Logic-based Program Verification

First-Order Logic

Mădălina Erașcu   Tudor Jebelean

Research Institute for Symbolic Computation,
Johannes Kepler University, Linz, Austria

{merascu,tjebelea}@risc.jku.at

October 30 & November 6, 2013

Outline

Formula Clausification

Substitution & Unification

Resolution Principle for FOL

Outline

Formula Clausification

A clause is a disjunction of literals.

Examples: $\neg P[x] \vee Q[y, f[x]]$, $P[x]$

A set of clauses S is regarded as a conjunction of all clauses in S, where every variable in S is considered governed by a universal quantifier.

Example: Let

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

The standard form of the formula above, that is

$$\underset{x}{\forall}((\neg P[x, f[x]] \ \vee\ R[x, f[x], g[x]]) \ \wedge\ (Q(x, g[x]) \ \vee\ R[x, f[x], g[x]]))$$

can be represented by the following set of clauses

$$\{\neg P[x, f[x]] \ \vee\ R[x, f[x], g[x]], Q(x, g[x]) \ \vee\ R[x, f[x], g[x]]\}$$

Note that, if S is a set of clauses that represents a standard form of a formula F, then F is inconsistent iff S is inconsistent.

Formula Clausification

A clause is a disjunction of literals.

Examples: $\neg P[x] \vee Q[y, f[x]]$, $P[x]$

A set of clauses S is regarded as a conjunction of all clauses in S, where every variable in S is considered governed by a universal quantifier.

Example: Let

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

The standard form of the formula above, that is

$$\underset{x}{\forall}((\neg P[x, f[x]] \ \vee \ R[x, f[x], g[x]]) \ \wedge \ (Q(x, g[x]) \ \vee \ R[x, f[x], g[x]]))$$

can be represented by the following set of clauses

$$\{\neg P[x, f[x]] \ \vee \ R[x, f[x], g[x]], Q(x, g[x]) \ \vee \ R[x, f[x], g[x]]\}$$

Note that, if S is a set of clauses that represents a standard form of a formula F, then F is inconsistent iff S is inconsistent.

Formula Clausification

A clause is a disjunction of literals.

Examples: $\neg P[x] \vee Q[y, f[x]]$, $P[x]$

A set of clauses S is regarded as a conjunction of all clauses in S, where every variable in S is considered governed by a universal quantifier.

Example: Let

$$\forall_{x} \exists_{y,z} ((\neg P[x, y] \ \wedge \ Q[x, z]) \ \vee \ R[x, y, z])$$

The standard form of the formula above, that is

$$\forall_{x} ((\neg P[x, f[x]] \ \vee \ R[x, f[x], g[x]]) \ \wedge \ (Q(x, g[x]) \ \vee \ R[x, f[x], g[x]]))$$

can be represented by the following set of clauses

$$\{\neg P[x, f[x]] \ \vee \ R[x, f[x], g[x]], Q(x, g[x]) \ \vee \ R[x, f[x], g[x]]\}$$

Note that, if S is a set of clauses that represents a standard form of a formula F, then F is inconsistent iff S is inconsistent.

Formula Clausification

A clause is a disjunction of literals.

Examples: $\neg P[x] \vee Q[y, f[x]]$, $P[x]$

A set of clauses S is regarded as a conjunction of all clauses in S, where every variable in S is considered governed by a universal quantifier.

Example: Let

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

The standard form of the formula above, that is

$$\underset{x}{\forall}((\neg P[x,f[x]]\ \vee\ R[x,f[x],g[x]])\ \wedge\ (Q(x,g[x])\ \vee\ R[x,f[x],g[x]]))$$

can be represented by the following set of clauses

$$\{\neg P[x,f[x]]\ \vee\ R[x,f[x],g[x]], Q(x,g[x])\ \vee\ R[x,f[x],g[x]]\}$$

Note that, if S is a set of clauses that represents a standard form of a formula F, then F is inconsistent iff S is inconsistent.

Formula Clausification

A clause is a disjunction of literals.

Examples: $\neg P[x] \vee Q[y, f[x]]$, $P[x]$

A set of clauses S is regarded as a conjunction of all clauses in S, where every variable in S is considered governed by a universal quantifier.

Example: Let

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

The standard form of the formula above, that is

$$\underset{x}{\forall} ((\neg P[x, f[x]] \ \vee \ R[x, f[x], g[x]]) \ \wedge \ (Q(x, g[x]) \ \vee \ R[x, f[x], g[x]]))$$

can be represented by the following set of clauses

$$\{\neg P[x, f[x]] \ \vee \ R[x, f[x], g[x]], Q(x, g[x]) \ \vee \ R[x, f[x], g[x]]\}$$

Note that, if S is a set of clauses that represents a standard form of a formula F, then F is inconsistent iff S is inconsistent.

Formulas Clausification (cont'd)

Example :

Transform the formulas F_1, F_2,F_3,F_4, and $\neg G$ into a set of clauses, where

$$F_1: \quad \underset{x,y}{\forall}\ \underset{z}{\exists} P[x,y,z]$$

$$F_2: \quad \begin{array}{l} \underset{x,y,z,u,v,w}{\forall} (P[x,y,u] \ \wedge\ P[y,z,v] \ \wedge\ P[u,z,w] \ \Rightarrow\ P[x,v,w]) \\ \wedge \\ \underset{x,y,z,u,v,w}{\forall} (P[x,y,u] \ \wedge\ P[y,z,v] \ \wedge\ P[x,v,w] \ \Rightarrow\ P[u,z,w]) \end{array}$$

$$F_3: \quad \underset{x}{\forall} P[x,e,x] \wedge \underset{x}{\forall} P[e,x,x]$$

$$F_4: \quad \underset{x}{\forall} P[x,i[x],e] \wedge \underset{x}{\forall} P[i[x],x,e]$$

$$G: \quad \underset{x}{\forall} P[x,x,e] \ \Rightarrow\ \underset{u,v,w}{\forall} (P[u,v,w] \ \Rightarrow\ P[v,u,w])$$

Outline

Substitution & Unification

Motivation: apply resolution principle to FOL formulas.

Example: Let

$$C_1 : \quad P[x] \vee Q[x]$$
$$C_2 : \quad \neg P[f[x]] \vee R[x]$$

Substitution & Unification

Motivation: apply resolution principle to FOL formulas.

Example: Let

$$\begin{array}{ll} C_1 : & P[x] \vee Q[x] \\ C_2 : & \neg P[f[x]] \vee R[x] \end{array}$$

Substitution & Unification

Motivation: apply resolution principle to FOL formulas.

Example: Let

$$
\begin{array}{ll}
C_1: & P[x] \vee Q[x] \\
C_2: & \neg P[f[x]] \vee R[x]
\end{array}
$$

Let $x \to f[a]$ in C_1, $x \to a$ in C_2.

Substitution & Unification

Motivation: apply resolution principle to FOL formulas.

Example: Let

$$\begin{array}{ll} C_1: & P[x] \vee Q[x] \\ C_2: & \neg P[f[x]] \vee R[x] \end{array}$$

Let $x \rightarrow f[a]$ in C_1, $x \rightarrow a$ in C_2.

We have

$$\begin{array}{ll} C_1': & P[f[a]] \vee Q[f[a]] \\ C_2': & \neg P[f[a]] \vee R[a] \end{array}$$

Substitution & Unification

Motivation: apply resolution principle to FOL formulas.

Example: Let

$$C_1 : \quad P[x] \vee Q[x]$$
$$C_2 : \quad \neg P[f[x]] \vee R[x]$$

Let $x \to f[a]$ in C_1, $x \to a$ in C_2.

We have

$$C_1' : \quad P[f[a]] \vee Q[f[a]]$$
$$C_2' : \quad \neg P[f[a]] \vee R[a]$$

C_1' and C_2' are ground instances.

Substitution & Unification

Motivation: apply resolution principle to FOL formulas.

Example: Let

$$C_1: \quad P[x] \lor Q[x]$$
$$C_2: \quad \neg P[f[x]] \lor R[x]$$

Let $x \to f[a]$ in C_1, $x \to a$ in C_2.

