Artificial Intelligence

6. First Order Logic Inference

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute of Economics and Information Systems
& Institute of Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

1. **Unification**

2. **Forward Chaining**

3. **Backward Chaining**

4. **Resolution**

"Compound Expressions"

Formulas and function terms sometimes are described as **compound expressions**.

For a compound expression, its operator and its arguments is defined:

$$
\begin{array}{ll}
\text{op}(P(t_1,\ldots,t_n)) := P & \text{args}(P(t_1,\ldots,t_n)) := (t_1,\ldots,t_n) \\
\text{op}(f(t_1,\ldots,t_n)) := f & \text{args}(f(t_1,\ldots,t_n)) := (t_1,\ldots,t_n) \\
\text{op}(\neg\phi) := \neg & \text{args}(\neg\phi) := (\phi) \\
\text{op}(\phi \oplus \psi) := \oplus & \text{args}(\phi \oplus \psi) := (\phi, \psi), \quad \oplus \in \{\wedge, \vee, \rightarrow, \leftrightarrow\} \\
\text{op}(\forall x\phi) := \forall & \text{args}(\forall x\phi) := (x, \phi) \\
\text{op}(\exists x\phi) := \exists & \text{args}(\exists x\phi) := (x, \phi)
\end{array}
$$

Atomic terms, i.e., constants and variables, are not considered compound expressions.

Unification / Algorithm

unify(x, y, θ) :
if $\theta =$ failure
    return failure
elsif $x = y$
        return θ
elsif *is-variable*(x)
        return *unify-var*(x, y, θ)
elsif *is-variable*(y)
        return *unify-var*(y, x, θ)
elsif *is-compound*(x) and *is-compound*(y)
        return *unify*(*args*(x), *args*(y), *unify*(*op*(x), *op*(y), θ))
elsif *is-list*(x) and *is-list*(y)
        return *unify*$((x_2, \ldots, x_n), (y_2, \ldots, y_n)$, *unify*$(x_1, y_1, \theta))$
else
        return failure
fi

unify-var(*var*, x, θ) :
if θ(*var*) $\neq \emptyset$
    return *unify*(θ(*var*), x, θ)
elsif $\theta(x) \neq \emptyset$
        return *unify*(*var*, $\theta(x), \theta$)
elsif *occurs*(*var*, x)
        return *failure*
else
        return $\theta \cup \{\mathit{var} \mapsto x\}$

Unification / Example

$$
\begin{aligned}
&\text{unify}(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Mother}(y)), \emptyset) \\
&= \text{unify}((\text{John}, x), (y, \text{Mother}(y)), \text{unify}(\text{Knows}, \text{Knows}, \emptyset)) \\
&= \text{unify}((\text{John}, x), (y, \text{Mother}(y)), \emptyset) \\
&= \text{unify}((x), (\text{Mother}(y)), \text{unify}(\text{John}, y, \emptyset)) \\
&= \text{unify}((x), (\text{Mother}(y)), \{y/\text{John}\}) \\
&= \text{unify-var}(x, \text{Mother}(y), \{y/\text{John}\}) \\
&= \{y/\text{John}, x/\text{Mother}(y)\}
\end{aligned}
$$

1. Unification

2. **Forward Chaining**

3. Backward Chaining

4. Resolution

Generalized Modus Ponens

premise	conclusion	name
$\mathcal{F} \vdash F, \mathcal{F} \vdash F \rightarrow G$	$\mathcal{F} \vdash G$	$\rightarrow$ -elimination / modus ponens
$\mathcal{F} \vdash F$	$\mathcal{F} \vdash F\theta$	universial instantiation
$\mathcal{F} \vdash F, \mathcal{F} \vdash F' \rightarrow G, F\theta = F'\theta$	$\mathcal{F} \vdash G\theta$	generalized modus ponens

Lemma 1. *Generalized modus ponens is sound.*

Proof.

1. $\mathcal{F} \vdash F$ [assumption]
2. $\mathcal{F} \vdash F\theta$ [universal instantiation applied to 1]
3. $\mathcal{F} \vdash F' \rightarrow G$ [assumption]
4. $\mathcal{F} \vdash F'\theta \rightarrow G\theta$ [universal instantiation applied to 3]
5. $\mathcal{F} \vdash G\theta$ [$\rightarrow$ -elemination applied to 2,4]

$\square$

Generalized Modus Ponens / Example

Let the knowledge base $\mathcal{F}$ be

$$\text{King}(x) \wedge \text{Greedy}(x) \rightarrow \text{Evil}(x)$$
$$\text{King(John)}$$
$$\text{Greedy}(y)$$

Now use

$$\begin{aligned} F :=& \text{King(John)} \wedge \text{Greedy}(y) \\ F' :=& \text{King}(x) \wedge \text{Greedy}(x) \\ G :=& \text{Evil}(x) \end{aligned}$$

then for

$$\theta := \{x/\text{John}, y/\text{John}\}$$

we have

$$F\theta = \text{King(John)} \wedge \text{Greedy(John)} = F'\theta$$

and thus we can derive

$$G\theta = \text{Evil(John)}$$

Forward Chaining

Definitions for **conjunctive normal forms** (CNF), **Horn clauses** and **Horn formulas** are the same as in propositional logic.

