

Foundations of Artificial Intelligence

9. Predicate Logic

Syntax and Semantics, Normal Forms, Herbrand Expansion, Resolution

Joschka Boedecker and Wolfram Burgard and Marco Ragni



Albert-Ludwigs-Universität Freiburg

Contents

- 1 Motivation
- 2 Syntax and Semantics
- 3 Normal Forms
- 4 Resolution & Unification
- 5 Closing Remarks

We can already do a lot with propositional logic. It is, however, annoying that there is no structure in the atomic propositions.

Example:

“All blocks are red”

“There is a block A”

It should follow that “A is red”

But propositional logic cannot handle this.

Idea: We introduce individual variables, predicates, functions,

→ First-Order Predicate Logic (PL1)

The Alphabet of First-Order Predicate Logic

Symbols:

- Operators: $\neg, \vee, \wedge, \forall, \exists, =$
- Variables: $x, x_1, x_2, \dots, x', x'', \dots, y, \dots, z, \dots$
- Brackets: $()$, $[\]$, $()$, $[\]$
- Function symbols (e.g., $weight()$, $color()$)
- Predicate symbols (e.g., $Block()$, $Red()$)
- Predicate and function symbols have an arity (number of arguments).
 - 0-ary predicate = propositional logic atoms: P, Q, R, \dots
 - 0-ary function = constants: a, b, c, \dots
- We assume a countable set of predicates and functions of any arity.
- “=” is usually not considered a predicate, but a logical symbol

The Grammar of First-Order Predicate Logic (1)

Terms (represent objects):

1. Every variable is a term.
2. If t_1, t_2, \dots, t_n are terms and f is an n -ary function, then

$$f(t_1, t_2, \dots, t_n)$$

is also a term.

Terms without variables: **ground terms**.

Atomic Formulae (represent statements about objects)

1. If t_1, t_2, \dots, t_n are terms and P is an n -ary predicate, then $P(t_1, t_2, \dots, t_n)$ is an atomic formula.
2. If t_1 and t_2 are terms, then $t_1 = t_2$ is an atomic formula.

Atomic formulae without variables: **ground atoms**.

The Grammar of First-Order Predicate Logic (2)

Formulae:

1. Every atomic formula is a formula.
2. If φ and ψ are formulae and x is a variable, then

$$\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \Rightarrow \psi, \varphi \Leftrightarrow \psi, \exists x\varphi \text{ and } \forall x\varphi$$

are also formulae.

\forall, \exists are as strongly binding as \neg .

Propositional logic is part of the PL1 language:

1. Atomic formulae: only 0-ary predicates
2. Neither variables nor quantifiers.

Alternative Notation

| Here | Elsewhere |
|--------------------------------|--|
| $\neg\varphi$ | $\sim\varphi \quad \bar{\varphi}$ |
| $\varphi \wedge \psi$ | $\varphi \& \psi \quad \varphi \bullet \psi \quad \varphi, \psi$ |
| $\varphi \vee \psi$ | $\varphi \psi \quad \varphi ; \psi \quad \varphi + \psi$ |
| $\varphi \Rightarrow \psi$ | $\varphi \rightarrow \psi \quad \varphi \supset \psi$ |
| $\varphi \Leftrightarrow \psi$ | $\varphi \leftrightarrow \psi \quad \varphi \equiv \psi$ |
| $\forall x \varphi$ | $(\forall x) \varphi \quad \forall x \varphi$ |
| $\exists x \varphi$ | $(\exists x) \varphi \quad \exists x \varphi$ |

Meaning of PL1-Formulae

Our example: $\forall x[Block(x) \Rightarrow Red(x)], Block(a)$

For all objects x : If x is a block, then x is red and a is a block.

Generally:

- Terms are interpreted as objects.
- Universally-quantified variables denote all objects in the universe.
- Existentially-quantified variables represent one of the objects in the universe (made true by the quantified expression).
- Predicates represent subsets of the universe.

