Predicate Logic

1 Syntax and Semantics

The syntax of first order logic (FOL) is defined over a signature $\sigma = (V, F, P)$ where

- $V = \{x_1, x_2, \cdots\}$ is a countable set of variable symbols,
- $F = \{f_1^{k_1}, f_2^{k_2}, \cdots\}$ is a countable set of function symbols with arities $k_1, k_2, \cdots$, respectively and
- $P = \{p_1^{k_1}, p_2^{k_2}, p_3, \cdots\}$ countable set of predicate symbols with arities $k_1, k_2, \cdots$, respectively.
-

and propositional connectives $\{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$. A function symbol of arity 0 is called a **constant**. We use the following conventions:

- u, v, w, x, y, z stand for arbitrary variable symbols from the set V.
- a, b, c stand for arbitrary constants.
- f, g, h stand for arbitrary function symbols where the respective arities are inferred from the context.
- p, q, r, s stand for arbitrary predicate symbols where the respective arities are inferred from the context.

The set of **terms** is defined inductively as the smallest set satisfying the following conditions:

1. every variable $x \in V$ is a term,
2. if f is a function symbol in F with arity k and $t_1, t_2, \cdots, t_k$ are k terms then $f^k(t_1, t_2, \cdots, t_k)$ is a term and

3. nothing else is a term.

Let T be the set of all terms constructed using the above rules. The set of all **well-formed formulas** (**wff**s) of first order logic are defined inductively as the smallest set satisfying the following conditions:

1. If $p \in P$ is a predicate symbol with arity k and if $t_1, t_2, \cdots, t_k$ are terms then $p(t_1, t_2, \cdots, t_k)$ is a wff,
2. If α is a wff then $(\neg\alpha)$ is a wff,
3. If α, β are wffs then so are $(\alpha \vee \beta), (\alpha \wedge \beta), (\alpha \Rightarrow \beta)$ and $(\alpha \Leftrightarrow \beta)$,
4. If x is a variable and α is a wff then $(\forall x)\alpha$ and $(\exists x)\alpha$ are wffs and
5. Nothing else is a wff.

Atomic formulas are exactly those formulas that are built up using rule 1. Let Ψ be the set of all wffs constructed from the signature σ and the connectives $\{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$ using the above rules. We use wff and formula interchangeably. Ψ can alternatively be defined via Backus-Naur Form as follows:

$$\alpha, \beta \in \Phi ::= p(t_1, \cdots, t_k) \mid (\neg\alpha) \mid (\alpha \vee \beta) \mid (\alpha \wedge \beta) \mid (\alpha \Rightarrow \beta) \mid (\alpha \Leftrightarrow \beta) \mid (\forall x)\alpha \mid (\exists x)\alpha.$$

where $p \in P$ of arity k and $t_1, \cdots, t_k \in T$ are terms. All occurrences of a variable in a formula are distinguished into **free** and **bound** occurrences. An occurrence of the variable x in a formula α is bound if x occurs within a subformula of α of the form $(\exists x)\beta$ or $(\forall x)\beta$. The same variable x can occur both free and bound in a formula α.

A formula without the occurrence of a free variable is called **closed**. The symbols $\exists$ and $\forall$ are called **quantifiers** where $\exists$ is the existential quantifier and $\forall$ is the universal quantifier. The matrix of a formula α, denoted by α^*, is obtained by removing every occurrences of a quantifier and the variable that follows the quantifier.

Example 1.1 *Consider the formula*

$$\alpha = (\exists x)p(x, f(y))) \wedge (\neg(\forall y)q(y, g(a, h(z))))$$

- *The scope of the quantifier* $(\exists x)$ *is the subformula* $p(x, f(y))$
- *The scope of the quantifier* $(\forall y)$ *is the subformula* $q(y, g(a, h(z)))$
- *The occurrences of* x *in* α *are all bound.*

- *The occurrences of y in α are bound as well as free.*
- *The occurrences of x in α are all free.*
- *The matrix of the formula α is*

$$\alpha^* = p(x, f(y))) \wedge (\neg q(y, g(a, h(z))))$$

The semantics of predicate logic is defined via structures.

Definition 1.2 *A* **structure** *is a pair $A = (U_A, I_A)$ where U_A is an arbitrary, non-empty set and is called the* **ground set** *or* **universe** *of A. The* **interpretation** *I_A is a mapping that maps*

- *each k-ary predicate symbol p to a k-ary predicate on U_A, denoted by $I_A(p)$ (or p^A)*
- *each k-ary function symbol to a k-ary function on U_A, denoted by $I_A(f)$ (or f^A) and*
- *each variable x to an element of U_A, denoted by $I_A(x)$ (or x^A).*

Let α be a formula and $A = (U_A, I_A)$ be a structure. A is called suitable for α if I_A is defined for all predicate symbols occurring α, all function symbols occurring in α and all variables that occur free in α.

Let α be a formula and $A = (U_A, I_A)$ be a structure suitable to α. For each term t occurring in α we can denote its **value** under A by $A(t)$ (or t^A) and define it inductively as follows:

1. if t is a variable then $A(t) = I_A(t)$,
2. if t has the form $f(t_1, \cdots, t_k)$ then $A(t) = I_A(f)(A(t_1), \cdots, A(t_k))$

Similarly, we define the **truth value** of the formula α under the structure A, denoted by $A(\alpha)$, by an inductive definition:

- $$A(p(t_1, \cdots, t_k)) = \begin{cases} T & (A(t_1), \cdots, A(t_k)) \in I_A(p) \\ F & \text{otherwise} \end{cases}$$
- $$A(\neg\beta) = \begin{cases} T & A(\beta) = F \\ F & A(\beta) = T \end{cases}$$
- $$A(\alpha \vee \beta) = \begin{cases} T & \text{when } A(\alpha) = T \text{ or } A(\beta) = T \\ F & \text{otherwise} \end{cases}$$

- $$A(\alpha \wedge \beta) = \begin{cases} T & \text{when } A(\alpha) = T \text{ and } A(\beta) = T \\ F & \text{otherwise} \end{cases}$$

- $$A(\alpha \Rightarrow \beta) = \begin{cases} F & \text{when } A(\alpha) = T \text{ and } A(\beta) = F \\ T & \text{otherwise} \end{cases}$$

- $$A(\alpha \Leftrightarrow \beta) = \begin{cases} T & \text{when } A(\alpha) = A(\beta) \\ F & \text{otherwise} \end{cases}$$

- $$A(\forall x)\alpha = \begin{cases} T & \text{if for every } d \in U_A,\ A_{[x/d]}(\alpha) = T \\ F & \text{otherwise} \end{cases}$$

  Here, $A_{[x/d]}$ is the structure A' which is identical to A with the exception of $x^{A'}$: No matter whether I_A is defined for x or not we let $x^{A'} = d$.

