Introduction to AI

Logic for

Knowledge Representation and

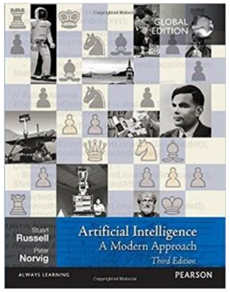
Automated Reasoning

Francesca Toni

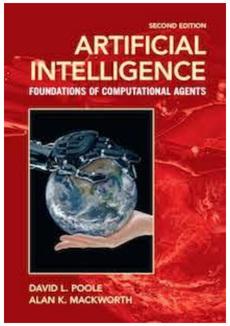
Outline

- Resolution and unification and their use for automated reasoning
- Foundations of logic programming for knowledge representation and automated reasoning

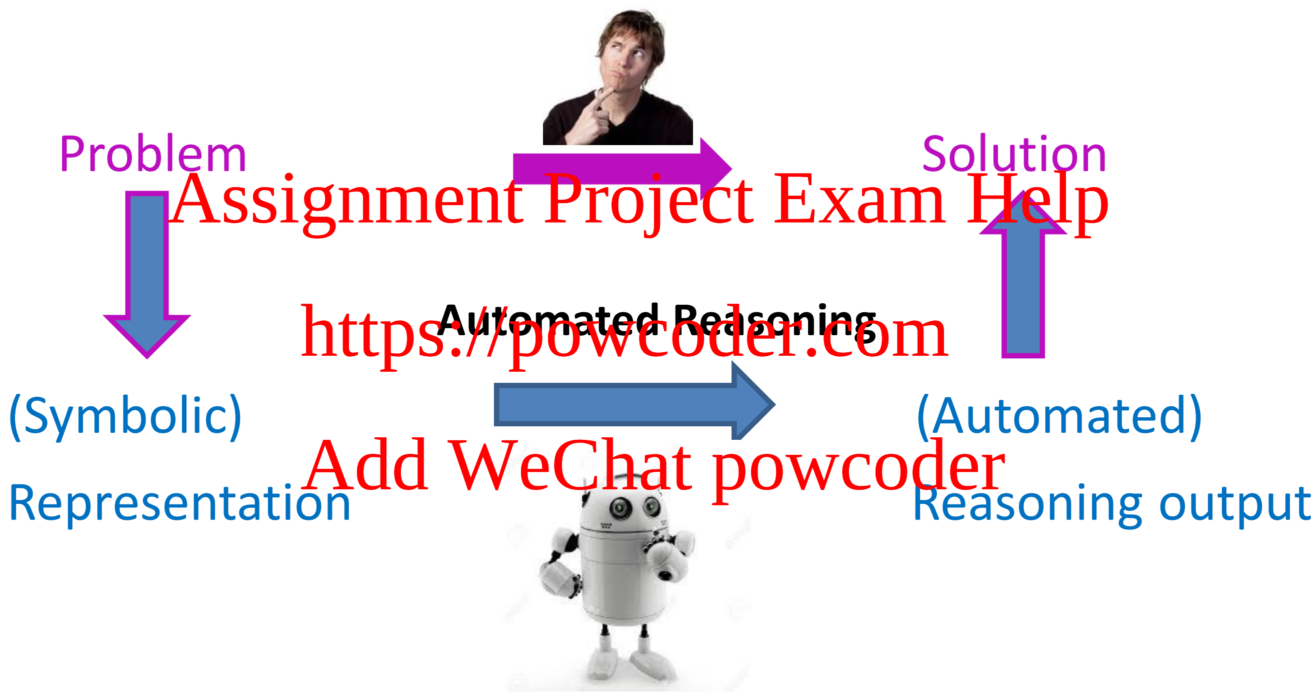
Recommended reading:
(most of) Chapters 7-9



Additional reading:
Chapter 5



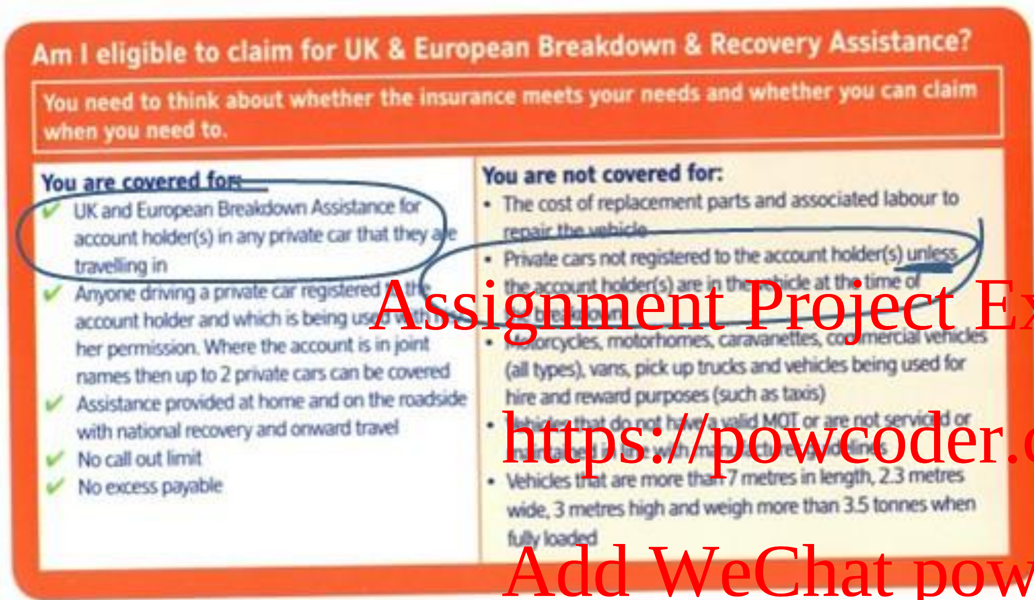
Knowledge representation and automated reasoning



In order to find a solution to a problem:

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

Example



Am I eligible to claim for UK & European Breakdown & Recovery Assistance?

You need to think about whether the insurance meets your needs and whether you can claim when you need to.

You are covered for:

- UK and European Breakdown Assistance for account holder(s) in any private car that they are travelling in
- Anyone driving a private car registered to the account holder and which is being used with his or her permission. Where the account is in joint names then up to 2 private cars can be covered
- Assistance provided at home and on the roadside with national recovery and onward travel
- No call out limit
- No excess payable

You are not covered for:

- The cost of replacement parts and associated labour to repair the vehicle
- Private cars not registered to the account holder(s) unless the account holder(s) are in the vehicle at the time of the breakdown
- Motorcycles, motorhomes, caravanettes, commercial vehicles (all types), vans, pick up trucks and vehicles being used for hire and reward purposes (such as taxis)
- Vehicles that do not have a valid MOT or are not serviced or maintained in line with manufacturer's guidelines
- Vehicles that are more than 7 metres in length, 2.3 metres wide, 3 metres high and weigh more than 3.5 tonnes when fully loaded

is Francesca covered?