Similar to propositional logic, we define [interpretations](#), [satisfiability](#), [models](#), [validity](#), ...

Interpretation: $I = \langle D, \bullet^I \rangle$ where D is an arbitrary, non-empty set and \bullet^I is a function that

- maps n -ary function symbols to functions over D :

$$f^I \in [D^n \mapsto D]$$

- maps individual constants to elements of D :

$$a^I \in D$$

- maps n -ary predicate symbols to relations over D :

$$P^I \subseteq D^n$$

Interpretation of ground terms:

$$(f(t_1, \dots, t_n))^I = f^I(t_1^I, \dots, t_n^I)$$

Satisfaction of ground atoms $P(t_1, \dots, t_n)$:

$$I \models P(t_1, \dots, t_n) \text{ iff } \langle t_1^I, \dots, t_n^I \rangle \in P^I$$

Example (1)

$$D = \{d_1, \dots, d_n \mid n > 1\}$$

$$a^I = d_1$$

$$b^I = d_2$$

$$c^I = \dots$$

$$\text{Block}^I = \{d_1\}$$

$$\text{Red}^I = D$$

$$I \models \text{Red}(b)$$

$$I \not\models \text{Block}(b)$$

Example (2)

$$D = \{1, 2, 3, \dots\}$$

$$1^I = 1$$

$$2^I = 2$$

...

$$\text{Even}^I = \{2, 4, 6, \dots\}$$

$$\text{succ}^I = \{(1 \mapsto 2), (2 \mapsto 3), \dots\}$$

$$I \models \text{Even}(2)$$

$$I \not\models \text{Even}(\text{succ}(2))$$

Semantics of PL1: Variable Assignment

Set of all variables V . Function $\alpha : V \mapsto D$

Notation: $\alpha[x/d]$ is the same as α apart from point x .

For $x : \alpha[x/d](x) = d$.

Interpretation of terms under I, α :

$$x^{I, \alpha} = \alpha(x)$$

$$a^{I, \alpha} = a^I$$

$$(f(t_1, \dots, t_n))^{I, \alpha} = f^I(t_1^{I, \alpha}, \dots, t_n^{I, \alpha})$$

Satisfaction of atomic formulae:

$$I, \alpha \models P(t_1, \dots, t_n) \text{ iff } \langle t_1^{I, \alpha}, \dots, t_n^{I, \alpha} \rangle \in P^I$$

Example

$$\alpha = \{(x \mapsto d_1), (y \mapsto d_2)\}$$

$$I, \alpha \models \textit{Red}(x)$$

$$I, \alpha[y/d_1] \models \textit{Block}(y)$$

Semantics of PL1: Satisfiability

A formula φ is **satisfied** by an **interpretation** I and a variable assignment α , i.e., $I, \alpha \models \varphi$:

$$I, \alpha \models \top$$

$$I, \alpha \not\models \perp$$

$$I, \alpha \models \neg\varphi \text{ iff } I, \alpha \not\models \varphi$$

...

and all other propositional rules as well as

$$I, \alpha \models P(t_1, \dots, t_n) \quad \text{iff} \quad \langle t_1^{I, \alpha}, \dots, t_n^{I, \alpha} \rangle \in P^{I, \alpha}$$

$$I, \alpha \models \forall x\varphi \quad \text{iff} \quad \text{for all } d \in D, I, \alpha[x/d] \models \varphi$$

$$I, \alpha \models \exists x\varphi \quad \text{iff} \quad \text{there exists a } d \in D \text{ with } I, \alpha[x/d] \models \varphi$$

Example

$$T = \{Block(a), Block(b), \forall x(Block(x) \Rightarrow Red(x))\}$$

$$D = \{d_1, \dots, d_n \mid n > 1\}$$

$$a^I = d_1$$

$$b^I = d_2$$

$$Block^I = \{d_1\}$$

$$Red^I = D$$

$$\alpha = \{(x \mapsto d_1), (y \mapsto d_2)\}$$

Questions:

1. $I, \alpha \models Block(b) \vee \neg Block(b)$?
2. $I, \alpha \models Block(x) \Rightarrow (Block(x) \vee \neg Block(y))$?
3. $I, \alpha \models Block(a) \wedge Block(b)$?
4. $I, \alpha \models \forall x(Block(x) \Rightarrow Red(x))$?
5. $I, \alpha \models \top$?

$$\forall x [R(\boxed{y}, \boxed{z}) \wedge \exists y ((\neg P(y, x) \vee R(y, \boxed{z})))]$$

The boxed appearances of y and z are **free**. All other appearances of x, y, z are **bound**.