- $$A(\exists x)\alpha = \begin{cases} T & \text{if for some } d \in U_A,\ A_{[x/d]}(\alpha) = T \\ F & \text{otherwise} \end{cases}$$

We denote $A(\alpha) = T$ by $A \models \alpha$ (A models α) and $A(\alpha) = F$ by $A \not\models \alpha$ (A does not model α).

Given a formula $\alpha \in \Psi$, we say that α is **satisfiable** if there exists a structure $A = (U_A, I_A)$ such that $A \models \alpha$. Similarly, we say that α is **unsatisfiable** if there **does not** exist a structure $A = (U_A, I_A)$ such that $A \models \alpha$. We say that α is **valid** if for every possible structure $A = (U_A, I_A)$, $A \models \alpha$.

Example 1.3 *Consider the formula $\alpha = (\exists x)p(x, f(y))$. The occurrence of x is bound whereas that of y is free. We want to define a structure $A = (U_A, I_A)$ suitable for α such that $A \models \alpha$.*

- *Let $U_A = \{d_1, d_2\}$.*
- *As y is free y^A should be defined, let $y^A = d_1$.*
- *We assign the following function to f, $f^A : U_A \to U_A$ such that $f(d_1) = d_2$ and $f(d_2) = d_1$.*
- *We assign the following predicate to p, $p^A \subseteq U_A \times U_A$ such that $p^A = \{(d_1, d_1), (d_2, d_2)\}$.*

- *Observe that* $A_{[x/d_1]}(p(x, f(y))) = F$ *but* $A_{[x/d_2]}(p(x, f(y))) = T$.
- $A_{[x/d_1]}(p(x, f(y))) = F$ *because* $(d_1, d_2) \notin p^A$. $A_{[x/d_1]}$ *assigns* d_1 *to* x *as well as* y *and* $A(f(y)) = d_2 = A_{[x/d_2]}(f(y))$.
- $A_{[x/d_2]}(p(x, f(y))) = T$ *because* $(d_2, d_2) \in p^A$. $A_{[x/d_2]}$ *assigns* d_2 *to* x *whereas* y^A *remains* d_1 *and* $A(f(y)) = d_2 = A_{[x/d_2]}(f(y))$.
- $\because d_2$ *is witness which makes* $p(x, f(y))$ *true, i.e.,* $A_{[x/d_2]}(p(x, f(y))) = T$ *we have* $A(\alpha) = T$.

A valid formula may also be called a **tautology** and an unsatisfiable formula a **contradiction**. For an arbitrary α, $(\alpha \vee \neg\alpha)$ is the simplest example of a valid tautology, whereas $(\alpha \wedge \neg\alpha)$ is the simplest example of a contradiction. They are, respectively, denoted by $\top$ and $\bot$.

Exercise 1.4 *A formula* α *is a tautology if and only if* $\neg\alpha$ *is unsatisfiable.*

We can extend the satisfiability relation to set of formulas. Let $\Delta = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$ be a set of propositional formulas. A valuation $A : P \to \{T, F\}$ models Δ (written as $A \models \Delta$) if $A \models \alpha_1$ and $A \models \alpha_2$ and $\cdots$ and $A \models \alpha_n$. Equivalently, $A \models \Delta$ iff $A \models (\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n)$.

Definition 1.5 (Semantic Entailment) *Given* $\Delta = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$ *and another formula* α*, we say that* α *is a consequence of* Δ *(or* Δ *semantically entails* α *denoted by* $\Delta \models \alpha$*) if for every valuation* A*, if* $A \models \Delta$ *then* $A \models \alpha$.

2 Equivalences and Normal Forms

All the equivalences of the PL remain valid in FOL and can be freely used to manipulate FOL formulas in order to transform them into their normal forms. Apart from propositional equivalences we have, in FOL, equivalences which involve quantifiers, they are as follows:

$$\neg(\forall x)\alpha \equiv (\exists x)\neg\alpha$$

$$\neg(\exists x)\alpha \equiv (\forall x)\neg\alpha$$

The following equivalences encode the property that scope of a quantifier can be enlarged to include another subformula, say β, if the variable bounded by that quantifier does not figure free in β.

$$((\forall x)\alpha \wedge \beta) \equiv (\forall x)(\alpha \wedge \beta) \text{ where } x \text{ does not occur free in } \beta.$$

$$((\forall x)\alpha \vee \beta) \equiv (\forall x)(\alpha \vee \beta) \text{ where } x \text{ does not occur free in } \beta.$$

$$((\exists x)\alpha \wedge \beta) \equiv (\exists x)(\alpha \wedge \beta) \text{ where } x \text{ does not occur free in } \beta.$$

$$((\exists x)\alpha \vee \beta) \equiv (\exists x)(\alpha \vee \beta) \text{ where } x \text{ does not occur free in } \beta.$$

The following equivalences mean that universal quantifiers distribute over $\wedge$ and existential quantifiers distribute over $\vee$.

$$((\forall x)\alpha \wedge (\forall x)\beta) \equiv (\forall x)(\alpha \wedge \beta) \text{ where } x \text{ occurs free in } \beta.$$

$$((\exists x)\alpha \vee (\exists x)\beta) \equiv (\exists x)(\alpha \vee \beta) \text{ where } x \text{ occurs free in } \beta.$$

The following equivalences encode the property that successive quantifiers of same type can interchange their places.

$$(\forall x)(\forall y)\alpha \equiv (\forall y)(\forall x)\alpha$$

$$(\exists x)(\exists y)\alpha \equiv (\exists y)(\exists x)\alpha$$

The following non equivalences represent the fact that universal quantifiers do not distribute over $\vee$ and existential quantifiers do not distribute over $\wedge$.

$$((\forall x)\alpha \vee (\forall x)\beta) \not\equiv (\forall x)(\alpha \vee \beta) \text{ where } x \text{ occurs free in } \beta.$$

$$((\exists x)\alpha \wedge (\exists x)\beta) \not\equiv (\exists x)(\alpha \wedge \beta) \text{ where } x \text{ occurs free in } \beta.$$

2.1 Substitution

In order to construct normal forms of a given FOL formula we may need to rename variables. In general, we can substitute variables by terms (variables or function applications).