Yes (or No)

Nationwide

Logic program
("pure" Prolog program)

$$covered(X) \leftarrow ah(X), tr(X,C), pr(C), not\ \neg covered(X,C)$$

$$\neg covered(X,C) \leftarrow \neg reg(C,X), not\ in(X,C)$$

$$ah(ft) \quad tr(ft, alpha) \quad pr(alpha)$$

$$\neg reg(alpha, ft) \quad in(ft, alpha)$$

$covered(ft)$?

success (or failure)

SLDNF (Prolog)

From knowledge representation and automated reasoning to…

Verification

- Model checking
- ….

Machine Learning

- Inductive Logic Programming
- Differential Inductive Logic
- Neural Inductive Logic Programming
- Logic Tensor Networks
- …

Explanation

Logic for
Knowledge Representation and (Automated) Reasoning

- 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 languages (representation)
  - automated theorem provers (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	$\nexists$ complete algorithm for arithmetic
1960	Davis/Putnam	"practical" algorithm for propositional logic
1965	Robinson	"practical" algorithm for FOL—resolution
1982	Martelli/Montanari	"practical" algorithm for unification

Clausal form

- Resolution works with (sets/conjunctions of) clauses:

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

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:

$$p_1 \wedge \ldots \wedge p_m \rightarrow q_1 \vee \ldots \vee q_n$$

often written as:

$$q_1 \vee \ldots \vee q_n \leftarrow p_1 \wedge \ldots \wedge p_m$$

Clausal form: note

- every formula in *propositional logic* is logically equivalent to a conjunction of clauses (conjunctive normal form), e.g.:

$$(A \wedge B) \vee \neg C$$

$$(A \vee \neg C) \wedge (B \vee \neg C)$$

$$(A \wedge (B \rightarrow C)) \rightarrow D$$

$$(\neg A \vee B \vee D) \wedge (\neg A \vee \neg C \vee D)$$

- every sentence in *first-order logic* can be written equivalently as a conjunction of clauses (universal quantification + conjunctive normal form + Skolemization), e.g.:

$$\exists x\, P(x)$$

$$P(SK_1)$$

$$\exists x\, (P(x) \wedge \forall y\, Q(x, y))$$

$$\forall y\, (P(SK_2) \wedge (Q(SK_2, y))$$

Resolution inference rule: propositional case

This rule combines two clauses to make a new one.

Basic propositional version:

$$\frac{\alpha \lor \beta,\ \neg \beta \lor \gamma}{\alpha \lor \gamma} \qquad \text{or equivalently} \qquad \frac{\neg \alpha \rightarrow \beta,\ \beta \rightarrow \gamma}{\neg \alpha \rightarrow \gamma}$$

It is applied repeatedly until the empty clause is derived.

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

- resolution with $\neg A \lor B$ and $\neg C \lor A$ gives $B \lor \neg C$
- resolution with C and $B \lor \neg C$ 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 (i.e. S ⊨ 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′ to obtain ☐

- *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 (with no variables): ground instantiation
- for σ a substitution, $\alpha\sigma$ is the formula obtained from α by applying σ

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

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

First-order case: 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 (the given set of sentences), 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 **{x/z, y/z}** 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

There is a (very efficient) unification algorithm which checks whether any two formulas can be unified, and produces a most general unifier if they can. (Details omitted – but Prolog implements (most of) it)

Unification is very powerful. Some examples:

p(x, y, F(z))
p(F(y),A, x)

unifier {x/F(A),y/A,z/A}

p(x, x, F(F(A)))
p(y, F(z), F(y))

unifier {x/F(A),y/F(A),z/A}

p(x, F(A), y)
p(F(y), x,B)

cannot unify: A $\neq$ B for constants A and B

p(x, x, F(A))
p(F(y), y, z)

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 $\alpha^j, \neg\beta^k$

Special case: definite clauses

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

$$p \leftarrow q_1, q_2, \ldots, q_n \qquad [\text{equivalently } \neg q_1 \vee \ldots \vee \neg q_n \vee p]$$

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

– p is the *head* and $q_1, \ldots, q_n$ the *body* of the clause.

– if n = 0, the clause $p \leftarrow$ can be written as p - called a *fact*.

Note: Definite clauses are often called '**Horn clauses**'. Strictly speaking though this is incorrect, as Horn clauses also include $\leftarrow q_1, \ldots, q_n$.

$\leftarrow q_1, \ldots, q_n$ is logically equivalent to $\neg q_1 \vee \ldots \vee \neg q_n$, which is logically equivalent to $\neg(q_1 \wedge \ldots \wedge q_n)$.

Note: a set of definite clauses is often referred to as a **(positive) logic program**

Generalized Modus Ponens (GMP)

For S a set of definite clauses, Generalized Modus Ponens is given by:

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

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

(Note: this is a special case of (several steps of) resolution.)

E.g.

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

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 (a logic program) 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?*

Definite clauses and GMP: 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.

Definite clauses for Colonel West

- 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)

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 $\leftarrow q_1, \ldots, q_n$ then apply **GMP** to derive $q_1, \ldots, q_n$ from S'

- **Forward chaining**:
  - Split S' into a set of facts E and a set of rules (definite clauses) Pr.
  - 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 $q_1, \ldots, q_n$ are in E succeed.)

Forward chaining for Colonel West

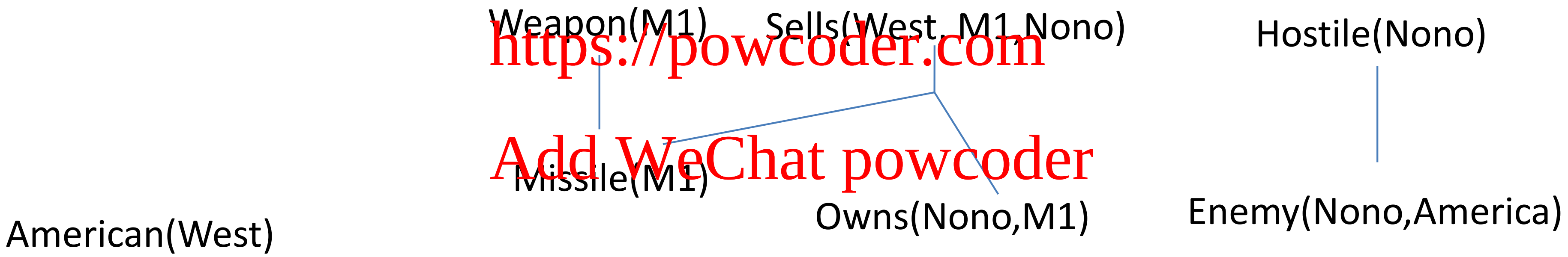
Missile(M1)

American(West)

