q_n$ from S'

- **Forward chaining**:
  - Split S' into a set of facts E and a set of rules (definite clauses) Pr.
  - Then apply the rules in Pr to the facts in E to derive (using GMP) a new set of implied facts E'
  - Add E' to E.
  - Repeat until no new facts are generated.
  - (If $q_1, ..., q_n$ are in E succeed.)
- Intuitively: When a new fact p is added to E, for each rule such that p unifies with a premise if the other premises are known then add the conclusion to E and continue chaining.

- Bottom-up computation is widely used in **deductive databases**

Forward chaining for Colonel West

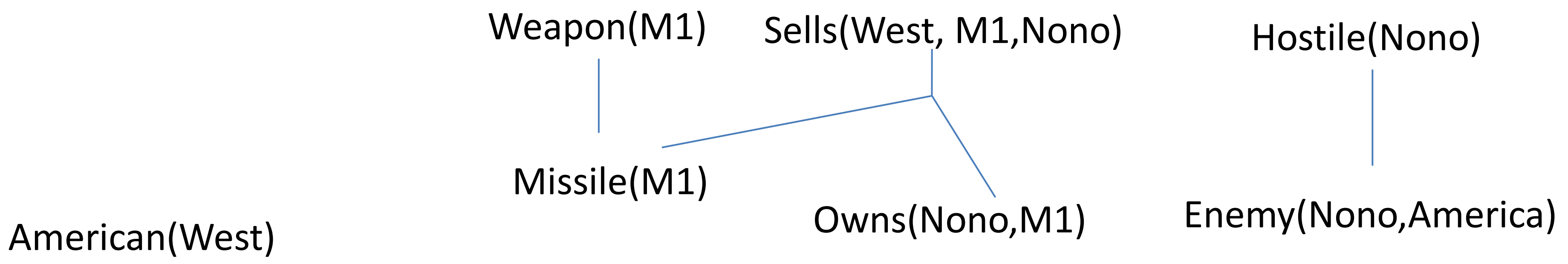
Missile(M1)

American(West)

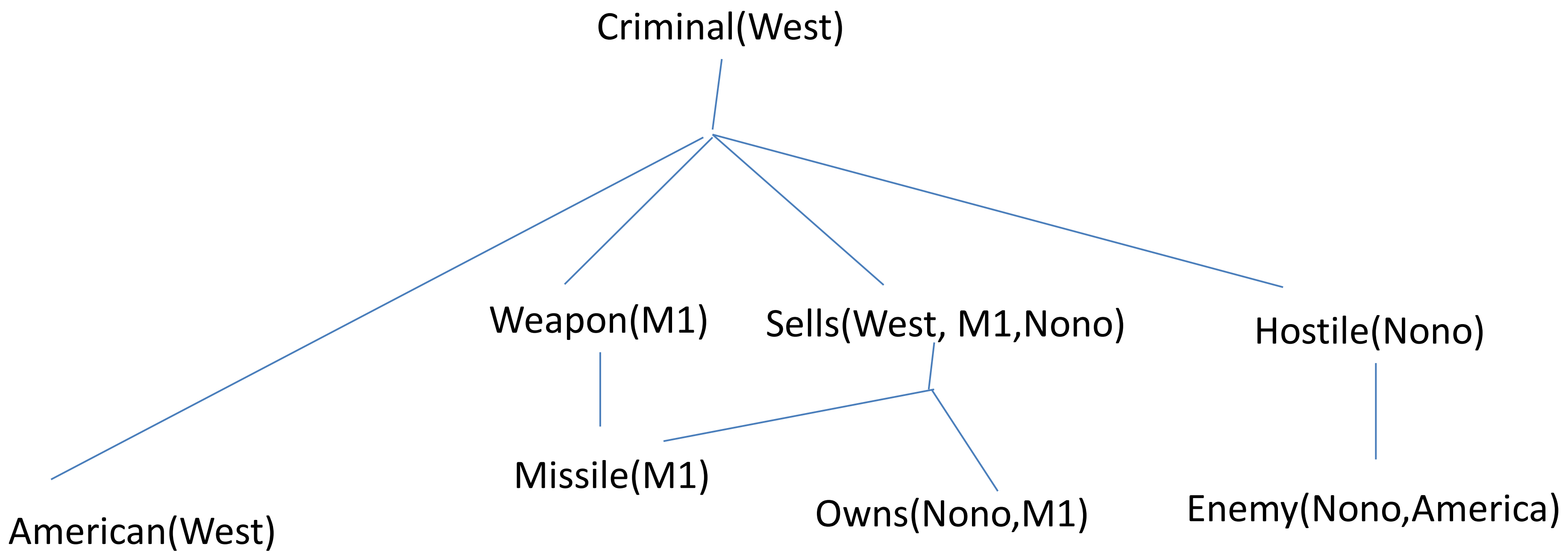
Owns(Nono,M1)

Enemy(Nono,America)

Forward chaining for Colonel West



Forward chaining for Colonel West



Forward chaining: observations

1. If a rule matched the facts on iteration k then it will still match the facts on iteration k + 1. (Lots of recomputation!) However...
2. In iteration k+1 it is only necessary to consider rules which have at least one condition in their body matching a fact obtained at iteration k.
3. If we have a particular query in mind that we want to answer, bottom up computation is likely to produce a lot of irrelevant facts.

Reasoning using Resolution: backward chaining (top-down computation)

