ARTIFICIAL INTELLIGENCE

2023/2024 Semester 2

Inference in first-order logic:

Chapter 9

Logic as a representation of the World

entails

Representation: Sentences

Sentence

Refers to
(Semantics)

World

Facts

follows

Fact

Inference in FOL

- All rules of inference for propositional logic apply to first-order logic

- We just need to reduce FOL sentences to PL sentences by instantiating variables and removing quantifiers

Remember: propositiona logic

◇ **Modus Ponens** or **Implication-Elimination**: (From an implication and the premise of the implication, you can infer the conclusion.)

$$\frac{\alpha \Rightarrow \beta, \qquad \alpha}{\beta}$$

◇ **And-Elimination**: (From a conjunction, you can infer any of the conjuncts.)

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

◇ **And-Introduction**: (From a list of sentences, you can infer their conjunction.)

$$\frac{\alpha_1, \ \alpha_2, \quad \ldots, \quad \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$

◇ **Or-Introduction**: (From a sentence, you can infer its disjunction with anything else at all.)

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

◇ **Double-Negation Elimination**: (From a doubly negated sentence, you can infer a positive sentence.)

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

◇ **Unit Resolution**: (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

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

◇ **Resolution**: (This is the most difficult. Because β cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.)

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

Reminder

- Ground term: A term that does not contain a variable.
  - A constant symbol
  - A function applies to some ground term
- {x/a}: substitution/binding list

Outline

- Propositional vs. First-Order Inference
- Unification
- Forward chaining
- Backward chaining
- Resolution

Propositional vs. First-Order Inference

- Inference rules for quantifiers
- Reduction to propositional inference

Universal instantiation (UI)

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

$$\frac{\forall v \quad \alpha}{\mathrm{Subst}(\{v/g\}, \alpha)}$$

for any variable v and ground term g

- E.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ 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)

- For any sentence α, variable *v*, and constant symbol *k* that does not appear elsewhere in the knowledge base:

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

- E.g., $\exists x\ Crown(x) \wedge OnHead(x, John)$ yields:

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

provided C_1 is a new constant symbol, called a Skolem constant

Reduction to propositional inference

- Suppose the KB contains just the following:
  $\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
  King(John)
  Greedy(John) Brother(Richard,John)
- How can we reduce this to PL
-
- Let's Instantiate the universal sentence in all possible ways:
  King(John) $\wedge$ Greedy(John) $\Rightarrow$ Evil(John)
  King(Richard) $\wedge$ Greedy(Richard) $\Rightarrow$ Evil(Richard)
  King(John)
  Greedy(John)
  Brother(Richard,John)

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

Reduction contd.

- Every FOL KB can be propositionalized so as to preserve entailment
  - (A ground sentence is entailed by new KB iff entailed by original KB)
- **Idea**: propositionalize KB and query, apply resolution, return result
- **Problem**: with function symbols, there are infinitely many ground terms
  - e.g., Father(X) yields Father(John)
  - Father(Father(John))
  - Father(Father(Father(John))),etc

Reduction contd.

Theorem: Herbrand (1930). If a sentence α is entailed by an FOL KB, it is entailed by a finite subset of the propositionalized KB

Idea: For $n = 0$ to ∞ do

- create a propositional KB by instantiating with depth-n terms
- see if α is entailed by this KB

Problem: works if α is entailed, loops if α is not entailed

Theorem: Turing (1936), Church (1936)

Entailment for FOL is semidecidable (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.)

Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.

- E.g., from:
  $\forall x\ King(x) \land Greedy(x) \Rightarrow Evil(x)$
  $King(John)$
  $\forall y\ Greedy(y)$
  $Brother(Richard,John)$

- it seems obvious that *Evil*(*John*), but propositionalization produces lots of facts such as *Greedy*(*Richard*) that are irrelevant

- With p k-ary predicates and n constants, there are $p \cdot n^k$ instantiations.

Inference in FOL

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

- Can we prove this without full propositionalization as an intermediate step?

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*
- θ = {x/John, y/John} works

$$\text{Unify}(\alpha,\beta) = \theta \text{ if } \alpha\theta = \beta\theta \quad (\text{Subst}(\theta, \alpha) = \text{Subst}(\theta, \beta))$$

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

Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*
- θ = {x/John,y/John} works

  Unify(α,β) = θ if αθ = βθ

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

Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*
- θ = {x/John,y/John} works

  Unify(α,β) = θ if αθ = βθ

p	q	θ
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)}
Knows(John,x)	Knows(x,OJ)	{fail}

Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

Unification

Extra example for unification:

P	Q	σ
Student(x)	Student(Bob)	{x/Bob}
Sells(Bob, x)	Sells(x, coke)	{x/coke, x/Bob} Is it correct?

Unification

Extra example for unification

P	Q	σ
Student(x)	Student(Bob)	{x/Bob}
Sells(Bob, x)	Sells(y, coke)	{x/coke, y/Bob}

Unification

More Unification Examples

VARIABLE term

1 – unify(P(a,X), P(a,b)) $\sigma = \{X/b\}$

2 – unify(P(a,X), P(Y,b)) $\sigma = \{Y/a, X/b\}$

3 – unify(P(a,X), P(Y,f(a)) $\sigma = \{Y/a, X/f(a)\}$

4 – unify(P(a,X), P(X,b)) σ = failure

Note: If P(a,X) and P(X,b) are independent, then we can replace X with Y and get the unification to work.

Unification

- To unify *Knows(John,x)* and *Knows(y,z)*,
  θ = {y/John, x/z } or θ = {y/John, x/John, z/John}

- The first unifier is more general than the second.

- There is a single most general unifier (MGU) that is unique up to renaming of variables.

MGU = { y/John, x/z }

Unification

$$P[z, f(w), B]$$
$$P[x, f(A), B]$$
$$P[g(z), f(A), B]$$
$$P[C, f(A), B]$$

$$\Leftarrow P[x, f(y), B]$$

$$s_1 = \{x / z, y / w\}$$
$$s_2 = \{y / A\}$$
$$s_3 = \{x / g(z), y / A\}$$
$$s_4 = \{x / C, y / A\}$$

Unification

- $(\omega s1)s2 = \omega(s1s2), \quad (s1s2)s3 = s1(s2s3)$
  - Let *w* be P(x, y), s1 be {x/f(y)}, and s2 be {y/A} then,

$$(\omega s1)s2 = [P(f(y), y)]\{y/\mathrm{A}\} = P(f(\mathrm{A}), \mathrm{A}) \qquad and$$

$$\omega(s1s2) = [P(x, y)]\{x/ f(\mathrm{A}), y/\mathrm{A}\} = P(f(\mathrm{A}), \mathrm{A})$$

  - Substitutions are not, in general, commutative

$$\omega(s1s2) = P(f(\mathrm{A}), \mathrm{A})$$

$$\omega(s2s1) = [P(x, y)]\{y/\mathrm{A}, x/ f(\mathrm{y})\} = P(f(\mathrm{y}), \mathrm{A})$$

- Unifiable: a set of $\{\omega_i\}$ expressions is unifiable if there exists a substitution s such that $\omega_1 s = \omega_2 s = \omega_3 s = \cdots$
  - $s = \{\mathrm{x}/A, \mathrm{y}/B\}$ *unifies* $\{P[x, f(y), B], P[x, f(B), B]\}$, to yield