We have

$$C_1': \quad P[f[a]] \lor Q[f[a]]$$
$$C_2': \quad \neg P[f[a]] \lor R[a]$$

C_1' and C_2' are ground instances.

A resolvent of C_1' and C_2' is

$$C_3': \quad Q[f[a]] \lor R[a]$$

Substitution & Unification

Motivation: apply resolution principle to FOL formulas.

Example: Let

$$\begin{array}{ll} C_1 : & P[x] \vee Q[x] \\ C_2 : & \neg P[f[x]] \vee R[x] \end{array}$$

Let $x \to f[x]$ in C_1. We have

$$C_1^* : \quad P[f[x]] \vee Q[f[x]]$$

C_1^* is an instance of C_1.

A resolvent of

$$\begin{array}{ll} C_2 : & \neg P[f[x]] \vee R[x] \\ C_1^* : & P[f[x]] \vee Q[f[x]] \end{array}$$

is

$$C_3 : \quad Q[f[x]] \vee R[x]$$

C_3' is an instance of C_3. C_3 is the most general clause.

Substitution & Unification

Motivation: apply resolution principle to FOL formulas.

Example: Let

$$\begin{array}{ll} C_1: & P[x] \vee Q[x] \\ C_2: & \neg P[f[x]] \vee R[x] \end{array}$$

Let $x \rightarrow f[x]$ in C_1. We have

$$C_1^*: \quad P[f[x]] \vee Q[f[x]]$$

C_1^* is an instance of C_1.

A resolvent of

$$\begin{array}{ll} C_2: & \neg P[f[x]] \vee R[x] \\ C_1^*: & P[f[x]] \vee Q[f[x]] \end{array}$$

is

$$C_3: \quad Q[f[x]] \vee R[x]$$

C_3' is an instance of C_3. C_3 is the most general clause.

Substitution & Unification

Motivation: apply resolution principle to FOL formulas.

Example: Let

$$C_1: \quad P[x] \vee Q[x]$$
$$C_2: \quad \neg P[f[x]] \vee R[x]$$

Let $x \rightarrow f[x]$ in C_1. We have

$$C_1^*: \quad P[f[x]] \vee Q[f[x]]$$

C_1^* is an instance of C_1.

A resolvent of

$$C_2: \quad \neg P[f[x]] \vee R[x]$$
$$C_1^*: \quad P[f[x]] \vee Q[f[x]]$$

is

$$C_3: \quad Q[f[x]] \vee R[x]$$

C_3' is an instance of C_3. C_3 is the most general clause.

Substitution & Unification

Motivation: apply resolution principle to FOL formulas.

Example: Let

$$C_1 : \quad P[x] \vee Q[x]$$
$$C_2 : \quad \neg P[f[x]] \vee R[x]$$

Let $x \to f[x]$ in C_1. We have

$$C_1^* : \quad P[f[x]] \vee Q[f[x]]$$

C_1^* is an instance of C_1.

A resolvent of

$$C_2 : \quad \neg P[f[x]] \vee R[x]$$
$$C_1^* : \quad P[f[x]] \vee Q[f[x]]$$

is

$$C_3 : \quad Q[f[x]] \vee R[x]$$

C_3' is an instance of C_3. C_3 is the most general clause.

Substitution & Unification

Motivation: apply resolution principle to FOL formulas.

Example: Let

$$\begin{array}{ll} C_1 : & P[x] \vee Q[x] \\ C_2 : & \neg P[f[x]] \vee R[x] \end{array}$$

Let $x \to f[x]$ in C_1. We have

$$C_1^* : \quad P[f[x]] \vee Q[f[x]]$$

C_1^* is an instance of C_1.

A resolvent of

$$\begin{array}{ll} C_2 : & \neg P[f[x]] \vee R[x] \\ C_1^* : & P[f[x]] \vee Q[f[x]] \end{array}$$

is

$$C_3 : \quad Q[f[x]] \vee R[x]$$

C_3' is an instance of C_3. C_3 is the most general clause.

Substitution (cont'd)

A substitution σ is a finite set of the form $\{v_1 \to t_1, ..., v_n \to t_n\}$ where every t_i is a term different from v_i and no two elements in the set have the same variable v_i.

Let σ be defined as above and E be an expression. Then $E\sigma$ is an expression obtained from E by replacing simultaneously each occurrence of v_i in E by the term t_i

Example: Let $\sigma = \{x \to z, z \to h[a, y]\}$ and $E = f[z, a, g[x], y]$. Then $E\sigma = f[h[a, y], a, g[z], y]$.

Substitution (cont'd)

A substitution σ is a finite set of the form $\{v_1 \to t_1, ..., v_n \to t_n\}$ where every t_i is a term different from v_i and no two elements in the set have the same variable v_i.

Let σ be defined as above and E be an expression. Then $E\sigma$ is an expression obtained from E by replacing simultaneously each occurrence of v_i in E by the term t_i

Example: Let $\sigma = \{x \to z, z \to h[a, y]\}$ and $E = f[z, a, g[x], y]$. Then $E\sigma = f[h[a, y], a, g[z], y]$.

Substitution (cont'd)

A substitution σ is a finite set of the form $\{v_1 \to t_1, ..., v_n \to t_n\}$ where every t_i is a term different from v_i and no two elements in the set have the same variable v_i.

Let σ be defined as above and E be an expression. Then $E\sigma$ is an expression obtained from E by replacing simultaneously each occurrence of v_i in E by the term t_i

Example: Let $\sigma = \{x \to z, z \to h[a, y]\}$ and $E = f[z, a, g[x], y]$. Then $E\sigma = f[h[a, y], a, g[z], y]$.

Substitution (cont'd)

Let

$$\theta = \{x_1 \to t_1, ..., x_n \to t_n\}$$
$$\lambda = \{y_1 \to u_1, ..., y_n \to u_n\}$$

Then the composition of θ and λ $(\theta \circ \lambda)$ is obtained from the set

$$\{x_1 \to t_1\lambda, ..., x_n \to t_n\lambda, y_1 \to u_1, ..., y_n \to u_n\}$$

by deleting any element $x_j \to t_j\lambda$ for which $x_j = t_j\lambda$ and any element $y_i \to u_i$ such that y_i is among $\{x_1, ..., x_n\}$.

Substitution (cont'd)

Example 1:

$$\theta = \{x \to f[y], y \to z\}$$
$$\lambda = \{x \to a, y \to b, z \to y\}$$

Then

$$\theta \circ \lambda = \{x \to f[b], y \to y, x \to a, y \to b, z \to y\}$$
$$= \{x \to f[b], z \to y\}$$

Example 2:

$$\theta_1 = \{x \to a, y \to f[z], z \to y\}$$
$$\theta_2 = \{x \to b, y \to z, z \to g[x]\}$$

Then

$$\theta_1 \circ \theta_2 = \{x \to a, y \to f[g[x]], z \to z, x \to b, y \to z, z \to g[x]\}$$
$$= \{x \to a, y \to f[g[x]]\}$$

Substitution (cont'd)

Example 1:

$$\theta = \{x \to f[y], y \to z\}$$
$$\lambda = \{x \to a, y \to b, z \to y\}$$

Then

$$\theta \circ \lambda = \{x \to f[b], y \to y, x \to a, y \to b, z \to y\}$$
$$= \{x \to f[b], z \to y\}$$

Example 2:

$$\theta_1 = \{x \to a, y \to f[z], z \to y\}$$
$$\theta_2 = \{x \to b, y \to z, z \to g[x]\}$$

Then

$$\theta_1 \circ \theta_2 = \{x \to a, y \to f[g[x]], z \to z, x \to b, y \to z, z \to g[x]\}$$
$$= \{x \to a, y \to f[g[x]]\}$$

Substitution (cont'd)

