Prolog (part 1)

Andrea Galassi

Languages and Algorithms for Artificial Intelligence

Case study from Russel-Norvig's AIMA 3rd edition, Chapter 9

Exercises courtesy of Prof. Federico Chesani

Key concepts of Prolog: syntax

- Variables `X1` `_`
- Constants `alice`
- Functors `female/1`
- Predicates `female(X1)` `!`
- Lists `[]` `[a|l]` `[a,b]` `[a,b|l]`

- Facts `Head .`
- Rules `Head :- Body .`

```
female(alice).
sister(X1,X2) :- sibling(X1,X2), female(X1) .
```

Key concepts of Prolog: principles

- Closed world assumption:
  - All the truths are entailed by the knowledge base
  - Everything else is false

- Unique name assumption:
  - Every constant and ground term refers to a distinct object
  - W.r.t. FOL it is not necessary anymore to specify which objects are distinct

Key concepts of Prolog: SLD solution

- Backward chaining algorithm

  Starts with the goal and tries to prove it

  (actually, negates the goal and tries to prove it)

- Depth-first search, left-most selection rule

- Clauses are considered in the order they are written
  - The same clauses that lead to a solution, in a different order may lead to an infinite loop

From Natural Language to Prolog (1/5)

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

Goal:

Col. West is a criminal

From Natural Language to Prolog (2/5)

From the paragraph:

- .. 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
  - Owns(Nono,M1)
  - Missile(M1)

- … all of its missiles were sold to it by Colonel West
  - $Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

From Natural Language to Prolog (3/5)

- West, who is American … :
  - American(West)
- The country Nono, an enemy of America …:
  - Enemy(Nono,America)

Implict knowledge

- Missiles are weapons:
  - Missile(x) $\Rightarrow$ Weapon(x)
- An enemy of America counts as “hostile”
  - Enemy(x,America) $\Rightarrow$ Hostile(x)

From Natural Language to Prolog (4/5)

Knowledge Base

- American(x) $\wedge$ Weapon(y) $\wedge$ Sells(x,y,z) $\wedge$ Hostile(z) $\Rightarrow$ Criminal(x)
- Owns(Nono,M1)
- Missile(M1)
- Missile(x) $\wedge$ Owns(Nono,x) $\Rightarrow$ Sells(West,x,Nono)
- American(West)
- Enemy(Nono,America)
- Missile(x) $\Rightarrow$ Weapon(x)
- Enemy(x,America) $\Rightarrow$ Hostile(x)

Goal: Criminal(West)

From Natural Language to Prolog (5/5)