Formulae with no free variables are called **closed** formulae or **sentences**. We form theories from closed formulae.

Note: With closed formulae, the concepts *logical equivalence*, *satisfiability*, and *implication*, etc. are not dependent on the variable assignment α (i.e., we can always ignore all variable assignments).

With closed formulae, α can be left out on the left side of the model relationship symbol:

$$I \models \varphi$$

An interpretation I is called a **model** of φ under α if

$$I, \alpha \models \varphi$$

A PL1 formula φ can, as in propositional logic, be **satisfiable**, **unsatisfiable**, **falsifiable**, or **valid**.

Analogously, two formulae are **logically equivalent** ($\varphi \equiv \psi$) if for all I, α :

$$I, \alpha \models \varphi \text{ iff } I, \alpha \models \psi$$

Note: $P(x) \not\equiv P(y)$!

Logical Implication is also analogous to propositional logic.

Question: How can we define **derivation**?

Prenex Normal Form

Because of the quantifiers, we cannot produce the CNF form of a formula directly.

First step: Produce the prenex normal form

quantifier prefix + (quantifier-free) matrix

$Qx_1Qx_2Qx_3 \dots Qx_n \varphi$

Equivalences for the Production of Prenex Normal Form

$$(\forall x\varphi) \wedge \psi \equiv \forall x(\varphi \wedge \psi) \text{ if } x \text{ not free in } \psi$$

$$(\forall x\varphi) \vee \psi \equiv \forall x(\varphi \vee \psi) \text{ if } x \text{ not free in } \psi$$

$$(\exists x\varphi) \wedge \psi \equiv \exists x(\varphi \wedge \psi) \text{ if } x \text{ not free in } \psi$$

$$(\exists x\varphi) \vee \psi \equiv \exists x(\varphi \vee \psi) \text{ if } x \text{ not free in } \psi$$

$$\forall x\varphi \wedge \forall x\psi \equiv \forall x(\varphi \wedge \psi)$$

$$\exists x\varphi \vee \exists x\psi \equiv \exists x(\varphi \vee \psi)$$

$$\neg\forall x\varphi \equiv \exists x\neg\varphi$$

$$\neg\exists x\varphi \equiv \forall x\neg\varphi$$

... and propositional logic equivalents

Production of Prenex Normal Form

1. Eliminate \Rightarrow and \Leftrightarrow
2. Move \neg inwards
3. Move quantifiers outwards

Example:

$$\begin{aligned} & \neg\forall x[(\forall xP(x)) \Rightarrow Q(x)] \\ \rightarrow & \neg\forall x[\neg(\forall xP(x)) \vee Q(x)] \\ \rightarrow & \exists x[(\forall xP(x)) \wedge \neg Q(x)] \end{aligned}$$

And now?

Renaming of Variables

$\varphi[\frac{x}{t}]$ is obtained from φ by replacing all free appearances of x in φ by t .

Lemma: Let y be a variable that does not appear in φ . Then it holds that

$$\forall x\varphi \equiv \forall y\varphi[\frac{x}{y}] \text{ and } \exists x\varphi \equiv \exists y\varphi[\frac{x}{y}]$$

Theorem: There exists an algorithm that calculates the prenex normal form of any formula.

Idea: **Elimination of existential quantifiers** by applying a function that produces the “right” element.