Here, atoms are formulas

$$P(t_1, t_2, \ldots, t_n)$$

where P is a predicate symbol and t_i are any terms (including variables).

A Horn clause C is called **definite** it it contains exactly one positive literal, i.e., implications of type

$$(\bigvee_{i=1}^{n} \neg L_i) \equiv (\bigwedge_{i=1}^{n} L_i \rightarrow \text{false})$$

are not possible.

If the knowledge base consists of **Horn clauses** only, then generalized modus ponens can be used just like modus ponens to infer statements iteratively by forward chaining.

Example

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

Prove that Col. West is a criminal.

Example (2/4)

The law says that it is a crime for an American to sell weapons to hostile nations.

$\forall \text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Hostile}(z) \wedge \text{Sell}(x, y, z) \rightarrow \text{Criminal}(x)$

The country Nono,

Country(Nono)

an enemy of America,

Enemy(Nono, America)

has some missiles,

$$\exists x \text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x)$$

and all of its missiles were sold to it by Colonel West,

$$\forall x \text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \rightarrow \text{Sell}(\text{West}, x, \text{Nono})$$

who is American.

American(West)

Example (3/4)

Additional background knowledge:
Missiles are weapons.

$$\forall x \text{Missile}(x) \rightarrow \text{Weapon}(x)$$

Enemies of America are hostile.

$$\forall x \text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)$$

Prove that Col. West is a criminal

$$\text{Criminal(West)}?$$

Example (4/4)

The knowledge base can be simplified by

- existential instantiation and
- omitting universal quantifiers
(as all free variables are universally quantified anyway)

American(x) ∧ Weapon(y) ∧ Hostile(z) ∧ Sell$(x, y, z) \rightarrow$ Criminal(x)
Country(Nono)
Enemy(Nono, America)
Missile(M_1) ∧ Owns(Nono, M_1)
Missile(x) ∧ Owns(Nono, x) $\rightarrow$ Sell(West, x, Nono)
American(West)
Missile$(x) \rightarrow$ Weapon(x)
Enemy(x, America) $\rightarrow$ Hostile(x)

$\rightsquigarrow$ This knowledge base consists of definite Horn clauses only !

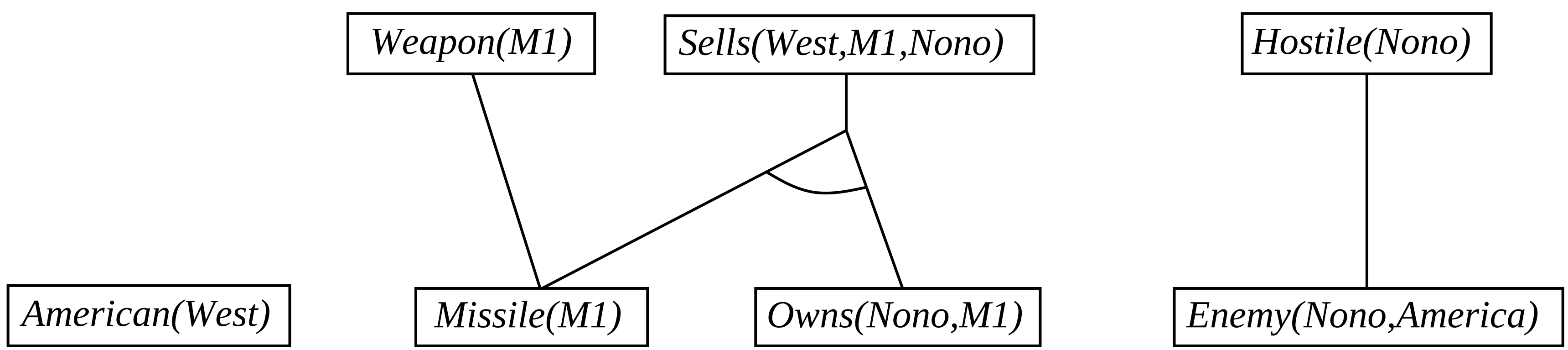
Forward Chaining

```
entails-fc(FOL definite horn formula F, query atom Q) :
C := ∅
C' := clauses(F)
while C' ≠ ∅ do
        C := C ∪ C'
        C' := ∅
        for C ∈ C do
            C' := standardize-apart(C)
            for atoms A_1, A_2, ..., A_n ∈ C and θ with body(C')θ = (A_1 ∧ A_2 ∧ ... ∧ A_n)θ do
                H := head(C')θ
                if H ∉ C and H ∉ C'
                    C' := C' ∪ {H}
                    if unify(H, Q) return true fi
                fi
            od
        od
od
return false
```