Example 1:

$$\theta = \{x \to f[y], y \to z\}$$
$$\lambda = \{x \to a, y \to b, z \to y\}$$

Then

$$\theta \circ \lambda = \{x \to f[b], y \to y, x \to a, y \to b, z \to y\}$$
$$= \{x \to f[b], z \to y\}$$

Example 2:

$$\theta_1 = \{x \to a, y \to f[z], z \to y\}$$
$$\theta_2 = \{x \to b, y \to z, z \to g[x]\}$$

Then

$$\theta_1 \circ \theta_2 = \{x \to a, y \to f[g[x]], z \to z, x \to b, y \to z, z \to g[x]\}$$
$$= \{x \to a, y \to f[g[x]]\}$$

Substitution (cont'd)

Example 1:

$$\theta = \{x \to f[y], y \to z\}$$
$$\lambda = \{x \to a, y \to b, z \to y\}$$

Then

$$\begin{aligned}\theta \circ \lambda &= \{x \to f[b], y \to y, x \to a, y \to b, z \to y\} \\ &= \{x \to f[b], z \to y\}\end{aligned}$$

Example 2:

$$\theta_1 = \{x \to a, y \to f[z], z \to y\}$$
$$\theta_2 = \{x \to b, y \to z, z \to g[x]\}$$

Then

$$\begin{aligned}\theta_1 \circ \theta_2 &= \{x \to a, y \to f[g[x]], z \to z, x \to b, y \to z, z \to g[x]\} \\ &= \{x \to a, y \to f[g[x]]\}\end{aligned}$$

Unification

A substitution θ is called a unifier for a set $\{E_1, ..., E_k\}$ iff $E_1\theta = ... = E_k\theta$. The set $\{E_1, ..., E_k\}$ is said to be unifiable iff there exists an unifier for it.

A unifier σ for a set $\{E_1, ..., E_k\}$ of expressions is a most general unifier iff for each unifier θ for the set there is a substitution λ such that $\theta = \sigma \circ \lambda$.

Example: The set $\{P[a, y], P[x, f[b]]\}$ is unifiable since $\sigma = \{x \to a, y \to f[b]\}$ is a unifier for the set.

Unification

A substitution θ is called a unifier for a set $\{E_1, ..., E_k\}$ iff $E_1\theta = ... = E_k\theta$. The set $\{E_1, ..., E_k\}$ is said to be unifiable iff there exists an unifier for it.

A unifier σ for a set $\{E_1, ..., E_k\}$ of expressions is a most general unifier iff for each unifier θ for the set there is a substitution λ such that $\theta = \sigma \circ \lambda$.

Example: The set $\{P[a, y], P[x, f[b]]\}$ is unifiable since $\sigma = \{x \to a, y \to f[b]\}$ is a unifier for the set.

Unification

A substitution θ is called a unifier for a set $\{E_1, ..., E_k\}$ iff $E_1\theta = ... = E_k\theta$. The set $\{E_1, ..., E_k\}$ is said to be unifiable iff there exists an unifier for it.

A unifier σ for a set $\{E_1, ..., E_k\}$ of expressions is a most general unifier iff for each unifier θ for the set there is a substitution λ such that $\theta = \sigma \circ \lambda$.

Example: The set $\{P[a, y], P[x, f[b]]\}$ is unifiable since $\sigma = \{x \to a, y \to f[b]\}$ is a unifier for the set.

Unification Algorithm

Unification algorithm for finding a most general unifier (mgu), or its nonexistence, for a finite set of nonempty expressions.

The disagreement set of a nonempty set W of expressions is obtained by

- locating the first symbol (starting from the left) at which not all the expressions in W have exactly the same symbol and then
- extracting from each expression in W the subexpression that begins with the symbol occupying that position.

Example: The disagreement set of $\{P[a, x, f[g[y]]], P[z, f[z], f[u]]\}$ is $\{a, z\}$.

Unification Algorithm

Unification algorithm for finding a most general unifier (mgu), or its nonexistence, for a finite set of nonempty expressions.

The disagreement set of a nonempty set W of expressions is obtained by

- locating the first symbol (starting from the left) at which not all the expressions in W have exactly the same symbol and then
- extracting from each expression in W the subexpression that begins with the symbol occupying that position.

Example: The disagreement set of $\{P[a, x, f[g[y]]], P[z, f[z], f[u]]\}$ is $\{a, z\}$.

Unification Algorithm

Unification algorithm for finding a most general unifier (mgu), or its nonexistence, for a finite set of nonempty expressions.

The disagreement set of a nonempty set W of expressions is obtained by

- locating the first symbol (starting from the left) at which not all the expressions in W have exactly the same symbol and then
- extracting from each expression in W the subexpression that begins with the symbol occupying that position.

Example: The disagreement set of $\{P[a, x, f[g[y]]], P[z, f[z], f[u]]\}$ is $\{a, z\}$.

Unification Algorithm

Unification algorithm for finding a most general unifier (mgu), or its nonexistence, for a finite set of nonempty expressions.

The disagreement set of a nonempty set W of expressions is obtained by

- locating the first symbol (starting from the left) at which not all the expressions in W have exactly the same symbol and then
- extracting from each expression in W the subexpression that begins with the symbol occupying that position.

Example: The disagreement set of $\{P[a, x, f[g[y]]], P[z, f[z], f[u]]\}$ is $\{a, z\}$.

Unification Algorithm

Unification algorithm for finding a most general unifier (mgu), or its nonexistence, for a finite set of nonempty expressions.

The disagreement set of a nonempty set W of expressions is obtained by

- locating the first symbol (starting from the left) at which not all the expressions in W have exactly the same symbol and then
- extracting from each expression in W the subexpression that begins with the symbol occupying that position.

Example: The disagreement set of $\{P[a, x, f[g[y]]], P[z, f[z], f[u]]\}$ is $\{a, z\}$.

Unification Algorithm (cont'd)

Unification Algorithm

1. $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$
2. If W_k is singleton then stop; σ_k is mgu of W. Otherwise find the disagreement set D_k of W_k.
3. If there exists $v_k,\ t_k \in D_k$ s.t. v_k is a variable which does not occur in t_k, go to 4. Otherwise, stop; W is not unifiable.
4. Let $\sigma_{k+1} = \sigma_k \circ \{v_k \to t_k\}$ and $W_{k+1} = W_k\{v_k \to t_k\}$.
5. $k = k + 1$ and go to 2.

Example: Find a most general unifier for

1. $W = \{P[a, x, f[g[y]]],\ P[z, f[z], f[u]]\}$
2. $W = \{Q[a],\ Q[b]\}$
3. $W = \{P[x],\ P[f[x]]\}$
4. $W = \{P[x],\ Q[y]\}$

Unification Algorithm (cont'd)

Unification Algorithm

1. $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$
2. If W_k is singleton then stop; σ_k is mgu of W. Otherwise find the disagreement set D_k of W_k.
3. If there exists $v_k,\ t_k \in D_k$ s.t. v_k is a variable which does not occur in t_k, go to 4. Otherwise, stop; W is not unifiable.
4. Let $\sigma_{k+1} = \sigma_k \circ \{v_k \to t_k\}$ and $W_{k+1} = W_k\{v_k \to t_k\}$.
5. $k = k + 1$ and go to 2.

Example: Find a most general unifier for

1. $W = \{P[a, x, f[g[y]]],\ P[z, f[z], f[u]]\}$
2. $W = \{Q[a],\ Q[b]\}$
3. $W = \{P[x],\ P[f[x]]\}$
4. $W = \{P[x],\ Q[y]\}$

Unification Algorithm (cont'd)

Unification Algorithm

1. $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$
2. If W_k is singleton then stop; σ_k is mgu of W. Otherwise find the disagreement set D_k of W_k.
3. If there exists $v_k,\ t_k \in D_k$ s.t. v_k is a variable which does not occur in t_k, go to 4. Otherwise, stop; W is not unifiable.
4. Let $\sigma_{k+1} = \sigma_k \circ \{v_k \to t_k\}$ and $W_{k+1} = W_k\{v_k \to t_k\}$.
5. $k = k + 1$ and go to 2.