Definition 2.1 (Substitution) *Let α be a formula, x a variable, and t a term. Then, $\alpha[x/t]$ denotes the formula, obtained from α, by substituting t for every free occurrence of x in α.*

We treat *substitutions* (denoted by $sub = [x/t]$) as independent objects which describe mappings from the set of formulas to set of formulas. Such substitutions can be concatenated, e.g. $sub = [x/t_1][y/t_2]$ describes the effect of first substituting all free occurrences of x by t_1 and then substituting all free occurrences of y by t_2.

2.2 Rectified Normal Form

A formula of predicate logic is in rectified normal form if

- negations are applied only at the predicates,
- no variable occurs both bound and free and
- all quantifiers in the formula refer to different variables.

Let us work out the following example to understand how an arbitrary FOL formula is rectified.

Example 2.2

$$\neg\Big((\exists x)\big((\forall y)p(x,y,y) \wedge (\forall y)(\exists z)p(y,z,x)\big)\Big)$$

- *Push the negation to the predicate level. We use the equivalences* $\neg(\forall x)\alpha \equiv (\exists x)\neg\alpha$ *and* $\neg(\exists x)\alpha \equiv (\forall x)\neg\alpha$ *for these transformations.*

$$\Big((\forall x)\neg\big((\forall y)p(x,y,y) \wedge (\forall y)(\exists z)p(y,z,x)\big)\Big)$$

$$\Big((\forall x)\big(\neg(\forall y)p(x,y,y) \vee \neg(\forall y)(\exists z)p(y,z,x)\big)\Big)$$

$$\Big((\forall x)\big((\exists y)\neg p(x,y,y) \vee (\exists y)\neg(\exists z)p(y,z,x)\big)\Big)$$

$$\Big((\forall x)\big((\exists y)\neg p(x,y,y) \vee (\exists y)(\forall z)\neg p(y,z,x)\big)\Big)$$

- *The subformula* $(\exists y)\neg p(x,y,y) \vee (\exists y)(\forall z)\neg p(y,z,x)$ *is of the form* $(\exists x)\alpha \vee (\exists)\beta$*, where* $\alpha = \neg p(x,y,y)$ *and* $\beta = (\forall z)\neg p(y,z,x)$*. Using the equivalence* $(\exists x)\alpha \vee (\exists)\beta \equiv (\exists x)(\alpha \vee \beta)$ *we can convert the above formula to*

$$\Big((\forall x)\big((\exists y)(\neg p(x,y,y) \vee (\forall z)\neg p(y,z,x))\big)\Big)$$

- *Thus we have transformed the original formula to rectified normal form where negations are applied only at the predicate level, no variable occurs both bound and free and all quantifiers in the formula refer to different variables.*

We work out another example where we need to rename bound variables to find the rectified normal form.

Example 2.3

$$(\forall x)(\exists y)p(x, f(y)) \wedge (\forall y)(q(x, y) \wedge r(x))$$

- *The same variable y occurs in two different quantifiers. Therefore, the above formula is not in rectified normal form.*
- *Note that the scope of $(\exists y)$ is the subformula $p(x, f(y))$ and the scope of $(\forall y)$ is $(q(x, y) \wedge r(x))$.*
- *We can rename one of the bound occurrences of y as follows:*

$$(\forall x)(\exists z)p(x, f(z)) \wedge (\forall y)(q(x, y) \wedge r(x)).$$

- *Note that y in $(\forall y)(q(x, y) \wedge r(x))$ does not change because this subformula is not in the scope of the original existential quantifier as already pointed out.*
- *Furthermore, note that, the scope of $(\forall x)$ is $(\exists y)p(x, f(y))$. Hence, occurrences of x in $(q(x, y) \wedge r(x))$ are free. Therefore, x occurs free as well as bound.*
- *This can be rectified by renaming the bound occurrences of x as follows:*

$$(\forall w)(\exists z)p(w, f(z)) \wedge (\forall y)(q(x, y) \wedge r(x)).$$

- *Thus we have transformed the original formula to rectified normal form where negations are applied only at the predicate level, no variable occurs both bound and free and all quantifiers in the formula refer to different variables.*

2.3 Prenex Normal Form

A formula of the predicate logic is in prenex normal form if it is written as a string of quantifiers (referred to as the **prefix**) followed by a quantifier-free part (referred to as the **matrix**).

In order to convert a rectified formula to prenex form we use the following equivalences of predicate logic:

$$((\forall x)\alpha \wedge \beta) \equiv (\forall x)(\alpha \wedge \beta) \text{ where } x \text{ does not occur free in } \beta.$$

$$((\forall x)\alpha \vee \beta) \equiv (\forall x)(\alpha \vee \beta) \text{ where } x \text{ does not occur free in } \beta.$$

$((\exists x)\alpha \wedge \beta) \equiv (\exists x)(\alpha \wedge \beta)$ where x does not occur free in β.

$((\exists x)\alpha \vee \beta) \equiv (\exists x)(\alpha \vee \beta)$ where x does not occur free in β.

The above two formulas which we converted to rectified normal form can be, in turn, transformed to prenex form as follows: Consider the first formula.

$$(\forall x)(\exists y)\big(\neg p(x,y,y) \vee (\forall z)\neg p(y,z,x)\big)$$

This is equivalent to the following formula, using commutativity equivalence of PL,

$$(\forall x)(\exists y)\big((\forall z)\neg p(y,z,x) \vee \neg p(x,y,y)\big)$$

The subformula $\big((\forall z)\neg p(y,z,x) \vee \neg p(x,y,y)\big)$ is of the form $(\forall z)\alpha \vee \beta$ where z does not figure in β, which can be converted to the formula $(\forall z)(\alpha \vee \beta)$ using the second equivalence given above. Therefore, we get the following transformed formula:

$$(\forall x)(\exists y)(\forall z)\big(\neg p(y,z,x) \vee \neg p(x,y,y)\big)$$

Clearly, this formula is in prenex normal form, where the prefix is $(\forall x)(\exists y)(\forall z)$ and the matrix is $\big(\neg p(y,z,x) \vee \neg p(x,y,y)\big)$.