Prolog Knowledge Base

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
owns(nono,m1).
missile(m1).
sells(west,Y,nono) :- missile(Y), owns(nono,Y).
american(west).
enemy(nono,america).
weapon(Y) :- missile(Y).
hostile(X) :- enemy(X,america).
```

Goal:

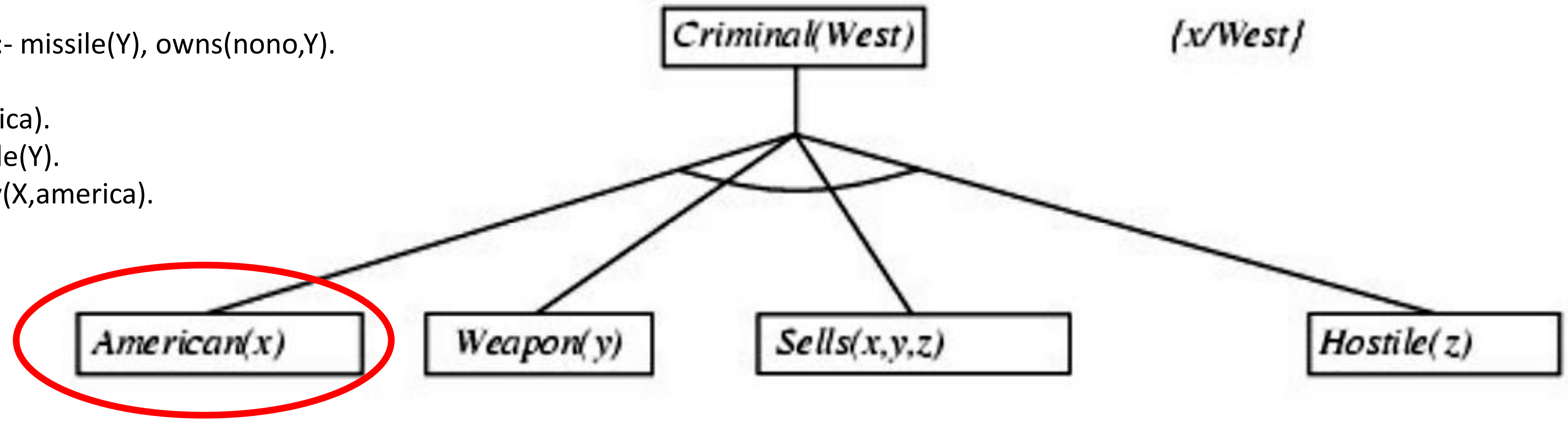
```
?- criminal(west)
```

Solving

criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
owns(nono,m1).
missile(m1).