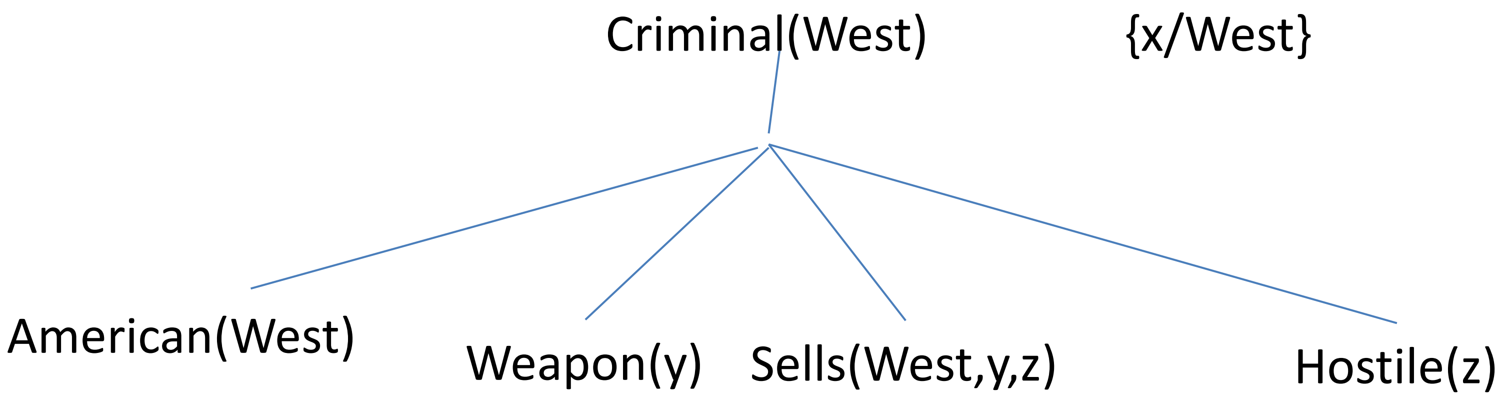
- To prove that $S \models P$:
  1. compute the conjunctive normal form S′ of S
  2. compute the conjunctive normal form NP′ of $\neg P$
  3. If S′ is a set of definite clause and NP′ is a Horn clause $\leftarrow q_1, \ldots, q_n$ then apply **GMP** to derive $q_1, \ldots, q_n$ from S′

- **Backward chaining**: To solve goal G wrt θ
  – if there is a matching fact G′ in S′, "add" mgu σ to θ ($G\sigma = G'\sigma$)
  – for each rule $G' \leftarrow G_1, \ldots, G_m$ in S′ whose head G′ matches G via mgu σ' ($G\sigma' = G'\sigma'$), solve goals $G_1\sigma', \ldots, G_m\sigma'$ wrt θ after "adding" σ' to θ
  – Repeat until there are no goals to solve, return θ
  – (Initially the goals to be solved are $q_1, \ldots, q_n$ and $\theta = \{\}$)