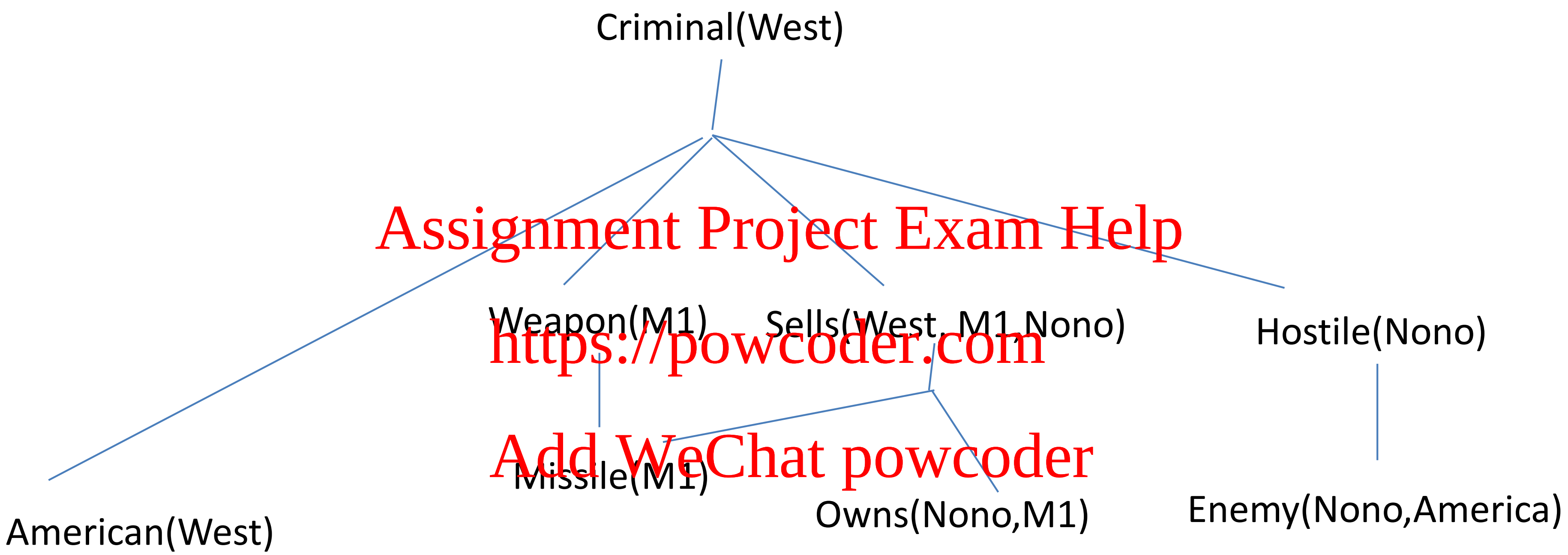
Owns(Nono,M1)

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)

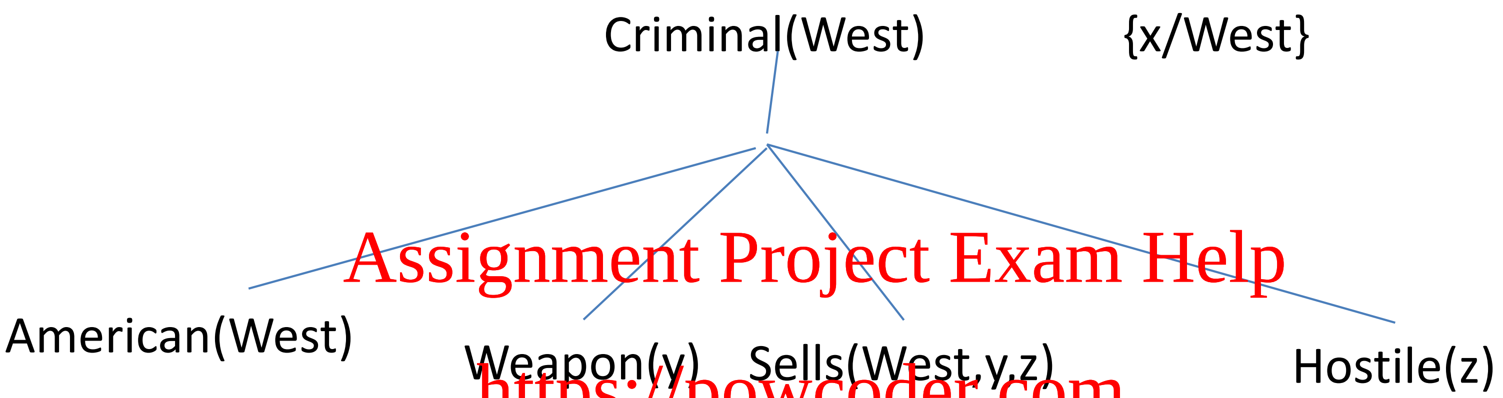
- To prove that S $\models$ P:
  1. compute the conjunctive normal form S′ of S
  2. compute the conjunctive normal form NP′ of $\neg$P
  3. If S′ is a set of definite clause and NP′ is a Horn clause $\leftarrow q_1, \ldots, q_n$ then apply **GMP backwards** to derive $q_1, \ldots, q_n$ from S′