sells(west,Y,nono) :- missile(Y), owns(nono,Y).
american(west).
enemy(nono,america).
weapon(Y) :- missile(Y).
hostile(X) :- enemy(X,america).

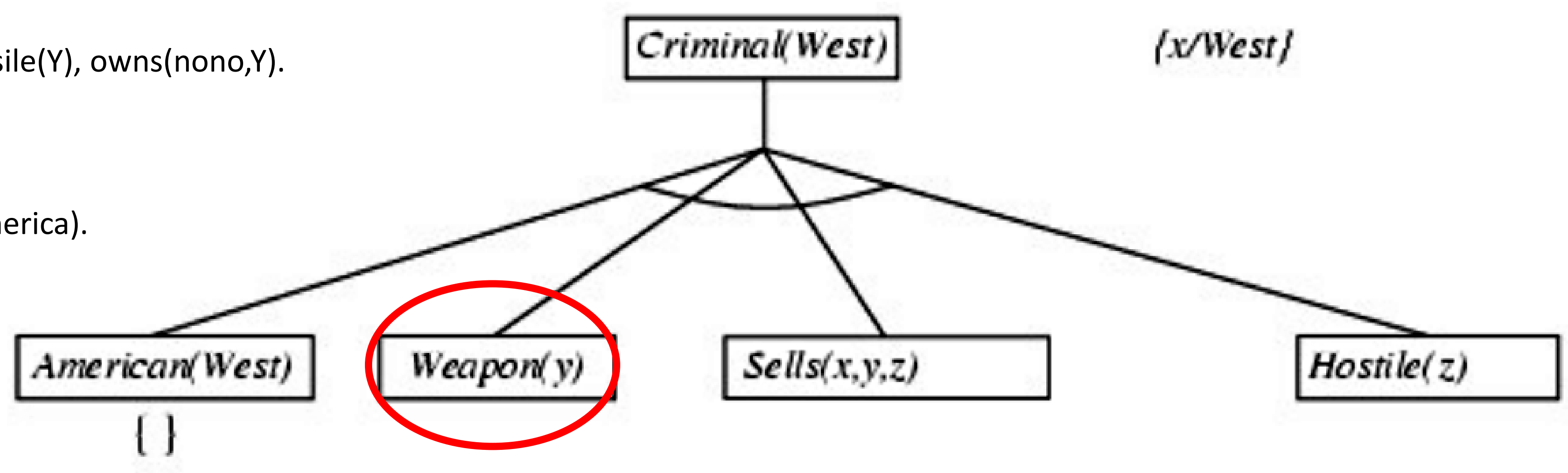
Solving

criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
owns(nono,m1).
missile(m1).
sells(west,Y,nono) :- missile(Y), owns(nono,Y).
american(west).
enemy(nono,america).
weapon(Y) :- missile(Y).
hostile(X) :- enemy(X,america).



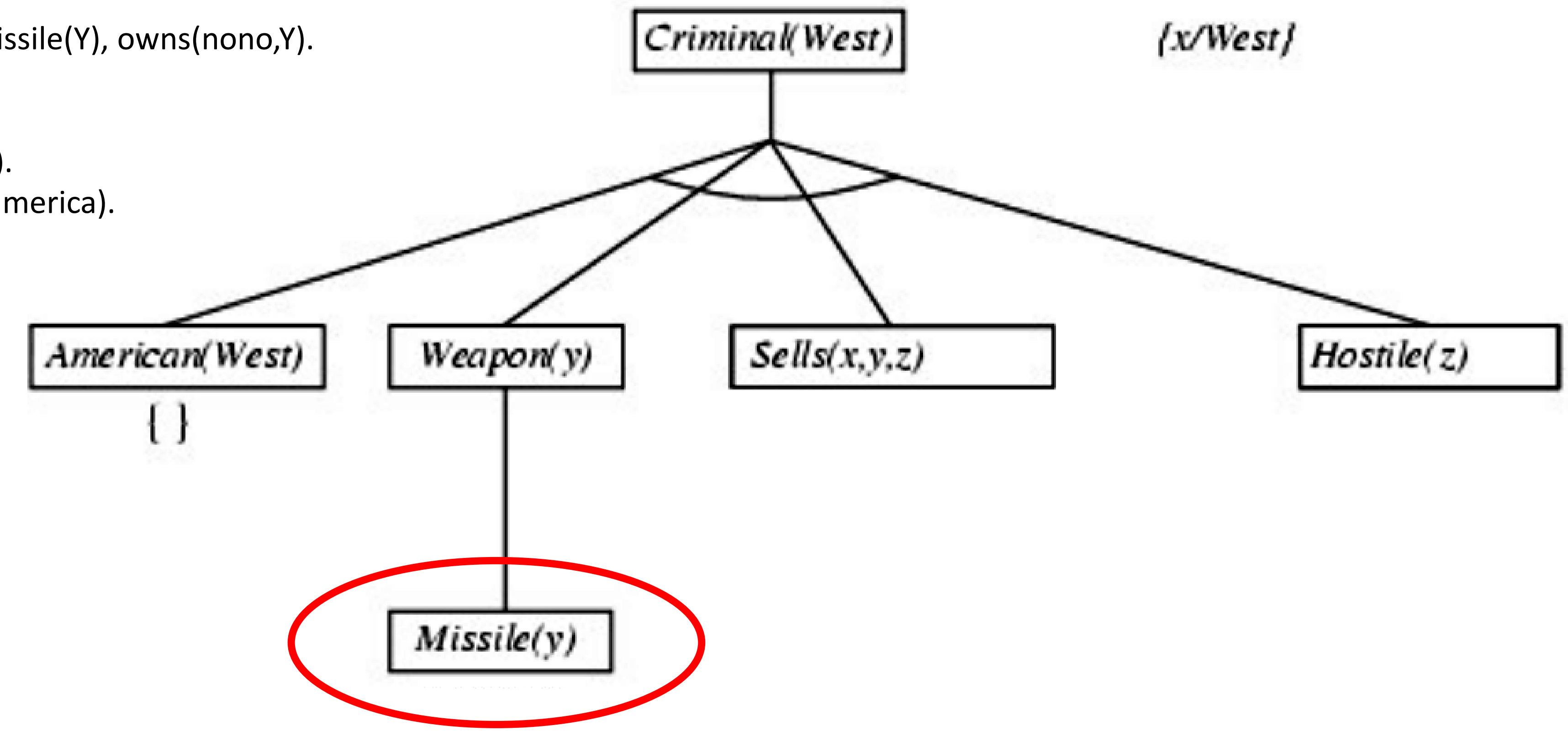