Theorem (**Skolem Normal Form**): Let φ be a closed formula in prenex normal form such that all quantified variables are pair-wise distinct and the function symbols g_1, g_2, \dots do not appear in φ . Let

$$\varphi = \forall x_1 \cdots \forall x_i \exists y \psi,$$

then φ is satisfiable iff

$$\varphi' = \forall x_1 \cdots \forall x_i \psi \left[\frac{y}{g_i(x_1, \dots, x_i)} \right]$$

is satisfiable.

Example: $\forall x \exists y [P(x) \Rightarrow Q(y)] \rightarrow \forall x [P(x) \Rightarrow Q(g(x))]$

Skolem Normal Form

Skolem Normal Form: Prenex normal form without existential quantifiers.

Notation: φ^* is the SNF of φ .

Theorem: It is possible to calculate the Skolem normal form of every closed formula φ .

Example: $\exists x((\forall xP(x)) \wedge \neg Q(x))$ develops as follows:

$$\exists y((\forall xP(x)) \wedge \neg Q(y))$$

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

$$\forall x(P(x) \wedge \neg Q(g_0))$$

Note: This transformation is **not an equivalence transformation**; it **only preserves satisfiability!**

Note: ... and is **not unique**.

Production of Clausal Form from SNF

We have: Skolem Normal Form

quantifier prefix + (quantifier-free) matrix

$\forall x_1 \forall x_2 \forall x_3 \cdots \forall x_n \varphi$

1. Put Matrix φ into CNF using propositional logic equivalences.
2. Eliminate universal quantifiers.
3. Eliminate conjunction symbol.
4. Rename variables so that no variable appears in more than one clause.

Theorem: It is possible to calculate the clausal form of every closed formula φ .

Note: Same remarks as for SNF

Conversion to Clausal Form (1)

Everyone who loves all animals is loved by someone:

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

1. Eliminate biconditionals and implications

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

2. Move \neg inwards: $\neg\forall xp \equiv \exists x\neg p$, $\neg\exists xp \equiv \forall x\neg p$

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

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

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

Conversion to Clausal Form (2)

3. Standardize variables: each quantifier should use a different one

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

4. Prenex norm form: all quantifiers in front of the matrix:

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

5. Skolemize: Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

$$\forall x ([Animal(f(x)) \wedge \neg Loves(x, f(x))] \vee [Loves(g(x), x)])$$

6. Distribute \wedge over \vee :

$$\forall x ([Animal(f(x)) \vee Loves(g(x), x)] \wedge [\neg Loves(x, f(x)) \vee Loves(g(x), x)])$$

Conversion to Clausal Form (3)

7. Eliminate universal quantification (implicitly assumed):

$$([\textit{Animal}(f(x)) \vee \textit{Loves}(g(x), x)] \wedge [\neg \textit{Loves}(x, f(x)) \vee \textit{Loves}(g(x), x)])$$

8. Eliminate conjunction (and transform to set of disjunctions):

$$\{[\textit{Animal}(f(x)) \vee \textit{Loves}(g(x), x)], [\neg \textit{Loves}(x, f(x)) \vee \textit{Loves}(g(x), x)]\}$$

9. Normalize variables:

$$\{[\textit{Animal}(f(x)) \vee \textit{Loves}(g(x), x)], [\neg \textit{Loves}(y, f(y)) \vee \textit{Loves}(g(y), y)]\}$$

Assumption: KB is a **set of clauses**.

Due to commutativity, associativity, and idempotence of \vee , **clauses** can also be understood as **sets of literals**. The **empty set of literals** is denoted by \square (and denotes falsity).

Set of clauses: Δ

Set of literals: C, D

Literal: l

Negation of a literal: \bar{l}

$$\frac{C_1 \dot{\cup} \{l\}, C_2 \dot{\cup} \{\bar{l}\}}{C_1 \cup C_2}$$

$C_1 \cup C_2$ are called **resolvents** of the **parent clauses** $C_1 \dot{\cup} \{l\}$ and $C_2 \dot{\cup} \{\bar{l}\}$. l and \bar{l} are the **resolution literals**.