- **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

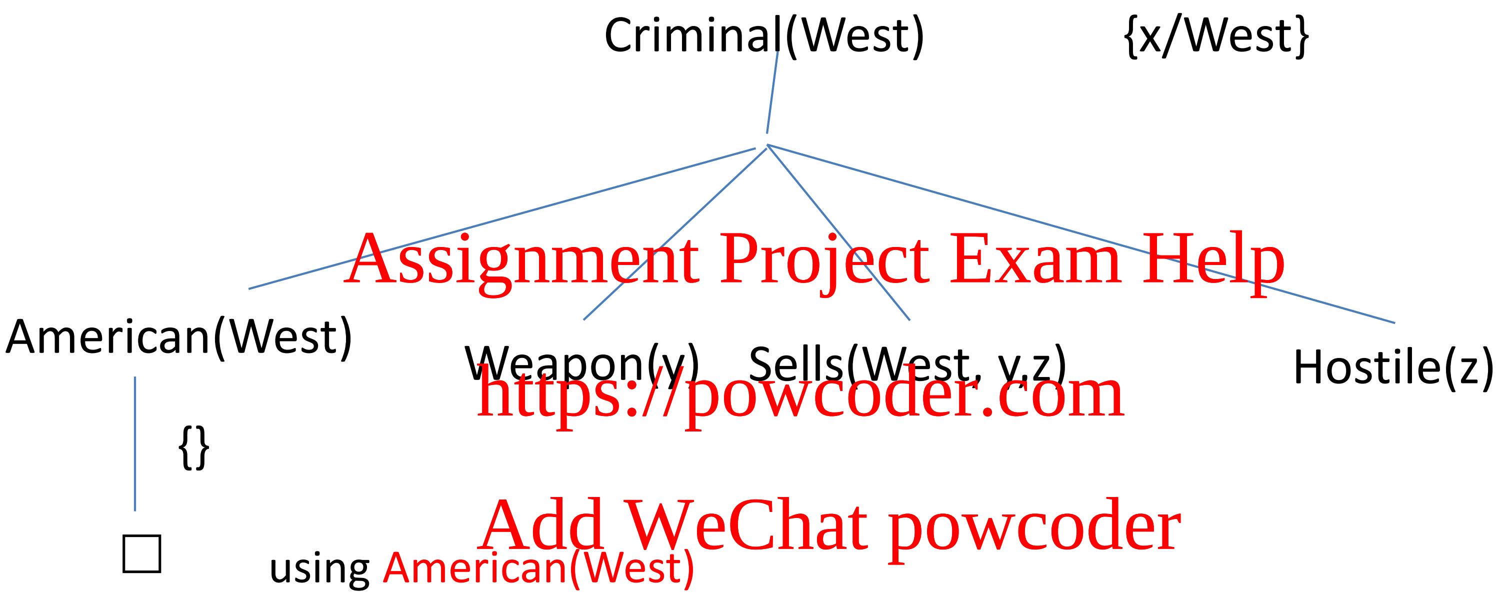
Criminal(West) {}

Backward chaining for Colonel West

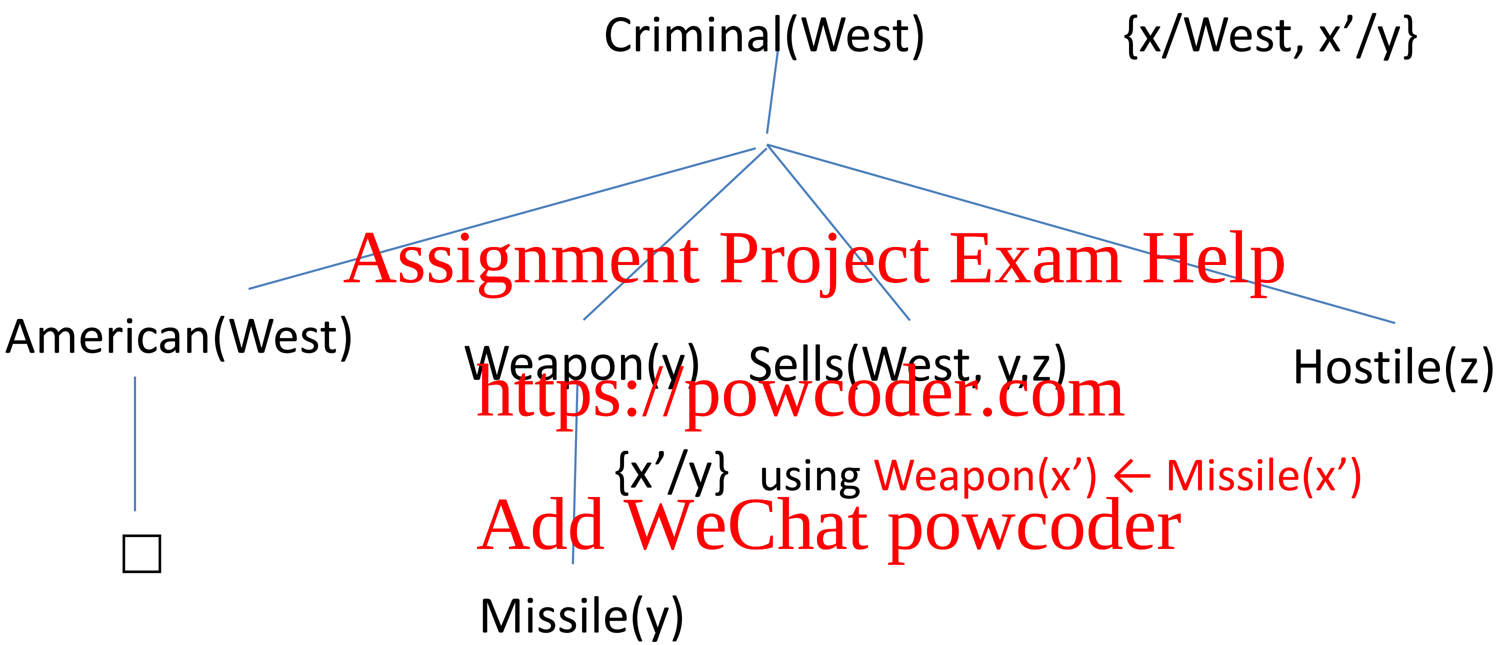


using Criminal(x) ← American(x), Weapon(y), Sells(x, y, z), Hostile(z)