$$\{P[\mathrm{A}, f(B), B]\}$$

The 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

The 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
```

Generalized Modus Ponens (GMP)

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

p_1' is *King*(*John*) ; p_2' is *Greedy*(*y*)

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

Substitution θ is {x/John, y/John}

Subst(θ ,q) is *Evil*(*John*)

GMP used with KB of definite clauses (exactly one positive literal)

- All variables assumed universally quantified

Soundness of GMP

- Need to show that

$$p_1', \ldots, p_n', (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models \text{Subst}(\theta, q)$$

provided that $\text{Subst}(\theta, p_i') = \text{Subst}(\theta, p_i)$ for all i

- Lemma: For any sentence p, we have $p \models \text{Subst}(\theta, p)$ by UI

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models$
   $\text{Subst}(\theta, p_1 \wedge \ldots \wedge p_n \Rightarrow q) = (\text{Subst}(\theta, p_1) \wedge \ldots \wedge \text{Subst}(\theta, p_n)$
   $\Rightarrow \text{Subst}(\theta, q))$

2. $p_1', \ldots, p_n' \models p_1' \wedge \ldots \wedge p_n' \models \text{Subst}(\theta, p_1') \wedge \ldots \wedge \text{Subst}(\theta, p_n')$

3. From 1 and 2, $\text{Subst}(\theta, q)$ follows by ordinary Modus Ponens

Inference with GMP

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

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

Forward Chaining

- Remember this from propositional logic?
  - Start with atomic sentences in KB
  - Apply Modus Ponens
    - add new sentences to KB
    - discontinue when no new sentences
  - Hopefully find the sentence you are looking for in the generated sentences

Lifting forward chaining

- First-order definite clauses
  - all sentences are defined this way to simplify processing
    - disjunction of literals with exactly one positive
    - clause is either atomic or an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal

$$King(x) \land Greedy(x) \Rightarrow Evil(x)$$
$$King(John) .$$
$$Greedy(y) .$$

Example knowledge base

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

- Prove that Col. West is a criminal

Example knowledge base contd.

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

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

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

$Owns(Nono,M_1)$ and $Missile(M_1)$

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

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

Missiles are weapons:

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

An enemy of America counts as "hostile“:

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

West, who is American …

$American(West)$

The country Nono, an enemy of America …

$Enemy(Nono,America)$

Forward-chaining

- Starting from the facts
  - find all rules with satisfied premises
  - add their conclusions to known facts
  - repeat until
    - query is answered
    - no new facts are added

Forward chaining algorithm

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

  repeat until *new* is empty
    $new \leftarrow \{\}$
    for each sentence r **in** KB **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
      for each θ such that $\text{SUBST}(\theta, p_1 \wedge \ldots \wedge p_n) = \text{SUBST}(\theta, p'_1 \wedge \ldots \wedge p'_n)$
          for some $p'_1, \ldots, p'_n$ in KB
        $q' \leftarrow \text{SUBST}(\theta, q)$
        if q' is not a renaming of some sentence already in KB or *new* **then do**
          add q' to *new*