Now, we move to the second rectified formula,

$$(\forall w)(\exists z)p(w,f(z)) \wedge (\forall y)(q(x,y) \wedge r(x)).$$

This formula is of the form $(\forall w)\alpha \wedge \beta$, where $\alpha = (\exists z)p(w,f(z))$ and $\beta = (\forall y)(q(x,y) \wedge r(x))$. Using the first equivalence given above we can transform the given formula to

$$(\forall w)\Big((\exists z)p(w,f(z)) \wedge (\forall y)(q(x,y) \wedge r(x))\Big).$$

The subformula $(\exists z)p(w,f(z)) \wedge (\forall y)(q(x,y) \wedge r(x))$ is of the form $(\exists z)\alpha \wedge \beta$, where $\alpha = p(w,f(z))$ and $\beta = (\forall y)(q(x,y) \wedge r(x))$. This subformula can be transformed to $(\exists z)(p(w,f(z)) \wedge (\forall y)(q(x,y) \wedge r(x)))$. Therefore, original formula is equivalent to

$$(\forall w)\Big((\exists z)\big(p(w,f(z)) \wedge (\forall y)(q(x,y) \wedge r(x))\big)\Big).$$

Using the commutativity equivalence of PL, we get

$$(\forall w)\Big((\exists z)\big((\forall y)(q(x,y) \wedge r(x)) \wedge p(w,f(z))\big)\Big).$$

The subformula $\big((\forall y)(q(x,y) \wedge r(x)) \wedge p(w, f(z))\big)$ is of the form $(\forall y)\alpha \wedge \beta$, where y does not occur in β and $\alpha = (q(x,y) \wedge r(x))$ and $\beta = p(w, f(z))$. As explained above the given subformula is equivalent to $(\forall y)\big((q(x,y) \wedge r(x)) \wedge p(w, f(z))\big)$. Consequently, the original formula is transformed to,

$$(\forall w)\Big((\exists z)(\forall y)\big((q(x,y) \wedge r(x)) \wedge p(w, f(z))\big)\Big).$$

Removing redundant parentheses and using associative property of PL we get,

$$(\forall w)(\exists z)(\forall y)\big(q(x,y) \wedge r(x) \wedge p(w, f(z))\big).$$

2.4 Skolem Normal Form

A formula in prenex normal form is in Skolem form if it is of the kind

$$(\forall y_1)(\forall y_2)\cdots(\forall y_m)\alpha^*.$$

An skolem normal form formula contains only universal quantifiers in its prefix. If there is an existential quantifier $(\exists u)$ in the prefix with k universal quantifiers preceding it then it can be removed by introducing a fresh function h with arity k. Every FOL formula in prenex form can be **Skolemized** to its equivalent Skolem normal form using an algorithm which we explain via an example, $(\forall x)(\exists y)(\forall z)(\exists w)\big(\neg p(z,w) \vee q(f(x), y)\big)$.

- We scan the prefix of the formula from left to right and consider the first existential quantifier we encounter, in the present case it is $(\exists y)$, replace y in the matrix by $h(x)$. The modified formula is

$$(\forall x)(\forall z)(\exists w)\big(\neg p(z,w) \vee q(f(x), h(x))\big).$$

- The next existential quantifier we encounter is $(\exists w)$ with $(\forall x)(\forall z)$ preceding it. We replace w in the matrix by $i(x,z)$. The transformed formula is,

$$(\forall x)(\forall z)\big(\neg p(z, i(x,z)) \vee q(f(x), h(x))\big).$$

If an existential quantifier, say $(\exists u)$, occurs in the front of the prefix of a formula in the prenex form then $(\exists v)$ is removed by substituting u by a constant, say a, in the matrix of the formula. For example consider the negation of the above prenex formula,

$$\neg(\forall x)(\exists y)(\forall z)(\exists w)\big(\neg p(z,w) \vee q(f(x), y)\big)$$

This formula is not even rectified. We first rectify it and then convert it back to prenex form. Thereafter, we shall skolemize it. First, we push the negation inside so that it is asserted only at the predicate level.

$$(\exists x)\neg(\exists y)(\forall z)(\exists w)\big(\neg p(z,w) \vee q(f(x),y)\big)$$

$$(\exists x)(\forall y)\neg(\forall z)(\exists w)\big(\neg p(z,w) \vee q(f(x),y)\big)$$

$$(\exists x)(\forall y)(\exists z)\neg(\exists w)\big(\neg p(z,w) \vee q(f(x),y)\big)$$

$$(\exists x)(\forall y)(\exists z)(\forall w)\neg\big(\neg p(z,w) \vee q(f(x),y)\big)$$

Applying De Morgan's law we get,

$$(\exists x)(\forall y)(\exists z)(\forall w)\big(\neg\neg p(z,w) \wedge \neg q(f(x),y)\big)$$

which is equivalent to

$$(\exists x)(\forall y)(\exists z)(\forall w)\big(p(z,w) \wedge \neg q(f(x),y)\big)$$

This formula is now in prenex form which we can skolemize.

- We scan the prefix of the formula from left to right and consider the first existential quantifier we encounter, in the present case it is $(\exists x)$ with no preceding universal quantifier. We replace x in the matrix by a constant, say a. The modified formula is

$$(\forall y)(\exists z)(\forall w)\big(p(z,w) \wedge \neg q(f(a),y)\big).$$

- The next existential quantifier we encounter is $(\exists z)$ with $(\forall y)$ preceding it. We replace z in the matrix by $j(y)$. The transformed formula is

$$(\forall y)(\forall w)\big(p(j(y),w) \wedge \neg q(f(x),y)\big).$$

3 Resolution

3.1 Ground Resolution for Predicate Logic

In order to do ground resolution in FOL, we need to define **Herbrand Universe** and **Herbrand expansion** of an arbitrary formula. Let α be a closed formula in Skolem form, the Herbrand universe of α, denoted by $HU(\alpha)$, is the set of all **variable-free terms** that can be built from the contents of α. If there are no constants in α then we choose an arbitrary constant a and then build up $HU(\alpha)$. $HU(\alpha)$ is defined inductively as follows:

- Every constant occurring in α is also in $HU(\alpha)$. If α does not contain any constant then $a \in HU(\alpha)$.
- For every k-ary function symbol f occurring in α, for all terms $t_1, t_2, \cdots, t_k$ already in $HU(\alpha)$, $f(t_1, \cdots, t_k)$ is also in $HU(\alpha)$.