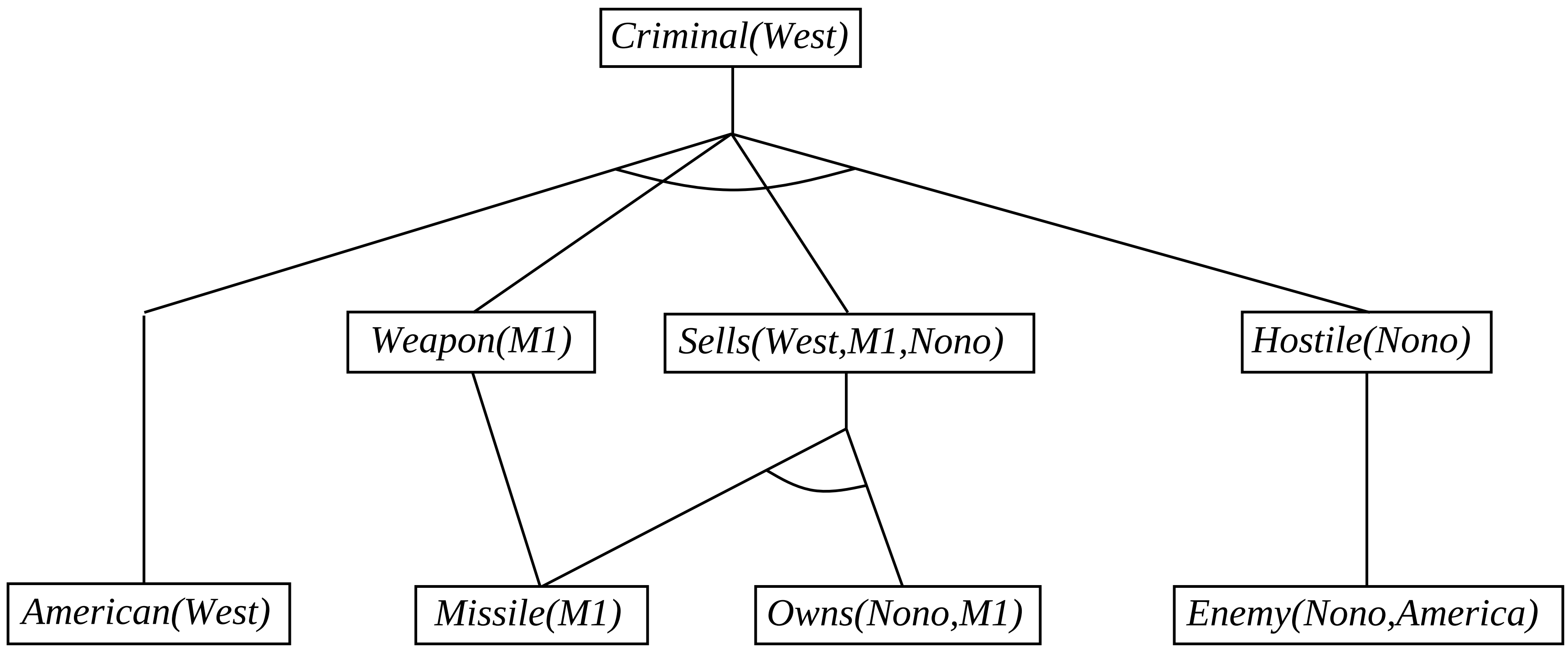
Forward Chaining / Example

American(West) *Missile(M1)* *Owns(Nono,M1)* *Enemy(Nono,America)*

Forward Chaining / Example



Forward Chaining / Example



1. Unification
2. Forward Chaining
3. **Backward Chaining**
4. Resolution

Backward Chaining

Backward chaining works the other way around:

- keep a list of yet unsatisfied atoms Q
  – starting with the query atom.

- try to find rules whichs head match atoms in Q (after unification) and
  replace the atom from Q by the atoms of the body of the matching rule.

- proceed recursively until no more atoms have to be satisfied.

Backward chaining keeps track of the substitution needed during the proof.