          $\phi \leftarrow \text{UNIFY}(q', a)$
          if ϕ is not *fail* **then return** ϕ
    add *new* to KB
  return *false*

First iteration of forward chaining

- Look at the implication sentences first

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

  - must satisfy unknown premises
  - We can satisfy this rule

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

    - by substituting {x/M1}
    - and adding Sells(West, M1, Nono) to KB

First iteration of forward chaining

- We can satisfy

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

  - with {x/M1}
  - and Weapon (M1) is added
- We can satisfy

  - w $Enemy(x, America) \Rightarrow Hostile(x)$
  - and Hostile {Nono} is added

Second iteration of forward chaining

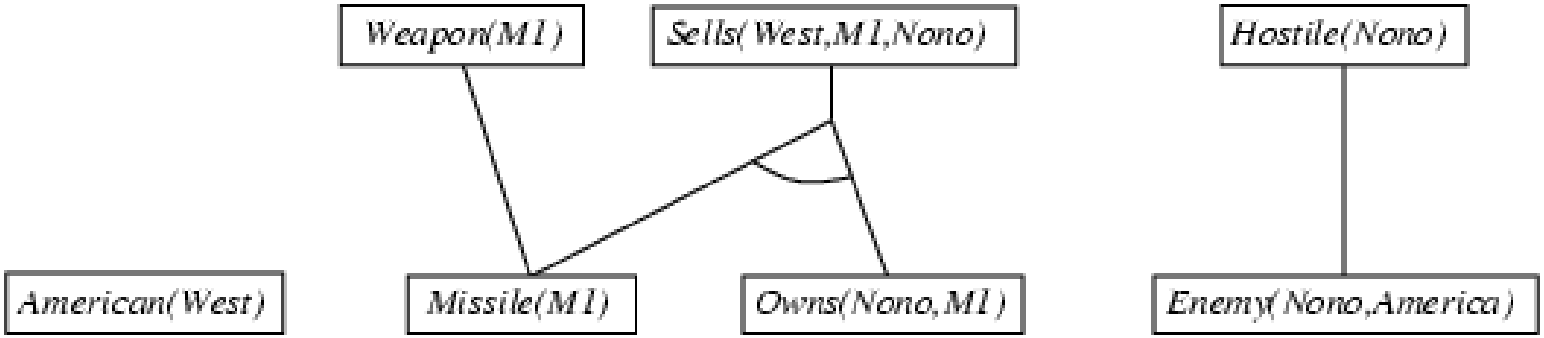
- We can satisfy

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

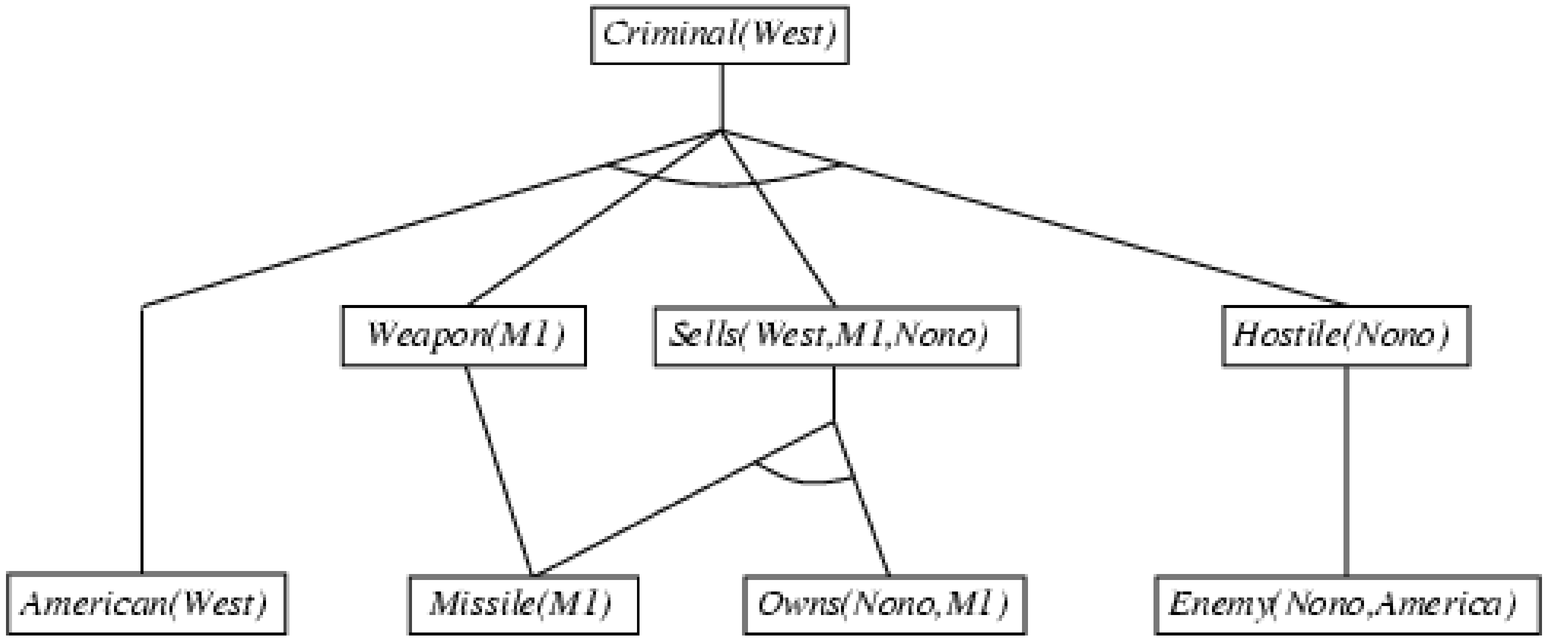
  - with {x/West, y/M1, z/Nono}
  - and Criminal (West) is added

Forward chaining proof

Forward chaining proof



Forward chaining proof



Another Example (from Konelsky)

- Nintendo example.
  - Nintendo says it is Criminal for a programmer to provide emulators to people. My friends don’t have a Nintendo 64, but they use software that runs N64 games on their PC, which is written by Reality Man, who is a programmer.

Forward Chaining

- The knowledge base initially contains:
  - Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x, z, y) $\Rightarrow$ Criminal(x)
  - Use(friends, x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Provide(Reality Man, friends, x)
  - Software(x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Emulator(x)

Forward Chaining

$$Programmer(x) \wedge Emulator(y) \wedge People(z) \wedge Provide(x,z,y) \Rightarrow Criminal(x) \quad (1)$$

$$Use(friends, x) \wedge Runs(x, N64\ games) \Rightarrow Provide(Reality\ Man, friends, x) \quad (2)$$

$$Software(x) \wedge Runs(x, N64\ games) \Rightarrow Emulator(x) \quad (3)$$

- Now we add atomic sentences to the KB sequentially, and call on the forward-chaining procedure:

Forward Chaining

$\text{Programmer}(x) \wedge \text{Emulator}(y) \wedge \text{People}(z) \wedge \text{Provide}(x,z,y)$
$\Rightarrow \text{Criminal}(x)$ (1)

$\text{Use}(\text{friends}, x) \wedge \text{Runs}(x, \text{N64 games})$
$\Rightarrow \text{Provide}(\text{Reality Man}, \text{friends}, x)$ (2)

$\text{Software}(x) \wedge \text{Runs}(x, \text{N64 games})$
$\Rightarrow \text{Emulator}(x)$ (3)

Programmer(Reality Man) (4)

- This new premise unifies with (1) with **subst(**{x/Reality Man}, Programmer(x)**)** but not all the premises of (1) are yet known, so nothing further happens.

Forward Chaining

Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y)
$\Rightarrow$ Criminal(x) (1)

Use(friends, x) $\wedge$ Runs(x, N64 games)

$\Rightarrow$ Provide(Reality Man, friends, x) (2)

Software(x) $\wedge$ Runs(x, N64 games)

$\Rightarrow$ Emulator(x) (3)

Programmer(Reality Man) (4)

- Continue adding atomic sentences: People(friends)

Forward Chaining

$\text{Programmer}(x) \wedge \text{Emulator}(y) \wedge \text{People}(z) \wedge \text{Provide}(x,z,y) \Rightarrow \text{Criminal}(x)$ (1)

$\text{Use}(\text{friends}, x) \wedge \text{Runs}(x, \text{N64 games})$

$\Rightarrow \text{Provide}(\text{Reality Man}, \text{friends}, x)$ (2)

$\text{Software}(x) \wedge \text{Runs}(x, \text{N64 games})$

$\Rightarrow \text{Emulator}(x)$ (3)

Programmer(Reality Man) (4)

People(friends) (5)

- This also unifies with (1) with **subst(**{z/friends}, People(z)**)** but other premises are still missing.