Example 3.1 *Consider the following formulas α and β.*

$$\alpha = (\forall x)(\forall y)(\forall z)p(x, f(y), g(z, x))$$

$$\beta = (\forall x)(\forall y)q(c, f(x), h(y, b))$$

The formula α does not contain any constants, therefore, the Herbrand universe of α is built using an arbitrary constant a.

$$HU(\alpha) = \{a, f(a), g(a, a), g(a, f(a)), g(f(a), a), g(f(a), f(a)), f(f(a)), \cdots\}$$

The formula β contains two constants b and c. $HU(\beta)$ is built using $\{b, c\}$.

$$HU(\beta) = \{b, c, f(b), f(c), h(b, b), h(c, b), h(f(b), b), h(f(c), b), f(f(b)), \cdots\}$$

Observe that if α does not contain any function symbols then $HU(\alpha)$ is finite else it is infinite.

The Herbrand expansion of a closed skolem formula

$$\alpha = (\forall y_1)(\forall y_2) \cdots (\forall y_k)\alpha^*,$$

denoted by $HE(\alpha)$, is the set of all ground instances of α obtained by substituting the variables in α^* by the elements in Herbrand universe of α.

$$HE(\alpha) = \{\alpha^*[y_1/t_1][y_2/t_2] \cdots [y_k/t_k] \mid t_1, t_2, \cdots, t_k \in HU(\alpha)\}.$$

Example 3.2 *For the given formula*

$$\beta = (\forall x)(\forall y)q(c, f(x), h(y, b))$$

with

$$HU(\beta) = \{b, c, f(b), f(c), h(b, b), h(c, b), h(f(b), b), h(f(c), b), f(f(b)), \cdots\}$$

the Herbrand expansion of β contains the following ground instances

$$q(c, f(b), h(b, b)) \quad \textit{using the substitution} \quad [x/b][y/b]$$

$$q(c, f(b), h(c, b)) \quad \textit{using the substitution} \quad [x/b][y/c]$$

$$
\begin{array}{rll}
q(c, f(c), h(c,b)) & \textit{using the substitution} & [x/c][y/c] \\
q(c, f(c), h(b,b)) & \textit{using the substitution} & [x/c][y/b] \\
q(c, f(f(b)), h(c,b)) & \textit{using the substitution} & [x/f(b)][y/c] \\
q(c, f(f(b)), h(f(c),b)) & \textit{using the substitution} & [x/f(b)][y/f(c)] \\
q(c, f(h(b,b)), h(c,b)) & \textit{using the substitution} & [x/h(b,b)][y/c]
\end{array}
$$

A ground instance of α does not have any occurrences of variables in it, and can be treated as a formula in PL.

Theorem 3.3 *For each closed formula α in Skolem form, α is satisfiable if and only if the set of formulas in $HE(\alpha)$ is satisfiable, where $HE(\alpha)$ is interpreted as a set of PL wffs.*

Now, we can describe the **ground resolution procedure** as follows:

Input: A closed formula α in Skolem form with its matrix α^* in CNF.

1. Compute $HE(\alpha)$;
2. Enumerate $HE(\alpha)$ as $\alpha_1, \alpha_2, \ldots$;
3. $i = 0$;
4. $M = 0$;
5. **repeat**
   - 5.1 $i = i + 1$;
   - 5.2 Compute clausal form F_i of α_i;
   - 5.3 $M = M \cup \{F_i\}$;
   - 5.4 $M = Res^*(M)$;
6. **until**$(\square \in M)$;
7. Output "unsatisfiable" and halt;

3.2 Unification

The predicate logic version of resolution involves a technique called **unification** which was invented by **J. A. Robinson**. The idea is to resolve clauses in predicate logic to clauses in predicate logic instead of ground instances as discussed in the previous section. Each step in predicate resolution involves substitution of variables (say x) in the literals to variables (different from x) or terms (not containing x), not necessarily comprising only objects from the corresponding Herbrand universe.

In order to resolve two clauses C_1 and C_2, we search for a **substitution** which **unifies** subsets of literals in these clauses. For example, consider two clauses $C_1 = \{p(x), \neg q(g(x))\}$ and $C_2 = \{\neg p(f(y))\}$. The substitution $[x/f(y)]$ unifies the literal set $\{p(x), p(f(y)\}$. The outcome of unification is a set containing exactly one literal $\{p(f(y))\}$. Applying this substitution on C_1 and C_2 we obtain $C_1' = C_1[x/f(y)] = \{p(f(y)), \neg q(g(f(y)))\}$ and $C_2' = C_2[x/f(y)] = \{\neg p(f(y))\}$. C_1' and C_2' can now be resolved to $\{\neg q(g(f(y)))\}$.

Definition 3.4 (Unifier) *A substitution sub is a* **unifier** *for a (finite) set of literals* $\mathcal{L} = \{L_1, L_2, \cdots, L_k\}$ *if* $L_1 sub = L_2 sub = ...L_k sub$.

If $\mathcal{L}sub$ denotes the set obtained by applying sub to every literal in $\mathcal{L}$, then sub is a unifier if and only if $|\mathcal{L}sub| = 1$. The following unification algorithm finds the most general unifier for a set of literals $\mathcal{L} = \{L_1, L_2, \cdots, L_k\}$.