Backward chaining for Colonel West

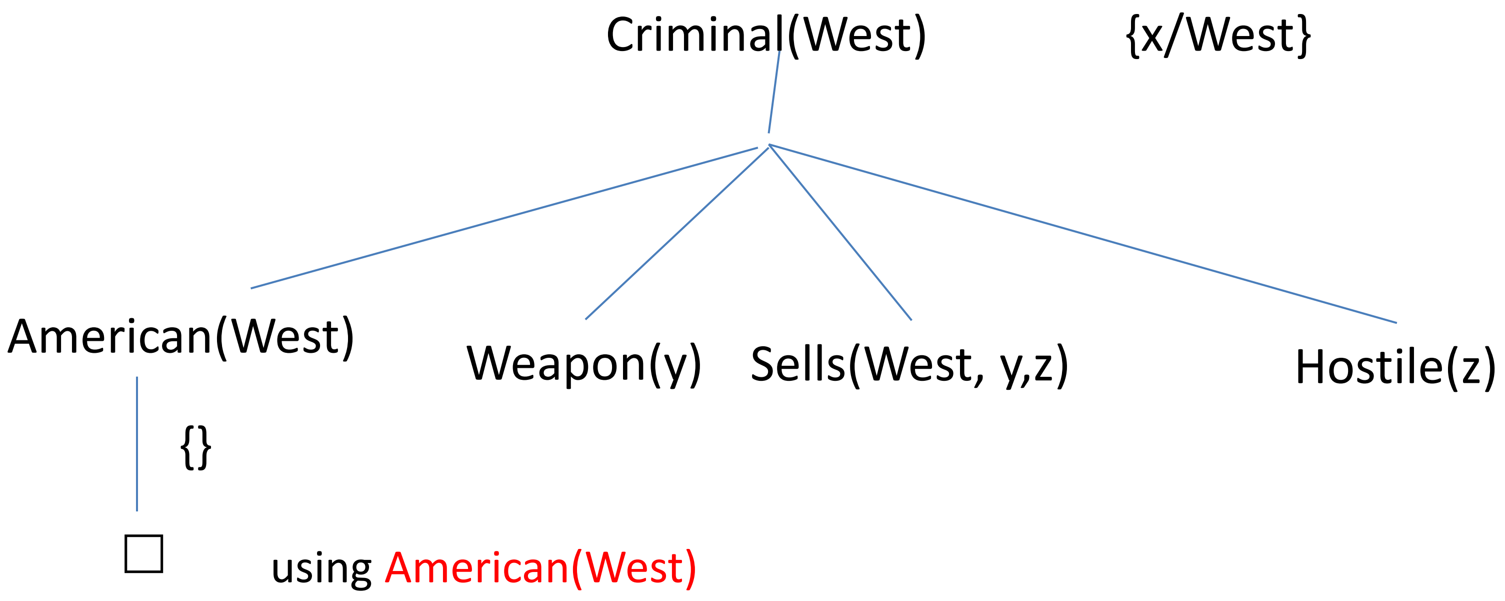
Criminal(West) {}

Backward chaining for Colonel West

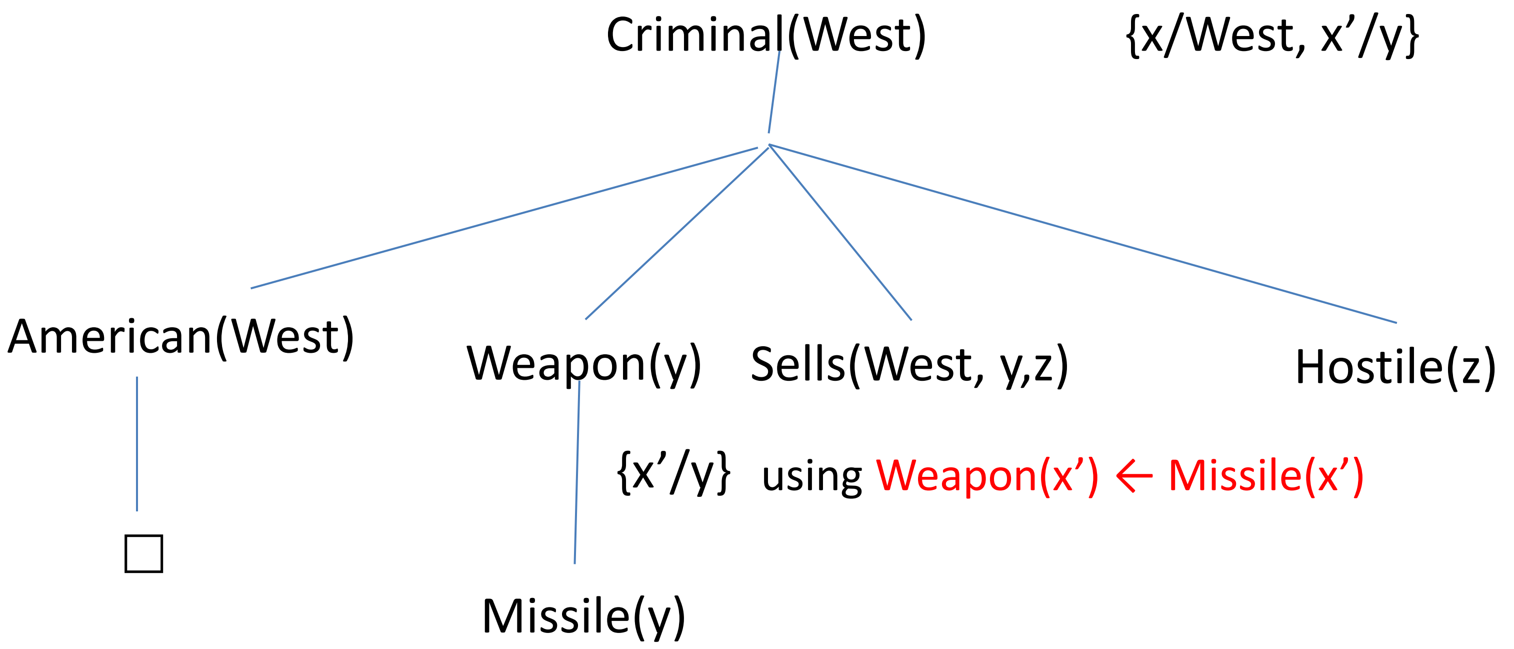


using Criminal(x) ← American(x), Weapon(y), Sells(x, y, z), Hostile(z)