Forward Chaining

Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y)
$\Rightarrow$ Criminal(x) (1)

Use(friends, x) $\wedge$ Runs(x, N64 games)

$\Rightarrow$ Provide(Reality Man, friends, x) (2)

Software(x) $\wedge$ Runs(x, N64 games)

$\Rightarrow$ Emulator(x) (3)

Programmer(Reality Man) (4)

People(friends) (5)

- Add:
  - Software(U64)

Forward Chaining

Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y)
$\Rightarrow$ Criminal(x) (1)

Use(friends, x) $\wedge$ Runs(x, N64 games)
$\Rightarrow$ Provide(Reality Man, friends, x) (2)

Software(x) $\wedge$ Runs(x, N64 games)
$\Rightarrow$ Emulator(x) (3)

Programmer(Reality Man) (4)

People(friends) (5)

Software(U64) (6)

- This new premise unifies with (3) but the other premise is not yet known.

Forward Chaining

Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y)
$\Rightarrow$ Criminal(x) (1)

Use(friends, x) $\wedge$ Runs(x, N64 games)
$\Rightarrow$ Provide(Reality Man, friends, x) (2)

<u>Software(x)</u> $\wedge$ Runs(x, N64 games)
$\Rightarrow$ Emulator(x) (3)

Programmer(Reality Man) (4)

People(friends) (5)

Software(U64) (6)

- Add:
  - Use(friends, U64)

Forward Chaining

$\text{Programmer}(x) \wedge \text{Emulator}(y) \wedge \text{People}(z) \wedge \text{Provide}(x,z,y) \Rightarrow \text{Criminal}(x)$ (1)

$\text{Use}(\text{friends}, x) \wedge \text{Runs}(x, \text{N64 games}) \Rightarrow \text{Provide}(\text{Reality Man}, \text{friends}, x)$ (2)

$\text{Software}(x) \wedge \text{Runs}(x, \text{N64 games}) \Rightarrow \text{Emulator}(x)$ (3)

Programmer(Reality Man) (4)

People(friends) (5)

Software(U64) (6)

Use(friends, U64) (7)

- This premise unifies with (2) but one still lacks.

Forward Chaining

Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y) $\Rightarrow$ Criminal(x) (1)

Use(friends, x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Provide(Reality Man, friends, x) (2)

Software(x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Emulator(x) (3)

Programmer(Reality Man) (4)

People(friends) (5)

Software(U64) (6)

Use(friends, U64) (7)

- Add:
  - Runs(U64, N64 games)

Forward Chaining

Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y) $\Rightarrow$ Criminal(x) (1)
Use(friends, x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Provide(Reality Man, friends, x) (2)
Software(x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Emulator(x) (3)

Programmer(Reality Man) (4)
People(friends) (5)
Software(U64) (6)
Use(friends, U64) (7)
Runs(U64, N64 games) (8)

- This new premise unifies with (2) and (3).

Forward Chaining

Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y) $\Rightarrow$ Criminal(x) (1)
Use(friends, x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Provide(Reality Man, friends, x) (2)
Software(x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Emulator(x) (3)

Programmer(Reality Man) (4)
People(friends) (5)
Software(U64) (6)
Use(friends, U64) (7)
Runs(U64, N64 games) (8)

- Premises (6), (7) and (8) satisfy the implications fully.

Forward Chaining

Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y) $\Rightarrow$ Criminal(x) (1)

Use(friends, x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ **Provide(Reality Man, friends, x)** (2)

Software(x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ **Emulator(x)** (3)

Programmer(Reality Man) (4)

People(friends) (5)

Software(U64) (6)

Use(friends, U64) (7)

Runs(U64, N64 games) (8)

- So we can infer the consequents, which are now added to the knowledge.

Forward Chaining

Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y)$\Rightarrow$ Criminal(x) (1)

Use(friends, x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ **Provide(Reality Man, friends, x)** (2)

Software(x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ **Emulator(x)** (3)

Programmer(Reality Man) (4)

People(friends) (5)

Software(U64) (6)

Use(friends, U64) (7)

Runs(U64, N64 games) (8)

Provide(Reality Man, friends, U64) (9)

Emulator(U64) (10)

- Addition of these new facts triggers further forward chaining.

Forward Chaining

$\text{Programmer}(x) \wedge \text{Emulator}(y) \wedge \text{People}(z) \wedge \text{Provide}(x,z,y) \Rightarrow \text{Criminal}(x)$ (1)

$\text{Use}(\text{friends}, x) \wedge \text{Runs}(x, \text{N64 games}) \Rightarrow$ **Provide(Reality Man, friends, x)** (2)

$\text{Software}(x) \wedge \text{Runs}(x, \text{N64 games}) \Rightarrow$ **Emulator(x)** (3)

Programmer(Reality Man) (4)

People(friends) (5)

Software(U64) (6)

Use(friends, U64) (7)

Runs(U64, N64 games) (8)

Provide(Reality Man, friends, U64) (9)

Emulator(U64) (10)

Criminal(Reality Man) (11)

- Which results in the final conclusion:

Criminal(Reality Man)

Forward Chaining

- Forward Chaining acts like a breadth-first search at the top level, with depth-first sub-searches.

- Since the search space spans the entire KB, a large KB must be organized in an intelligent manner in order to enable efficient searches in reasonable time.

Properties of forward chaining

- Sound and complete for first-order definite clauses
- Datalog = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations
- May not terminate in general if α is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable
- Forward chaining is widely used in deductive databases

Analyze this algorithm

- Sound?
  - Does it only derive sentences that are entailed?
  - Yes, because only Modus Ponens is used and it is sound
- Complete?
  - Does it answer every query whose answers are entailed by the KB?
  - Yes if the clauses are definite clauses

Proving completeness

- Assume KB only has sentences with no function symbols
  - What's the most number of iterations through algorithm?
  - Depends on the number of facts that can be added
    - Let *k* be the arity, the max number of arguments of any predicate and
    - Let *p* be the number of predicates
    - Let *n* be the number of constant symbols
  - At most pn^k distinct ground facts
  - Fixed point is reached after this many iterations
  - A proof by contradiction shows that the final KB is complete

Complexit of this algorithm

- Three sources of complexity
  - inner loop requires finding all unifiers such that premise of rule unifies with facts of database
    - this "pattern matching" is expensive
  - must check every rule on every iteration to check if its premises are satisfied
  - many facts are generated that are irrelevant to goal

Efficient forward chaining

- Matching rules against known facts
- Incremetal forward chaining
- Irrelevant facts

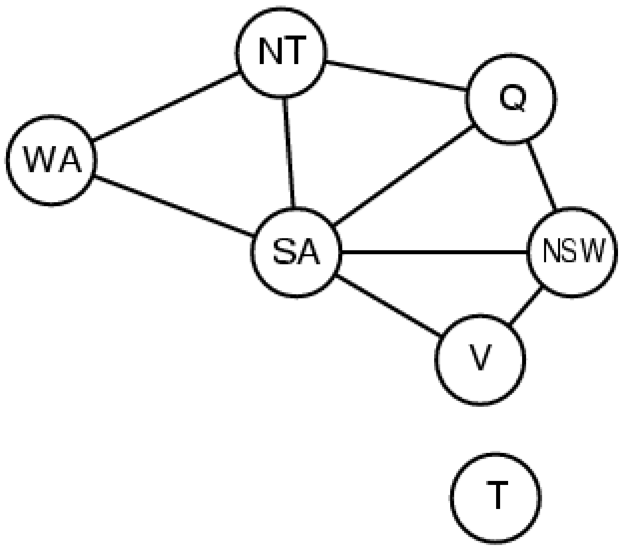
Matching rules against known facts

Pattern matching

- Conjunct ordering
  - Missile (x) ^ Owns (Nono, x) => Sells (West, x, Nono)
    - Look at all items owned by Nono, call them X
    - for each element x in X, check if it is a missile

    - Look for all missiles, call them X
    - for each element x in X, check if it is owned by Nono
- Optimal ordering is NP-hard, similar to matrix mult

Hard matching example



Diff(wa,nt) $\wedge$ *Diff(wa,sa)* $\wedge$ *Diff(nt,q)* $\wedge$
Diff(nt,sa) $\wedge$ *Diff(q,nsw)* $\wedge$ *Diff(q,sa)* $\wedge$