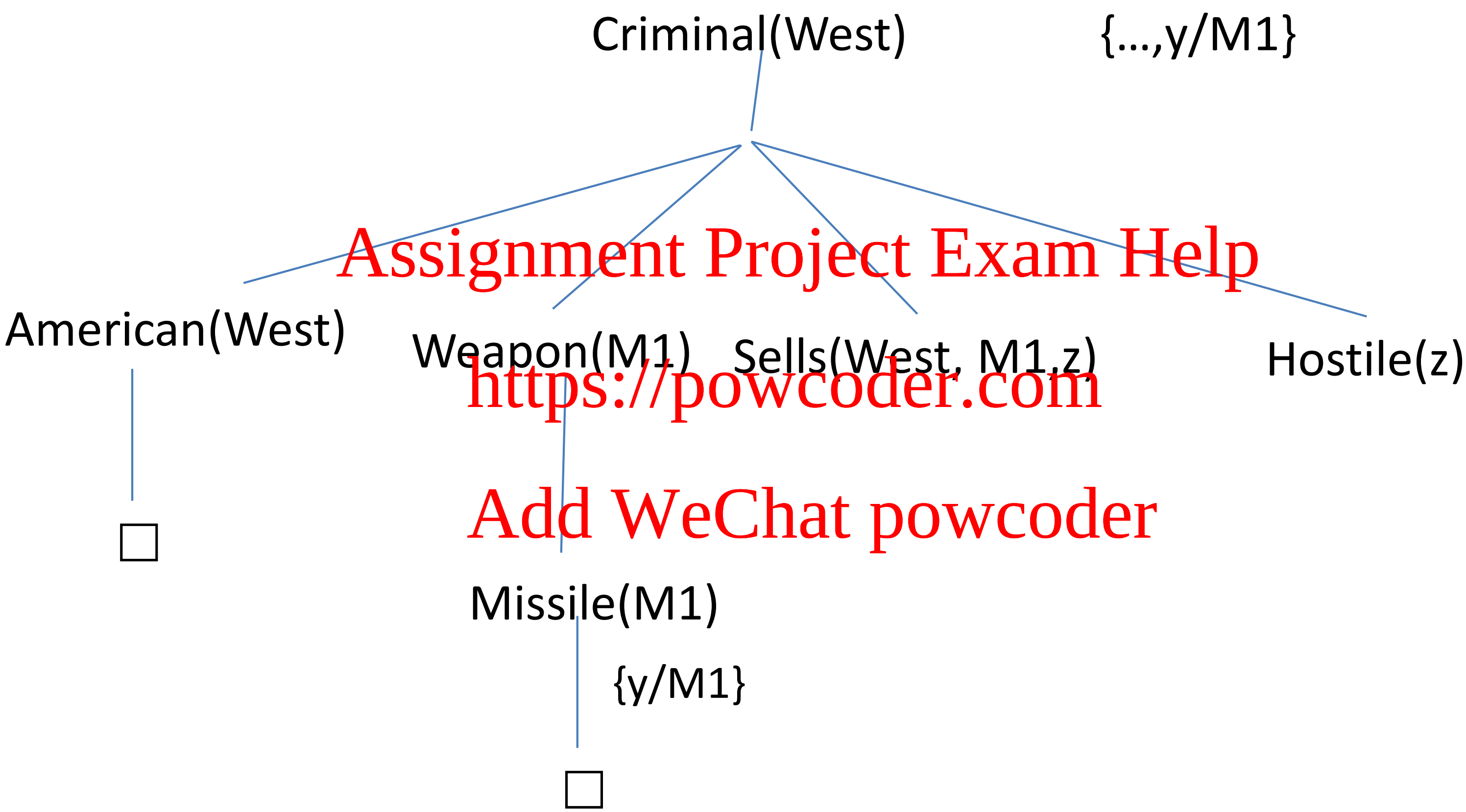
Backward chaining for Colonel West



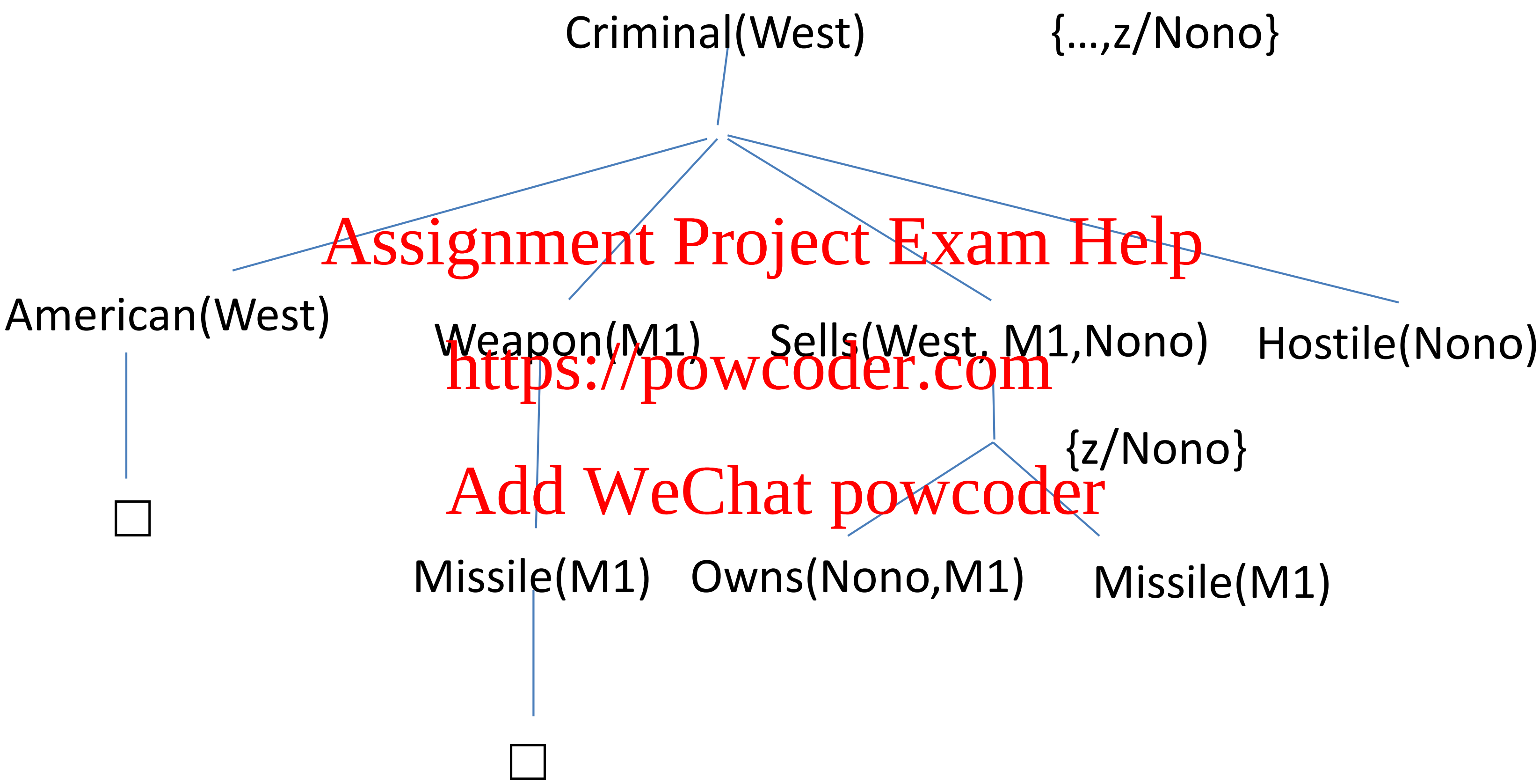
Backward chaining for Colonel West



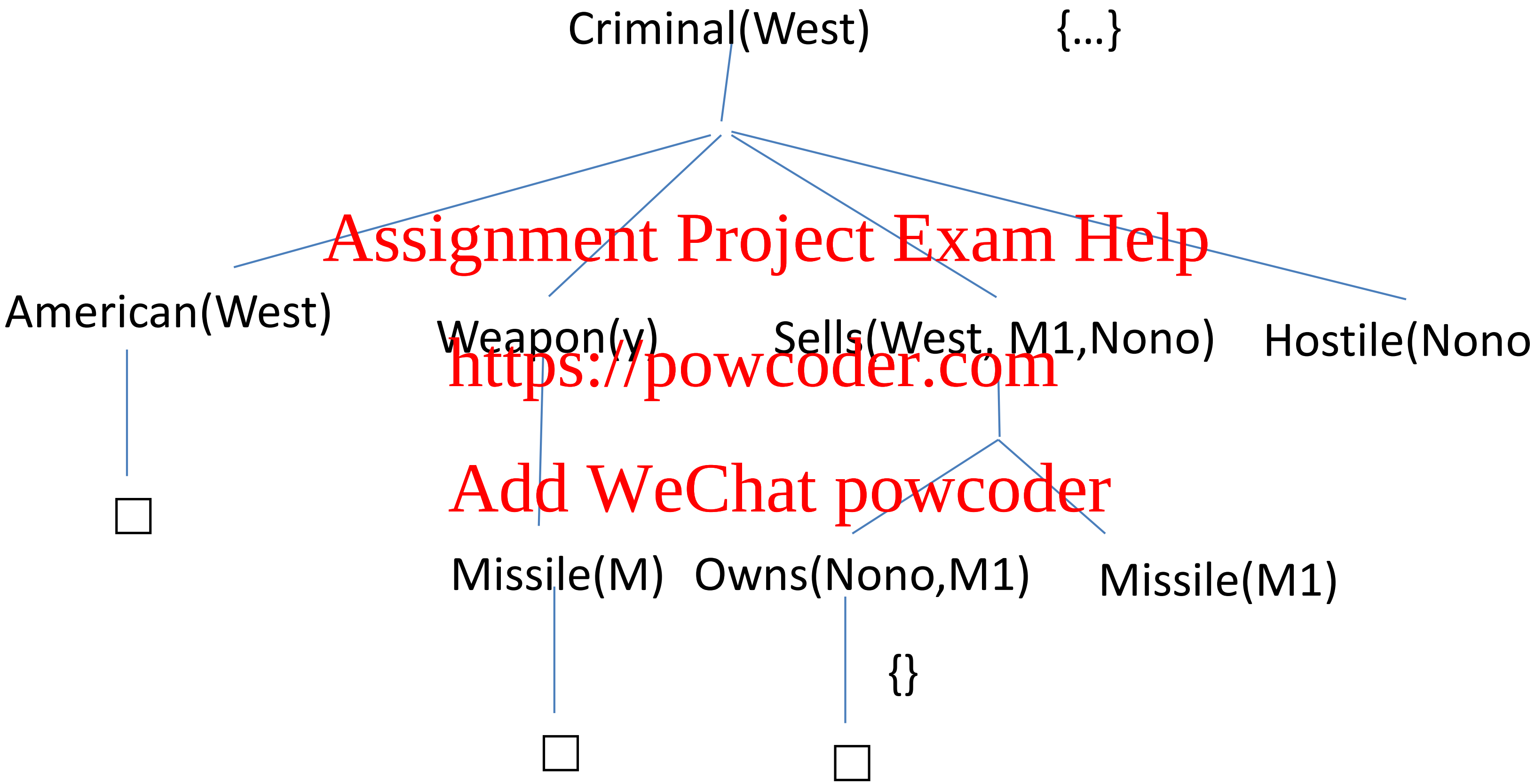
Backward chaining for Colonel West