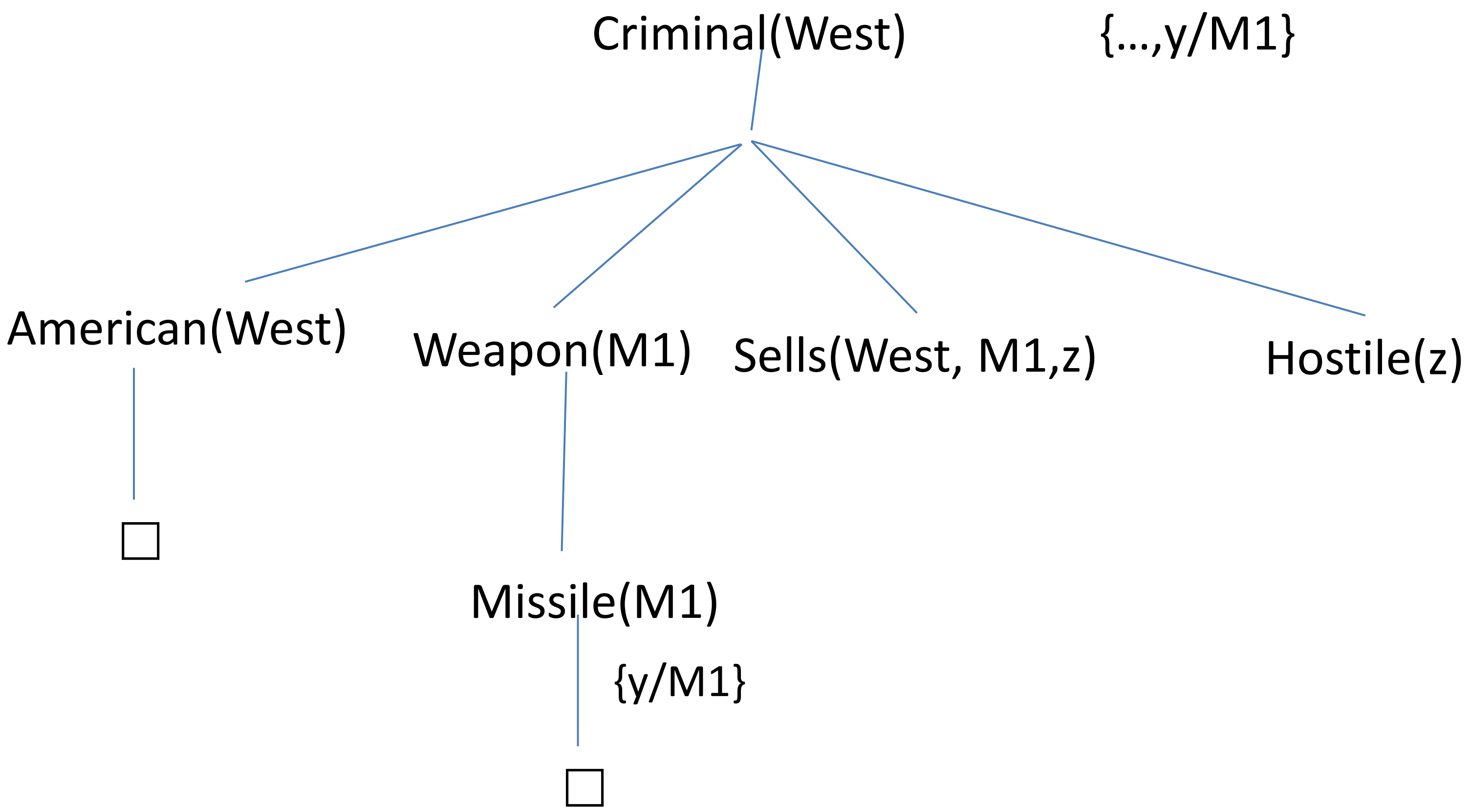
Backward chaining for Colonel West



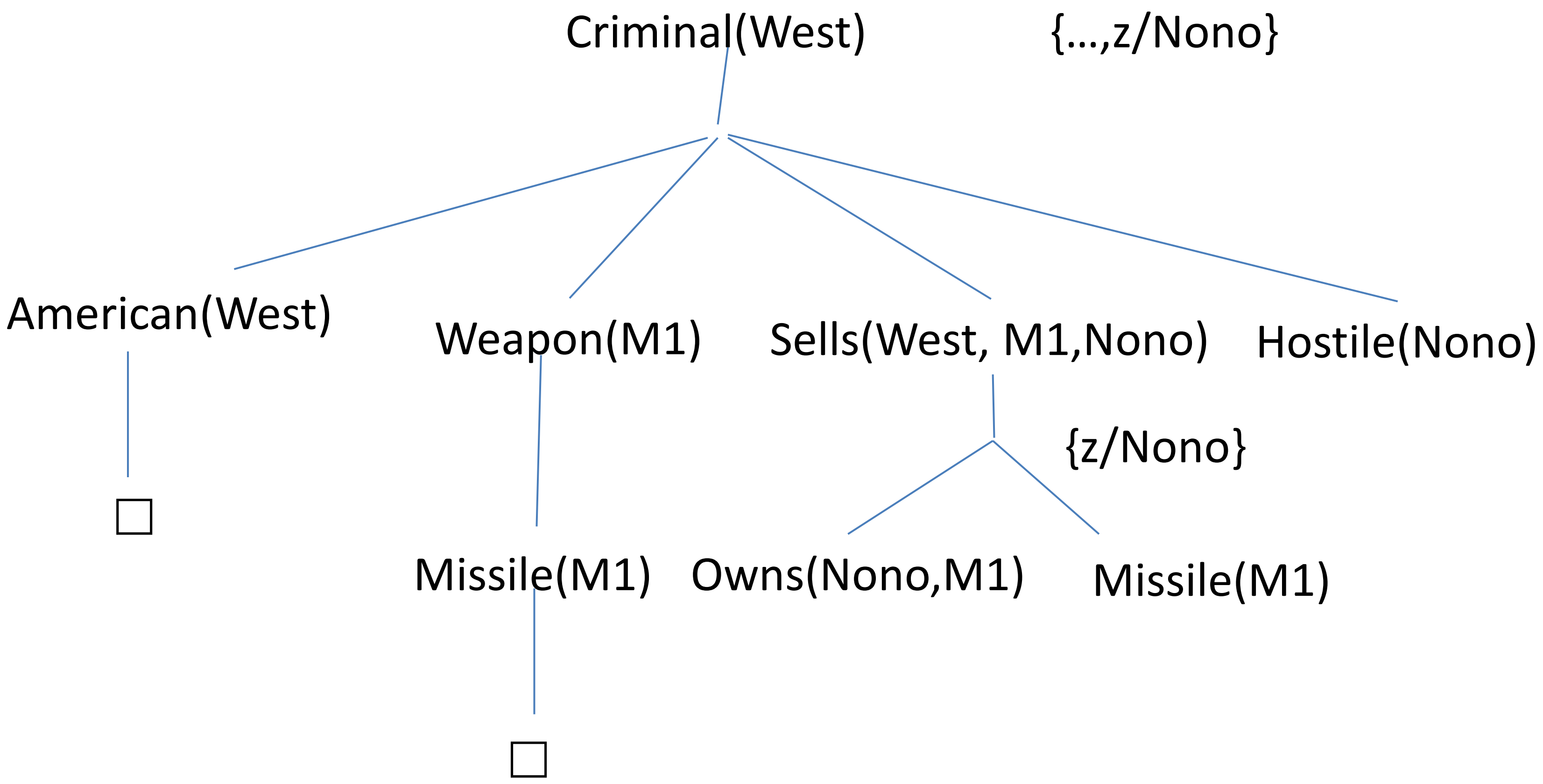
Backward chaining for Colonel West



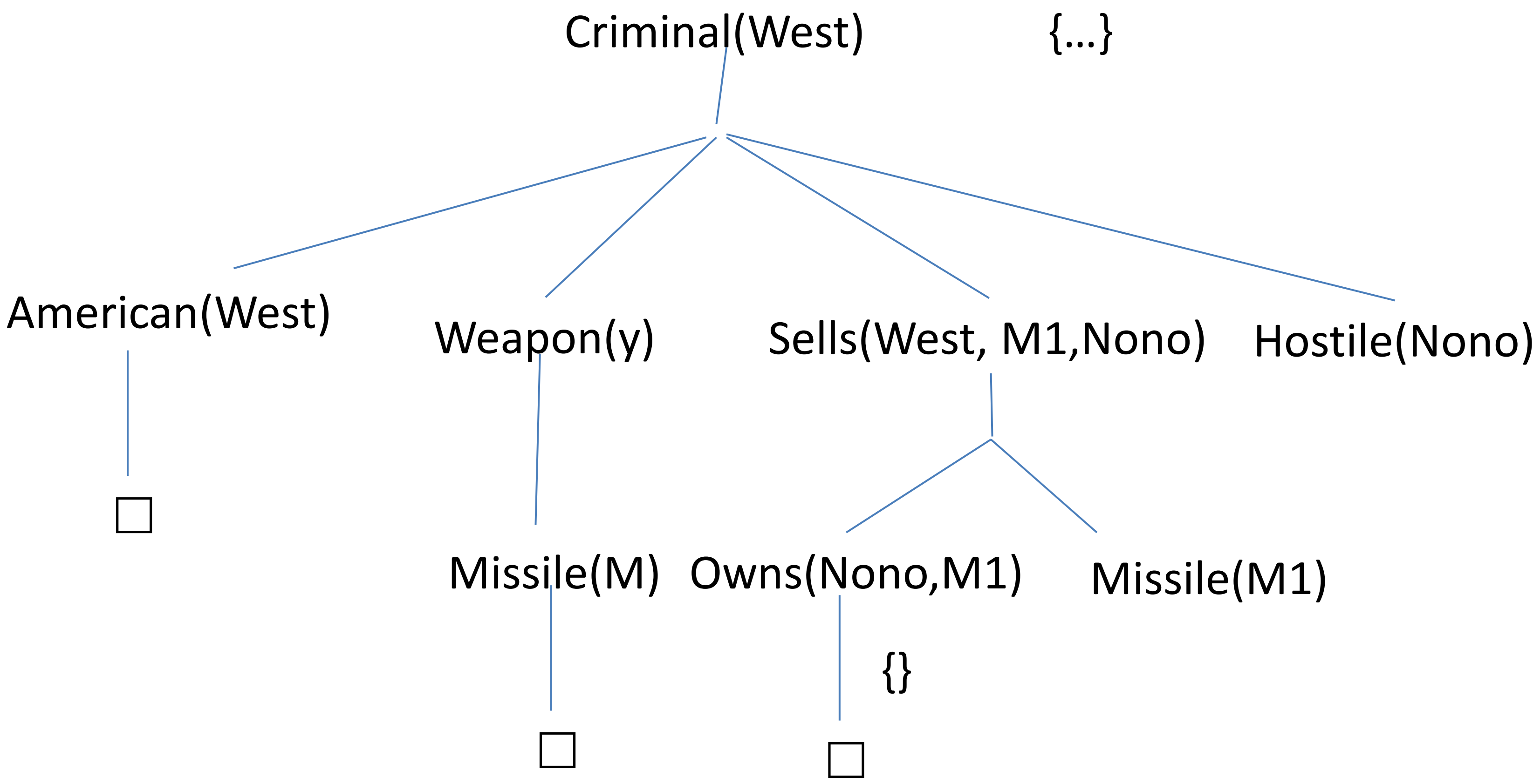
Backward chaining for Colonel West



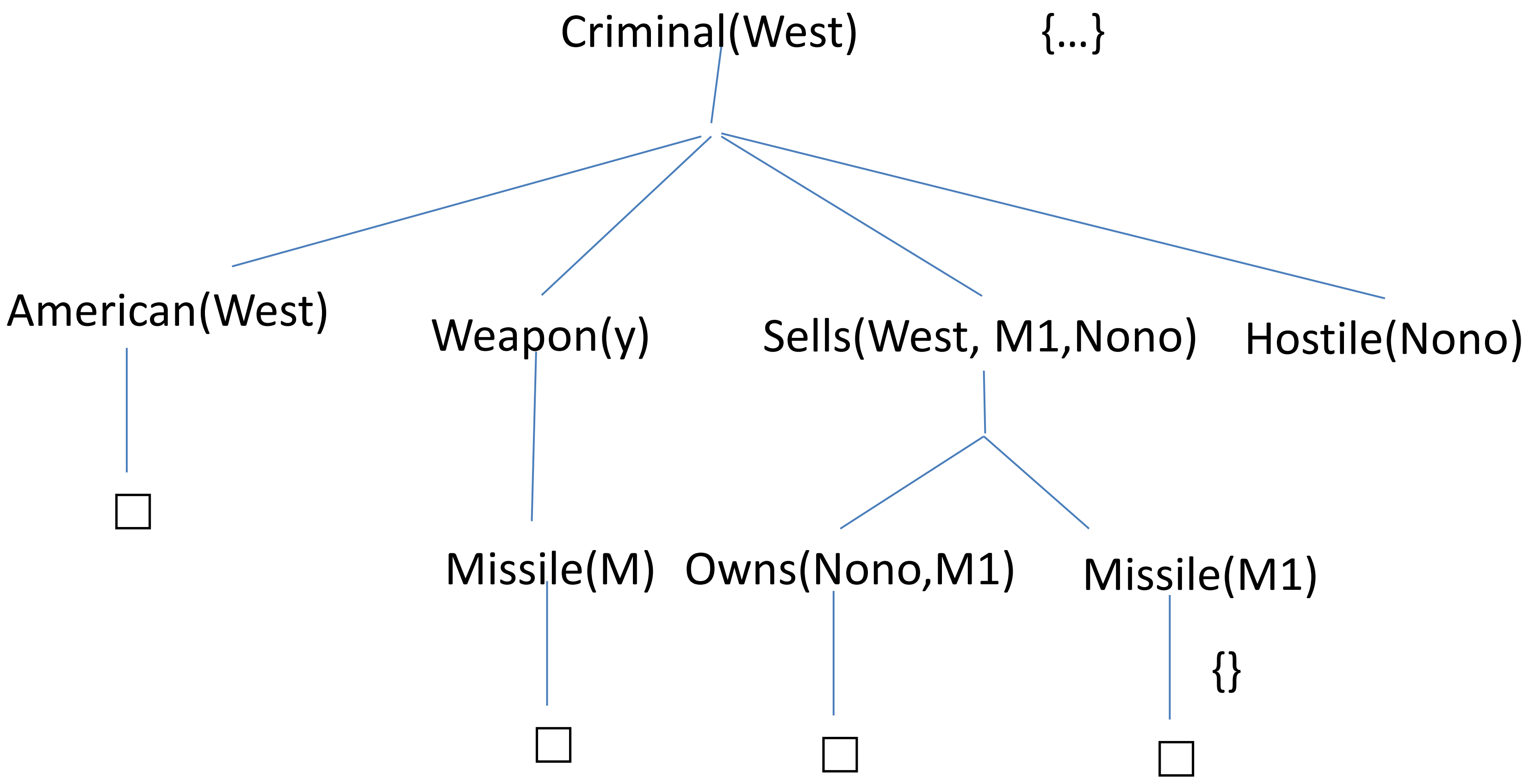
Backward chaining for Colonel West



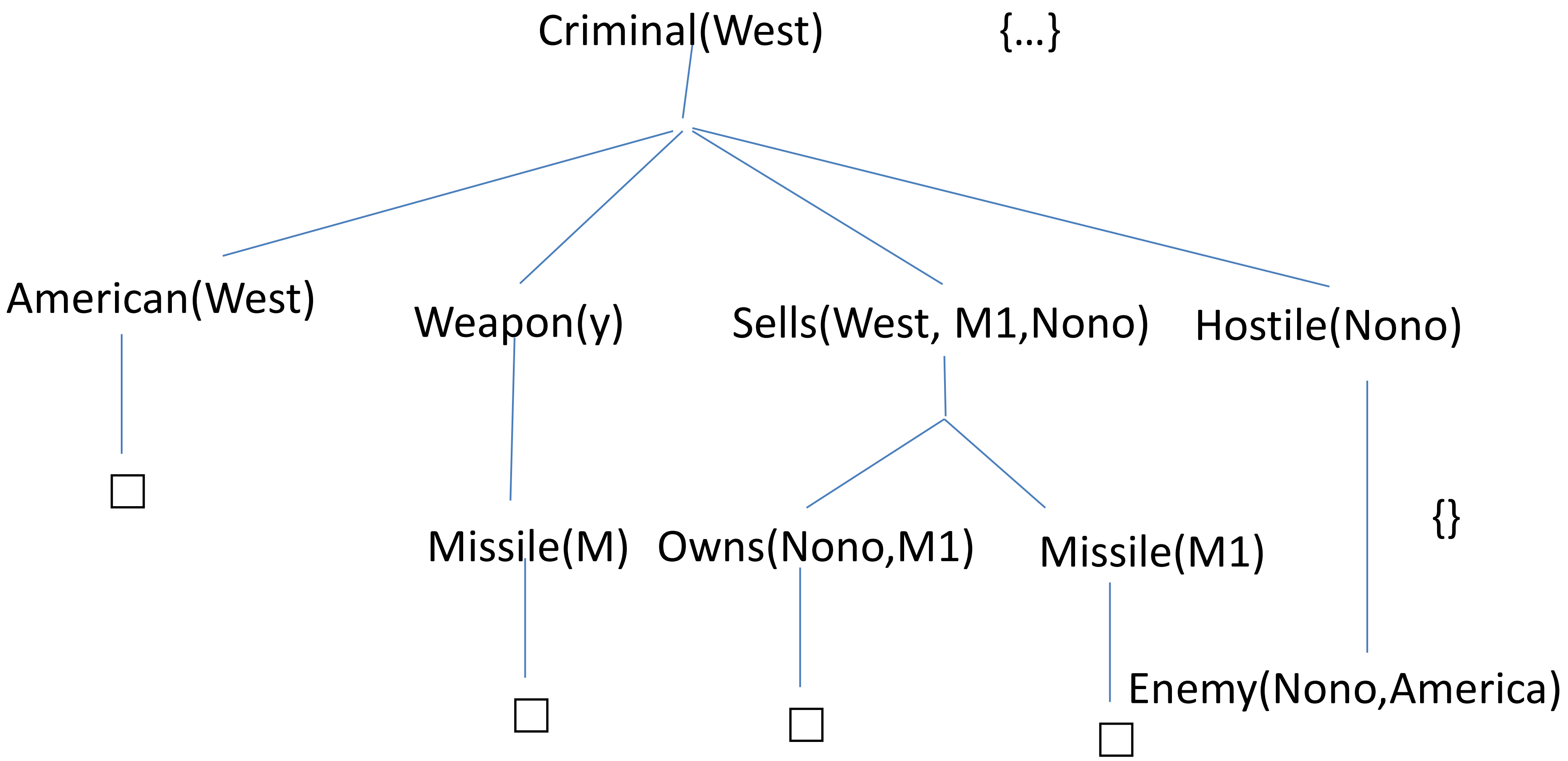
Backward chaining for Colonel West



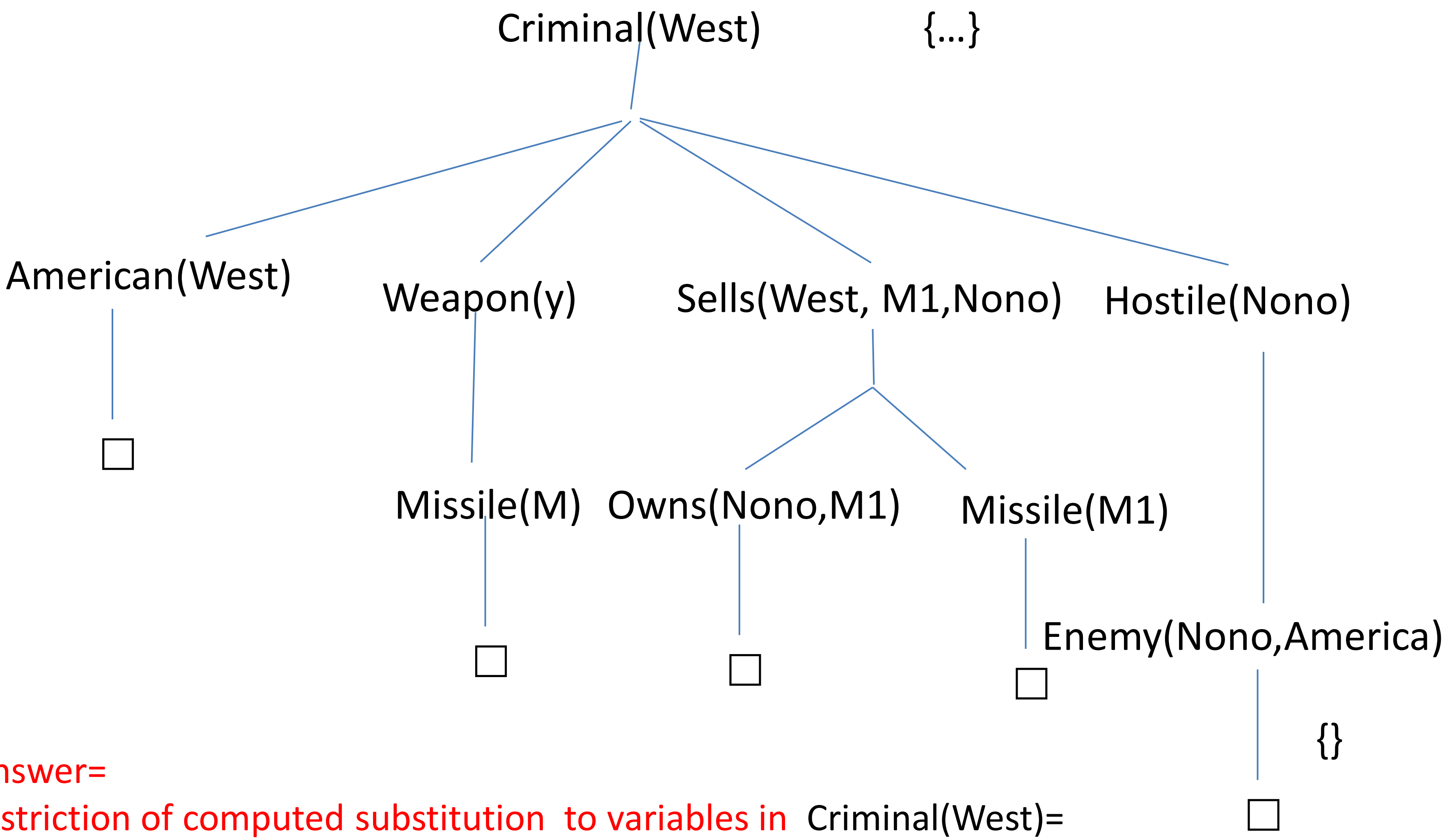
Backward chaining for Colonel West



Backward chaining for Colonel West



Backward chaining for Colonel West



Answer=