Example: $\{a, b, \neg c\}$ resolves with $\{a, d, c\}$ to $\{a, b, d\}$.

Notation: $R(\Delta) = \Delta \cup \{C \mid C \text{ is a resolvent of two clauses from } \Delta\}$

First-order Resolution: What Changes?

Examples

$\{\{Nat(s(0)), \neg Nat(0)\}, \{Nat(0)\}\} \vdash \{Nat(s(0))\}$

$\{\{Nat(s(0)), \neg Nat(x)\}, \{Nat(0)\}\} \vdash \{Nat(s(0))\}$

$\{\{Nat(s(x)), \neg Nat(x)\}, \{Nat(0)\}\} \vdash \{Nat(s(0))\}$

We need **unification**, a way to make literals identical.

Based on the notion of **substitution**, e.g., $\{\frac{x}{0}\}$.

A **substitution** $s = \{ \frac{v_1}{t_1}, \dots, \frac{v_n}{t_n} \}$ **substitutes variables** v_i **by terms** t_i (t_i must not contain v_i).

Applying a substitution s to an expression φ yields the expression φs which is φ with all occurrences of v_i replaced by t_i for all i .

Substitution Examples

$$P(x, f(y), b)$$

$$P(z, f(w), b) \quad s = \left\{ \frac{x}{z}, \frac{y}{w} \right\}$$

$$P(x, f(a), b) \quad s = \left\{ \frac{y}{a} \right\}$$

$$P(g(z), f(a), b) \quad s = \left\{ \frac{x}{g(z)}, \frac{y}{a} \right\}$$

$$P(c, f(a), a)$$

Reminder: x, y, z, \dots are variables, a, b, c, \dots are constants, f, g, \dots are functions.

Composing Substitutions

Composing substitutions s_1 and s_2 gives s_1s_2 which is that substitution obtained by first applying s_2 to the terms in s_1 and adding remaining term/variable pairs (**not occurring** in s_1) to s_1 .

$$\text{Example: } \left\{ \frac{z}{g(x,y)} \right\} \left\{ \frac{x}{a}, \frac{y}{b}, \frac{w}{c}, \frac{z}{d} \right\} = \left\{ \frac{z}{g(a,b)}, \frac{x}{a}, \frac{y}{b}, \frac{w}{c} \right\}$$

Application example: $P(x, y, z) \rightarrow P(a, b, g(a, b))$

For a formula φ and substitutions s_1, s_2

$$(\varphi s_1) s_2 = \varphi(s_1 s_2)$$

$$(s_1 s_2) s_3 = s_1(s_2 s_3)$$

$$s_1 s_2 \neq s_2 s_1$$

associativity

no commutativity!

Unification

Unifying a set of expressions $\{w_i\}$

Find substitution s such that $w_i s = w_j s$ for all i, j

Example

$\{P(x, f(y), b), P(x, f(b), b)\}$

$s = \{\frac{y}{b}, \frac{z}{a}\}$ not the simplest unifier

$s = \{\frac{y}{b}\}$ most general unifier (mgu)

The **most general unifier**, the **mgu**, g of $\{w_i\}$ has the property that if s is any unifier of $\{w_i\}$ then there exists a substitution s' such that

$$\{w_i\}s = \{w_i\}gs'$$

Property: The common expression produced is unique up to **alphabetic variants** (variable renaming) for all mgus.