1. $sub = [\ ]$; (we begin with the empty substitution)
2. **while** $(|\mathcal{L}sub| > 1)$ **do**
   - 2.1 Scan the literals in $\mathcal{L}sub$ from left to right, until the first position is found where in at least two literals (say L_1 and L_2) the corresponding symbols are different;
   - 2.2 **if** none of these symbols is a variable **then** output "non-unifiable" and halt;
   - 2.3 **else**
     - 2.3.1 Let x be the variable in L_1, and let t be term, different from x, at the same position in L_2;
     - 2.3.2 **if** x occurs in t **then** output "non-unifiable" and halt;
     - 2.3.3 **else** $sub = sub[x/t]$;
3. output sub as the most general unifier of $\mathcal{L}$;

Example 3.5 *Let us unify the following set of predicate literals.*

$$L = \{p(x, y), p(f(a), g(x)), p(f(z), g(f(z)))\}$$

- *Initialize sub to* [ ];
- *Scan the first two literals* $p(x, y)$ *and* $p(f(a), g(x))$. *Then, we try to make these two literals identical. Replacing* x *by* $f(a)$ *modifies the set* L *as follows:* $L[x/f(a)] = \{p(f(a), y), p(f(a), g(f(a))), p(f(z), g(f(z)))\}$ *and* $sub = [x/f(a)]$.
- *Replacing* y *by* $g(f(a))$ *modifies the set as follows:* $L[x/f(a)][y/g(f(a))] = \{p(f(a), g(f(a))), p(f(z), g(f(z)))\}$. *The first two literals are unified now and the sub is* $[x/f(a)][y/g(f(a))]$.
- *Scan the first two literals in* $L[x/f(a)][y/g(f(a))]$, $p(f(a), g(f(a)))$ *and* $p(f(z), g(f(z)))$. *We try to make these literals identical.*
- *Replacing* z *by* a *gives the following set:* $L[x/f(a)][y/g(f(a))][z/a] = \{p(f(a), g(f(a)))\}$.
- *The substitution which unifies the set* $L = \{p(x, y), p(f(a), g(x)), p(f(z), g(f(z)))\}$ *is* $sub = [x/f(a)][y/g(f(a))][z/a]$.
- *Note that* $|Lsub| = 1$.

Exercise 3.6 *Apply the unification algorithm to the following set of literals:*

1. $L = \{\neg P(f(z, g(a, y)), h(z)), \neg P(f(f(u, v), w), h(f(a, b)))\}$

3.3 Resolution for Predicate Logic

Using the unification principle, we can now formulate the resolution principle for predicate logic.

Let C_1 and C_2 be two sets of predicate clauses. Then R is called a **resolvent** of C_1, C_2 if the following holds:

1. There exists certain substitutions s_1 and s_2 which are variable renamings so that $C_1 s_1$ and $C_2 s_2$ do not contain the same variable.
2. There is a set of literals $L_1, \ldots, L_m \in C_1 s_1 (m \geq 1)$ and $L'_1, \ldots, L'_n \in C_2 s_2 (n \geq 1)$ such that $\mathbf{L} = \{L_1, \ldots, L_m, L'_1, \ldots, L'_n\}$ is **unifiable**. Let sub be the most general unifier for $\mathbf{L}$.

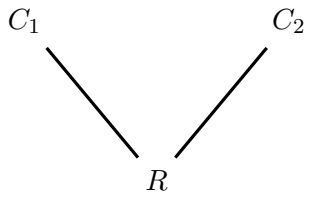

Figure 1: Resolvent in Predicate Resolution

3. R has the form

$$R = ((C_1 s_1 - \{L_1, \ldots, L_m\}) \cup (C_2 s_2 - \{L'_1, \ldots, L'_n\}))sub$$

Exercise 3.7 *Find all resolvents of the following two clauses C_1 and C_2*

1. $C_1 = \{P(f(x)), \neg Q(z), P(z)\}, C_2 = \{\neg P(x), R(g(x), a)\}$
2. $C_1 = \{\neg P(x, y), \neg P(f(a), g(u, b)), Q(x, u)\}$, $C_2 = \{P(f(x), g(a, b)), \neg Q(f(a), b), \neg Q(a, b)\}$

We express the situation described by the definition above as a diagram, in the Figure 1. It turns out that resolution in propositional logic is a special case of resolution in predicate logic where $s_1 = s_2 = sub = [\ ]$ and $m = n = 1$. Therefore, we adopt the notation introduced for the resolution in propositional logic and extend the notion $Res(F)$ for clause sets in predicate logic:

$$Res(F) = F \cup \{R \mid R \text{ is a resolvent of } C_1, C_2 \in F\},$$

$$Res^0(F) = F$$

$$Res^{n+1}(F) = Res(Res^n(F)) \text{ for } n \geq 0 \text{ and}$$

$$Res^*(F) = \bigcup_{n \geq 0} Res^n(F).$$

As in propositional calculus, it is clear that $Res^*(F)$ if and only if there is a sequence $C_1, C_2, \ldots, C_n$ of clauses such that $C_n = \Box$, and for $i = 1, 2, \ldots, n$, C_i is either element of F or C_i is a resolvent of two clauses C_j and C_k with $j, k < i$.

We can also observe that resolutions in propositional logic (for ground instances of clauses in predicate logic) can be "lifted" to certain resolutions in predicate logic. This "lifting lemma" allows us to transform a resolution refutation on clauses in propositional logic to a resolution refutation on clauses in predicate logic.

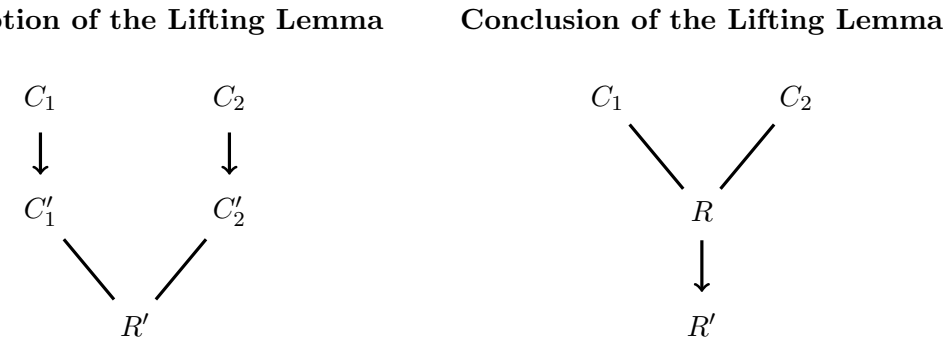

Figure 2: Lifting Lemma

Lifting Lemma

Let C_1, C_2 be two clauses in predicate logic and let C_1', C_2' be two arbitrary ground instances thereof which are resolvable (in the sense of propositional logic). Let R' be a resolvent of C_1', C_2'. Then, there exists a clause R which is a resolvent of C_1, C_2 (in the sense of predicate logic) so that R' is a ground instance of R.

The diagrams in the Figure 2 demonstrate the situation described by the statement of Lifting lemma.

