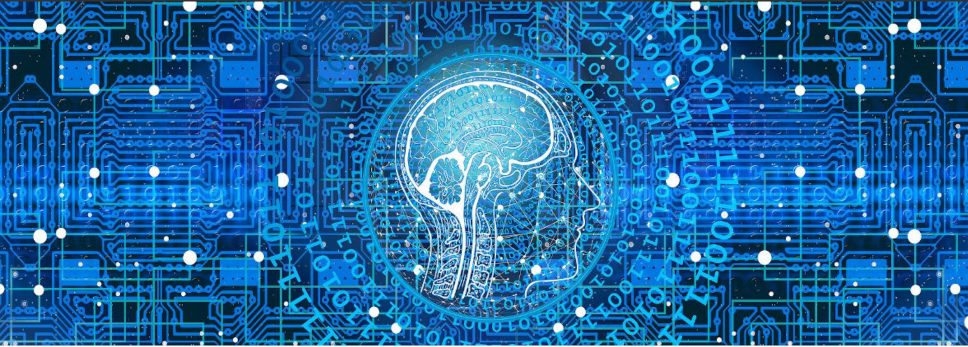


Chapter 9: Predicate Logic

Source A: Foundations of Artificial Intelligence by J. Boedecker, W. Burgard, F. Hutter, B. Nebel (Uni. Freiburg)

Motivation

- 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)

Contents

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

The Alphabet of First-Order Predicate Logic

- Symbols:
- Operators: $\neg, \vee, \wedge, \forall, \exists, =, \Rightarrow, \Leftrightarrow$
- Variables: $x, x_1, x_2, ..., x', x'', ..., y, ..., z, ...$
- 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: $P, Q, R, ...$
  - 0-ary function $=$ constants: $a, b, c, ...$
- 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, e.g. this is a table):

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

$$f(t_1, t_2, ..., t_n)$$

  is also term.
- Terms without variables: ground terms (e.g. $f()$ or $f(g(), h(), ..)$)

Atomic Formulae (represent statements about objects)

- 1. If $t_1, t_2, ..., t_n$ are terms and P is an n-ary predicate, then $P(t_1, t_2, ..., t_n)$ 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

$$\neg\phi, \phi \wedge \psi, \phi \vee \psi, \phi \Rightarrow \psi, \phi \iff \psi, \exists x \phi \text{ and } \forall x \phi$$

  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
$\neg\varphi$	$\sim\varphi$ $\overline{\varphi}$
$\varphi \wedge \psi$	$\varphi \& \psi$ $\varphi \bullet \psi$ φ, ψ
$\varphi \vee \psi$	$\varphi \vert \psi$ $\varphi ; \psi$ $\varphi + \psi$
$\varphi \Rightarrow \psi$	$\varphi \rightarrow \psi$ $\varphi \supset \psi$
$\varphi \Leftrightarrow \psi$	$\varphi \leftrightarrow \psi$ $\varphi \equiv \psi$
$\forall x \varphi$	$(\forall x)\varphi \wedge x\varphi$
$\exists x \varphi$	$(\exists x)\varphi \vee x\varphi$

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 (e.g. some objects of the universe are Blocks...).

Similar to propositional logic, we define interpretations, satisfiability, models, validity, ...

Semantics of PL1-Logic

Interpretation: $I =< D, \cdot^I >$ where D is an arbitrary, non-empty set (i.e. universe) and $\cdot^I$ is a function that

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

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

- 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:

$$(f(t_1, ..., t_n))^I = f^I(t_1^I, ..., t_n^I)$$

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

$$I \models P(t_1, ..., t_n) \text{ iff } < t_1^I, ..., t_n^I > \in P^I$$

Example (1)

$$\begin{aligned}
D &= \{d_1, \ldots, d_n \mid n > 1\} \\
a^I &= d_1 \\
b^I &= d_2 \\
c^I &= \ldots \\
Block^I &= \{d_1\} \\
Red^I &= D \\
I &\models Red(b) \\
I &\not\models Block(b)
\end{aligned}$$

Example (2)

$$
\begin{aligned}
D &= \{1, 2, 3, \ldots\} \\
1^I &= 1 \\
2^I &= 2 \\
&\ldots \\
Even^I &= \{2, 4, 6, \ldots\} \\
succ^I &= \{(1 \mapsto 2), (2 \mapsto 3), \ldots\} \\
I &\models Even(2) \\
I &\not\models Even(succ(2))
\end{aligned}
$$