Disagreement Set

The **disagreement set** of a set of expressions $\{w_i\}$ is the set of sub-terms $\{t_i\}$ of $\{w_i\}$ at the first position in $\{w_i\}$ for which the $\{w_i\}$ disagree

Examples

$\{P(x, a, f(y)), P(v, b, z)\}$ gives $\{x, v\}$

$\{P(x, a, f(y)), P(x, b, z)\}$ gives $\{a, b\}$

$\{P(x, y, f(y)), P(x, b, z)\}$ gives $\{y, b\}$

Unification Algorithm

UNIFY(*Terms*):

- 1 $k \leftarrow 0$
- 2 $T_k = \text{Terms}$
- 3 $s_k = \emptyset$
- 4 If T_k is a singleton, then return s_k .
- 5 Let D_k be the disagreement set of T_k .
- 6 If there exists a var v_k and a term t_k in D_k such that v_k does not occur in t_k , continue. Otherwise, exit with failure.
- 7 $s_{k+1} \leftarrow s_k \left\{ \frac{v_k}{t_k} \right\}$
- 8 $T_{k+1} \leftarrow T_k \left\{ \frac{v_k}{t_k} \right\}$
- 9 $k \leftarrow k + 1$
- 10 Continue with step 4.

Example

$\{P(x, f(y), y), P(z, f(b), b)\}$

$$\frac{C_1 \dot{\cup} \{l_1\}, C_2 \dot{\cup} \{\bar{l}_2\}}{[C_1 \cup C_2]s}$$

where $s = \text{mgu}(l_1, l_2)$, the most general unifier $[C_1 \cup C_2]s$ is the **resolvent** of the **parent clauses** $C_1 \dot{\cup} \{l_1\}$ and $C_2 \dot{\cup} \{\bar{l}_2\}$.

$C_1 \dot{\cup} \{l_1\}$ and $C_2 \dot{\cup} \{\bar{l}_2\}$ do not share variables l_1 and l_2 are the **resolution literals**.

Examples: $\{\{Nat(s(0)), \neg Nat(0)\}, \{Nat(0)\}\} \vdash \{Nat(s(0))\}$

$\{\{Nat(s(0)), \neg Nat(x)\}, \{Nat(0)\}\} \vdash \{Nat(s(0))\}$

$\{\{Nat(s(x)), \neg Nat(x)\}, \{Nat(0)\}\} \vdash \{Nat(s(0))\}$

Some Further Examples

Resolve $\{P(x), Q(f(x))\}$ and $\{R(g(x)), \neg Q(f(a))\}$

Standardizing the variables apart gives $\{P(x), Q(f(x))\}$ and $\{R(g(y)), \neg Q(f(a))\}$

Substitution $s = \{\frac{x}{a}\}$ Resolvent $\{P(a), R(g(y))\}$

Resolve $\{P(x), Q(x, y)\}$ and $\{\neg P(a), \neg R(b, z)\}$

Standardizing the variables apart

Substitution $s = \{\frac{x}{a}\}$ and Resolvent $\{Q(a, y), \neg R(b, z)\}$

$$\frac{C_1 \dot{\cup} \{l_1\} \dot{\cup} \{l_2\}}{[C_1 \cup \{l_1\}]s}$$

where $s = mgu(l_1, l_2)$ is the most general unifier.

Needed because:

$$\{\{P(u), P(v)\}, \{\neg P(x), \neg P(y)\}\} \models \square$$

but \square cannot be derived by binary resolution

Factoring yields:

$\{P(u)\}$ and $\{\neg P(x)\}$ whose resolvent is \square .

Notation: $R(\Delta) = \Delta \cup \{C \mid C \text{ is a resolvent or a factor of two clauses from } \Delta\}$

We say D can be derived from Δ , i.e.,

$$\Delta \vdash D,$$

if there exist $C_1, C_2, C_3, \dots, C_n = D$ such that $C_i \in R(\Delta \cup \{C_1, \dots, C_{i-1}\})$ for $1 \leq i \leq n$.

Lemma: ([soundness](#)) If $\Delta \vdash D$, then $\Delta \models D$.

Lemma: resolution is [refutation-complete](#):

Δ is unsatisfiable implies $\Delta \vdash \square$.

Theorem: Δ is unsatisfiable iff $\Delta \vdash \square$.

Technique: to prove that $\Delta \models C$ negate C and prove that $\Delta \cup \{\neg C\} \vdash \square$.

Recursive Enumeration and Decidability

Based on the result, we can construct a **semi-decision procedure** for **validity**, i.e., we can give a (rather inefficient) algorithm that *enumerates* all valid formulae step by step.