Diff(nsw,v) $\wedge$ *Diff(nsw,sa)* $\wedge$ *Diff(v,sa)*
$\Rightarrow$ *Colorable()*

Diff(Red,Blue)	*Diff (Red,Green)*
Diff(Green,Red)	*Diff(Green,Blue)*
Diff(Blue,Red)	*Diff(Blue,Green)*

- *Colorable*() is inferred iff the CSP has a solution
- Matching is NP-hard

Matching rules against known facts

- Matching rules against Known facts
  - We can remind ourselves that most rules in real-world knowledge bases are small and simple, conjunct ordering
  - We can consider subclasses of rules for which matching is efficient, most constrained variable
  - We can work hard to eliminate redundant rule matching attempts in the forward chaining algorithm, which is the subject of the next section

Incremental forward chaining

- Pointless (redundant) repetition
  - Some rules generate new information
    - this information may permit unification of existing rules
  - some rules generate preexisting information
    - we need not revisit the unification of the existing rules

- Every new fact inferred on iteration t must be derived from at least one new fact inferred on iteration t-1

$\Rightarrow$match each rule whose premise contains a newly added positive literal

Irrelevant facts

- Some facts are irrelevant and occupy computation of forward-chaining algorithm
  - What if Nono example included lots of facts about food preferences?
    - Not related to conclusions drawn about sale of weapons
    - How can we eliminate them?
      - Backward chaining is one way

Backward Chaining

- Start with the premises of the goal
  - Each premise must be supported by KB
  - Start with first premise and look for support from KB
    - looking for clauses with a head that matches premise
    - the head's premise must then be supported by KB
- A recursive, depth-first, algorithm
  - Suffers from repetition and incompleteness

Backward Chaining

- The algorithm:
  - a knowledge base KB
  - a desired conclusion c or question q
  - finds all sentences that are answers to q in KB *or* proves c
    - if q is directly provable by premises in KB, infer q and remember how q was inferred (building a list of answers).
    - find all implications that have q as a consequent.
    - for each of these implications, find out whether all of its premises are now in the KB, in which case infer the consequent and add it to the KB, remembering how it was inferred. If necessary, attempt to prove the implication also via backward chaining
    - premises that are conjuncts are processed one conjunct at a time

Backward chaining algorithm

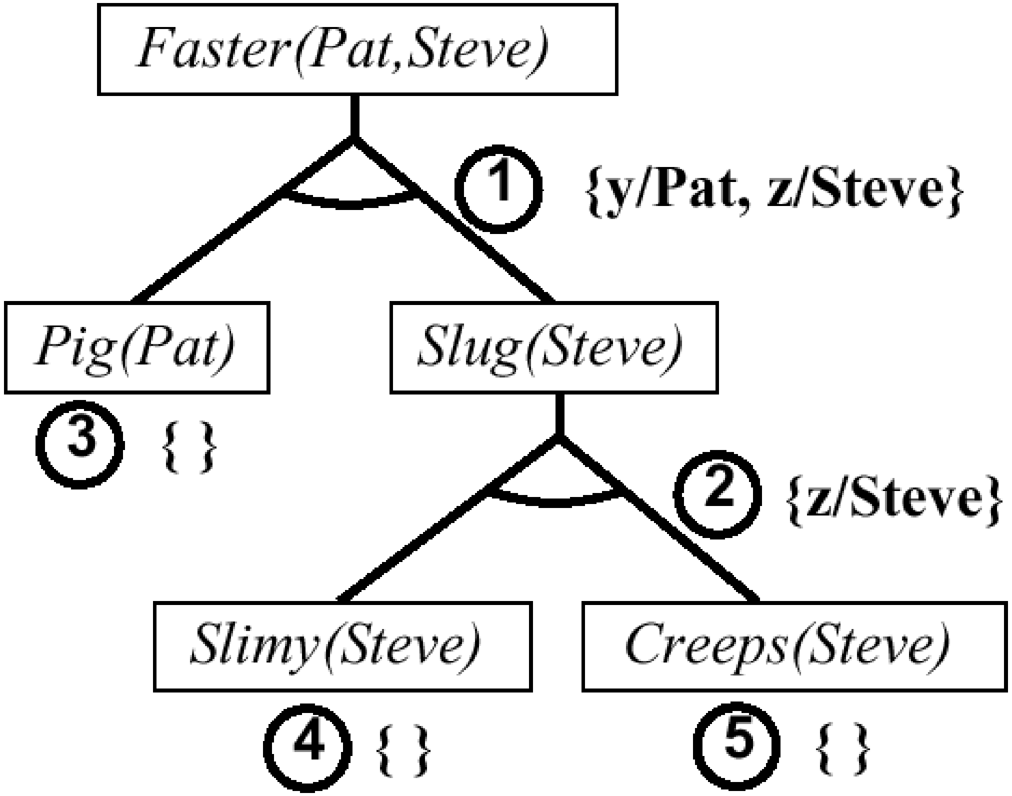
function FOL-BC-ASK(KB, goals, θ) **returns** a set of substitutions
  inputs: KB, a knowledge base
    $goals$, a list of conjuncts forming a query
    θ, the current substitution, initially the empty substitution { }
  local variables: ans, a set of substitutions, initially empty

  if $goals$ is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST(θ, FIRST($goals$))
  for each r **in** KB where STANDARDIZE-APART(r) = ($p_1 \wedge \ldots \wedge p_n \Rightarrow q$)
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
   $ans \leftarrow$ FOL-BC-ASK(KB, $[p_1, \ldots, p_n | \text{REST}(goals)]$, COMPOSE($\theta$, θ')) $\cup$ ans
  return ans

- SUBST(COMPOSE(θ_1, θ_2), p) = SUBST(θ_2, SUBST(θ_1, p))

Backward chaining example

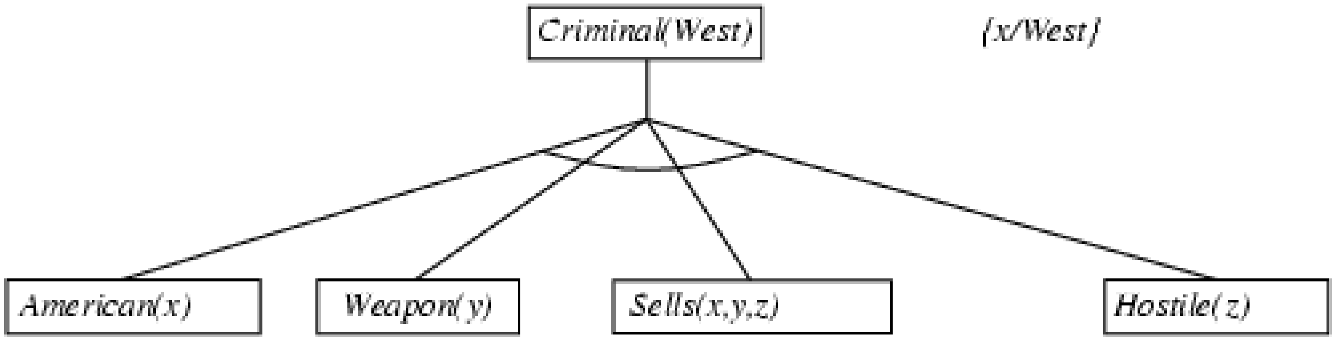
$\underline{1.}$ $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
$\underline{2.}$ $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
$\underline{3.}$ $Pig(Pat)$ $\quad$ $\underline{4.}$ $Slimy(Steve)$ $\quad$ $\underline{5.}$ $Creeps(Steve)$



