

First-Order Logic Resolution

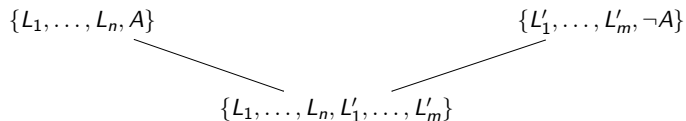
Resolution for first-order logic

Gilmore's algorithm is correct and complete,
but useless in practice.

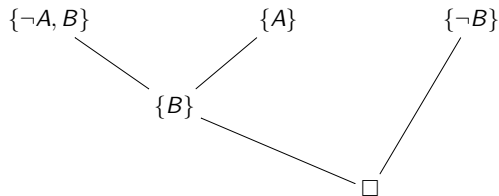
We upgrade resolution to make it work for predicate logic.

Recall: resolution in propositional logic

Resolution step:



Resolution graph:



A set of clauses is **unsatisfiable** iff the **empty clause** can be derived.

Adapting Gilmore's Algorithm

Gilmore's Algorithm:

Let F be a closed formula in Skolem form
and let F_1, F_2, F_3, \dots be an enumeration of $E(F)$.

$n := 0$;

repeat $n := n + 1$

until $(F_1 \wedge F_2 \wedge \dots \wedge F_n)$ is unsatisfiable;

– *this can be checked with any calculus for propositional logic*

return “unsatisfiable”

“any calculus” \rightsquigarrow use **resolution** for the unsatisfiability test

Terminology

Literal/clause/CNF is defined as for propositional logic but with the atomic formulas of predicate logic.

A **ground term/formula/etc** is a term/formula/etc that does not contain any variables.

An **instance** of a term/formula/etc is the result of applying a substitution to a term/formula/etc.

A **ground instance** is an instance that does not contain any variables.

Clause Herbrand expansion

Let $F = \forall y_1 \dots \forall y_n F^*$ be a closed formula in Skolem form with F^* in CNF, and let C_1, \dots, C_m be the clauses of F^* .

The **clause Herbrand expansion** of F is the set of ground clauses

$$CE(F) = \bigcup_{i=1}^m \{C_i[t_1/y_1] \dots [t_n/y_n] \mid t_1, \dots, t_n \in T(F)\}$$

Lemma

$CE(F)$ is unsatisfiable iff $E(F)$ is unsatisfiable.

Proof Informally speaking, " $CE(F) \equiv E(F)$ ".

Ground resolution algorithm

Let F be a closed formula in Skolem form with F^* in CNF.

Let C_1, C_2, C_3, \dots be an enumeration of $CE(F)$.

```
 $n := 0;$   
 $S := \emptyset;$   
repeat  
     $n := n + 1;$   
     $S := S \cup \{C_n\};$   
until  $S \vdash_{Res} \square$   
  
return “unsatisfiable”
```

Note: The search for \square can be performed incrementally every time S is extended.

Example

$$F^* = \{\{\neg P(x), \neg P(f(a)), Q(y)\}, \{P(y)\}, \{\neg P(g(b, x)), \neg Q(b)\}\}$$

Ground resolution theorem

The correctness of the ground resolution algorithm can be rephrased as follows:

Theorem

A formula $F = \forall y_1 \dots \forall y_n F^$ with F^* in CNF is unsatisfiable iff there is a sequence of ground clauses $C_1, \dots, C_m = \square$ such that for every $i = 1, \dots, m$*

- ▶ *either C_i is a ground instance of a clause $C \in F^*$,
i.e. $C_i = C[t_1/y_1] \dots [t_n/y_n]$ where $t_1, \dots, t_n \in T(F)$,*
- ▶ *or C_i is a resolvent of two clauses C_a, C_b with $a < i$ and $b < i$*

Where do the ground substitutions come from?

Better:

- ▶ allow substitutions with variables
- ▶ only instantiate clauses enough to allow one (new kind of) resolution step

Example

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

Substitutions as functions

Substitutions are functions from variables to terms:

$[t/x]$ maps x to t (and all other variables to themselves)

Functions can be composed.

Composition of substitutions is denoted by juxtaposition:

$[t_1/x][t_2/y]$ first substitutes t_1 for x and then substitutes t_2 for y .

Example

$$(P(x,y))[f(y)/x][b/y] = (P(f(y),y))[b/y] = P(f(b),b)$$

Similarly we can compose arbitrary substitutions σ_1 and σ_2 :

$\sigma_1\sigma_2$ is the substitution that applies σ_1 first and then σ_2 .

Substitutions are functions. Therefore

$$\sigma_1 = \sigma_2 \quad \text{iff} \quad \text{for all variables } x, x\sigma_1 = x\sigma_2$$

Substitutions as functions

Definition

The **domain** of a substitution: $dom(\sigma) = \{x \mid x\sigma \neq x\}$

Example

$$dom([a/x][b/y]) = \{x, y\}$$

Substitutions are defined to have **finite domain**.