Theorem: The set of **valid** (and **unsatisfiable**) **formulae** in PL1 is **recursively enumerable**.

What about *satisfiable* formulae?

Theorem (**undecidability of PL1**): It is **undecidable**, whether a formula of PL1 is **valid**.

(Proof by reduction from PCP)

Corollary: The set of **satisfiable formulae** in PL1 is **not recursively enumerable**.

In other words: If a formula is valid (or follows logically from a set of formulae), we can effectively confirm this. Otherwise, we can end up in an infinite loop (producing resolvents without end).

From Russell and Norvig:

The law says 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

... 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.

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

Missiles are weapons:

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

An enemy of America counts as "hostile":

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

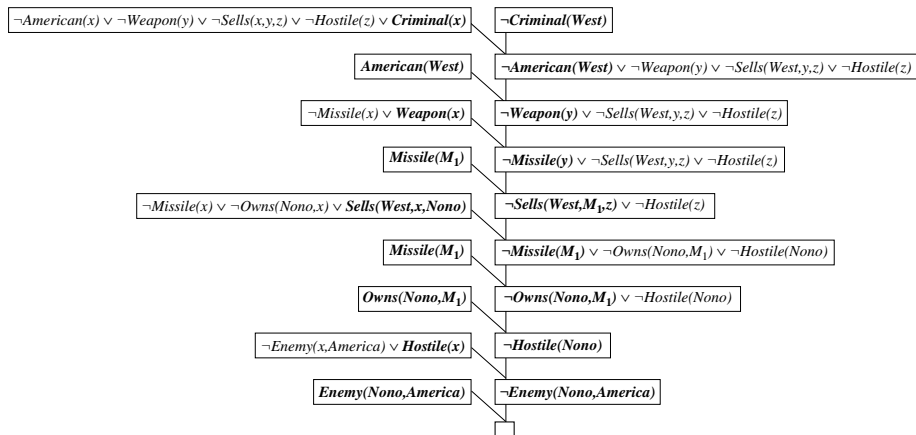
West, who is American ...

$American(West)$

The country Nono, an enemy of America

$Enemy(Nono, America)$

An Example



Closing Remarks: Processing

- **PL1-Resolution**: forms the basis of
 - most state of the art theorem provers for PL1
 - the programming language **Prolog**
 - only Horn clauses
 - considerably more efficient methods.
 - not dealt with : search/resolution strategies
- **Finite theories**: In applications, we often have to deal with a fixed set of objects. **Domain closure axiom**:
$$\forall x[x = c_1 \vee x = c_2 \vee \dots \vee x = c_n]$$
 - Translation into finite propositional theory is possible.

Closing Remarks: Possible Extensions

- PL1 is definitely very expressive, but in some circumstances we would like more ...
- **Second-Order Logic:** Also over predicate quantifiers
$$\forall x, y[(x = y) \Leftrightarrow \{\forall p[p(x) \Leftrightarrow p(y)]\}]$$
- Validity is no longer semi-decidable
- **Lambda Calculus:** Definition of predicates, e.g.,
 $\lambda x, y[\exists z P(x, z) \wedge Q(z, y)]$ defines a new predicate of arity 2
- Reducible to PL1 through Lambda-Reduction
- **Uniqueness quantifier:** $\exists! x \varphi(x)$ - there is exactly one x ...
- Reduction to PL1:
$$\exists x[\varphi(x) \wedge \forall y(\varphi(y) \Rightarrow x = y)]$$

Summary

- PL1 makes it possible to structure statements, thereby giving us considerably **more expressive power than propositional logic**.
- Formulae consist of **terms** and **atomic formulae**, which, together with **connectors** and **quantifiers**, can be put together to produce formulae.
- Interpretations in PL1 consist of a **universe** and an **interpretation function**.
- **Resolution** is sound and **refutation complete**
- **Validity** in PL1 is **not decidable** (it is only semi-decidable)