Solving

criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
owns(nono,m1).
missile(m1).
sells(west,Y,nono) :- missile(Y), owns(nono,Y).
american(west).
enemy(nono,america).
weapon(Y) :- missile(Y).
hostile(X) :- enemy(X,america).



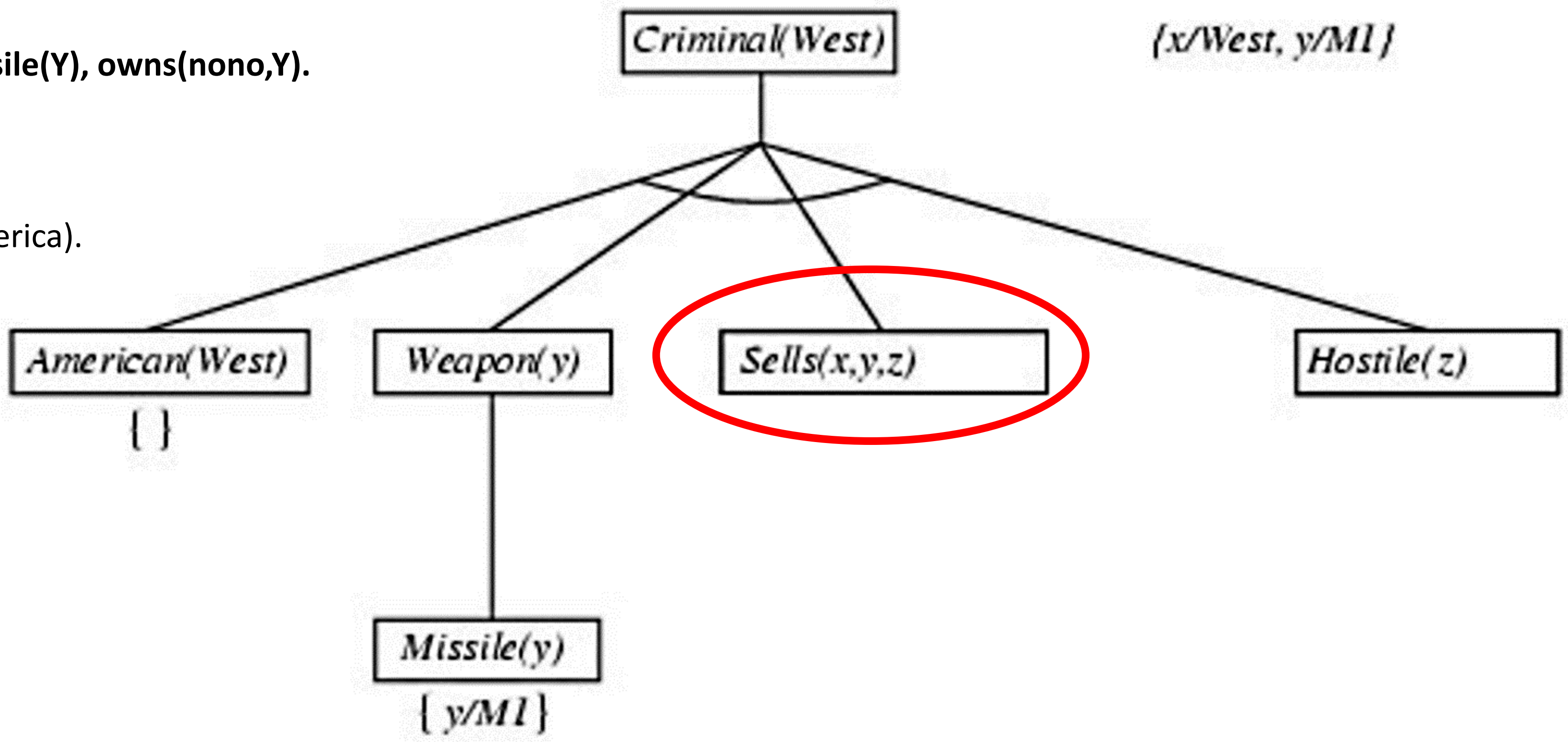
Solving

criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
owns(nono,m1).
missile(m1).
sells(west,Y,nono) :- missile(Y), owns(nono,Y).
american(west).