Backward Chaining / Algorithm

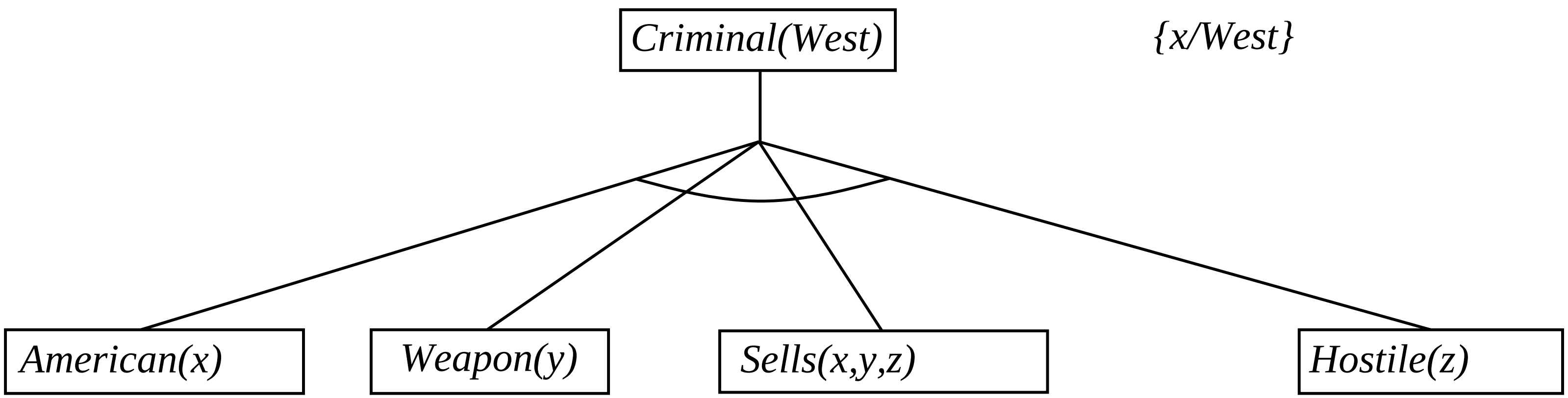
entails-bc(FOL definite horn formula F, query atom Q) :
return entails-bc-goals(clauses(F), $\{Q\}, \emptyset) \neq \emptyset$

entails-bc-goals(set of FOL definite Horn clauses $\mathcal{C}$, set of FOL atoms Q, θ) :
if $Q = \emptyset$ **return** $\{\theta\}$ **fi**
$\Theta := \emptyset$
for $C \in \mathcal{C}$ **do**
    $C' :=$ standardize-apart(C)
    $\theta' :=$ unify(head(C'), $Q[1]\theta$)
    if $\theta' \neq$ failure
        $\Theta := \Theta \cup$ entails-bc-goals($\mathcal{C}$, atoms(body(C')) $\cup (Q \setminus \{C\}), \theta \cup \theta'$)
    fi
od
return Θ