Semantics of PL1: Variable Assignment

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

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

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

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

Satisfaction of atomic formulae:

$$I, \alpha \models P(t_1, ..., t_n) \text{ iff } < t_1^{I,\alpha}, ..., t_n^{I,\alpha} > \in P^I$$

Example

$$Block^I = d_1$$

$$Red^I = D$$

$$\alpha = (x \rightarrow d_1), (y \rightarrow d_2)$$

$$I, \alpha \models Red(x)$$

$$I, \alpha[y/d1] \models Block(y)$$

Semantics of PL1: Satisfiability

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

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

$$I, \alpha \not\models \bot$$

$$I, \alpha \models \neg\phi \text{ iff } I, \alpha \models \phi$$

and all other propositional rules as well as

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

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

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

Example

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

$$D = \{d_1, ..., d_n | n > 1\}$$

$$a^I = d_1$$

$$b^I = d_2$$

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

$$Red^I = D$$

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

Example

Questions:

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

Free and Bound Variables

$$\forall x\,\big[R(\boxed{y},\boxed{z}) \land \exists y\big((\neg P(y,x) \lor R(y,\boxed{z})\big)\big]$$

- 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:

$$I \models \phi$$

Terminology

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

$$I, \alpha \models \phi$$

- A PL1 formula ϕ can, as in propositional logic, be satisfiable, unsatisfiable, falsifiable, or valid.
- Analogously, two formulae are logically equivalent $(\phi \equiv \psi)$ if for all

$$I, \alpha : I, \alpha \models \phi \text{ iff } I, \alpha \models \psi$$

- Note: $P(x)! \equiv P(y)$
- Logical Implication is also analogous to propositional logic.
- Question: How can we define derivation?

Normal Forms

Normal Forms

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

- $(\forall x \phi) \wedge \psi \equiv \forall x(\phi \wedge \psi)$ if x not free in ψ
- $(\forall x \phi) \vee \psi \equiv \forall x(\phi \vee \psi)$ if x not free in ψ
- $(\exists x \phi) \wedge \psi \equiv \exists x(\phi \wedge \psi)$ if x not free in ψ
- $(\exists x \phi) \vee \psi \equiv \exists x(\phi \vee \psi)$ if x not free in ψ
- $\forall x \phi \wedge \forall x \psi \equiv \forall x(\phi \wedge \psi)$
- $\exists x \phi \vee \exists x \psi \equiv \exists x(\phi \vee \psi)$
- $\neg \forall x \phi \equiv \exists x \neg \phi$
- $\neg \exists x \phi \equiv \forall x \neg \phi$
- ... and propositional logic equivalents

Production of Prenex Normal Form

- 1. Eliminate $\Rightarrow$ and $\iff$
- 2. Move $\neg$ inwards
- 3. Move quantifiers outwards

Example

$$\neg\forall x[(\forall x P(x)) \Rightarrow Q(x)]$$

$$\rightarrow \neg\forall x[\neg(\forall x P(x)) \vee Q(x)]$$

$$\rightarrow \exists x[(\forall x P(x)) \wedge \neg Q(x)]$$

And now?

Renaming of Variables

$\phi[\frac{x}{t}]$ 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

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

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

Skolemization

- 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 $g_1, g_2, ...$ do not appear in ϕ. Let

  $$\phi = \forall x_1 ... \forall x_i \exists y \psi,$$

  then ϕ is satisfiable iff

  $$\phi' = \forall x_1 ... \forall x_i \psi[\frac{y}{g_i(x_1, ..., x_i)}]$$

  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:

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

develops as follows:

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

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

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

- Note: This transformation is not an equivalence transformation; it only preserves satisfiability! ...and is not unique.

Production of Clausal Form from SNF

- We have: Skolem Normal Form
  quantifier prefix + (quantifier-free) matrix

$$\forall x_1 \forall x_2 \forall x_3 \forall x_n \phi$$

- 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 is loved by all animals, is loved by someone

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

1. Eliminate biconditionals and implications gfdg

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

2. Move ¬ inwards:

$$\neg\forall x p \equiv \exists x \neg p, \neg\exists x p \equiv \forall x \neg p$$

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

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

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

Conversion to Clausal Form (2)

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

$$\forall x[\exists y(Animal(y) \land \neg Loves(x,y))] \lor [\exists z Loves(z,x)]$$

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

$$\forall x \exists y \exists z[(Animal(y) \land \neg Loves(x,y))] \lor [Loves(z,x)]$$