enemy(nono,america).
weapon(Y) :- missile(Y).
hostile(X) :- enemy(X,america).



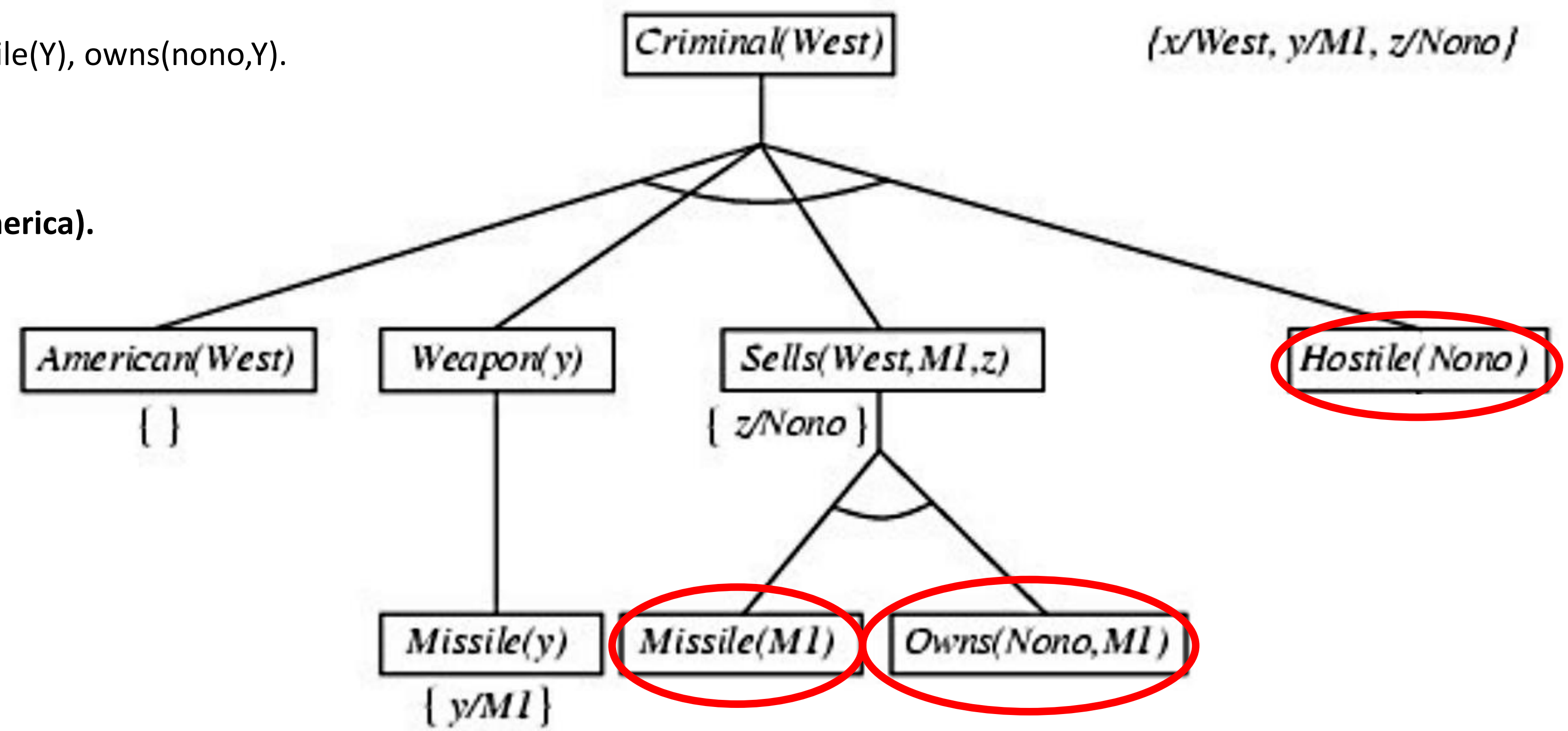
Solving

criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
owns(nono,m1).
missile(m1).
sells(west,Y,nono) :- missile(Y), owns(nono,Y).
american(west).
enemy(nono,america).
weapon(Y) :- missile(Y).
hostile(X) :- enemy(X,america).



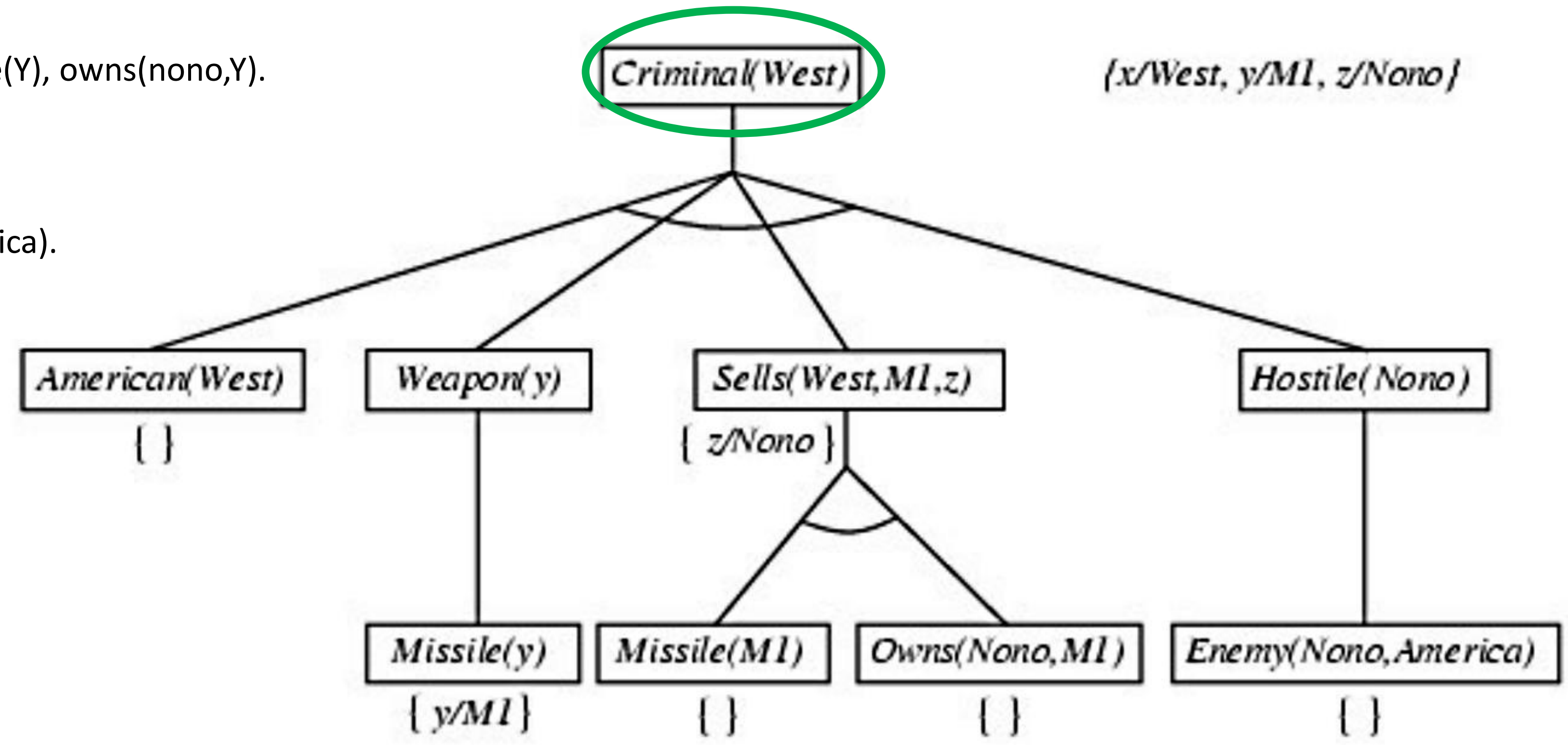
Solving

criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
owns(nono,m1).
missile(m1).
sells(west,Y,nono) :- missile(Y), owns(nono,Y).
american(west).
enemy(nono,america).
weapon(Y) :- missile(Y).
hostile(X) :- enemy(X,america).



Solving

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
owns(nono,m1).
missile(m1).
sells(west,Y,nono) :- missile(Y), owns(nono,Y).
american(west).
enemy(nono,america).
weapon(Y) :- missile(Y).
hostile(X) :- enemy(X,america).
```