Example: Find a most general unifier for

1. $W = \{P[a, x, f[g[y]]],\ P[z, f[z], f[u]]\}$
2. $W = \{Q[a],\ Q[b]\}$
3. $W = \{P[x],\ P[f[x]]\}$
4. $W = \{P[x],\ Q[y]\}$

Unification Algorithm (cont'd)

Unification Algorithm

1. $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$
2. If W_k is singleton then stop; σ_k is mgu of W. Otherwise find the disagreement set D_k of W_k.
3. If there exists $v_k,\ t_k \in D_k$ s.t. v_k is a variable which does not occur in t_k, go to 4. Otherwise, stop; W is not unifiable.
4. Let $\sigma_{k+1} = \sigma_k \circ \{v_k \to t_k\}$ and $W_{k+1} = W_k\{v_k \to t_k\}$.
5. $k = k + 1$ and go to 2.

Example: Find a most general unifier for

1. $W = \{P[a, x, f[g[y]]],\ P[z, f[z], f[u]]\}$
2. $W = \{Q[a],\ Q[b]\}$
3. $W = \{P[x],\ P[f[x]]\}$
4. $W = \{P[x],\ Q[y]\}$

Unification Algorithm (cont'd)

Unification Algorithm

1. $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$
2. If W_k is singleton then stop; σ_k is mgu of W. Otherwise find the disagreement set D_k of W_k.
3. If there exists $v_k,\ t_k \in D_k$ s.t. v_k is a variable which does not occur in t_k, go to 4. Otherwise, stop; W is not unifiable.
4. Let $\sigma_{k+1} = \sigma_k \circ \{v_k \to t_k\}$ and $W_{k+1} = W_k\{v_k \to t_k\}$.
5. $k = k + 1$ and go to 2.

Example: Find a most general unifier for

1. $W = \{P[a, x, f[g[y]]],\ P[z, f[z], f[u]]\}$
2. $W = \{Q[a],\ Q[b]\}$
3. $W = \{P[x],\ P[f[x]]\}$
4. $W = \{P[x],\ Q[y]\}$

Unification Algorithm (cont'd)

Unification Algorithm

1. $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$
2. If W_k is singleton then stop; σ_k is mgu of W. Otherwise find the disagreement set D_k of W_k.
3. If there exists $v_k,\ t_k \in D_k$ s.t. v_k is a variable which does not occur in t_k, go to 4. Otherwise, stop; W is not unifiable.
4. Let $\sigma_{k+1} = \sigma_k \circ \{v_k \to t_k\}$ and $W_{k+1} = W_k\{v_k \to t_k\}$.
5. $k = k + 1$ and go to 2.

Example: Find a most general unifier for

1. $W = \{P[a, x, f[g[y]]],\ P[z, f[z], f[u]]\}$
2. $W = \{Q[a],\ Q[b]\}$
3. $W = \{P[x],\ P[f[x]]\}$
4. $W = \{P[x],\ Q[y]\}$

Unification Algorithm (cont'd)

Unification Algorithm

1. $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$
2. If W_k is singleton then stop; σ_k is mgu of W. Otherwise find the disagreement set D_k of W_k.
3. If there exists $v_k,\ t_k \in D_k$ s.t. v_k is a variable which does not occur in t_k, go to 4. Otherwise, stop; W is not unifiable.
4. Let $\sigma_{k+1} = \sigma_k \circ \{v_k \to t_k\}$ and $W_{k+1} = W_k\{v_k \to t_k\}$.
5. $k = k + 1$ and go to 2.

Example: Find a most general unifier for

1. $W = \{P[a, x, f[g[y]]],\ P[z, f[z], f[u]]\}$
2. $W = \{Q[a],\ Q[b]\}$
3. $W = \{P[x],\ P[f[x]]\}$
4. $W = \{P[x],\ Q[y]\}$

Unification Algorithm (cont'd)

Unification Algorithm

1. $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$
2. If W_k is singleton then stop; σ_k is mgu of W. Otherwise find the disagreement set D_k of W_k.
3. If there exists $v_k,\ t_k \in D_k$ s.t. v_k is a variable which does not occur in t_k, go to 4. Otherwise, stop; W is not unifiable.
4. Let $\sigma_{k+1} = \sigma_k \circ \{v_k \to t_k\}$ and $W_{k+1} = W_k\{v_k \to t_k\}$.
5. $k = k + 1$ and go to 2.

Example: Find a most general unifier for

1. $W = \{P[a, x, f[g[y]]],\ P[z, f[z], f[u]]\}$
2. $W = \{Q[a],\ Q[b]\}$
3. $W = \{P[x],\ P[f[x]]\}$
4. $W = \{P[x],\ Q[y]\}$

Unification Algorithm (cont'd)

Unification Algorithm

1. $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$
2. If W_k is singleton then stop; σ_k is mgu of W. Otherwise find the disagreement set D_k of W_k.
3. If there exists $v_k,\ t_k \in D_k$ s.t. v_k is a variable which does not occur in t_k, go to 4. Otherwise, stop; W is not unifiable.
4. Let $\sigma_{k+1} = \sigma_k \circ \{v_k \to t_k\}$ and $W_{k+1} = W_k\{v_k \to t_k\}$.
5. $k = k + 1$ and go to 2.

Example: Find a most general unifier for

1. $W = \{P[a, x, f[g[y]]],\ P[z, f[z], f[u]]\}$
2. $W = \{Q[a],\ Q[b]\}$
3. $W = \{P[x],\ P[f[x]]\}$
4. $W = \{P[x],\ Q[y]\}$

Unification Algorithm (cont'd)

Unification Algorithm

1. $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$
2. If W_k is singleton then stop; σ_k is mgu of W. Otherwise find the disagreement set D_k of W_k.
3. If there exists $v_k,\ t_k \in D_k$ s.t. v_k is a variable which does not occur in t_k, go to 4. Otherwise, stop; W is not unifiable.
4. Let $\sigma_{k+1} = \sigma_k \circ \{v_k \to t_k\}$ and $W_{k+1} = W_k\{v_k \to t_k\}$.
5. $k = k + 1$ and go to 2.

Example: Find a most general unifier for

1. $W = \{P[a, x, f[g[y]]],\ P[z, f[z], f[u]]\}$
2. $W = \{Q[a],\ Q[b]\}$
3. $W = \{P[x],\ P[f[x]]\}$
4. $W = \{P[x],\ Q[y]\}$

Unification Algorithm (cont'd)

Unification Algorithm

1. $k := 0$, $W_k := W$, $\sigma_k := \varepsilon$
2. If W_k is singleton then stop; σ_k is mgu of W. Otherwise find the disagreement set D_k of W_k.
3. If there exists $v_k,\ t_k \in D_k$ s.t. v_k is a variable which does not occur in t_k, go to 4. Otherwise, stop; W is not unifiable.
4. Let $\sigma_{k+1} = \sigma_k \circ \{v_k \to t_k\}$ and $W_{k+1} = W_k\{v_k \to t_k\}$.
5. $k = k + 1$ and go to 2.

Example: Find a most general unifier for

1. $W = \{P[a, x, f[g[y]]],\ P[z, f[z], f[u]]\}$
2. $W = \{Q[a],\ Q[b]\}$
3. $W = \{P[x],\ P[f[x]]\}$
4. $W = \{P[x],\ Q[y]\}$

Outline

Formula Clausification

Substitution & Unification

Resolution Principle for FOL

Resolution Principle for FOL

If two or more literals (with the same sign) of a clause C have σ the mgu, then $C\sigma$ is called a factor of C.

Example: Let $C: \; P[x] \;\vee\; P[a] \;\vee\; Q[f[x]] \;\vee Q[f[a]]$ be a clause. Then the mgu is $\sigma = \{x \to a\}$ and $C\sigma: \; P[a] \;\vee Q[f[a]]$ is a factor of C.

Let C_1 and C_2 be two clauses with *no variables in common*. Let L_1 and L_2 be two literals in C_1 and C_2, respectively. If L_1 and $\neg L_2$ have mgu σ, then the clause $C_1\sigma \;\vee\; C_2\sigma$ is called a binary resolvent of C_1 and C_2.