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

$$\forall x[Animal(f(x)) \land \neg Loves(x,f(x))] \lor [Loves(g(x),x)]$$

6. Distribute $\land$ over $\lor$:

$$\forall x[Animal(f(x)) \lor Loves(g(x),x)] \land [\neg Loves(x,f(x)) \lor Loves(g(x),x)]$$

Conversion to Clausal Form (3)

7. Eliminate universal quantification (implicitely assumed):

$$[Animal(f(x)) \lor Loves(g(x), x)] \land [\neg Loves(x, g(x)) \lor Loves(g(x), x)]$$

8. Elimiate conjunction (and transform to set of disjunctions:

$$[Animal(f(x)) \lor Loves(g(x), x)], [\neg Loves(x, g(x)) \lor Loves(g(x), x)]$$

9. Normalize variables:

$$[Animal(f(x)) \lor Loves(g(x), x)], [\neg Loves(y, g(y)) \lor Loves(g(y), y)]$$

Resolution & Unification

Resolution & Unification

Clauses and Resolution

- 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 $\Box$ (and denotes falsity).
- Set of clauses: $\triangle$ (e.g. our knowledge base)
- Set of literals: C, D
- Literal: l
- Negation of a literal: $\bar{l}$

Propositional Resolution

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

- $C_1 \cup C_2$ are called resolvents of the parent clauses $C_1 \dot\cup \{l\}$ and $C_2 \dot\cup \{l\}$. l and $\bar{l}$ are the resolution literals.
- Example: $\{a, b, \neg c\}$ resolves with $\{a, d, c\}$ to $\{a, b, d\}$.
- Note: The resolvent is not equivalent to the parent clauses, but it follows from them!
- Notation: $R(\triangle) = \triangle \cup \{C | C$ is a resolvent of two clauses from $\triangle\}$

First-order Resolution: What Changes

Examples (remember, in front we have the universial quantifiers)

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

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

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

- We need unification, a way to make literals identical.
- Based on the notion of substitution, e.g., $\{\frac{x}{0}\}$.

Note: inside the clauses, the "," means disjunction, while outside the clauses the "," means conjunction

Substitutions

- A substitution $s = \{\frac{v_1}{t_1}, ..., \frac{v_n}{t_n}\}$ 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

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

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

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

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

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

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

Composing Substitutions

- Composing substitutions s_1 and s_2 gives $s_1 s_2$ 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.
- Example:
$$\{\frac{z}{g(x,y)}\}\{\frac{x}{a},\frac{y}{b},\frac{w}{c},\frac{z}{d}\} = \{\frac{z}{g(a,b)},\frac{x}{a},\frac{y}{b},\frac{w}{c}\}$$
- Application example: $P(x,y,z) \rightarrow P(a,b,g(a,b))$

Properties of substitutions

For a formula ϕ and substitutions s_1, s_2

$$(\phi s_1)s_2 = \phi(s_1 s_2)$$

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

$s_1 s_2 \neq s_2 s_1$ 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 to alphabetic variants (variable renaming) for all mgus.

Disagreement Set

Unification algorithm will look for disagreements in expressions:

- 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

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

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

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

Unification Algorithm

Unify(Terms):

- 1. Step: $k \leftarrow 0$
- 2. Step: $T_k = Terms$
- 3. Step: $s_k = \varnothing$
- Loop:
  - 4 Step: If T_k is a singleton, then return s_k.
  - 5 Step: Let D_k be the disagreement set of T_k.
  - 6 Step: 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 Step: $s_{k+1} \leftarrow s_k\{\frac{v_k}{t_k}\}$
  - 8 Step: $T_{k+1} \leftarrow T_k\{\frac{v_k}{t_k}\}$
  - 9 Step: $k \leftarrow k + 1$
  - 10 Step: Continue with step 4.

Example

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

Binary Resolution

$$\frac{C_1 \dot{\cup} \{l_1\}, C_2 \dot{\cup} \{\overline{l_2}\}}{[C_1 \dot{\cup} C_2]s}$$

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

1st Example

- 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 $\{P(a), R(g(y))\}$

2nd Example

- 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 $\{Q(a, y), \neg R(b, z)\}$

Factoring

One more rule in PRL:

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

where $s = mgu(l_1, l_2)$, the most general unifier. (Remember that $\dot{\cup}\{l_1\}$ indicates that l_1 doesnt appear before.)

Why do we need this? Consider

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

but $\Box$ cannot be derived by binary resolution
Factoring yields:
$\{P(u)\}$ and $\{\neg P(x)\}$ whose resolvent is $\Box$.