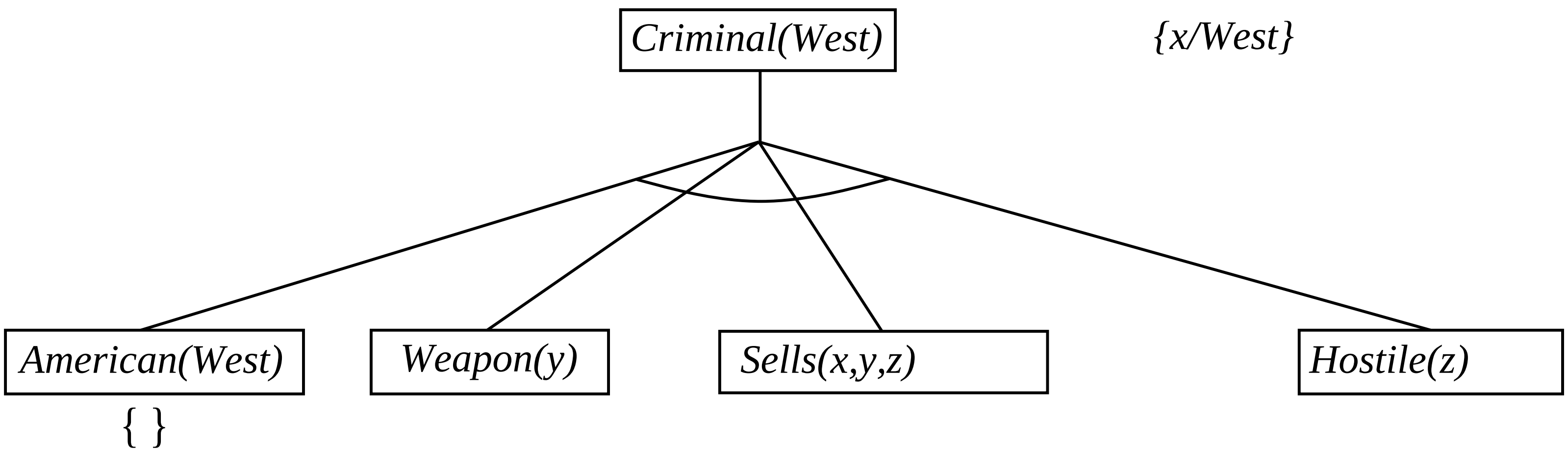
Backward Chaining / Example

$\boxed{Criminal(West)}$

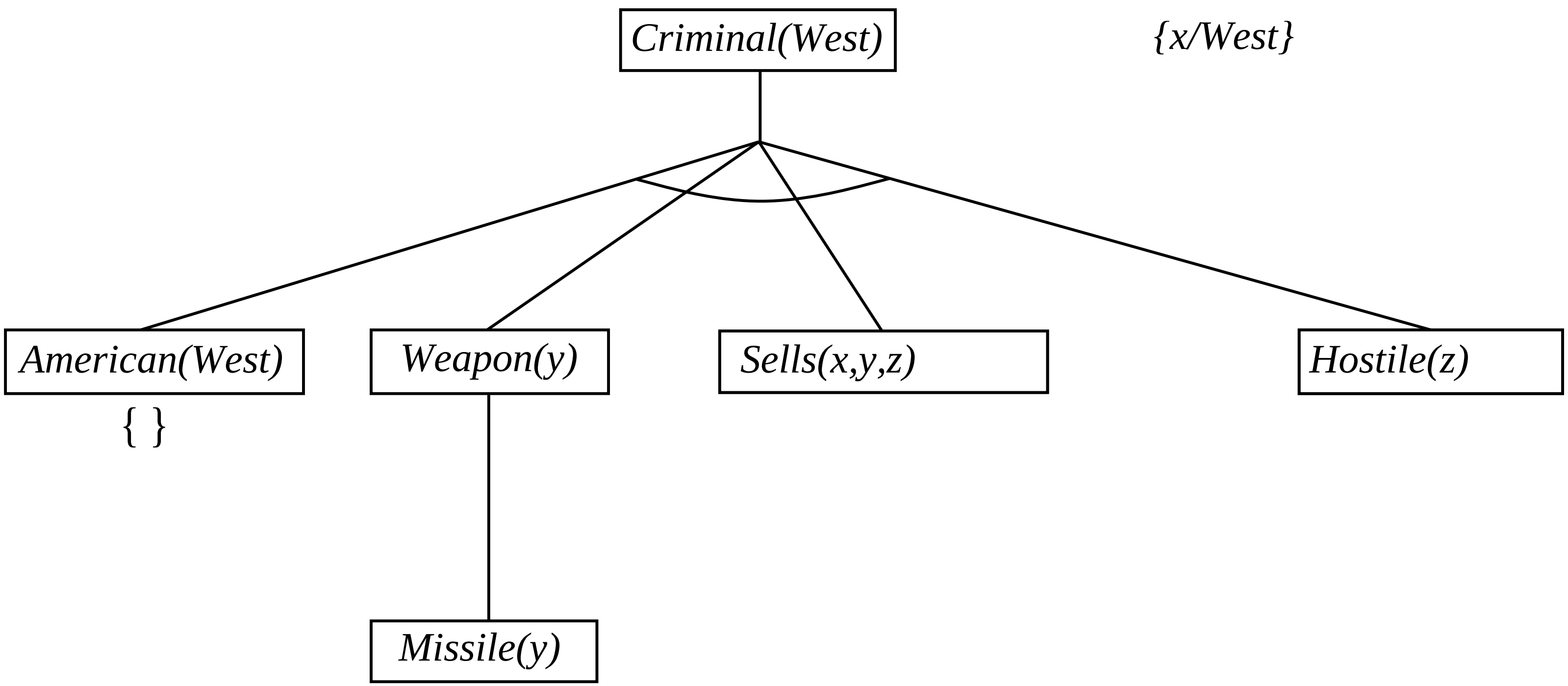
Backward Chaining / Example



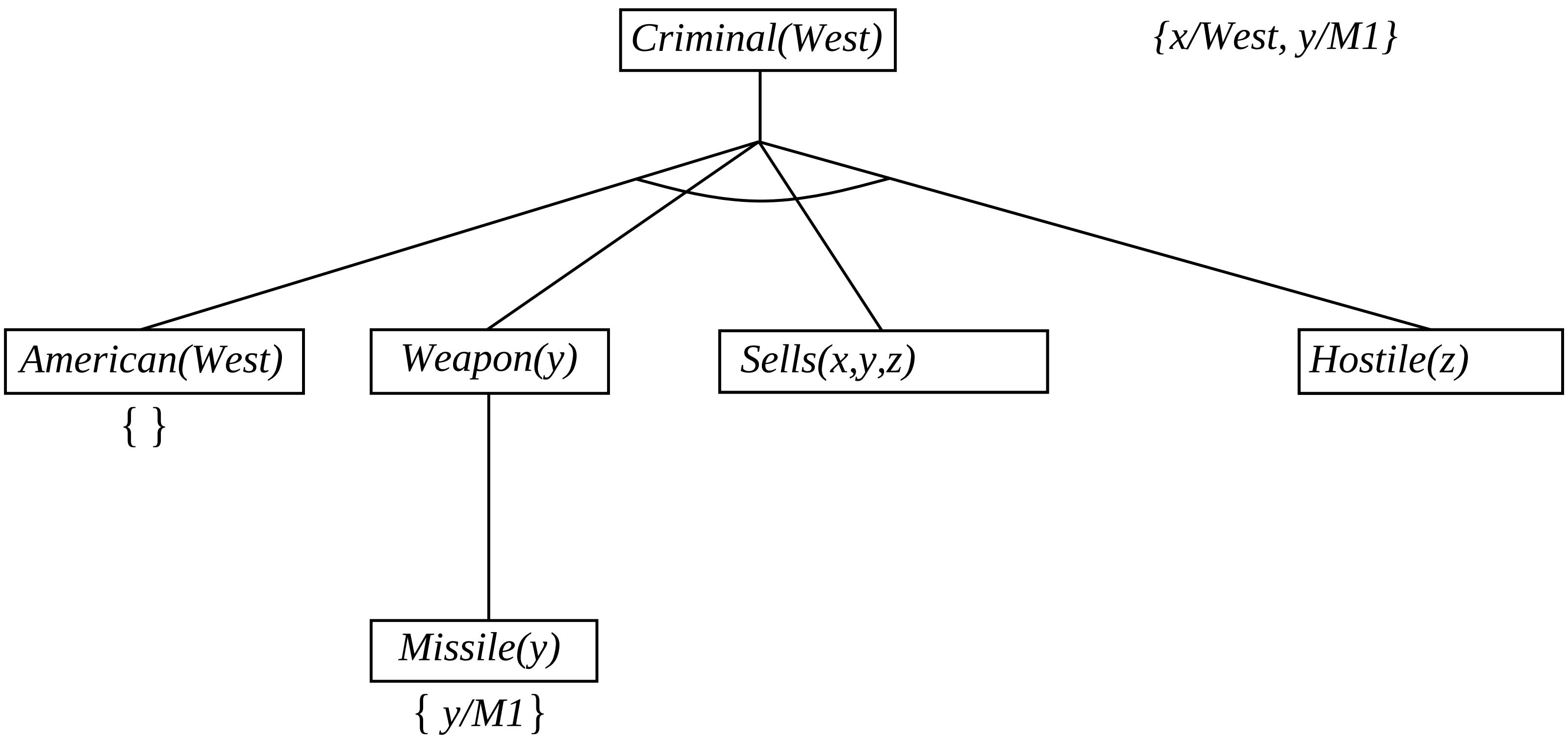
Backward Chaining / Example



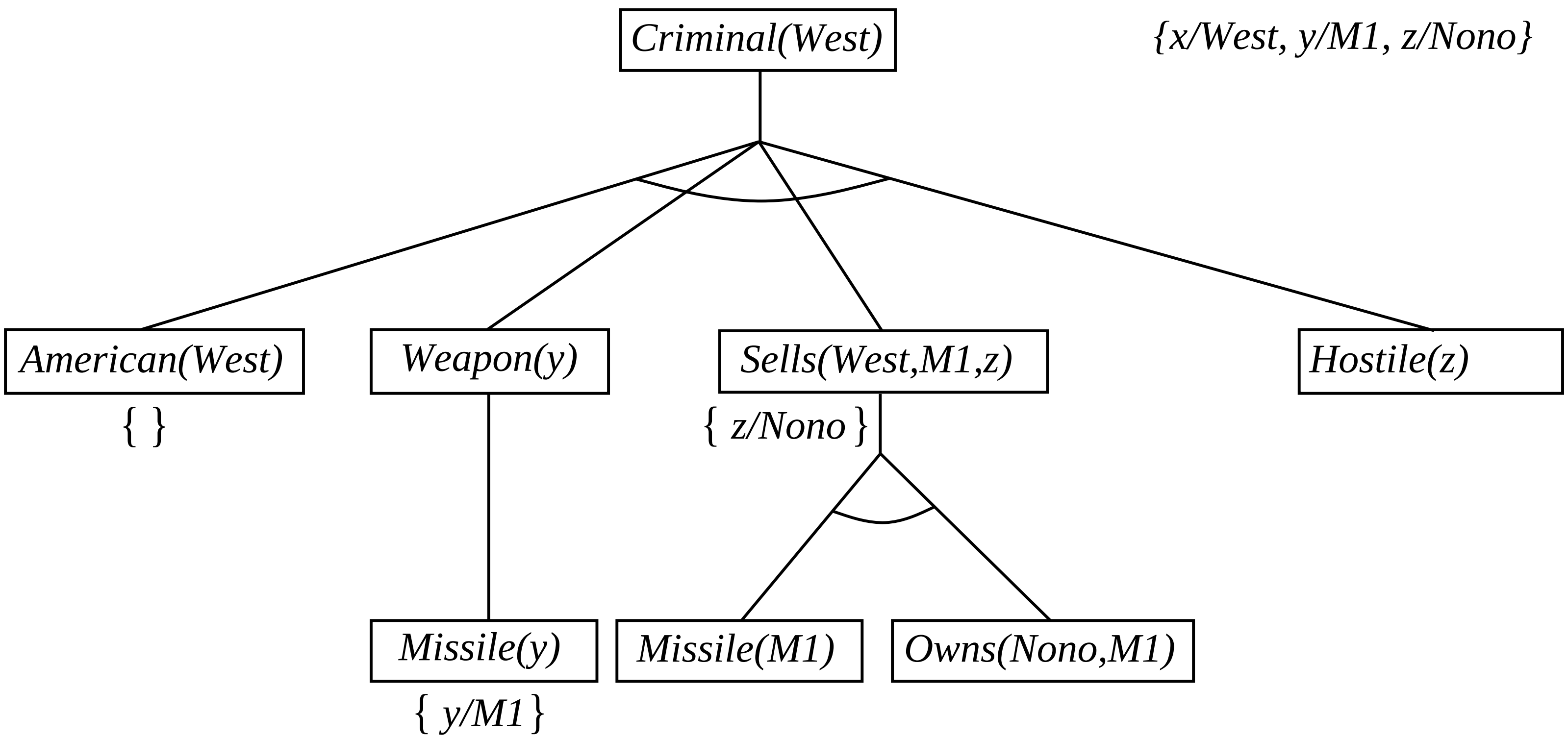
Backward Chaining / Example



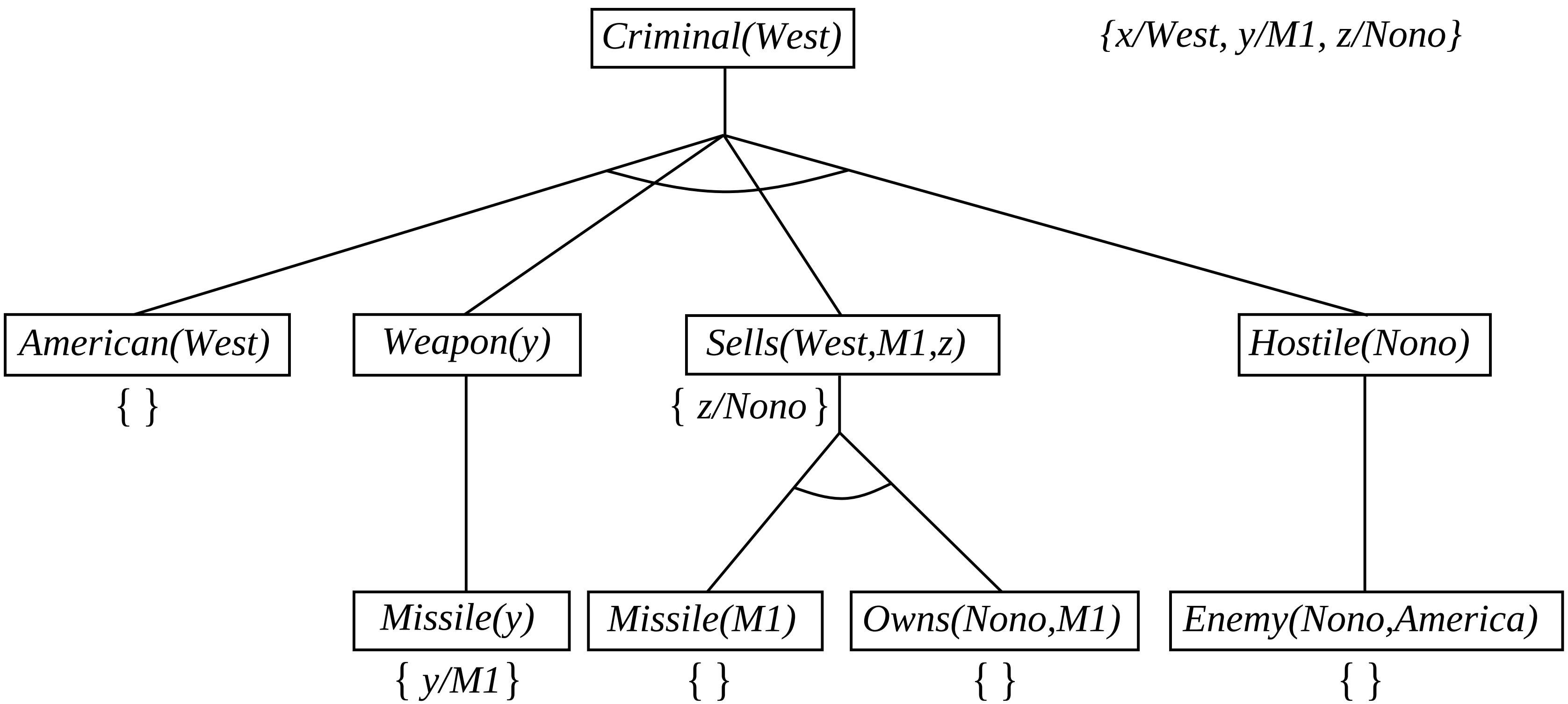
Backward Chaining / Example



Backward Chaining / Example



Backward Chaining / Example



Logic Programming: Prolog

Prolog: logical programming language (PROgrammation en LOGique; Alain Colmerauer and Philippe Roussel, ca. 1972)

Allows knowlegde bases (= programs) consisting of definite Horn clauses.

Uses depth-first, left-to-right backward chaining (with several improvements).

Example:

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

Negation as Failure