Example: Let

$$C_1: \quad P[x] \vee Q[x]$$

$$\sout{C_2: \quad \neg P[a] \vee R[x]}$$

By renaming x with y in C_2, we have

$$C_2: \qquad \neg P[a] \vee R[y]$$

Let $\sigma = \{x \to a\}$ a mgu of the literals $P[x]$ and $\neg P[a]$. Then a binary resolvent of C_1 and C_2 is $Q[a] \vee R[y]$.

Resolution Principle for FOL

If two or more literals (with the same sign) of a clause C have σ the mgu, then $C\sigma$ is called a factor of C.

Example: Let $C: \; P[x] \; \vee \; P[a] \; \vee \; Q[f[x]] \; \vee Q[f[a]]$ be a clause. Then the mgu is $\sigma = \{x \to a\}$ and $C\sigma: \; P[a] \; \vee Q[f[a]]$ is a factor of C.

Let C_1 and C_2 be two clauses with *no variables in common*. Let L_1 and L_2 be two literals in C_1 and C_2, respectively. If L_1 and $\neg L_2$ have mgu σ, then the clause $C_1\sigma \; \vee \; C_2\sigma$ is called a binary resolvent of C_1 and C_2.

Example: Let

$$C_1 : \quad P[x] \vee Q[x]$$

~~$C_2 : \quad \neg P[a] \vee R[x]$~~

By renaming x with y in C_2, we have

$$C_2 : \qquad \neg P[a] \vee R[y]$$

Let $\sigma = \{x \to a\}$ a mgu of the literals $P[x]$ and $\neg P[a]$. Then a binary resolvent of C_1 and C_2 is $Q[a] \vee R[y]$.

Resolution Principle for FOL

If two or more literals (with the same sign) of a clause C have σ the mgu, then $C\sigma$ is called a factor of C.

Example: Let $C: \; P[x] \; \vee \; P[a] \; \vee \; Q[f[x]] \; \vee Q[f[a]]$ be a clause. Then the mgu is $\sigma = \{x \to a\}$ and $C\sigma: \; P[a] \; \vee Q[f[a]]$ is a factor of C.

Let C_1 and C_2 be two clauses with *no variables in common*. Let L_1 and L_2 be two literals in C_1 and C_2, respectively. If L_1 and $\neg L_2$ have mgu σ, then the clause $C_1\sigma \; \vee \; C_2\sigma$ is called a binary resolvent of C_1 and C_2.

Example: Let

$$C_1: \quad P[x] \vee Q[x]$$

$$C_2: \quad \neg P[a] \vee R[x]$$

By renaming x with y in C_2, we have

$$C_2: \qquad \neg P[a] \vee R[y]$$

Let $\sigma = \{x \to a\}$ a mgu of the literals $P[x]$ and $\neg P[a]$. Then a binary resolvent of C_1 and C_2 is $Q[a] \vee R[y]$.

Resolution Principle for FOL

If two or more literals (with the same sign) of a clause C have σ the mgu, then $C\sigma$ is called a factor of C.

Example: Let $C: \; P[x] \; \vee \; P[a] \; \vee \; Q[f[x]] \; \vee Q[f[a]]$ be a clause. Then the mgu is $\sigma = \{x \to a\}$ and $C\sigma: \; P[a] \; \vee Q[f[a]]$ is a factor of C.

Let C_1 and C_2 be two clauses with *no variables in common*. Let L_1 and L_2 be two literals in C_1 and C_2, respectively. If L_1 and $\neg L_2$ have mgu σ, then the clause $C_1\sigma \; \vee \; C_2\sigma$ is called a binary resolvent of C_1 and C_2.

Example: Let

$$C_1: \quad P[x] \vee Q[x]$$
$$C_2: \quad \neg P[a] \vee R[x]$$

By renaming x with y in C_2, we have

$$C_2: \qquad \neg P[a] \vee R[y]$$

Let $\sigma = \{x \to a\}$ a mgu of the literals $P[x]$ and $\neg P[a]$. Then a binary resolvent of C_1 and C_2 is $Q[a] \vee R[y]$.

Resolution Principle for FOL

If two or more literals (with the same sign) of a clause C have σ the mgu, then $C\sigma$ is called a factor of C.

Example: Let $C: \; P[x] \; \vee \; P[a] \; \vee \; Q[f[x]] \; \vee Q[f[a]]$ be a clause. Then the mgu is $\sigma = \{x \to a\}$ and $C\sigma: \; P[a] \; \vee Q[f[a]]$ is a factor of C.

Let C_1 and C_2 be two clauses with *no variables in common*. Let L_1 and L_2 be two literals in C_1 and C_2, respectively. If L_1 and $\neg L_2$ have mgu σ, then the clause $C_1\sigma \; \vee \; C_2\sigma$ is called a binary resolvent of C_1 and C_2.

Example: Let

$$C_1: \quad P[x] \vee Q[x]$$

$$\sout{C_2: \quad \neg P[a] \vee R[x]}$$

By renaming x with y in C_2, we have

$$C_2: \qquad \neg P[a] \vee R[y]$$

Let $\sigma = \{x \to a\}$ a mgu of the literals $P[x]$ and $\neg P[a]$. Then a binary resolvent of C_1 and C_2 is $Q[a] \vee R[y]$.

Resolution Principle for FOL

If two or more literals (with the same sign) of a clause C have σ the mgu, then $C\sigma$ is called a factor of C.

Example: Let $C: \ P[x] \ \vee \ P[a] \ \vee \ Q[f[x]] \ \vee Q[f[a]]$ be a clause. Then the mgu is $\sigma = \{x \to a\}$ and $C\sigma: \ P[a] \ \vee Q[f[a]]$ is a factor of C.

Let C_1 and C_2 be two clauses with *no variables in common*. Let L_1 and L_2 be two literals in C_1 and C_2, respectively. If L_1 and $\neg L_2$ have mgu σ, then the clause $C_1\sigma \ \vee \ C_2\sigma$ is called a binary resolvent of C_1 and C_2.

Example: Let

$$C_1: \quad P[x] \vee Q[x]$$

$$C_2: \quad \neg P[a] \vee R[x]$$

By renaming x with y in C_2, we have

$$C_2: \qquad \neg P[a] \vee R[y]$$

Let $\sigma = \{x \to a\}$ a mgu of the literals $P[x]$ and $\neg P[a]$. Then a binary resolvent of C_1 and C_2 is $Q[a] \vee R[y]$.

Resolution Principle for FOL (cont'd)

Resolution: (Robinson, 1965)

- is an inference rule which generates resolvents from a set of clauses
- is a refutation proof procedure: empty clause is tried to be derived from a set of clauses
- is refutationally complete: a set of clauses is unsatisfiable iff the empty clause can be derived

How does resolution work?

Given: formulas F_1, ..., F_n

Prove: G by resolution.

1. Bring F_1, ..., F_n, ..., $\neg G$ into standard form and write the clauses which are obtained
2. Start derivation and try to obtain the empty clause from the set C of clauses.
3. In the derivation use resolution inference rule and factoring rules to derive new clauses; these new clauses are added to C.
4. If the empty clause appears, stop: Contradiction found, G is proved.
5. If no step can be made and the empty clause is not found, then H can not be proved.

Resolution Principle for FOL (cont'd)

Resolution: (Robinson, 1965)

- is an inference rule which generates resolvents from a set of clauses
- is a refutation proof procedure: empty clause is tried to be derived from a set of clauses
- is refutationally complete: a set of clauses is unsatisfiable iff the empty clause can be derived

How does resolution work?

Given: formulas F_1, ..., F_n

Prove: G by resolution.

1. Bring F_1, ..., F_n, ..., $\neg G$ into standard form and write the clauses which are obtained
2. Start derivation and try to obtain the empty clause from the set C of clauses.
3. In the derivation use resolution inference rule and factoring rules to derive new clauses; these new clauses are added to C.
4. If the empty clause appears, stop: Contradiction found, G is proved.
5. If no step can be made and the empty clause is not found, then H can not be proved.