Derivations

- Notation: $R(\triangle) = \triangle \cup \{C|C$ is a resolvent or a factor of two clauses from $\triangle\}$
- We say D can be derived from $\triangle$, i.e.,

$$\triangle \vdash D,$$

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

Properties of Resolution

Lemma: (soundness) If $\triangle \vdash D$, then $\triangle \models D$.

Lemma: resolution is refutation-complete: $\triangle$ is unsatisfiable implies $\triangle \vdash \square$

Theorem: $\triangle$ is unsatisfiable iff $\triangle \vdash \square$.

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

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.
- 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).

Example

- 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 (1/3)

- ...it is a crime for an American to sell weapons to hostile nations:

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

- Nono ...has some missiles, i.e., $\exists x Owns(Nono, x) \land Missile(x)$:

$$Owns(Nono, M1) \text{ and } Missile(M1)$$

- ...all of its missiles were sold to it by Colonel West.

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

- Missiles are weapons:

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

- An enemy of America counts as "hostile":

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

Example (2/3)

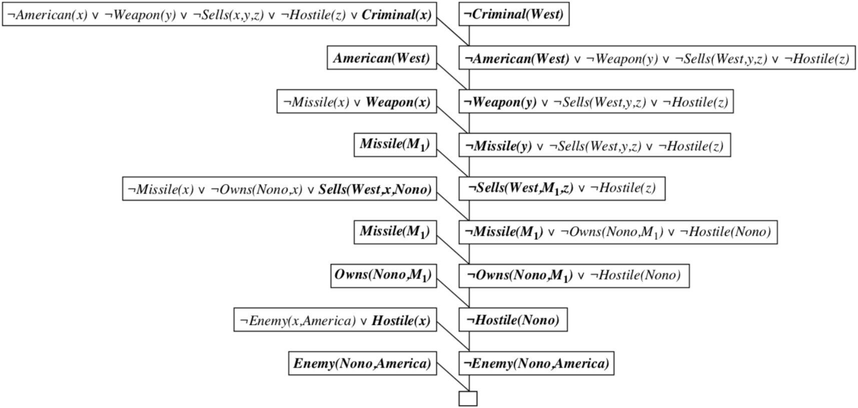
- West, who is American ...

$$American(West)$$

- The country Nono, an enemy of America

$$Enemy(Nono, America)$$

Example (3/3)



$\neg American(x) \lor \neg Weapon(y) \lor \neg Sells(x,y,z) \lor \neg Hostile(z) \lor \mathbf{Criminal(x)}$

$\mathbf{\neg Criminal(West)}$

$\mathbf{American(West)}$

$\mathbf{\neg American(West)} \lor \neg Weapon(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)$

$\neg Missile(x) \lor \mathbf{Weapon(x)}$

$\mathbf{\neg Weapon(y)} \lor \neg Sells(West,y,z) \lor \neg Hostile(z)$

$\mathbf{Missile(M_1)}$

$\mathbf{\neg Missile(y)} \lor \neg Sells(West,y,z) \lor \neg Hostile(z)$

$\neg Missile(x) \lor \neg Owns(Nono,x) \lor \mathbf{Sells(West,x,Nono)}$

$\mathbf{\neg Sells(West,M_1,z)} \lor \neg Hostile(z)$

$\mathbf{Missile(M_1)}$

$\mathbf{\neg Missile(M_1)} \lor \neg Owns(Nono,M_1) \lor \neg Hostile(Nono)$

$\mathbf{Owns(Nono,M_1)}$

$\mathbf{\neg Owns(Nono,M_1)} \lor \neg Hostile(Nono)$

$\neg Enemy(x,America) \lor \mathbf{Hostile(x)}$

$\mathbf{\neg Hostile(Nono)}$

$\mathbf{Enemy(Nono,America)}$

$\mathbf{\neg Enemy(Nono,America)}$

Closing Remarks

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:

$$\forall x[x = c_1 \vee x = c_2 \vee ... \vee x = c_n]$$

- Translation into finite propositional theory is possible.

Possible Extensions

- PL1 is definitely very expressive, but in some circumstances we would like more ...
- Second-Order Logic: Also over predicate quantifiers

$$\forall x, y[(x = y) \iff \{\forall p[p(x) \iff p(y)]\}]$$

- 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: $\exists! x \phi(x)$ - there is exactly one x ...
  - Reduction to PL1:

$$\exists x[\phi(x) \wedge \forall y(\phi(y) \Rightarrow x = y)]$$

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)