Now, we can assert the resolution theorem for predicate logic as follows:

Theorem 3.8 *Let α be a closed formula in Skolem form with its matrix α^* in CNF. Then, α is unsatisfiable if and only $\square \in Res^*(\alpha^*)$.*

Let α be a formula in Skolem form, whose matrix α^* can be written in clausal form as follows:

$$F = \Big\{\{\neg P(x), Q(x), R(x, f(x))\}, \{\neg P(x), Q(x), S(f(x))\}, \{T(a)\}, \{P(a)\},$$

$$\{\neg R(a, z), T(z)\}, \{\neg T(x), \neg Q(x)\}, \{\neg T(y), \neg S(y)\}\Big\}$$

We can show that α (or F) is unsatisfiable by giving a deduction of empty clause $\square$ as follows:

1. $\{T(a)\}$ clause in F
2. $\{\neg T(x), \neg Q(x)\}$ clause in F
3. $\{\neg Q(a)\}$ resolvent of (1) and (2)
4. $\{\neg P(x), Q(x), S(f(x))\}$ clause in F
5. $\{P(a)\}$ clause in F
6. $\{Q(a), S(f(a))\}$ resolvent of (4) and (5)
7. $\{S(f(a))\}$ resolvent of (3) and (6)
8. $\{\neg P(x), Q(x), R(x, f(x))\}$ clause in F
9. $\{Q(a), R(a, f(a))\}$ resolvent of (5) and (8)
10. $\{R(a, f(a))\}$ resolvent of (3) and (9)
11. $\{\neg R(a, z), T(z)\}$ clause in F
12. $\{T(f(a))\}$ resolvent of (10) and (11)
13. $\{\neg T(y), \neg S(y)\}$ clause in F
14. $\{\neg S(f(a))\}$ resolvent of (12) and (13)
15. $\square$ resolvent of (7) and (14)

4 Forward Chaining in Predicate Logic

We have already seen the forward chaining algorithm for Propositional logic. In that case we work with **definite clauses**—-Horn clauses that have exactly one positive literal. The knowledge base comprises exclusively of definite clauses. We start with atomic sentences (wffs) in the knowledge base and apply Modus Ponens repeatedly in the forward direction, adding new atomic sentences, until no further inferences can be made. It turns out that reasoning with forward chaining can be more efficient than resolution theorem proving. In this section, we extend forward chaining algorithms to first-order definite clauses.

First-order Definite Clauses

First-order definite clauses closely resemble propositional definite clauses: they are disjunction of literals of which exactly one is positive. Unlike propositional literals, first-order literals can include variables, in which case those variables are assumed to be universally quantified. Typically, we omit universal quantifiers when writing definite clauses.

Not every knowledge base can be converted into a set of definite clauses, because of the single-positive-literal restriction, but many can. Below, we give an example of a knowledge base comprised solely of definite clauses. Given this knowledge base, let us call it KB, we want to prove that "West is criminal". That is, $KB \vdash Criminal(West)$.

1. "It is crime for an American to sell weapons to hostile nations"

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

2. "Nono has some missiles"

   $$Owns(Nono, M_1)$$

   $$Missile(M_1)$$

3. "All of Nono's missiles were sold to it by Colonel West"

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

4. "Missiles are Weapons"

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

5. "Enemies of America are Hostile nations"

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

6. "Colonel West is an American"

   $$American(West)$$

7. "Nono is an enemy of America"

   $$Enemy(Nono, America)$$

This knowledge base contains no function symbols and is therefore an instance of the class of **Datalog** knowledge bases—that is, first-order definite clauses with no function symbols. We know that the absence of function symbols makes inference much easier.

function FOL-FC-ASK(KB,α) **returns** a substitution or *false*

inputs: KB, the knowledge base, a set of first-order definite clauses
α, the query, an atomic sentence
local variables: *new*, the new sentences inferred on each iteration
repeat

1. $new = \emptyset$;
2. **for each** sentence r **in** KB
   - 2.1. convert r to definite clause form $p_1 \wedge p_2 \wedge \cdots \wedge p_n \Rightarrow q$;
   - 2.2. **for each** substitution s such that $(p_1 \wedge p_2 \wedge \cdots \wedge p_n)s = (p'_1 \wedge p'_2 \wedge \cdots \wedge p'_n)s$ for some $p'_1 p'_2 \cdots p'_n \in KB$
     - 2.2.1. $q' = qs$
     - 2.2.2. **if** q' not a renaming some sentence already in KB or *new*
       - 2.2.2.1. add q' to *new*;
       - 2.2.2.2. Find the most general unifier *sub* of q' and α;
       - 2.2.2.3. **if** $sub \neq [\,]$ **then return** *sub*;
3. add *new* to KB;

until (*new* is empty);
return *false*.

Figure 3: A Forward-chaining Algorithm

A Simple Forward Chaining Algorithm

A simple forward chaining algorithm is given in the Figure 3. Starting from the known facts (definite clauses with no negative literals), it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts. The process continues until the query is answered (generated) or no new facts are added. We will use the KB and the query given above to show how the forward chaining algorithm works, i.e., how FOL-FC-ASK works to decide whether $KB \vdash Criminal(West)$. The facts (atomic sentences) are $2, 7$ and 8, whereas the integrity constraints (implication sentences) are $1, 4, 5$ and 6. Two iterations of the algorithms are needed to infer $Criminal(West)$.

- On the first iteration, rule 1 has unsatisfied premises.
  Rule 4 is satisfied with the substitution $\{x/M_1\}$, and $Sells(West, M_1, Nono)$