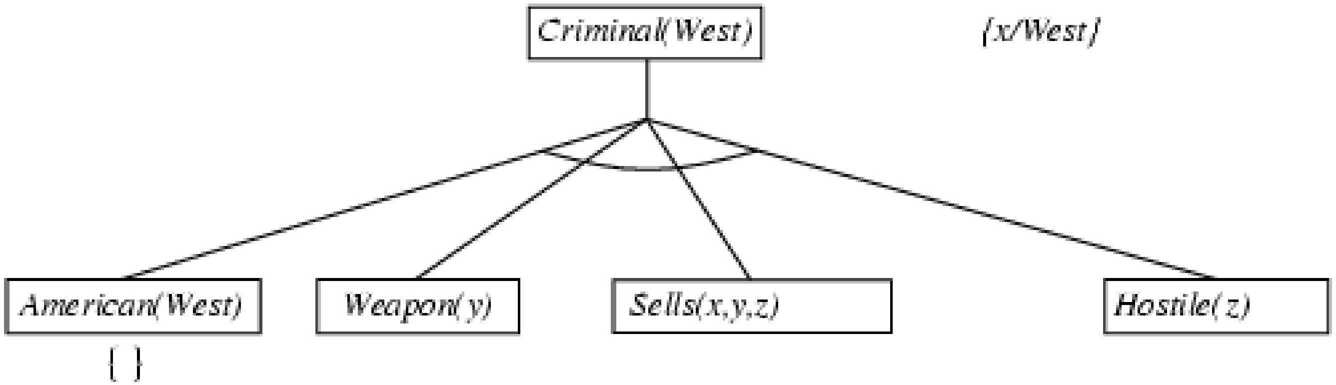
Backward chaining example

$\boxed{Criminal(West)}$

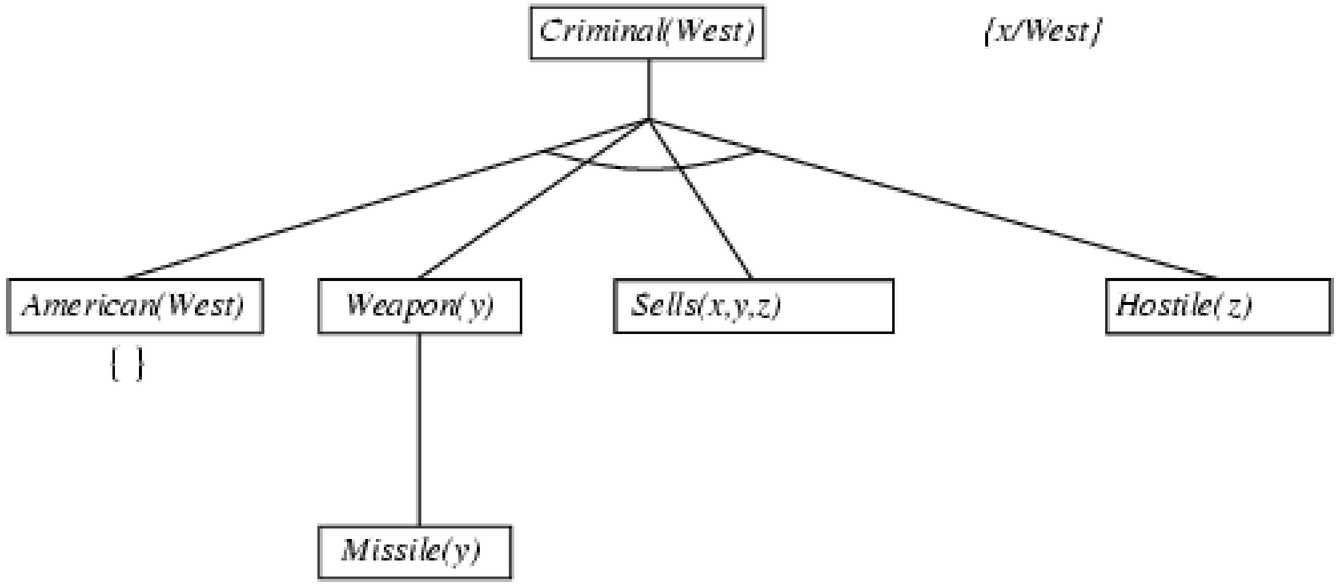
Backward chaining example



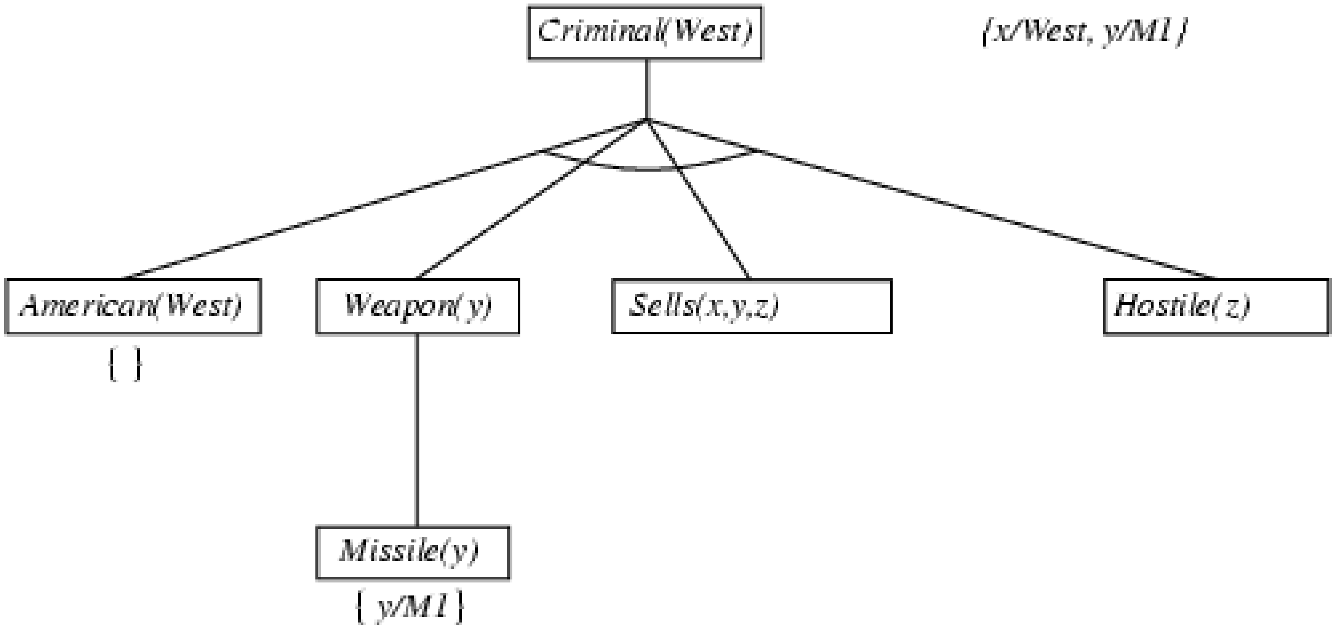
Backward chaining example



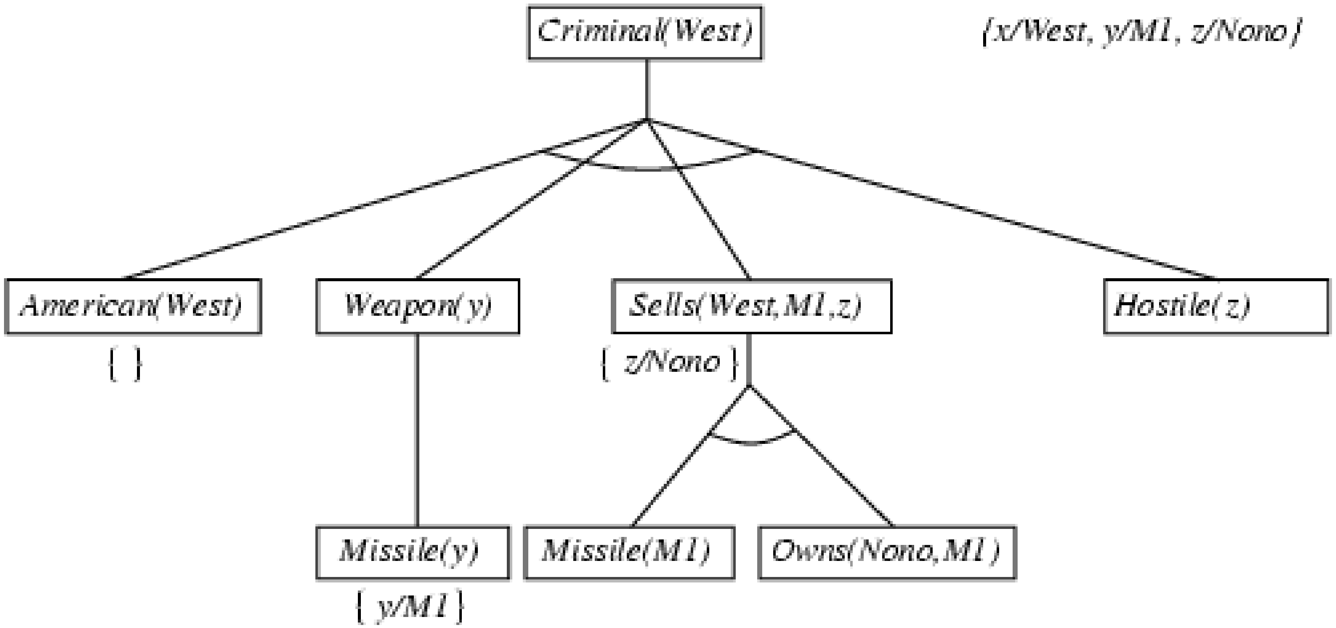
Backward chaining example



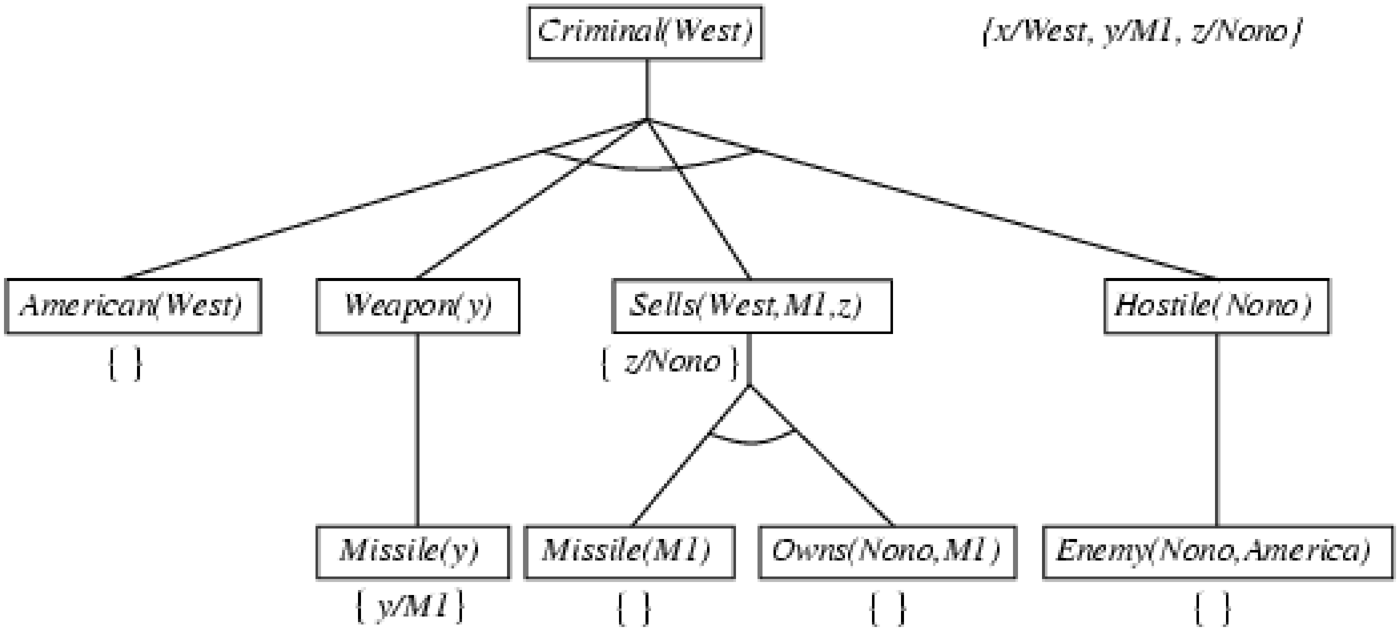
Backward chaining example



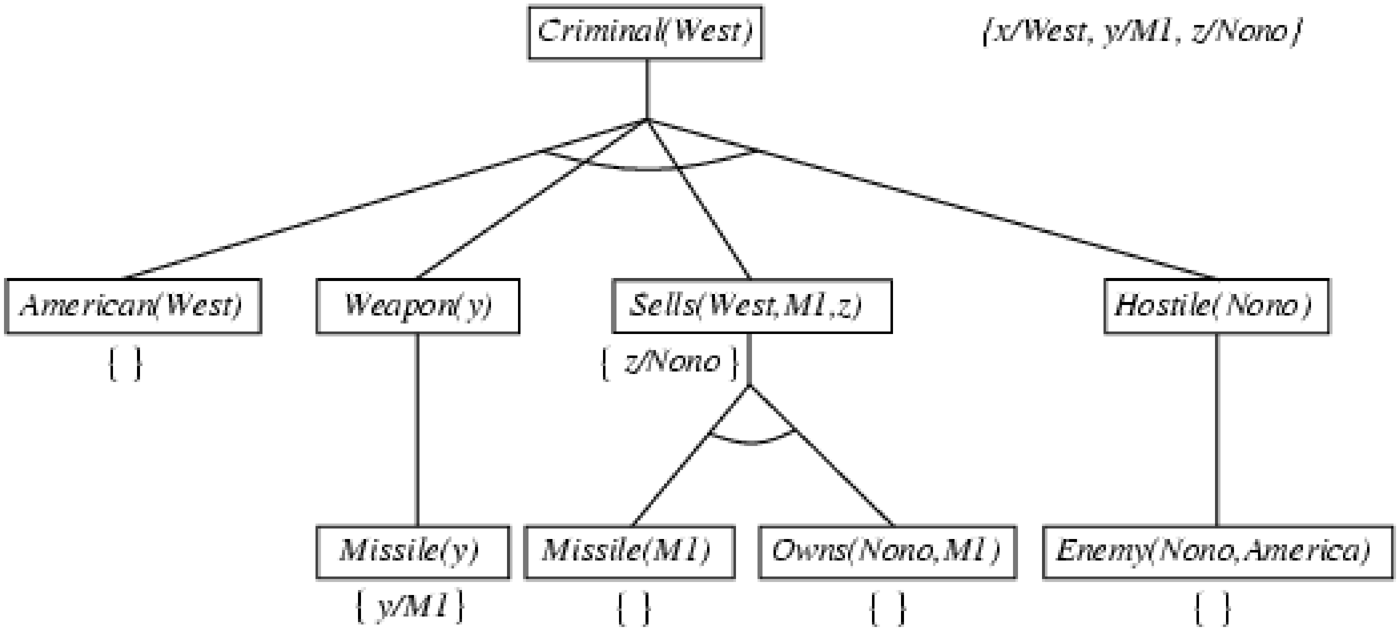
Backward chaining example



Backward chaining example



Backward chaining example



Backward Chaining Example

- Question: Has Reality Man done anything criminal?
- We will use the same knowledge as in our forward-chaining version of this example:

  Programmer(x) $\wedge$ Emulator(y) $\wedge$ People(z) $\wedge$ Provide(x,z,y) $\Rightarrow$ Criminal(x)

  Use(friends, x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Provide(Reality Man, friends, x)

  Software(x) $\wedge$ Runs(x, N64 games) $\Rightarrow$ Emulator(x)

  Programmer(Reality Man)

  People(friends)

  Software(U64)

  Use(friends, U64)

  Runs(U64, N64 games)

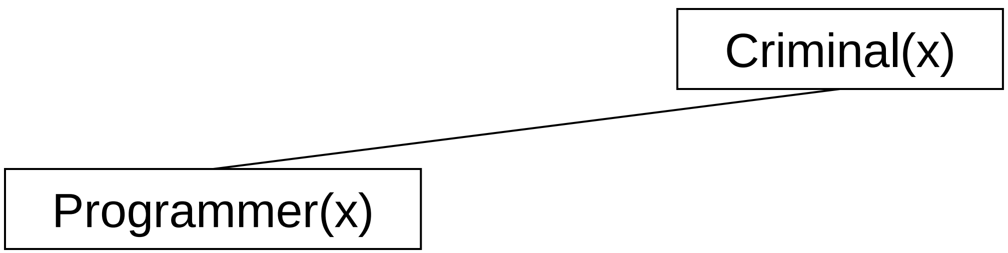
Backward Chaining Example

- Question: Has Reality Man done anything criminal?

$\boxed{\text{Criminal(x)}}$

Backward Chaining Example

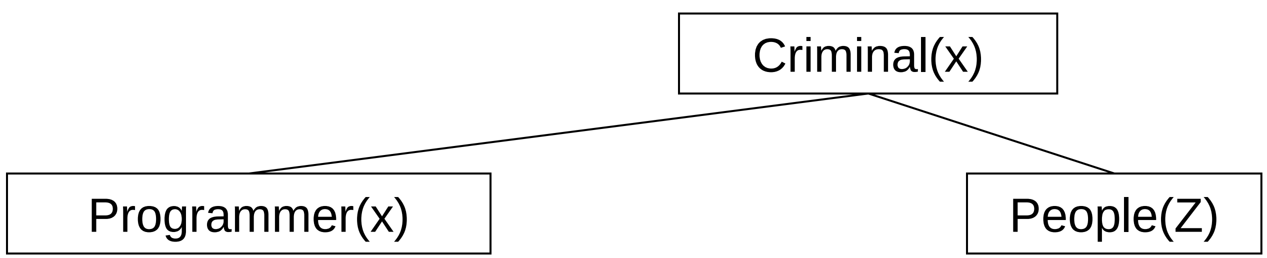
- Question: Has Reality Man done anything criminal?



Yes, {x/Reality Man}

Backward Chaining Example

- Question: Has Reality Man done anything criminal?

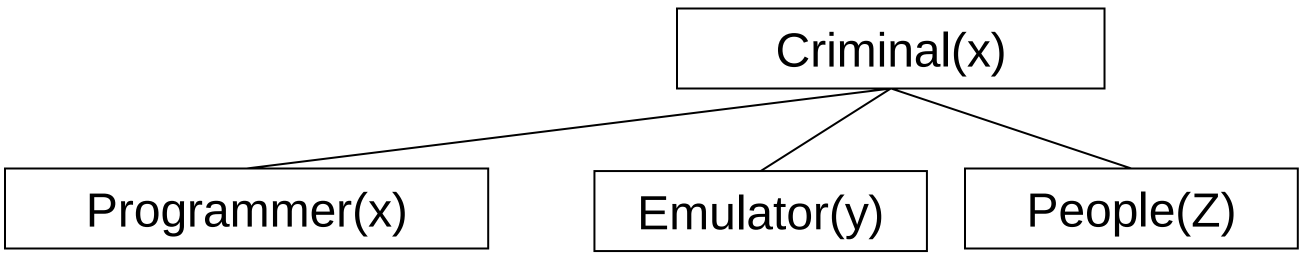


Yes, {x/Reality Man}

Yes, {z/friends}

Backward Chaining Example

- Question: Has Reality Man done anything criminal?

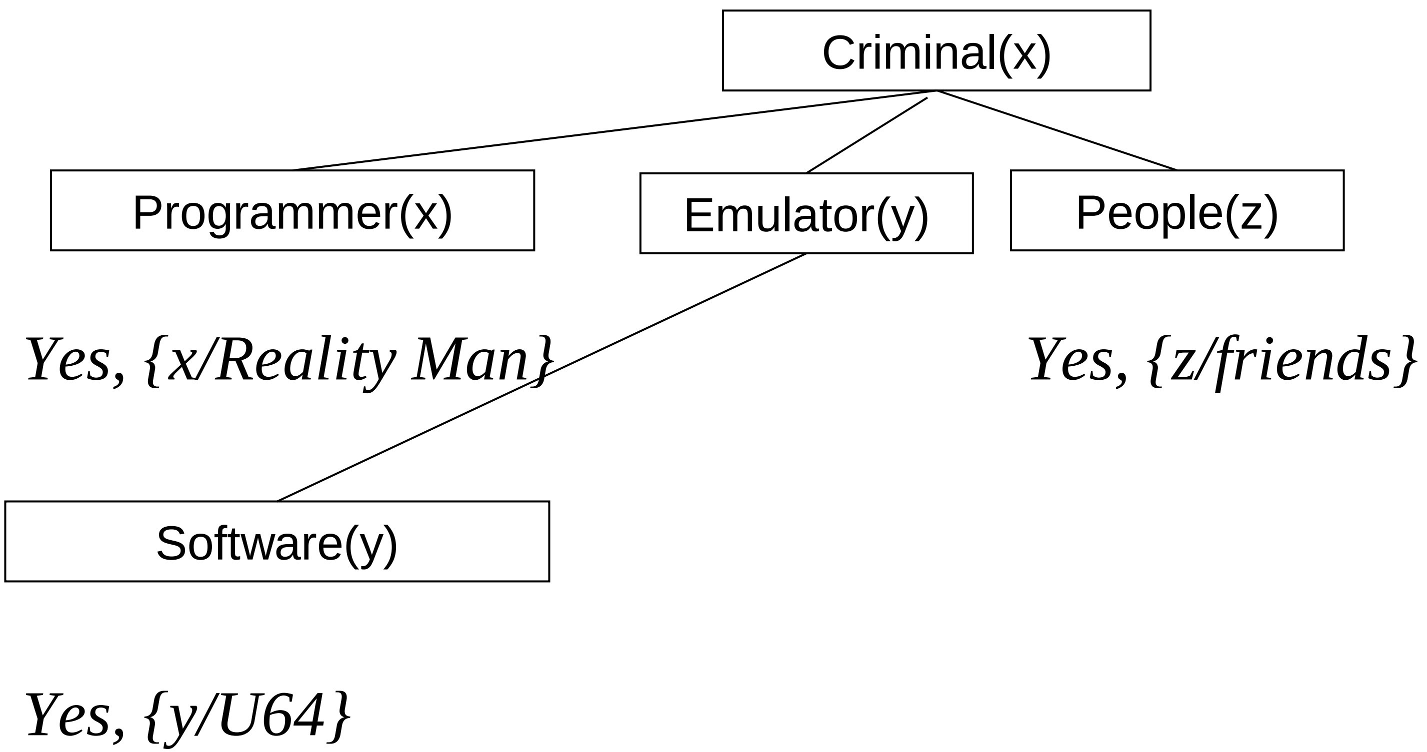


Yes, {x/Reality Man}

Yes, {z/friends}

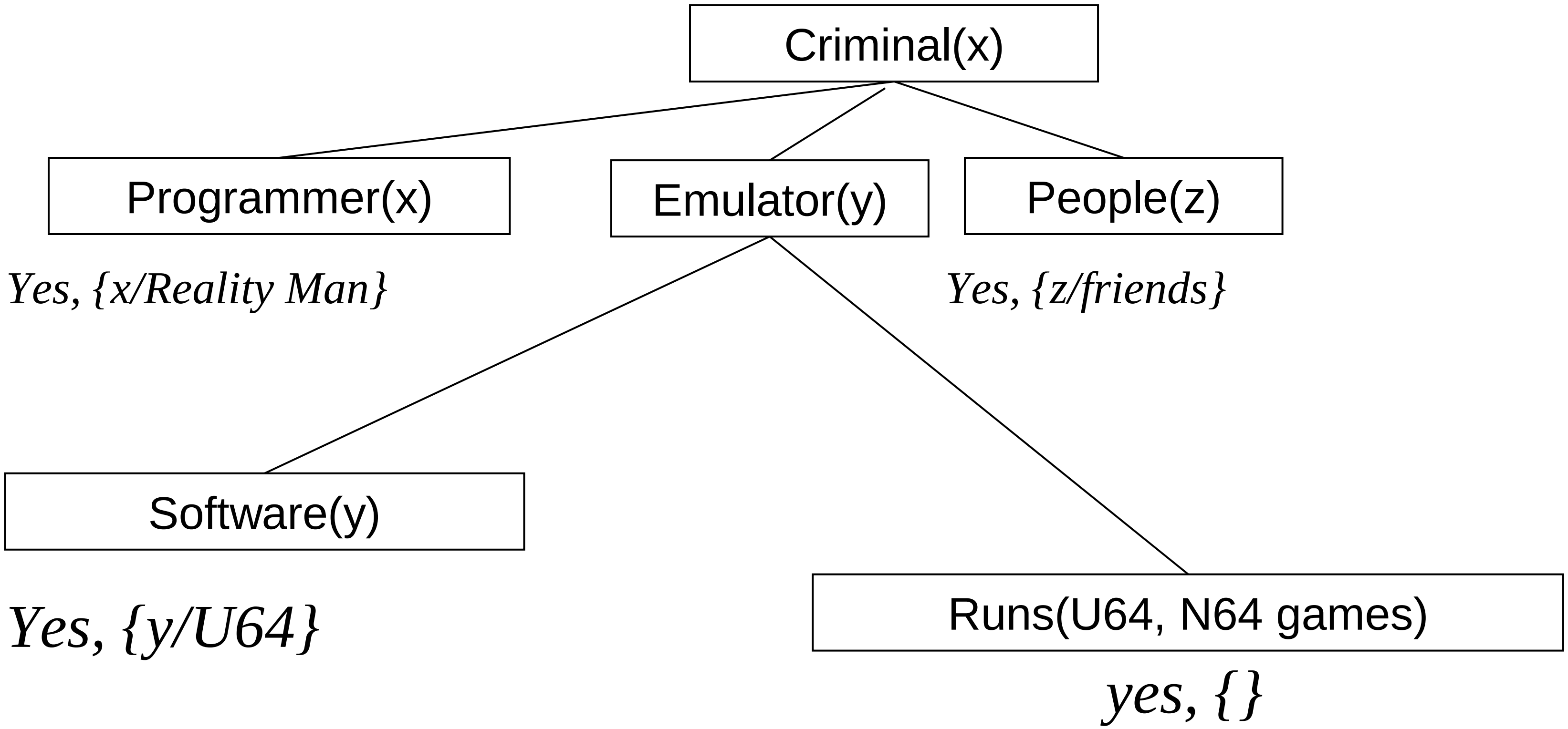
Backward Chaining Example

- Question: Has Reality Man done anything criminal?



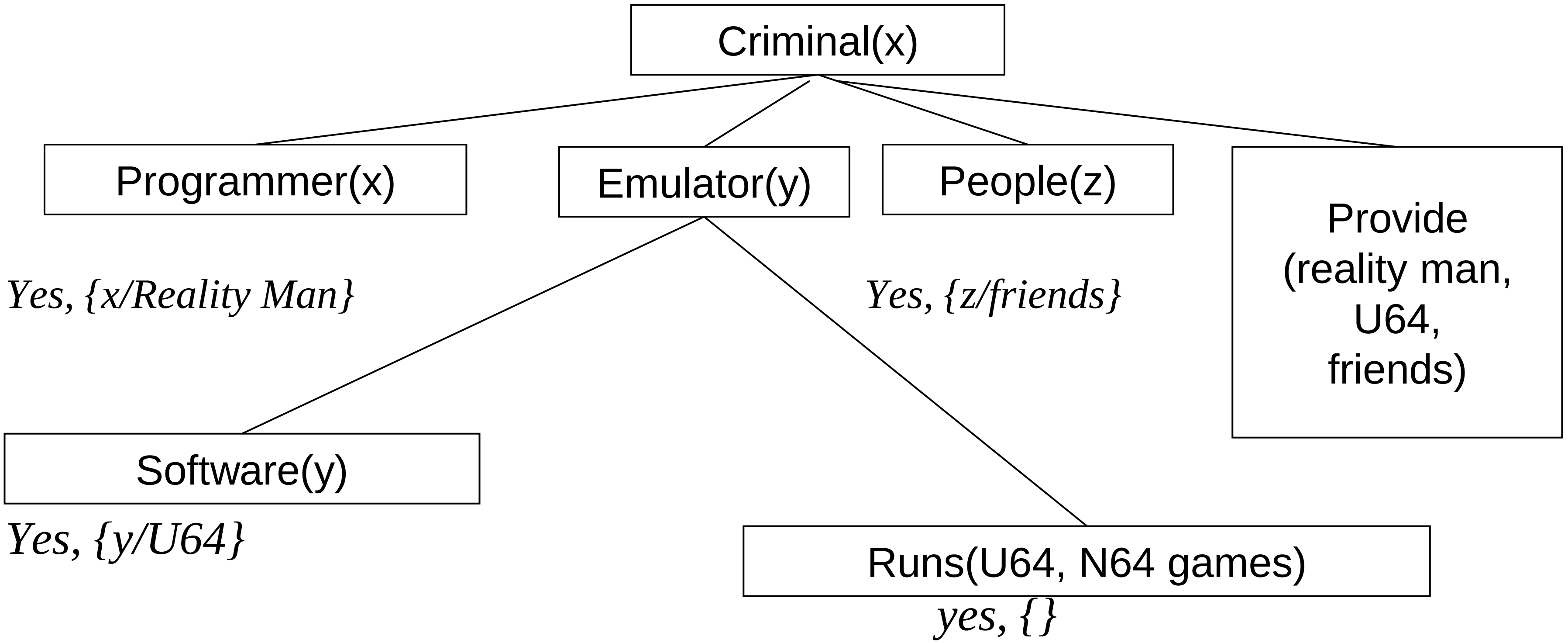
Backward Chaining Example

- Question: Has Reality Man done anything criminal?



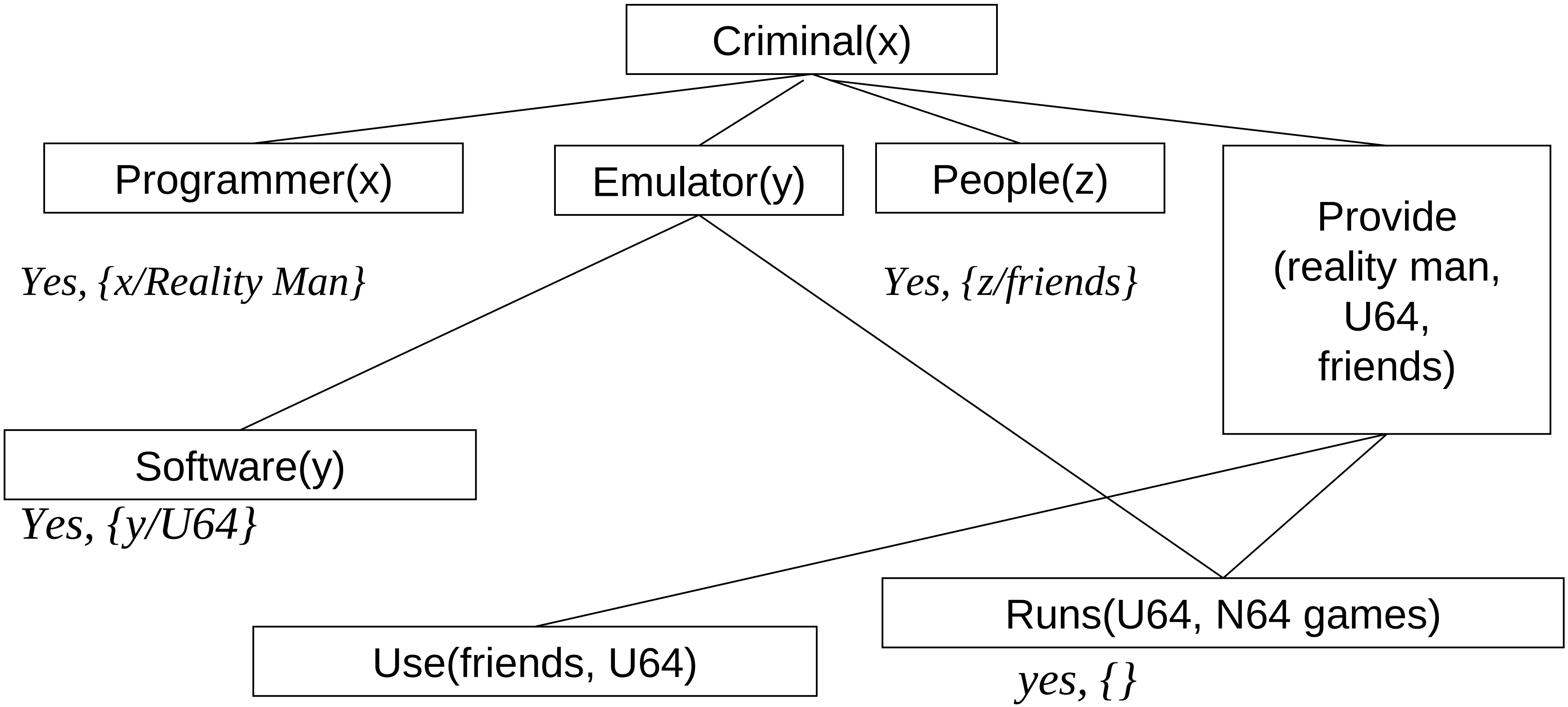
Backward Chaining Example

- Question: Has Reality Man done anything criminal?



Backward Chaining Example

- Question: Has Reality Man done anything criminal?



Backward Chaining

- Backward Chaining benefits from the fact that it is directed toward proving one statement or answering one question.
- In a focused, specific knowledge base, this greatly decreases the amount of superfluous work that needs to be done in searches.
- However, in broad knowledge bases with extensive information and numerous implications, many search paths may be irrelevant to the desired conclusion.
- Unlike forward chaining, where all possible inferences are made, a strictly backward chaining system makes inferences only when called upon to answer a query.

Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - $\Rightarrow$ fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - $\Rightarrow$ fix using caching of previous results (extra space)
- Widely used for logic programming

Logic programming

Sound bite: computation as inference on logical KBs

	Logic programming	Ordinary programming
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem instance as facts	Encode problem instance as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

Should be easier to debug $Capital(NewYork, US)$ than $x := x + 2$!

Logic programming: Prolog

- Algorithm = Logic + Control
- Basis: backward chaining with Horn clauses
  Widely used in Europe, Japan (basis of 5th Generation project)
- Program = set of clauses = `head :- literal`$_1$`, ... literal`$_n$.
-

