CSC520 - Artificial Intelligence

Lecture 14

Dr. Scott N. Gerard

North Carolina State University

Feb 27, 2025

Agenda

- Forward chaining
- Backward chaining
- Proof by resolution
- Logic Programming

Forward Chaining in FOL

```
function FOL-FC-ASK($KB, \alpha$) substitution or false
  inputs: $KB$, a set of definite clauses, $\alpha$, a sentence
  while true do
    $new \leftarrow \{\}$
    for rule in $KB$ do
      $(p_1 \wedge \ldots p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARS($rule$)
      for each $\theta$ such that SUBST($\theta, p_1 \wedge \ldots p_n$) = SUBST($\theta, p'_1 \wedge \ldots p'_n$)
              for some $p'_1, \ldots, p'_n$ in KB do
        $q' \leftarrow$ SUBST($\theta, q$)
        if $q'$ is not renaming of a sentence in $KB$ or $new$ then
          add $q'$ to $new$
          $\Phi \leftarrow$ UNIFY($q', \alpha$)
          if $\Phi$ is not $fail$ then return $\Phi$

    if new = {} then return $false$
    add $new$ to $KB$
```

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.
We would like to prove: Colonel West is a criminal.

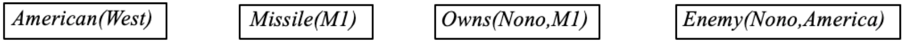
- $American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$
- $\exists x\ Owns(Nono, x) \land Missile(x)$
  - $Owns(Nono, M_1)$, $Missle(M_1)$, using existential instantiation
- $Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

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.
We would like to prove: Colonel West is a criminal.

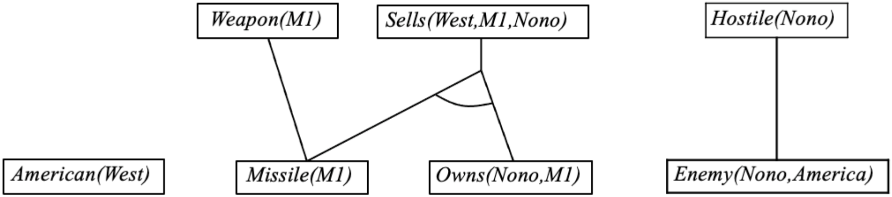
- $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
- $\exists x\ Owns(Nono, x) \wedge Missile(x)$
  - $Owns(Nono, M_1)$, $Missle(M_1)$, using existential instantiation
- $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- $Missile(x) \Rightarrow Weapon(x)$
- $Enemy(x, America) \Rightarrow Hostile(x)$
- $American(West)$
- $Enemy(Nono, America)$

Forward Chaining



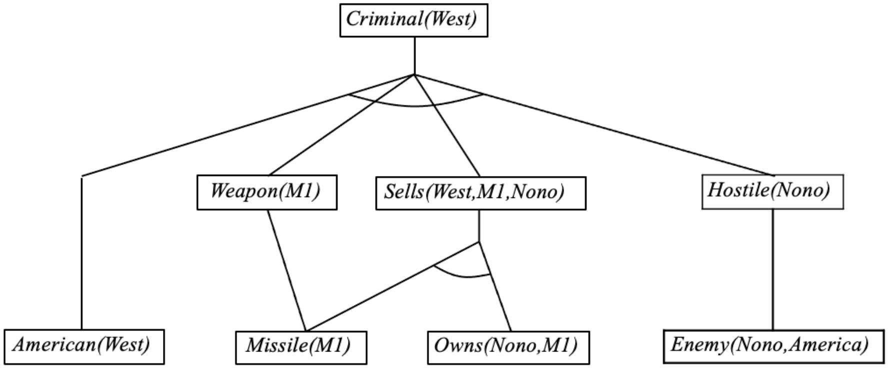
$Missile(x) \Rightarrow Weapon(x),\ \theta = \{x/M_1\}$
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono),\ \theta = \{x/M_1\}$
$Enemy(x, America) \Rightarrow Hostile(x),\ \theta = \{x/Nono\}$