Exercise 1: last element of a list (1/7)

Define a PROLOG predicate *lastel* that given a list L of integer, find the last element E of the list

```
lastel(L,E)
```

Exercise 1: last element of a list (2/7)

First step: define base cases

If the list is empty? The result is an empty list

```
lastel([], []).
```

If the list contains only one element? That element!

```
lastel([E], E).
```

Exercise 1: last element of a list (3/7)

```
lastel([], []).
lastel([E], E).
```

Second step: general case

If I have a list with more than one element, I can eliminate the first element (A) and consider only the remaining list:

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

Exercise 1: last element of a list (4/7)

Let’s test this program

```
lastel([], []).
lastel([E], E).
lastel([A|X],Y) :- lastel(X,Y).

?-lastel([3,4,5,7], Z)
```

What happens?

Exercise 1: last element of a list (5/7)

- First, since A is not used anywhere else in the program, it is better to use the unnamed variable _ in its place.

- Second, two solutions are found, why? How can it be fixed?

Exercise 1: last element of a list (6/7)

- First, since A is not used anywhere else in the program, it is better to use the unnamed variable **_** in its place.

- Second, two solutions are found, why? How can it be fixed?
  - When **`lastel`** is applied to a list with only one element, there are two possibility: apply the second clause or the third one.
  - In the second case, the last element is removed and the empty list is obtained.
  - We can fix this with the CUT operator in the second clause so to stop the search.

Exercise 1: last element of a list (7/7)

Final program:

```
lastel([], []).
lastel([E], E) :- !.
lastel([_|X],Y) :- lastel(X,Y).

?-lastel([3,4,5,7], Z)
```

Exercise 2: maximum element of a list

Define a PROLOG predicate *max* that given a list L of integer, find the greater element E of the list

```
max(L,E)
```

Two approaches: iterative and recursive

In the recursive approach we reach the end of the list and recursively evaluate every element.

In the iterative approach we evaluate every element while we reach the end.

Exercise 2: maximum element of a list

Recursive approach: we reach the end, then go back.

We define the base case:

`max([E],E):-!.`

To understand how to write the rest of the program let's think about a real scenario: **`?-max([7,5,6],X).`**

Let's start with the very last element of the list, we know that

`max([6],6)`.

Is proved true by the base case.

Exercise 2: maximum element of a list

Recursive approach: we reach the end, then go back.

```
max([E],E):-!.
```

So now we know that **`max(L,E)`** holds for E=6 and L=[6]

Now we consider the element that precedes our current sublist L.

We have **`max([5,6],6)`**

So, the maximum is the same because the new element is smaller than the maximum computed so far.

From this case we can write

Exercise 2: maximum element of a list

Recursive approach: we reach the end, then go back.

```
max([E],E):-!.
```

So now we know that **max(L,E)** holds for E=6 and L=[6]

Now we consider the element that precedes our current sublist L.

We have **max([5,6],6)**

So, the maximum is the same because the new element is smaller than the maximum computed so far.

From this case we can write

```
max([N|L],E):- max(L,E), E>=N.
```

Exercise 2: maximum element of a list

Recursive approach: we reach the end, then go back.

```
max([E],E):-!.
max([N|L],E):- max(L,E), E>=N.
```

So now we know that **`max(L,E)`** holds for E=6 and L=[5,6].

Once again, we consider the element that precedes our current sublist L.

We have **`max([7,5,6],7)`**

When the current maximum is not greater than the new element, the new element becomes the maximum

From this case we can write

Exercise 2: maximum element of a list

Recursive approach: we reach the end, then go back.

```
max([E],E):-!.
max([N|L],E):- max(L,E), E>=N.
```

So now we know that **max(L,E)** holds for E=6 and L=[5,6].

Once again, we consider the element that precedes our current sublist L.