restriction of computed substitution to variables in Criminal(West)=
{}

Another way of depicting a backward chaining for Colonel West

$$
\begin{array}{rl}
 & \leftarrow \text{Criminal(West)} \\
 & | \\
 & \leftarrow \text{American(West), Weapon(y), Sells(West, y, z),Hostile(z)} \\
 & | \\
 & \leftarrow \text{Weapon(y), Sells(West, y, z),Hostile(z)} \\
 & | \\
 & \leftarrow \text{Missile(y), Sells(West, y, z),Hostile(z)} \\
\{y/M1\} & | \\
 & \leftarrow \text{Sells(West,M1, z),Hostile(z)} \\
\{z/Nono\} & | \\
 & \leftarrow \text{Owns(Nono,M1),Missile(M1),Hostile(Nono)} \\
 & | \\
 & \leftarrow \text{Hostile(Nono)} \\
 & | \\
 & \leftarrow \text{Enemy(Nono,America)} \\
 & | \\
 & \square
\end{array}
$$

Answer: substitution {}

Backward chaining for Colonel West - strictly speaking

```
           ← Criminal(West)
{x/West} |
           ← American(West), Weapon(y), Sells(West, y, z),Hostile(z)
      {}   |
           ← Weapon(y), Sells(West, y, z),Hostile(z)
   {x’/y} |
           ← Missile(y), Sells(West, y, z),Hostile(z)
  {y/M1} |
           ← Sells(West,M1, z),Hostile(z)
{z/Nono} |
           ← Owns(Nono,M1),Missile(M1),Hostile(Nono)
     {}    |