Forward Chaining



$Missile(x) \Rightarrow Weapon(x),\ \theta = \{x/M_1\}$
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono),\ \theta = \{x/M_1\}$
$Enemy(x, America) \Rightarrow Hostile(x),\ \theta = \{x/Nono\}$

Forward Chaining

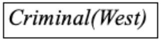


$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$,
$\theta = \{x/West, y/M_1, z/Nono\}$

Properties of Forward Chaining

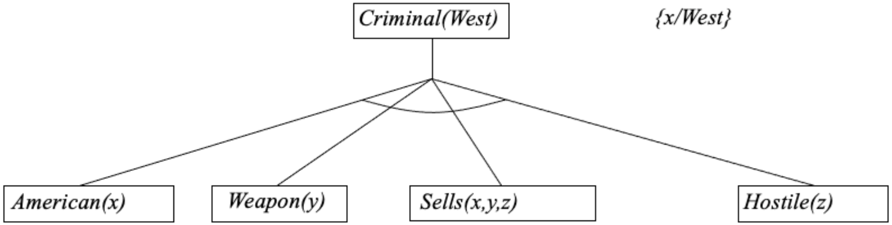
- Sound and complete for first-order definite clauses
- Datalog (1977) $=$ first-order definite clauses $+$ no functions (e.g., crime example)
- May not terminate in general if α is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable

Backward Chaining

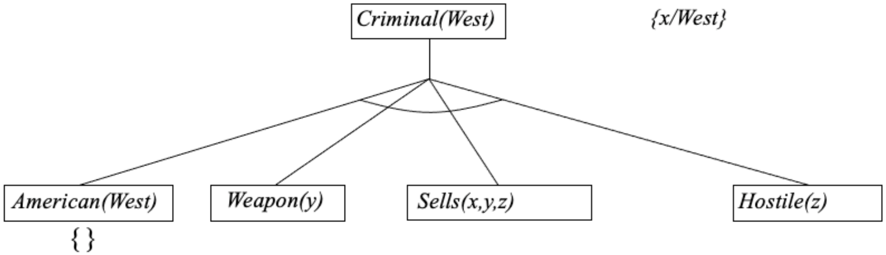


$$American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$$

Backward Chaining

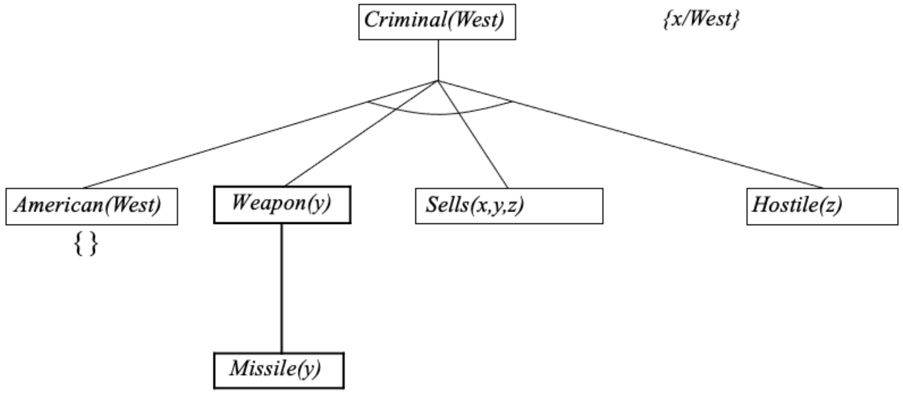


Backward Chaining

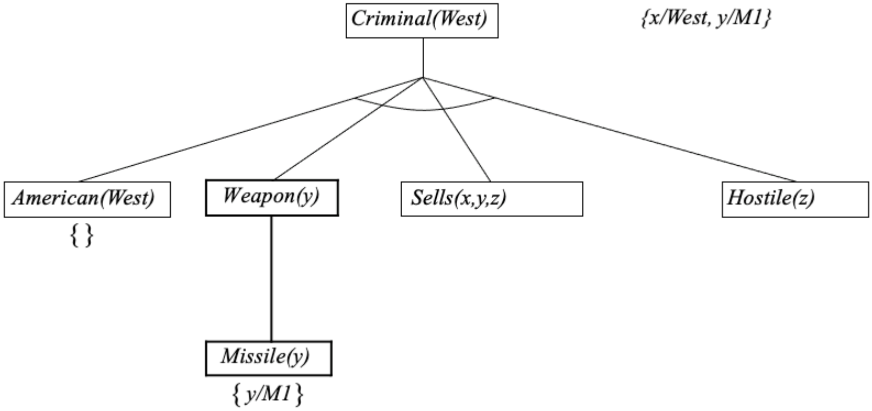


$$Missile(x) \Rightarrow Weapon(x)$$

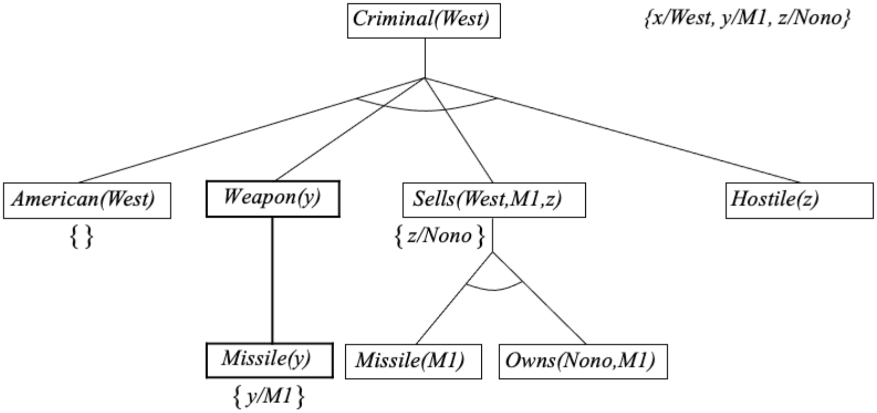
Backward Chaining



Backward Chaining

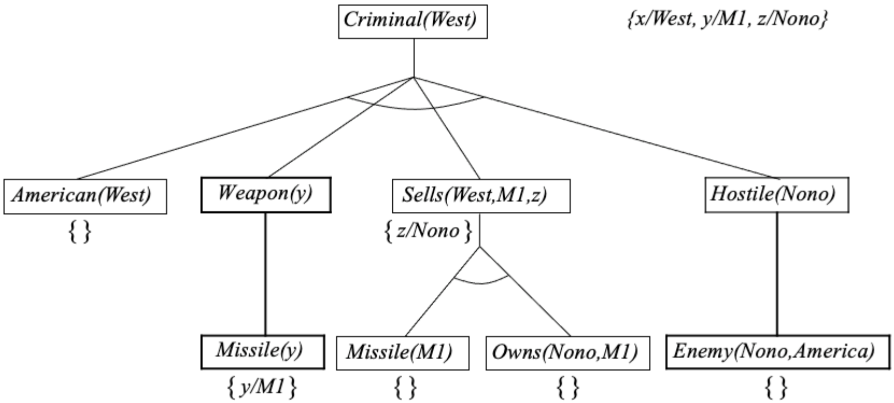


Backward Chaining



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

Backward Chaining



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

Properties of Backward Chaining

- Depth-first recursive proof search
- Incomplete due to infinite loops
- Inefficient due to repeated subgoals
- But it is widely used for logic programming

Logic Programming

- Encode information in a KB using a formal language e.g. FOL
- An inference engine runs on the KB to resolve queries
- Follows database semantics
  - Unique-names assumption: Every constant symbol refers to a distinct object
  - Closed world assumption: Atomic sentences not known are false
  - Domain closure assumption: A model contains no more objects than the constant symbols
- Without database semantics, "Richard has two brothers, John and Geoffrey"
  $Brother(John, Richard) \land Brother(Geoffrey, Richard) \land John \neq Geoffrey \land \forall x\ Brother(x, Richard) \Rightarrow (x = John \lor x = Geoffrey)$
- With database semantics:
  $Brother(John, Richard) \land Brother(Geoffrey, Richard)$