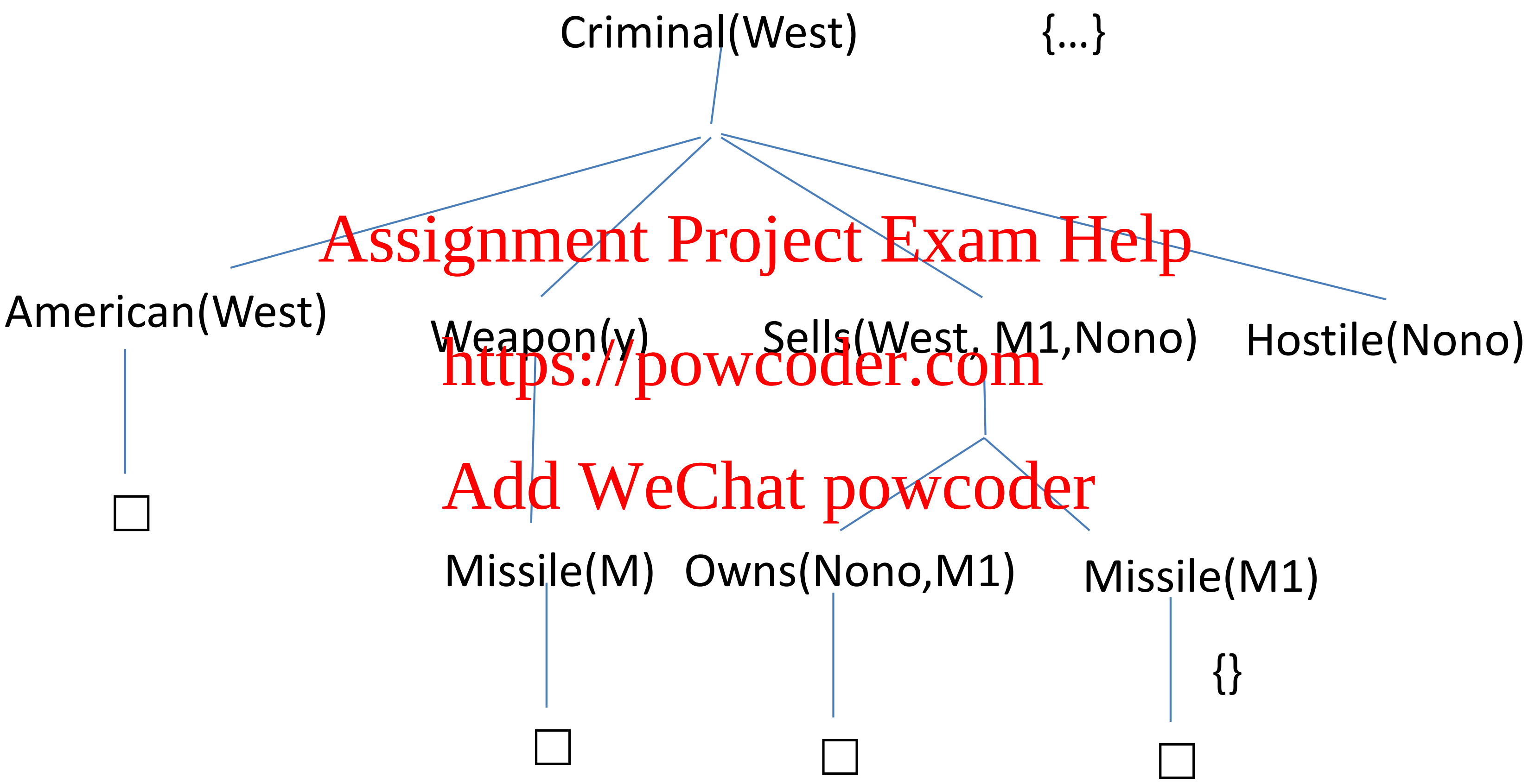
Backward chaining for Colonel West



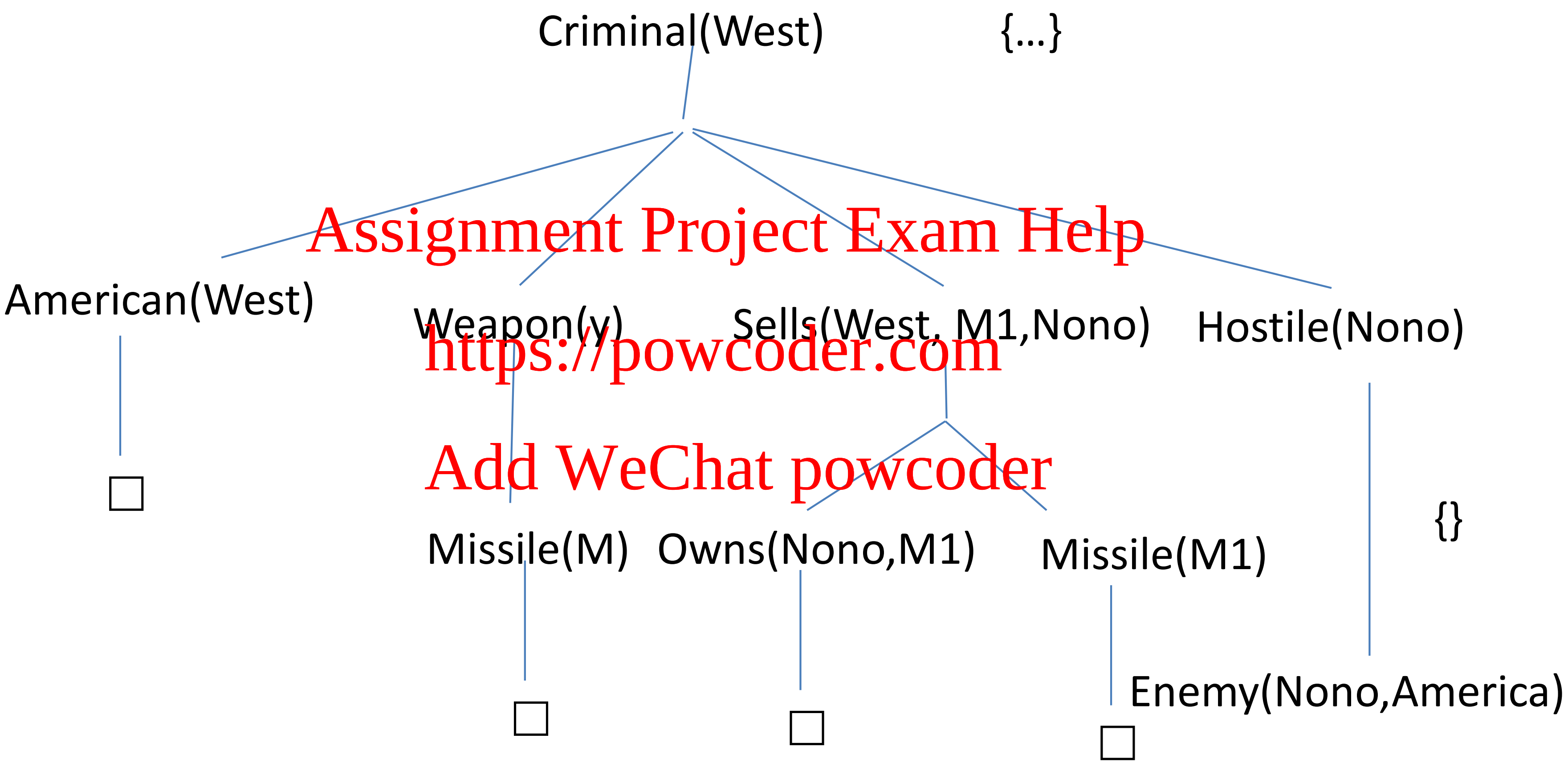
Backward chaining for Colonel West



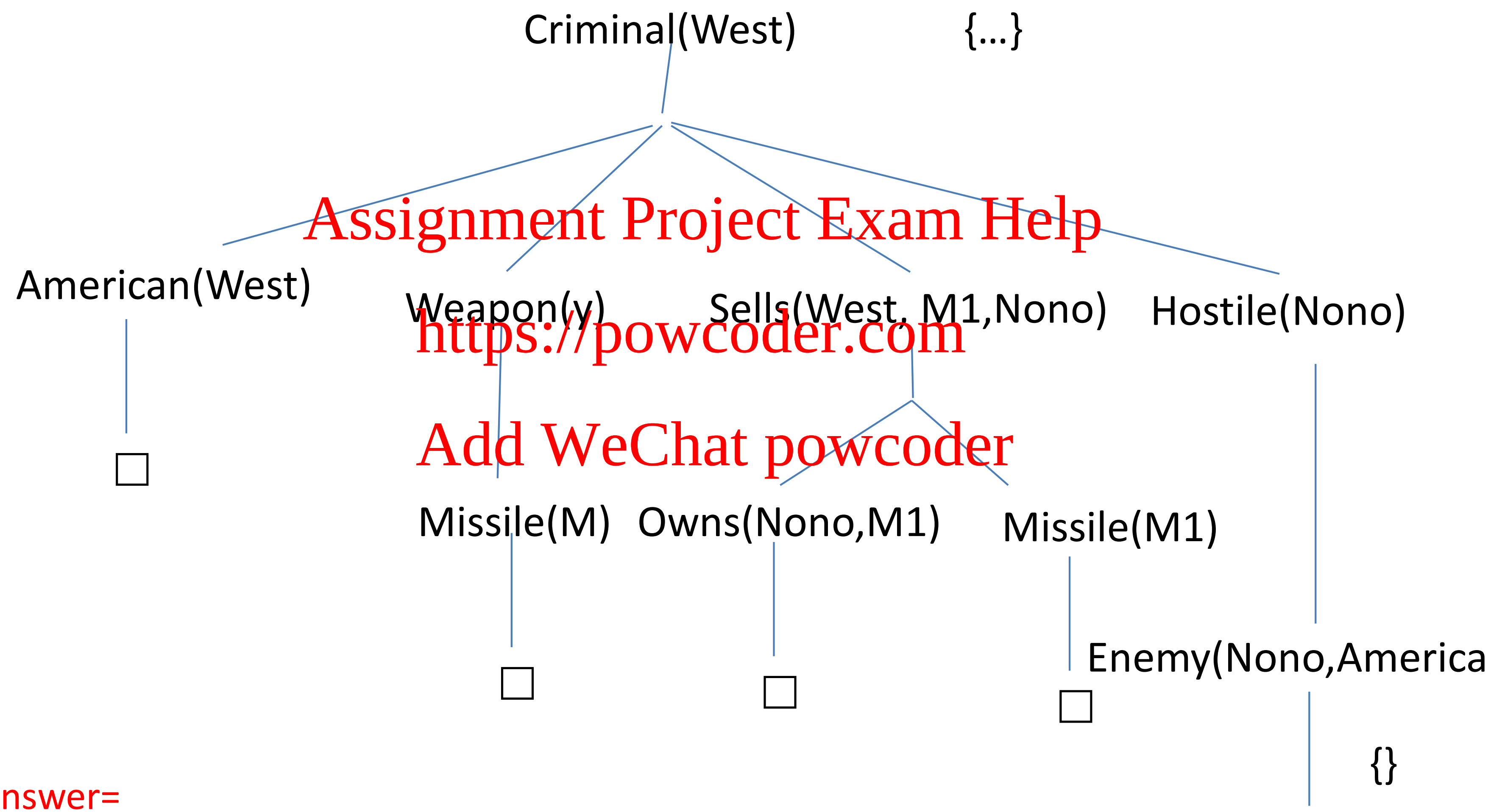
Backward chaining for Colonel West



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

← Criminal(West)
|
← American(West), Weapon(y), Sells(West, y, z),Hostile(z)
|
← Weapon(y), Sells(West, y, z),Hostile(z)
|
← 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)
|
□