Resolution Principle for FOL (cont'd)

Resolution: (Robinson, 1965)

- is an inference rule which generates resolvents from a set of clauses
- is a refutation proof procedure: empty clause is tried to be derived from a set of clauses
- is refutationally complete: a set of clauses is unsatisfiable iff the empty clause can be derived

How does resolution work?

Given: formulas F_1, ..., F_n

Prove: G by resolution.

1. Bring F_1, ..., F_n, ..., $\neg G$ into standard form and write the clauses which are obtained
2. Start derivation and try to obtain the empty clause from the set C of clauses.
3. In the derivation use resolution inference rule and factoring rules to derive new clauses; these new clauses are added to C.
4. If the empty clause appears, stop: Contradiction found, G is proved.
5. If no step can be made and the empty clause is not found, then H can not be proved.

Resolution Principle for FOL (cont'd)

Resolution: (Robinson, 1965)

- is an inference rule which generates resolvents from a set of clauses
- is a refutation proof procedure: empty clause is tried to be derived from a set of clauses
- is refutationally complete: a set of clauses is unsatisfiable iff the empty clause can be derived

How does resolution work?

Given: formulas F_1, ..., F_n

Prove: G by resolution.

1. Bring F_1, ..., F_n, ..., $\neg G$ into standard form and write the clauses which are obtained
2. Start derivation and try to obtain the empty clause from the set C of clauses.
3. In the derivation use resolution inference rule and factoring rules to derive new clauses; these new clauses are added to C.
4. If the empty clause appears, stop: Contradiction found, G is proved.
5. If no step can be made and the empty clause is not found, then H can not be proved.

Resolution Principle for FOL (cont'd)

Resolution: (Robinson, 1965)

- is an inference rule which generates resolvents from a set of clauses
- is a refutation proof procedure: empty clause is tried to be derived from a set of clauses
- is refutationally complete: a set of clauses is unsatisfiable iff the empty clause can be derived

How does resolution work?

Given: formulas F_1, ..., F_n

Prove: G by resolution.

1. Bring F_1, ..., F_n, ..., $\neg G$ into standard form and write the clauses which are obtained
2. Start derivation and try to obtain the empty clause from the set C of clauses.
3. In the derivation use resolution inference rule and factoring rules to derive new clauses; these new clauses are added to C.
4. If the empty clause appears, stop: Contradiction found, G is proved.
5. If no step can be made and the empty clause is not found, then H can not be proved.

Resolution Principle for FOL (cont'd)

Resolution: (Robinson, 1965)

- is an inference rule which generates resolvents from a set of clauses
- is a refutation proof procedure: empty clause is tried to be derived from a set of clauses
- is refutationally complete: a set of clauses is unsatisfiable iff the empty clause can be derived

How does resolution work?

Given: formulas F_1, ..., F_n

Prove: G by resolution.

1. Bring F_1, ..., F_n, ..., $\neg G$ into standard form and write the clauses which are obtained
2. Start derivation and try to obtain the empty clause from the set C of clauses.
3. In the derivation use resolution inference rule and factoring rules to derive new clauses; these new clauses are added to C.
4. If the empty clause appears, stop: Contradiction found, G is proved.
5. If no step can be made and the empty clause is not found, then H can not be proved.

Resolution Principle for FOL (cont'd)

Resolution: (Robinson, 1965)

- is an inference rule which generates resolvents from a set of clauses
- is a refutation proof procedure: empty clause is tried to be derived from a set of clauses
- is refutationally complete: a set of clauses is unsatisfiable iff the empty clause can be derived

How does resolution work?

Given: formulas F_1, ..., F_n

Prove: G by resolution.

1. Bring F_1, ..., F_n, ..., $\neg G$ into standard form and write the clauses which are obtained
2. Start derivation and try to obtain the empty clause from the set C of clauses.
3. In the derivation use resolution inference rule and factoring rules to derive new clauses; these new clauses are added to C.
4. If the empty clause appears, stop: Contradiction found, G is proved.
5. If no step can be made and the empty clause is not found, then H can not be proved.

Resolution Principle for FOL (cont'd)

Resolution: (Robinson, 1965)

- is an inference rule which generates resolvents from a set of clauses
- is a refutation proof procedure: empty clause is tried to be derived from a set of clauses
- is refutationally complete: a set of clauses is unsatisfiable iff the empty clause can be derived

How does resolution work?

Given: formulas F_1, ..., F_n

Prove: G by resolution.

1. Bring F_1, ..., F_n, ..., $\neg G$ into standard form and write the clauses which are obtained
2. Start derivation and try to obtain the empty clause from the set C of clauses.
3. In the derivation use resolution inference rule and factoring rules to derive new clauses; these new clauses are added to C.
4. If the empty clause appears, stop: Contradiction found, G is proved.
5. If no step can be made and the empty clause is not found, then H can not be proved.

Resolution Principle for FOL (cont'd)

Resolution: (Robinson, 1965)

- is an inference rule which generates resolvents from a set of clauses
- is a refutation proof procedure: empty clause is tried to be derived from a set of clauses
- is refutationally complete: a set of clauses is unsatisfiable iff the empty clause can be derived

How does resolution work?

Given: formulas F_1, ..., F_n

Prove: G by resolution.

1. Bring F_1, ..., F_n, ..., $\neg G$ into standard form and write the clauses which are obtained
2. Start derivation and try to obtain the empty clause from the set C of clauses.
3. In the derivation use resolution inference rule and factoring rules to derive new clauses; these new clauses are added to C.
4. If the empty clause appears, stop: Contradiction found, G is proved.
5. If no step can be made and the empty clause is not found, then H can not be proved.

Resolution Principle for FOL (cont'd)

Resolution: (Robinson, 1965)

- is an inference rule which generates resolvents from a set of clauses
- is a refutation proof procedure: empty clause is tried to be derived from a set of clauses
- is refutationally complete: a set of clauses is unsatisfiable iff the empty clause can be derived

How does resolution work?

Given: formulas F_1, ..., F_n

Prove: G by resolution.

1. Bring F_1, ..., F_n, ..., $\neg G$ into standard form and write the clauses which are obtained
2. Start derivation and try to obtain the empty clause from the set C of clauses.
3. In the derivation use resolution inference rule and factoring rules to derive new clauses; these new clauses are added to C.
4. If the empty clause appears, stop: Contradiction found, G is proved.
5. If no step can be made and the empty clause is not found, then H can not be proved.

Resolution Principle for FOL. Correctness & Completeness

Theorem

A set of clauses S is unsatisfiable iff there is a deduction of the empty clause from S.

Proof.

"$\Longrightarrow$" (Completeness)

...

"$\Longleftarrow$" (Correctness)

- Assume S is satisfiable and derive a contradiction.
- Since there exists a deduction from S, we have the resolvents $R_1, ... R_n$ obtained in this deduction.
- Since S is satisfiable there exists an interpretation satisfying each clause in S.
- Any resolvent of any two clauses in S is also satisfied by I, since these resolvents are logical consequences of the two clauses.
- Hence I satisfies $R_1, ... R_n$ which is impossible since one of R_i is the empty clause.

Resolution Principle for FOL. Correctness & Completeness

Theorem

A set of clauses S is unsatisfiable iff there is a deduction of the empty clause from S.

Proof.

"$\Longrightarrow$" (Completeness)

...

"$\Longleftarrow$" (Correctness)

- Assume S is satisfiable and derive a contradiction.
- Since there exists a deduction from S, we have the resolvents $R_1,\ldots R_n$ obtained in this deduction.
- Since S is satisfiable there exists an interpretation satisfying each clause in S.
- Any resolvent of any two clauses in S is also satisfied by I, since these resolvents are logical consequences of the two clauses.
- Hence I satisfies $R_1,\ldots R_n$ which is impossible since one of R_i is the empty clause.

Resolution Principle for FOL. Correctness & Completeness

Theorem

A set of clauses S is unsatisfiable iff there is a deduction of the empty clause from S.