Prolog

- Prolog is a logic programming language
  - Uses database semantics
- A program is a set of definite clauses
  - $A \wedge B \Rightarrow C$
  - In Prolog, `C :- A, B.`
- Prolog searches for a proof for the user queries
- Uses depth-first backward-chaining search without checking for infinite recursion
  - Some programs will fail to terminate

Elements of Prolog

- Constants have initial lowercase. Variables have initial uppercase.
- Comma is conjunction. Semicolon is disjunction. \+ is negation
- Facts
  - `american(west).`
  - `owns(nono, m1).`
  - `brother(richard, john).`
- Rules
  - `criminal(X) :- american(X), weapon(Y), sells(X, Y, Z), hostile(Z).`
  - `sibling(X, Y) :- brother(X, Y).`
- Lists
  - `likes(jim, [sue, john, doug]).`
  - `[H|T] = [a, b, c].`
- Math: Provides 'is' operator to evaluate mathematical expressions
  - `X is 2+2.`
  - `X > 0.`

Elements of Prolog

- Queries
  - After loading facts and rules, user poses queries
  - If query contains no variables, it is a binary question and prolog responds with true or false
  - If query contains variables, prolog prints valid variable mappings/bindings (if any) one at a time
    - ★ Enter ; to get next binding

Elements of SWI-Prolog

Basics

- SWI Prolog available for free
  - https://www.swi-prolog.org

```
%! start SWI-Prolog and load a file
./swipl assignment03.pl

%! load facts & rules from file
[my_prolog].

%! load facts & rules from console
[user].
male(bob).
ctrl-D
```

Elements of SWI-Prolog

Extra Goodies

```
%! Print a message
print("Hello").

%! find all solutions
?- findall(grand_parent(X,Y), writeln([X,Y])).
[richard,jack]
[richard,bob]
[jill,jack]
[jill,bob]
true.

%! find all solutions
?- findall(soln(X,Y), grand_parent(X,Y), Bag).
Bag = [soln(richard, jack), soln(richard, bob),
       soln(jill, jack), soln(jill, bob)].

?- findall(stuff(Z,X), (parent(X,Y), parent(Y,Z)), Bag).
Bag = [stuff(jack, richard), stuff(bob, richard),
       stuff(jack, jill), stuff(bob, jill)].

%! query inside a file
:- initialization forall(grand_parent(X,Y), writeln([X,Y])).
```

Prolog Program Examples
Family Domain

```
male(richard).
male(jack).
male(bob).
male(john).

female(jill).
female(alice).

parent(john, jack).
parent(alice, jack).

parent(john, bob).
parent(alice, bob).

parent(richard, john).
parent(jill, john).
```

```
father(F, C) :- male(F), parent(F, C).
mother(M, C) :- female(M), parent(M, C).

son(S, P) :- male(S), parent(P, S).
daughter(D, P) :- female(D), parent(P, D).

grand_parent(G, C) :- parent(G, P), parent(P, C).

?- grand_parent(X, Y).
X = richard,
Y = jack ;
X = richard,
Y = bob ;
X = jill,
Y = jack ;
X = jill,
Y = bob.

:- initialization forall(grand_parent(X,Y), writel
```

Prolog Program Examples

Searching in a list

```
on(Item, [Item|Rest]).
on(Item, [_|Tail]) :- on(Item, Tail).
```

```
?- on(3, [2, 4, 3, 6]).
true.
```

```
?- on(3, [2, 4, 5, 6]).
false.
```

Prolog Program Examples

Counting

```
count_to_10(10) :- write(10), nl.
count_to_10(X) :-
    X < 10, write(X), nl, Y is X + 1, count_to_10(Y).
```

```
?- count_to_10(4).
4
5
6
7
8
9
10
```

Class Exercise

- Suppose you are given these sentences.
  1. $\forall x \; Likes(x, Wine) \Rightarrow Likes(Dick, x)$
  2. $Likes(Ed, Wine)$

  Use proof by resolution to prove the sentence: $Likes(Dick, Ed)$.

- Convert following sentence to CNF.
  $\forall s \; [Breezy(s) \Rightarrow \exists r \; Adjacent(r, s) \wedge Pit(r)]$