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)
{} |
□

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)

□

SLD resolution

- The kind of resolution in the resolution view of Colonel West via backward chaining is SL (Selective Linear) resolution for Definite clauses – **SLD resolution**:

$$\frac{\overbrace{\neg\alpha^1 \vee \dots \vee \neg\alpha^j \vee \dots \vee \neg\alpha^m}^{\text{GOAL}}, \quad \overbrace{\alpha \vee \neg\beta^1 \vee \dots \vee \neg\beta^n}^{\text{definite clause in logic program}}}{(\neg\alpha^1 \vee \dots \vee \neg\beta^1 \vee \dots \vee \neg\beta^n \vee \dots \vee \neg\alpha^m)\theta}$$

where θ is a mgu of α^j , α

Alternative view of SLD resolution

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

$\leftarrow G$

θ_1

$\leftarrow G_1$

θ_2

$\vdots$

$\leftarrow G_{n-1}$

θ_n

$\Box$

- 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:

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

Alternative view of SLD resolution

- Each derivation step looks like this:

$$\leftarrow L_1,\ldots,L_{j-1}, B, L_{j+1},\ldots,L_n$$

θ_i | match B with $B' \leftarrow M_1,\ldots,M_k$, with $B\theta_i = B'\theta_i$

$$\leftarrow (L_1,\ldots,L_{j-1}, M_1,\ldots,M_k, L_{j+1},\ldots,L_n)\theta_i$$

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

Semantics of definite clauses/logic programs

- Classical models
- Herbrand models
- Immediate consequence operator

Note: semantically, each set of definite clauses S can be equated to the set of all its ground instances over the underlying **Herbrand universe**, i.e. the (possibly infinite) set of all ground terms that can be constructed from constant and function symbols in S

e.g. the Herbrand universe of $S=\{P(x) \leftarrow Q(F(x)), R(1) \leftarrow\}$ is $\{1, F(1), F(F(1)), \ldots\}$

From now on each set of definite clauses stands for the set of all its ground instances over its Herbrand universe

Classical models

Interpretations of (set of definite clauses) S:

- mappings from ground terms of S to elements of *some domain D* (ground terms denote elements of D)

e.g. for S={P(x) $\leftarrow$ Q(x),R(x), Q(1) $\leftarrow$, Q(Succ(1)) $\leftarrow$, R(1) $\leftarrow$}

D may be {1,2,3,…} where Succ(1) denotes 2 etc

Models of S

- interpretations of S in which every clause of S is true

Herbrand models

- These are models where ground terms denote themselves, i.e. whose domain is the Herbrand universe of (the given set of definite clauses) S

e.g. for S={P(x) ← Q(x), R(x), Q(1) ←, Q(Succ(1)) ←, R(1) ←}

both {P(1), Q(1), R(1), Q(Succ(1))}

and {P(1), Q(1), R(1), Q(Succ(1)), P(Succ(1))}

are Herbrand models (but only the former is **minimal**)

If S is a set of definite clauses, then

S has a model iff S has a Herbrand model

iff S has a minimal Herband model

So we can restrict attention to minimal Herbrand models

The immediate consequence operator

Let HB (the Herbrand Base of S) be the set of all ground atoms constructed from predicate symbols in S over the Herbrand universe of a set of definite clauses S:

for X$\subseteq HB$:

T_S(X)= {a ∈ HB | a ← b_1, …, b_m ∈ S, {b_1, …, b_m} ⊆ X}

e.g. for S={P(x) ← Q(x),R(x), Q(1) ←, Q(Succ(1)) ←, R(1) ←}

T_S({})= {R(1),Q(1), Q(Succ(1))}

T_S({R(1),Q(1)})= {R(1),Q(1),P(1), Q(Succ(1))}

T_S is continuous and admits a least fixed point, given by $T_S\uparrow^{\omega}$, and this is the minimal Herband model of S

Example of $T_S\uparrow^{\omega}$

$S=\{P(x) \leftarrow Q(x),R(x),\ Q(1) \leftarrow,\ Q(Succ(1)) \leftarrow,\ R(1) \leftarrow\}$

$T_S\uparrow^{1} = T_S(\{\}) = \{R(1),Q(1),\ Q(Succ(1))\}$

$T_S\uparrow^{2} = T_S(T_S\uparrow^{1}) = T_S(T_S(\{\})) = \{R(1),Q(1),\ Q(Succ(1)),P(1)\}$

$T_S\uparrow^{3} = T_S(T_S\uparrow^{2}) = T_S\uparrow^{2}$

...

$T_S\uparrow^{\omega} = T_S\uparrow^{2}$ = minimal Herbrand model of S

SLD resolution is complete

If S $\models$ Pσ (i.e. Pσ belongs to the minimal Herbrand model of S, or to the least fixed point of T_S) then there exists an SLD-refutation of P (to obtain □) with answer θ and a substitution ξ such that Pσ=P$\theta\xi$

Example: S={P(x) $\leftarrow$ Q(x),R(x), Q(1) $\leftarrow$, Q(Succ(1)) $\leftarrow$, R(1) $\leftarrow$}

$\leftarrow$ P(x)

$\leftarrow$ Q(x),R(x)

$\leftarrow$ R(1)

□ θ={x/1}

P(1) belongs to the minimal Herbrand model of S = {R(1),Q(1), Q(Succ(1)),P(1)} – so S $\models$ P(1)

SLD resolution is complete

If S $\models$ Pσ (i.e. Pσ belongs to the minimal Herbrand model of S, or to the least fixed point of T_S) then there exists an SLD-refutation of P (to obtain □) with answer θ and a substitution ξ such that Pσ=P$\theta\xi$

Example: S={P(x,y) $\leftarrow$ Q(x), Q(1) $\leftarrow$, R(2) $\leftarrow$}

$\leftarrow$ P(x,y)

$\leftarrow$ Q(x)

□ θ={x/1}

P(1,2) belongs to the minimal Herbrand model of S = {Q(1), R(2),P(1,1),P(1,2)}, ξ={y/2}

Summary - Logic for Knowledge Representation and Automated Reasoning

- Automated reasoning in FOL often needs one (or more) of:

  - Unification

  - Resolution
  - Generalized Modus Ponens/SLD resolution - Definite Clauses

  - Forward chaining (bottom-up computations)
  - Backward chaining (top-down, goal-directed computations)
- Completeness of (SLD) resolution

Note: Logic programming vs Prolog

- 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 (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.
```