Proof.

"$\Longrightarrow$" (Completeness)

...

"$\Longleftarrow$" (Correctness)

- Assume S is satisfiable and derive a contradiction.
- Since there exists a deduction from S, we have the resolvents R_1,...R_n obtained in this deduction.
- Since S is satisfiable there exists an interpretation satisfying each clause in S.
- Any resolvent of any two clauses in S is also satisfied by I, since these resolvents are logical consequences of the two clauses.
- Hence I satisfies R_1,...R_n which is impossible since one of R_i is the empty clause.

Resolution Principle for FOL. Correctness & Completeness

Theorem

A set of clauses S is unsatisfiable iff there is a deduction of the empty clause from S.

Proof.

"$\Longrightarrow$" (Completeness)

...

"$\Longleftarrow$" (Correctness)

- Assume S is satisfiable and derive a contradiction.
- Since there exists a deduction from S, we have the resolvents R_1,...R_n obtained in this deduction.
- Since S is satisfiable there exists an interpretation satisfying each clause in S.
- Any resolvent of any two clauses in S is also satisfied by I, since these resolvents are logical consequences of the two clauses.
- Hence I satisfies R_1,...R_n which is impossible since one of R_i is the empty clause.

Resolution Principle for FOL. Correctness & Completeness

Theorem

A set of clauses S is unsatisfiable iff there is a deduction of the empty clause from S.

Proof.

"$\Longrightarrow$" (Completeness)

...

"$\Longleftarrow$" (Correctness)

- Assume S is satisfiable and derive a contradiction.
- Since there exists a deduction from S, we have the resolvents R_1,...R_n obtained in this deduction.
- Since S is satisfiable there exists an interpretation satisfying each clause in S.
- Any resolvent of any two clauses in S is also satisfied by I, since these resolvents are logical consequences of the two clauses.
- Hence I satisfies R_1,...R_n which is impossible since one of R_i is the empty clause.

Resolution Principle for FOL. Correctness & Completeness

Theorem

A set of clauses S is unsatisfiable iff there is a deduction of the empty clause from S.

Proof.

"$\Longrightarrow$" (Completeness)

...

"$\Longleftarrow$" (Correctness)

- Assume S is satisfiable and derive a contradiction.
- Since there exists a deduction from S, we have the resolvents R_1,...R_n obtained in this deduction.
- Since S is satisfiable there exists an interpretation satisfying each clause in S.
- Any resolvent of any two clauses in S is also satisfied by I, since these resolvents are logical consequences of the two clauses.
- Hence I satisfies R_1,...R_n which is impossible since one of R_i is the empty clause.

Resolution Principle for FOL. Correctness & Completeness

Theorem

A set of clauses S is unsatisfiable iff there is a deduction of the empty clause from S.

Proof.

"$\Longrightarrow$" (Completeness)

...

"$\Longleftarrow$" (Correctness)

- Assume S is satisfiable and derive a contradiction.
- Since there exists a deduction from S, we have the resolvents R_1,...R_n obtained in this deduction.
- Since S is satisfiable there exists an interpretation satisfying each clause in S.
- Any resolvent of any two clauses in S is also satisfied by I, since these resolvents are logical consequences of the two clauses.
- Hence I satisfies R_1,...R_n which is impossible since one of R_i is the empty clause.

Resolution Principle for FOL. Correctness & Completeness

Theorem

A set of clauses S is unsatisfiable iff there is a deduction of the empty clause from S.

Proof.

"$\Longrightarrow$" (Completeness)

...

"$\Longleftarrow$" (Correctness)

- Assume S is satisfiable and derive a contradiction.
- Since there exists a deduction from S, we have the resolvents R_1,...R_n obtained in this deduction.
- Since S is satisfiable there exists an interpretation satisfying each clause in S.
- Any resolvent of any two clauses in S is also satisfied by I, since these resolvents are logical consequences of the two clauses.
- Hence I satisfies R_1,...R_n which is impossible since one of R_i is the empty clause.

Resolution Principle for FOL. Correctness & Completeness

Theorem

A set of clauses S is unsatisfiable iff there is a deduction of the empty clause from S.

Proof.

"$\Longrightarrow$" (Completeness)

...

"$\Longleftarrow$" (Correctness)

- Assume S is satisfiable and derive a contradiction.
- Since there exists a deduction from S, we have the resolvents R_1,...R_n obtained in this deduction.
- Since S is satisfiable there exists an interpretation satisfying each clause in S.
- Any resolvent of any two clauses in S is also satisfied by I, since these resolvents are logical consequences of the two clauses.
- Hence I satisfies R_1,...R_n which is impossible since one of R_i is the empty clause.

Resolution Principle for FOL. Correctness & Completeness (cont'd)

Lemma

Given two clauses C_1 and C_2, a resolvent C of C_1 and C_2 is a logical consequence of C_1 and C_2.

Proof.

Let

$$\begin{array}{ll} C_1: & L \vee C_1' \\ C_2: & \neg L \vee C_2' \end{array}$$

We have to prove that

$$L \vee C_1', \neg L \vee C_2' \quad \models \quad C_1' \vee C_2'$$

that is, for any interpretation I if $\langle L \vee C_1' \rangle_I = \langle \neg L \vee C_2' \rangle_I = \mathbb{T}$ then $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

- Case $\langle L \rangle_I = \mathbb{T}$. Then $\langle C_2' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.
- Case $\langle L \rangle_I = \mathbb{F}$. Then $\langle C_1' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

Resolution Principle for FOL. Correctness & Completeness (cont'd)

Lemma

Given two clauses C_1 and C_2, a resolvent C of C_1 and C_2 is a logical consequence of C_1 and C_2.

Proof.

Let

$$\begin{array}{ll} C_1: & L \vee C_1' \\ C_2: & \neg L \vee C_2' \end{array}$$

We have to prove that

$$L \vee C_1', \neg L \vee C_2' \models C_1' \vee C_2'$$

that is, for any interpretation I if $\langle L \vee C_1' \rangle_I = \langle \neg L \vee C_2' \rangle_I = \mathbb{T}$ then $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

- Case $\langle L \rangle_I = \mathbb{T}$. Then $\langle C_2' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.
- Case $\langle L \rangle_I = \mathbb{F}$. Then $\langle C_1' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

Resolution Principle for FOL. Correctness & Completeness (cont'd)

Lemma

Given two clauses C_1 and C_2, a resolvent C of C_1 and C_2 is a logical consequence of C_1 and C_2.

Proof.

Let

$$\begin{array}{ll} C_1: & L \vee C_1' \\ C_2: & \neg L \vee C_2' \end{array}$$

We have to prove that

$$L \vee C_1',\ \neg L \vee C_2' \models C_1' \vee C_2'$$

that is, for any interpretation I if $\langle L \vee C_1' \rangle_I = \langle \neg L \vee C_2' \rangle_I = \mathbb{T}$ then $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

- Case $\langle L \rangle_I = \mathbb{T}$. Then $\langle C_2' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.
- Case $\langle L \rangle_I = \mathbb{F}$. Then $\langle C_1' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

Resolution Principle for FOL. Correctness & Completeness (cont'd)

Lemma

Given two clauses C_1 and C_2, a resolvent C of C_1 and C_2 is a logical consequence of C_1 and C_2.

Proof.

Let

$$\begin{array}{ll} C_1 : & L \vee C_1' \\ C_2 : & \neg L \vee C_2' \end{array}$$

We have to prove that

$$L \vee C_1', \ \neg L \vee C_2' \ \models \ C_1' \vee C_2'$$

that is, for any interpretation I if $\langle L \vee C_1' \rangle_I = \langle \neg L \vee C_2' \rangle_I = \mathbb{T}$ then $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

- Case $\langle L \rangle_I = \mathbb{T}$. Then $\langle C_2' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.
- Case $\langle L \rangle_I = \mathbb{F}$. Then $\langle C_1' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

Resolution Principle for FOL. Correctness & Completeness (cont'd)

Lemma

Given two clauses C_1 and C_2, a resolvent C of C_1 and C_2 is a logical consequence of C_1 and C_2.