Prolog allows the usage of negated atoms in rule bodies interpreting them by **negation as failure**:

```
good(X) :- not evil(X)
```

Now the query `?- good(richard)` would evaluate to true as the opposite, `evil(richard)` cannot be proved.

This is also called **closed world assumption**:
if a fact is not encoded in the knowledge base
and cannot be inferred,
then it is considered not to be true.

Negation as failure renders Prolog **non-monotonic**:
if one adds formulas to the knowledge base, inferences may become untrue.

Example: add `evil(richard)` to the knowledge base,
now the query `?- good(richard)` evaluates to false.

In first order logics we could not derive any conclusions about `good(richard)`.

Prolog / Examples

Appending two lists to produce a third:
append(X,Y,Z) encodes that X appended to Y results in Z.

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

query:   append([1,2], [3,4], C) ?
answers: C=[1,2,3,4]

query:   append(A, B, [1,2]) ?
answers: A=[]    B=[1,2]
         A=[1]   B=[2]
         A=[1,2] B=[]
```

1. **Unification**

2. **Forward Chaining**

3. **Backward Chaining**

4. **Resolution**

FOL Resolvents

Full first-order version:

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

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

For example,

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

with $\ell_i = \neg\text{Rich}(x)$, $m_j = \text{Rich}(\text{Ken})$ and $\theta = \{x/\text{Ken}\}$

Apply resolution steps to $\text{CNF}(\text{KB} \wedge \neg\text{query})$; complete for FOL.

Conversion to CNF

Everyone who loves all animals is loved by someone: $\forall x[\forall y \text{Animal}(y) \implies \text{Loves}(x,y)] \implies [\exists y \text{Loves}(y,x)]$

1. Eliminate biconditionals and implications

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

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

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

Conversion to CNF

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

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

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

$$\forall x[\text{Animal}(F(x)) \wedge \neg\text{Loves}(x, F(x))] \vee \text{Loves}(G(x), x)$$

5. Drop universal quantifiers:

$$[\text{Animal}(F(x)) \wedge \neg\text{Loves}(x, F(x))] \vee \text{Loves}(G(x), x)$$

6. Distribute $\wedge$ over $\vee$:

$$[\text{Animal}(F(x)) \vee \text{Loves}(G(x), x)] \wedge [\neg\text{Loves}(x, F(x)) \vee \text{Loves}(G(x), x)]$$

Resolution / Example