We have **max([7,5,6],7)**

When the current maximum is not greater than the new element, the new element becomes the maximum

From this case we can write

```
max([N|L],N):- max(L,E), E<N.
```

Exercise 2: maximum element of a list

Recursive approach: we reach the end, then go back.

```
max([E],E):-!.
max([N|L],E):- max(L,E), E>=N.
max([N|L],N):- max(L,E), E<N.
```

Try the query **`?-max([7,5,6],X).`**

Exercise 2: maximum element of a list

Recursive approach: we reach the end, then go back.

```
max([E],E):-!.
max([N|L],E):- max(L,E), E>=N, !.
max([N|L],N):- max(L,E), E<N.
```

Once again, we need the CUT.

Exercise 2: maximum element of a list

Iterative approach: we evaluate every element while we reach the end.

For the iterative version we need to use a predicate with arity 3, so to “carry over” the temporary maximum.

```
max([N|L],E):- max(L,N,E).
```

At each step

- we analyze one element of the list
- we confirm or replace the temporary maximum N
- we move to examine the next element of the list.

Exercise 2: maximum element of a list

Iterative approach: we evaluate every element while we reach the end.

```
max([N|L],E):- max(L,N,E).
```

Final case: when we reach the final element of the list:

```
max([],E,E):- !.
```

In this approach we have two cases too: when the new element is bigger of the temporary maximum and when it is smaller.

Exercise 2: maximum element of a list

Iterative approach: we evaluate every element while we reach the end.

```
max([N|L],E):- max(L,N,E).
max([],E,E):- !.
```

It can be helpful to imagine the two scenarios with real cases.

Let's consider the lists [5,6,4] and [6,5,4]. In the first one we have

```
max([5|[6,4],X):- max([6,4],5,X).
```

In the next step we want to have **`max([4],6,X)`**, so we need to define a rule that links this two steps.

```
max([M|L],N,E):- M>=N,max(L,M,E),!.
```

Exercise 2: maximum element of a list

Iterative approach: we evaluate every element while we reach the end.

```
max([N|L],E):- max(L,N,E).
max([],E,E):- !.
max([M|L],N,E):- M>=N,max(L,M,E).
```

Similarly we in the opposite case

```
max([M|L],N,E):- M<N, max(L,N,E).
```

Exercise 2: maximum element of a list

Iterative approach: we evaluate every element while we reach the end.

Final program (adding a CUT!)

```
max([N|L],E):- max(L,N,E).
max([],E,E):- !.
max([M|L],N,E):- M>=N, max(L,M,E), !.
max([M|L],N,E):- M<N, max(L,N,E).
```

Exercise 3

Given a list L1 and a integer number N, write a PROLOG predicate question1(L1, N, L2) where L2 must be the list of elements in L1 that are list with 2 positive value between 1 and 9 which sum is N.

Example:

```
?- question1 (
     [ [3,1], 5, [2,1,1], [3], [1,1,1], a,[2,2] ],
     4, L2).

L2 = [[3,1], [2,2]]
```

Exercise 3

Solution:

```
question1([],_,[]).
question1([[A,B]|R ], N, [[A,B]|S]):-
    A>=1, A=<9, B>=1, B=<9,
    N is A + B, !,
    question1( R,N,S ).
question1([_|R], N,S ):- question1( R,N,S ).
```

Exercise 4

Write a Prolog predicate *consec* that given a list L and an element E, returns the element of L that follows E.

If E is the last element or it is not present in the list, the predicate must fail.

Example:

```
?- consec(3, [1,7,3,9,11],X).

X=9
```

Exercise 4

```
consec(El, [El| [X|_]], X):-!.
consec(El, [_|Tail],X):- consec(El,Tail,X).
```

Exercise 5: position of element

Write a Prolog predicate **`listPos`** that given a list L and an element E, returns the position of the first occurence of E inside L. The first element of the list is considered to be in position 0.

Example:

```
?- listPos([1,2,3,5,6,3], 3, X).
X=2
```

What if we want the positions of all the occurences of E?

Exercise 5: position of element

First occurence:

```
listPos([X|_], X, 0):-!.
listPos([_|T], X, Pos) :-
                            listPos(T, X, Pos1),
                            Pos is Pos1+1.
```

Exercise 5: position of element

All occurences:

```
listPos([X|_], X, 0).
listPos([_|T], X, Pos) :-
                            listPos(T, X, Pos1),
                            Pos is Pos1+1.
```