           ← Hostile(Nono)
     {}    |
           ← Enemy(Nono,America)
     {}    |
           □
```

Another query/goal for Colonel West

$\leftarrow$ Criminal(**x**)

|

$\leftarrow$ American(x), Weapon(y), Sells(West, y, z),Hostile(z)

{x/West} |

$\leftarrow$ Weapon(y), Sells(West, y, z),Hostile(z)

|

$\leftarrow$ Missile(y), Sells(West, y, z),Hostile(z)

|

$\leftarrow$ Sells(West,M1, z),Hostile(z)

|

$\leftarrow$ Owns(Nono,M1),Missile(M1),Hostile(Nono)

|

$\leftarrow$ Hostile(Nono)

|

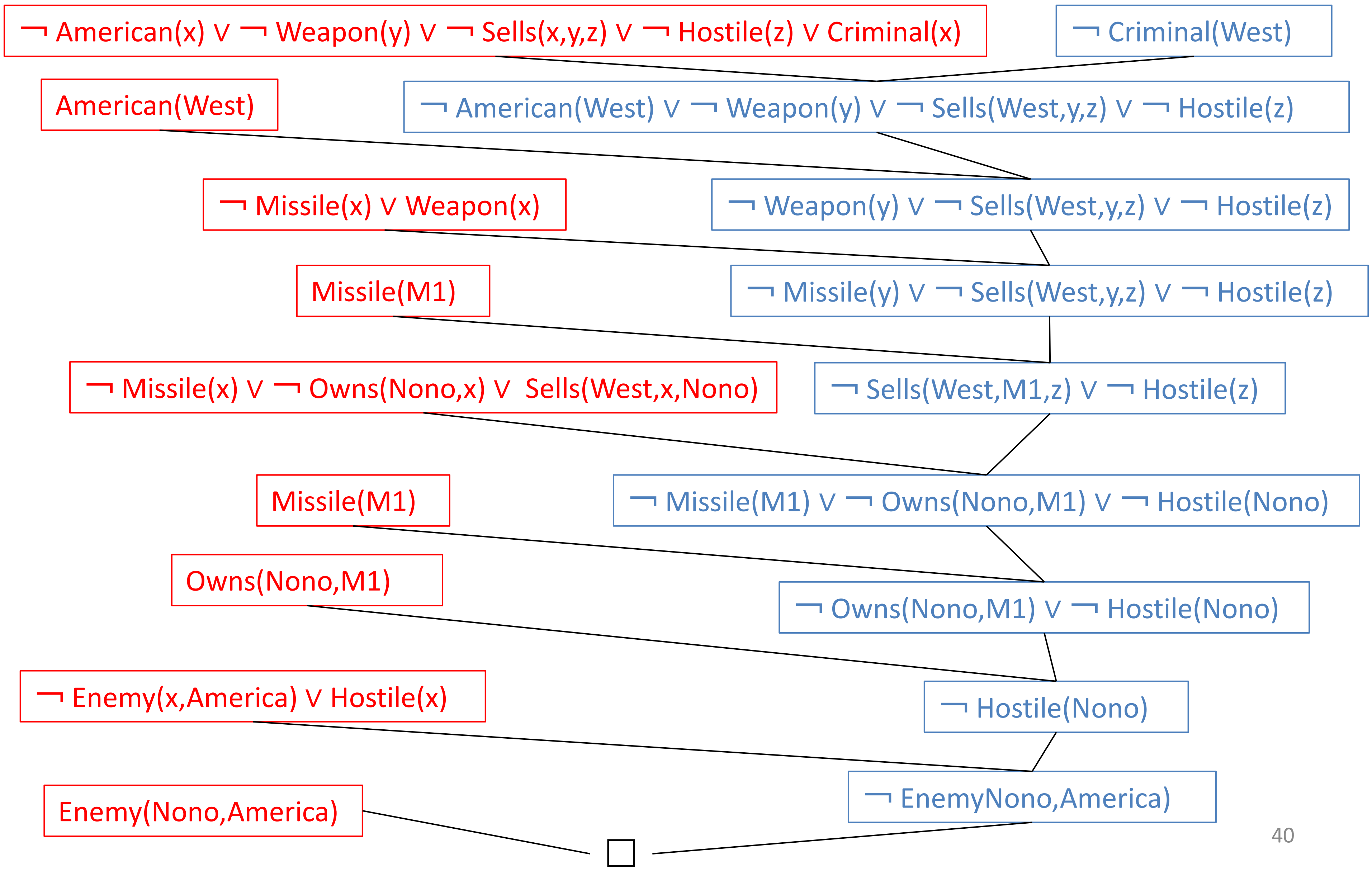
$\leftarrow$ Enemy(Nono,America)