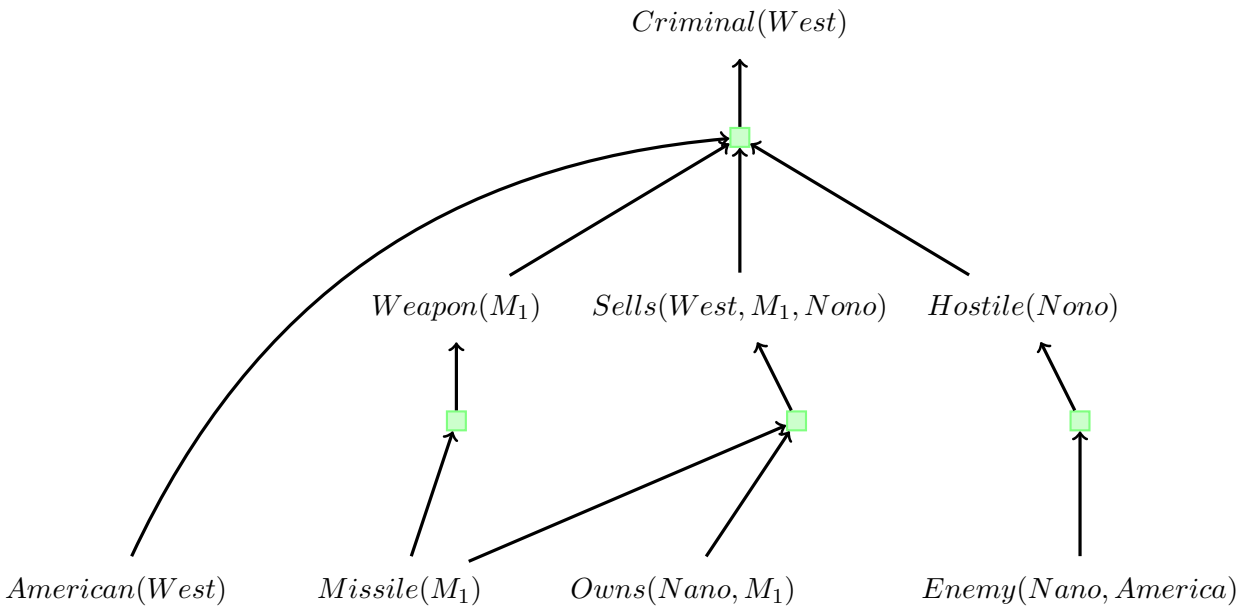

Figure 4: The AND-OR Graph for the crime example: forward chaining

is added.
Rule 5 is satisfied with the substitution $\{x/M_1\}$, and $Weapon(M_1)$ is added.
Rule 6 is satisfied with the substitution $\{x/Nono\}$, and $Hostile(Nono)$ is added.

- On the second iteration, rule 1 is satisfied with $\{x/West, y/M_1, z/Nono\}$, and $Criminal(West)$ is added.

We can draw the AND-OR graph of the KB as shown in the Figure 4. The initial facts, already in KB are at the leaf level. The facts inferred on the first iteration are in the middle level, and the facts inferred on the second iteration are at the top level.

5 Backward Chaining in Predicate Logic

The second major family of logical inference algorithms uses the **backward chaining** approach. We have already seen backward chaining for propositional logic, where the inference works backward from the goal, chaining

through rules ti find known facts support the proof. Backward chaining is used in **logic programming**, which is the most widely used form of automated reasoning.

The backward chaining algorithm is called with a list of goals containing a single element, the original query, and returns the set of all substitutions satisfying the query. The list of goals can be thought of as a "stack" waiting to be worked on; if *all* of them can be satisfied, then the current branch of the proof succeeds.

The algorithm takes the first goal in the list and finds every clause in the knowledge base whose positive literal, or **head**, unifies with the goal. Each such clause creates a new recursive call in which the premise, or **body**, of the clause is added to the goal stack.

Facts are clauses with a head but no body, so when a goal unifies with a known fact, no new subgoals are added to the stack and the goal is solved.

Figure 5 gives the proof tree for deriving $Criminal(West)$ from the KB of crime example. The proof tree is read depth first, left to right. To prove $Criminal(West)$, we have to prove the four conjuncts below it. Some of these are already in the KB, and others require further backward chaining.

The bindings for each successful unification are shown next to the corresponding subgoal. Note that, once one subgoal in a conjunction succeeds, its substitution is applied to subsequent subgoals.

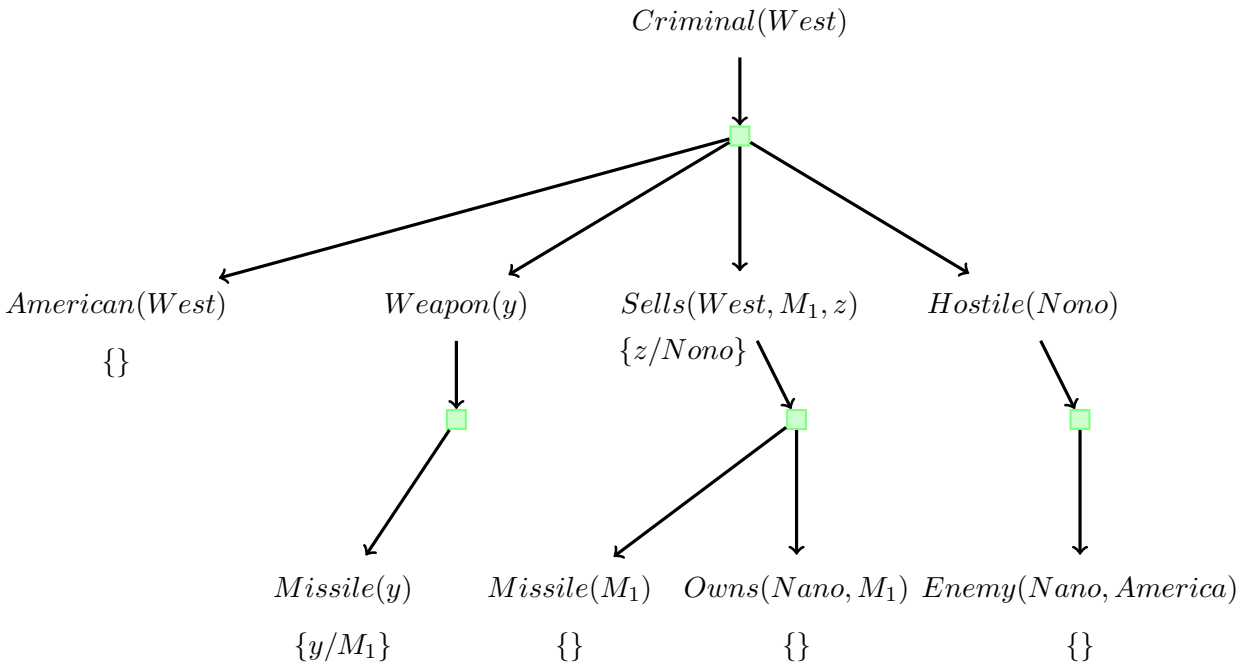

Figure 5: The AND-OR Graph for the crime example: backward chaining