```
criminal(X) :- american(X), weapon(Y),
          sells(X,Y,Z), hostile(Z).
```

Logic programming: Prolog

Prolog programs are sets of definite clauses witten in a notation somewhat different from standard first-order logic

FOL:

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

Prolog:

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

Prolog example

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

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

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

Logic programming: Prolog

- Depth-first, left-to-right backward chaining
- Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
- Built-in predicates that have side effects
  – (e.g., input and output predicates, assert/retract predicates)
- Closed-world assumption ("negation as failure")
  – e.g., given `alive(X) :- not dead(X).`
  – `alive(joe)` succeeds if `dead(joe)` fails

Prolog

- Appending two lists to produce a third:

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

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

- answers:
```
A=[]      B=[1,2]
A=[1]     B=[2]
A=[1,2]   B=[]
```

Resolution: brief summary

- Full first-order version:

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

where Unify(l_i, $\neg m_j$) = θ.

- The two clauses are assumed to be standardized apart so that they share no variables.
- For example,

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

with θ = {x/Ken}

- Apply resolution steps to CNF(KB ∧ ¬α); complete for FOL

Conversion to CNF

Everyone who loves all animals is loved by someone:

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

(1) Eliminate biconditionals and implications

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

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

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

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

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

(3) Standardize variables: each quantifier should use a different one

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

Conversion to CNF contd.

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

(5) Drop universal quantifiers:

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

(6) Distribute $\vee$ over $\wedge$:

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

(7) Create separate clauses

Animal(F(x)) $\vee$ Loves(G(x),x) $\neg$Loves(x, F(x)) $\vee$ Loves(G(x),x)

(8) Standardize variables

Animal(F(x)) $\vee$ Loves(G(x),x) $\neg$Loves(y, F(y)) $\vee$ Loves(G(y),y)

Example knowledge base contd.

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

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

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

$Owns(Nono,M_1)$ and $Missile(M_1)$

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

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

Missiles are weapons:

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

An enemy of America counts as "hostile“:

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

West, who is American …

$American(West)$

The country Nono, an enemy of America …

$Enemy(Nono,America)$

The Sentences in CNF are:

$$\neg American(x) \lor \neg Weapon(y) \lor \neg Sells(x, y, z) \lor \neg Hostile(z) \lor Criminal(x).$$

$$\neg Missile(x) \lor \neg Owns(Nono, x) \lor Sells(West, x, Nono).$$

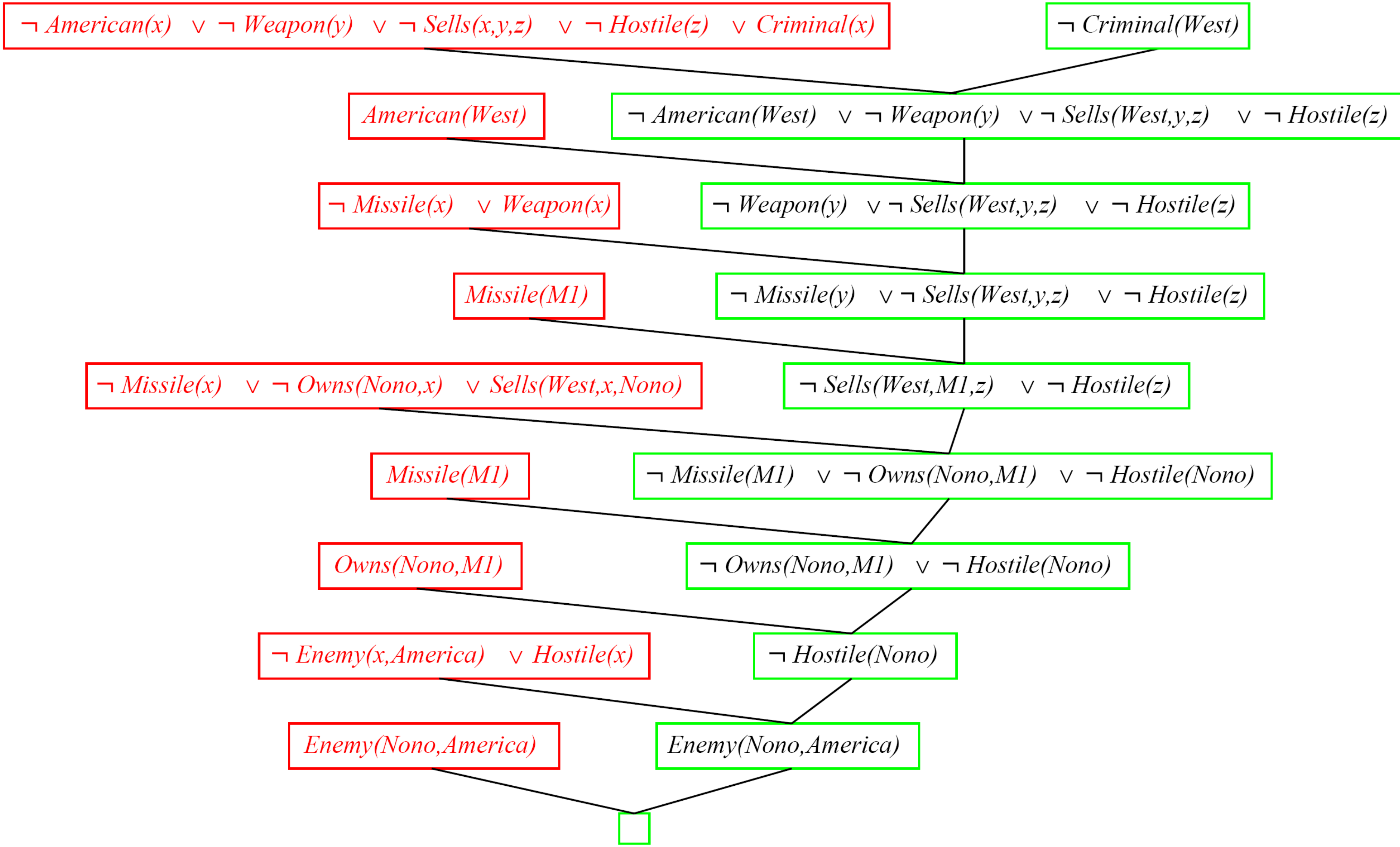
$$\neg Enemy(x, America) \lor Hostile(x).$$

$$\neg Missile(x) \lor Weapon(x).$$

$Owns(Nono, M_1).$ $Missile(M_1).$

$American(West).$ $Enemy(Nono, America).$

Resolution proof: definite clauses



Refinement Strategies

- **Unit preference**
  - Prefers to do resolutions where one of the sentences is a single literal.
  - Every resolution step must involve a unit clause.
  - May produce shorter clauses
  - Incomplete in general; Complete for Horn clauses
- **Set of support strategy**
  - Allows only those resolutions in which one of the clauses being resolved is in the set of support, i.e., those clauses that are either clauses coming from the negation of the theorem to be proved or descendants of those clauses.
  - Complete

Refinement Strategies

- **Linear input strategy**
  - at least one of the clauses being resolved is a member of the original set of clauses.
  - Not complete
- **Ancestry filtering strategy**
  - at least one member of the clauses being resolved either is a member of the original set of clauses or is an ancestor of the other clause being resolved.
  - Complete

Summary

- GMP
- Forward Chaining (前向链接)
- Backward Chaining (反向链接)
- Resolution (归结)

Conversion to CNF

(1) Eliminate biconditionals and implications

(2) Move $\neg$ inwards

(3) Standardize variables: each quantifier should use a different one

(4)Skolemize: a more general form of existential instantiation

(5) 将公式化为前束型

(6) Drop universal quantifiers

(7) Distribute $\vee$ over $\wedge$

(8)Create separate clauses

(9) Standardize variables

作业

,

- 9.4
- 9.6
- 9.21
- 9.23