|

□

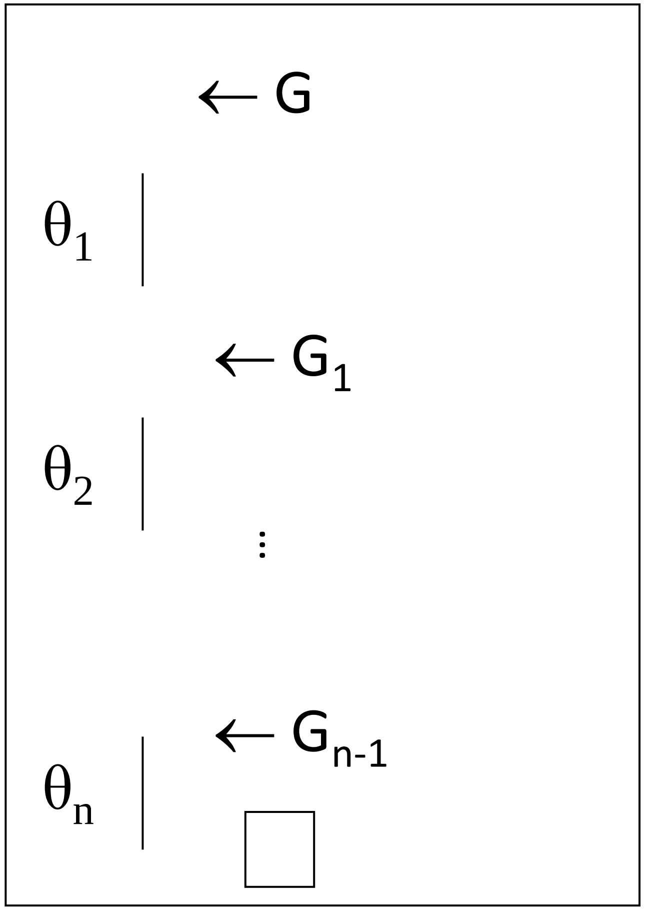
Answer: substitution {x/West}

Resolution view of Colonel West Example


¬ 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)
¬ EnemyNono,America)
□

More on Backward Chaining

The computation of a goal (query) G is a series of derivation steps:



- θ_i is the mgu at the i-th derivation step
- The answer computed is (the restriction to the vars of G of) the composition of all these mgus:

  $\theta = \theta_1 \circ \ldots \circ \theta_n$

More on Backward Chaining

- Each derivation step looks like this:

$$\begin{array}{ll} & \leftarrow L_1,\ldots,L_{j-1}, B, L_{j+1},\ldots,L_n \\ \theta_i & \Big|\ \text{match } B \text{ with } B' \leftarrow M_1,\ldots,M_k \text{, with } B\theta_i = B'\theta_i \\ & \leftarrow (L_1,\ldots,L_{j-1}, M_1,\ldots,M_k, L_{j+1},\ldots,L_n)\theta_i \end{array}$$

The sub-goal B selected for matching can be any one of the sub-goals in the current goal

- e.g. always choose the leftmost sub-goal.
- The answers computed are the same, whichever sub-goal is selected!

- Many possible choices for matching clause.
  - The choice might affect termination

Example: search for solutions

Admires(Ann, Bob)   Admires(Ann, Carla)   Admires(x,y) ← Lecturer(x), Lecturer(y)
Lecturer(Ann)   Lecturer(Dave)   Lecturer(Eric)
Rich(Carla)   Rich(Eric)   Rich(Ann) ← Rich(Carla)

Which rich person does Ann admire?

← Admires(Ann,z),Rich(z)

{z/Bob} |

← Rich(Bob)

failure   *but this is not the only possible derivation – backtrack*

retry ← Admires(Ann,z),Rich(z)

{z/Carla} |

← Rich(Carla)

|

☐ {z/Carla}   *any other answers? - backtrack*

Example: search for solutions

Admires(Ann, Bob)   Admires(Ann, Carla)   Admires(x,y) ← Lecturer(x), Lecturer(y)
Lecturer(Ann)   Lecturer(Dave)   Lecturer(Eric)
Rich(Carla)   Rich(Eric)   Rich(Ann) ← Rich(Carla)

Which rich person does Ann admire?

```
retry ← Admires(Ann,z),Rich(z)
            |
   ← Lecturer(Ann),Lecurer(z), Rich(z)
            |
   ← Lecurer(z), Rich(z)                                      retry  ← Lecurer(z), Rich(z)
 {z/Ann} |                                                           {z/Dave} |
  ← Rich(Ann)                                                        ← Rich(Ann)
             |                                                         fails
 ← Rich((Carla) )                                             retry  ← Lecurer(z), Rich(z)
            |                                                          {z/Eric} |
           □   {z/Ann}    any other answers? - backtrack                ← Rich(Eric)
                                                                              |
                                                                           □ {z/Eric}
```

Computed answers: {z/Carla} , {z/Ann} , {z/Eric}

Logic programming

- Computation as inference on a logical knowledge base (KB).
  - definite clauses + many extensions
- Bottom-up + top-down computations + many optimisations.
- Prolog: the most widely used programming language based upon logic programming.
  - a programming language!
  - Program = set of clauses: head :- literal$_1$,...,literal$_n$.
  - literals might include:
    - negation as failure (also in logic programming)
    - findall assert, IO features, etc (not in logic programming)
    - conventions on variables/constants etc
- Prolog has:
  - Efficient unification
  - Efficient retrieval of matching clauses by indexing techniques.
  - Depth-first, left-to-right search to find alternative solutions (with backtracking)
  - Built-in predicates for arithmetic etc., e.g., X is Y*Z+3

Colonel West in Prolog

```
criminal(X):- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
sells(west,Y,nono):- owns(nono,Y), missile(Y).
...
enemy(nono, america).
```

Query:

```
?- criminal(X).
X = west;
no.
```

Summary

- Knowledge representation and reasoning as inference in (subsets of) first-order logic (FOL)
- Inference in FOL often needs one (or more) of:
  - Unification
  - Resolution
  - Generalized Modus Ponens - Definite Clauses
  - Forward chaining (bottom-up computations)
  - Backward chaining (top-down, goal-directed computations)
- Efficient implementation of a subset of first-order logic can be built upon:
  - Logic programming and Prolog