Proof.

Let

$$\begin{array}{ll} C_1 : & L \vee C_1' \\ C_2 : & \neg L \vee C_2' \end{array}$$

We have to prove that

$$L \vee C_1',\ \neg L \vee C_2' \quad \models \quad C_1' \vee C_2'$$

that is, for any interpretation I if $\langle L \vee C_1' \rangle_I = \langle \neg L \vee C_2' \rangle_I = \mathbb{T}$ then $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

- Case $\langle L \rangle_I = \mathbb{T}$. Then $\langle C_2' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.
- Case $\langle L \rangle_I = \mathbb{F}$. Then $\langle C_1' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

Resolution Principle for FOL. Correctness & Completeness (cont'd)

Lemma

Given two clauses C_1 and C_2, a resolvent C of C_1 and C_2 is a logical consequence of C_1 and C_2.

Proof.

Let

$$\begin{array}{ll} C_1: & L \vee C_1' \\ C_2: & \neg L \vee C_2' \end{array}$$

We have to prove that

$$L \vee C_1', \ \neg L \vee C_2' \quad \models \quad C_1' \vee C_2'$$

that is, for any interpretation I if $\langle L \vee C_1' \rangle_I = \langle \neg L \vee C_2' \rangle_I = \mathbb{T}$ then $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

- Case $\langle L \rangle_I = \mathbb{T}$. Then $\langle C_2' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.
- Case $\langle L \rangle_I = \mathbb{F}$. Then $\langle C_1' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

Resolution Principle for FOL. Correctness & Completeness (cont'd)

Lemma

Given two clauses C_1 and C_2, a resolvent C of C_1 and C_2 is a logical consequence of C_1 and C_2.

Proof.

Let

$$\begin{array}{ll} C_1: & L \vee C_1' \\ C_2: & \neg L \vee C_2' \end{array}$$

We have to prove that

$$L \vee C_1', \ \neg L \vee C_2' \quad \models \quad C_1' \vee C_2'$$

that is, for any interpretation I if $\langle L \vee C_1' \rangle_I = \langle \neg L \vee C_2' \rangle_I = \mathbb{T}$ then $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

- Case $\langle L \rangle_I = \mathbb{T}$. Then $\langle C_2' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.
- Case $\langle L \rangle_I = \mathbb{F}$. Then $\langle C_1' \rangle_I = \mathbb{T}$. Hence $\langle C_1' \vee C_2' \rangle_I = \mathbb{T}$.

Resolution Principle for FOL. Examples

Example 0: Let

$$C_1 : \quad P[x] \vee Q[x]$$
$$C_2 : \quad \neg P[a] \vee R[x]$$

Apply resolution.

Example 1: Prove by resolution the following

$$\forall_x F[x] \vee \forall_x H[x] \quad \not\equiv \quad \forall_x (F[x] \vee H[x])$$

Example 2: Prove by resolution that G is a logical consequence of F_1 and F_2 where

$$F_1 : \quad \forall_x (C[x] \Rightarrow (W[x] \wedge R[x]))$$
$$F_2 : \quad \exists_x (C[x] \wedge O[x])$$
$$G : \quad \exists_x (O[x] \wedge R[x])$$

Resolution Principle for FOL. Examples

Example 0: Let

$$\begin{array}{ll} C_1: & P[x] \vee Q[x] \\ C_2: & \neg P[a] \vee R[x] \end{array}$$

Apply resolution.

Example 1: Prove by resolution the following

$$\underset{x}{\forall} F[x] \vee \underset{x}{\forall} H[x] \quad \not\equiv \quad \underset{x}{\forall} (F[x] \vee H[x])$$

Example 2: Prove by resolution that G is a logical consequence of F_1 and F_2 where

$$\begin{array}{ll} F_1: & \underset{x}{\forall} (C[x] \Rightarrow (W[x] \wedge R[x])) \\ F_2: & \underset{x}{\exists} (C[x] \wedge O[x]) \\ G: & \underset{x}{\exists} (O[x] \wedge R[x]) \end{array}$$

Resolution Principle for FOL. Examples

Example 0: Let

$$\begin{array}{ll} C_1: & P[x] \vee Q[x] \\ C_2: & \neg P[a] \vee R[x] \end{array}$$

Apply resolution.

Example 1: Prove by resolution the following

$$\underset{x}{\forall} F[x] \;\vee\; \underset{x}{\forall} H[x] \quad \not\equiv \quad \underset{x}{\forall}(F[x] \;\vee\; H[x])$$

Example 2: Prove by resolution that G is a logical consequence of F_1 and F_2 where

$$\begin{array}{ll} F_1: & \underset{x}{\forall}(C[x] \;\Rightarrow\; (W[x] \;\wedge\; R[x])) \\ F_2: & \underset{x}{\exists}(C[x] \wedge O[x]) \\ G: & \underset{x}{\exists}(O[x] \wedge R[x]) \end{array}$$

Resolution Principle for FOL. Examples (cont'd)

Example 3: Prove by resolution that G is a logical consequence of F_1 and F_2 where

$$
\begin{aligned}
F_1 :&\quad \underset{x}{\exists}\left(P[x] \;\wedge\; \underset{y}{\forall}(D[y] \;\Rightarrow\; L[x,y])\right)\\
F_2 :&\quad \underset{x}{\forall}\left(P[x] \;\Rightarrow\; \underset{y}{\forall}(Q[y] \;\Rightarrow\; \neg L[x,y])\right)\\
G :&\quad \underset{x}{\forall}(D[x] \;\Rightarrow\; \neg Q[x])
\end{aligned}
$$

Example 4: Prove by resolution that G is a logical consequence of F where

$$
\begin{aligned}
F :&\quad \underset{x\,y}{\forall\exists}(S[x,y] \;\wedge\; M[y]) \;\;\Rightarrow\;\; \underset{y}{\exists}(I[y] \;\wedge\; E[x,y])\\
G :&\quad \neg\underset{x}{\exists} I[x] \;\;\Rightarrow\;\; \underset{x,y}{\forall}\,(S[x,y] \Rightarrow \neg M[y])
\end{aligned}
$$

Resolution Principle for FOL. Examples (cont'd)

Example 3: Prove by resolution that G is a logical consequence of F_1 and F_2 where

$$F_1: \quad \underset{x}{\exists}\left(P[x] \wedge \underset{y}{\forall}(D[y] \Rightarrow L[x,y])\right)$$
$$F_2: \quad \underset{x}{\forall}\left(P[x] \Rightarrow \underset{y}{\forall}(Q[y] \Rightarrow \neg L[x,y])\right)$$
$$G: \quad \underset{x}{\forall}(D[x] \Rightarrow \neg Q[x])$$

Example 4: Prove by resolution that G is a logical consequence of F where

$$F: \quad \underset{x}{\forall}\underset{y}{\exists}(S[x,y] \wedge M[y]) \Rightarrow \underset{y}{\exists}(I[y] \wedge E[x,y])$$
$$G: \quad \neg\underset{x}{\exists}I[x] \Rightarrow \underset{x,y}{\forall}(S[x,y] \Rightarrow \neg M[y])$$

Resolution Principle for FOL. Examples (cont'd)

Example 5: Prove by resolution that G is a logical consequence of F_1, F_2, and F_3 where

$$
\begin{aligned}
F_1 : &\quad \underset{x}{\forall}(Q[x] \Rightarrow \neg P[x]) \\
F_2 : &\quad \underset{x}{\forall}\Big((R[x] \wedge \neg Q[x]) \Rightarrow \underset{y}{\exists}(T[x,y] \wedge S[y])\Big) \\
F_3 : &\quad \underset{x}{\exists}\Big(P[x] \wedge \underset{y}{\forall}(T[x,y] \Rightarrow P[y]) \wedge R[x]\Big) \\
G : &\quad \underset{x}{\exists}(S[x] \wedge P[x])
\end{aligned}
$$