Therefore every substitution can be written as a **simultaneous substitution** $[t_1/x_1, \dots, t_n/x_n]$.

Unifier and most general unifier

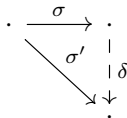
Let $L = \{L_1, \dots, L_k\}$ be a set of literals.

A substitution σ is a **unifier** of L if

$$L_1\sigma = L_2\sigma = \dots = L_k\sigma$$

i.e. if $|L\sigma| = 1$, where $L\sigma = \{L_1\sigma, \dots, L_k\sigma\}$.

A unifier σ of L is a **most general unifier (mgu)** of L if for every unifier σ' of L there is a substitution δ such that $\sigma' = \sigma\delta$.



Exercise

Unifiable?		Yes	No
$P(f(x))$	$P(g(y))$		x
$P(x)$	$P(f(y))$	x	
$P(x)$	$P(f(x))$		x
$P(x, f(y))$	$P(f(u), f(z))$	x	
$P(x, f(x))$	$P(f(y), y)$		x
$P(x, g(x), g^2(x))$	$P(f(z), w, g(w))$	x	
$P(x, f(y))$	$P(g(y), f(a))$	$P(g(a), z)$	x

Unification algorithm

Input: a set $L \neq \emptyset$ of literals

$\sigma := []$ (the empty substitution)

while $|L\sigma| > 1$ **do**

Find the first position at which two literals $L_1, L_2 \in L\sigma$ differ

if none of the two characters at that position is a variable

then return “non-unifiable”

else let x be the variable and t the term starting at that position

if x occurs in t

then return “non-unifiable”

else $\sigma := \sigma [t/x]$

return σ

Example

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

Correctness of the unification algorithm

Lemma

The unification algorithm terminates.

Proof Every iteration of the **while**-loop (possibly except the last) replaces a variable x by a term t not containing x , and so the number of variables occurring in $L\sigma$ decreases by one.

Lemma

If L is non-unifiable then the algorithm returns “non-unifiable”.

Proof If L is non-unifiable then the algorithm can never exit the loop normally.

Correctness/completeness of the unification algorithm

Lemma

*If L is unifiable then the algorithm returns the mgu of L
(and so in particular every unifiable set L has an mgu).*

Proof Assume L is unifiable and let n be the number of iterations of the loop on input L.

Let $\sigma_0 = []$, for $1 \leq i \leq n$ let σ_i be the value of σ after the i -th iteration of the loop.

We prove for every $0 \leq i \leq n$:

- (a) If $1 \leq i$, the i -th iteration does not return “non-unifiable”.
- (b) For every unifier σ' of L there is a substitution δ_i such that $\sigma' = \sigma_i \delta_i$.

By (a) the algorithm exits the loop normally after n iterations.

By (b) it returns a most general unifier.

Correctness/completeness of the unification algorithm

Proof of (a) and (b) by induction on i :

Basis ($i = 0$): For (a) there is nothing to prove.

For (b) take $\delta_0 = \sigma'$.

Step ($i \Rightarrow i + 1$)

For (a), since $|L\sigma_i| > 1$ and $L\sigma_i$ unifiable, x and t exist and x does not occur in t , and so “non-unifiable” is not returned.

For (b): Let σ' be a unifier of L . IH: $\sigma' = \sigma_i \delta_i$ for some δ_i .

δ_i must be of the form $[t_1/x_1, \dots, t_k/x_k, u/x]$ where x_1, \dots, x_k, x are distinct. Define $\delta_{i+1} = [t_1/x_1, \dots, t_k/x_k]$.

Note $u = x\delta_i = t\delta_i = t\delta_{i+1}$ ($\sigma_i \delta_i$ is unifier (IH), x not in t)

$$\begin{aligned} & \sigma_{i+1} \delta_{i+1} \\ = & \sigma_i [t/x] \delta_{i+1} && \text{(algorithm extends } \sigma_i \text{ with } [t/x]) \\ = & \sigma_i [t_1/x_1, \dots, t_k/x_k, t\delta_{i+1}/x] \\ = & \sigma_i [t_1/x_1, \dots, t_k/x_k, u/x] && \text{(Note } u = t\delta_{i+1}) \\ = & \sigma_i \delta_i \\ = & \sigma' && \text{(IH)} \end{aligned}$$

The standard view of unification

A unification problem is a pair of terms $s =? t$
(or a set of pairs $\{s_1 =? t_1, \dots, s_n =? t_n\}$)

A unifier is a substitution σ such that $s\sigma = t\sigma$
(or $s_1\sigma = t_1\sigma, \dots, s_n\sigma = t_n\sigma$)

Renaming

Definition

A substitution ρ is a **renaming** if for every variable x , $x\rho$ is a variable and ρ is injective on $dom(\rho)$.

Resolvents for first-order logic

A clause R is a **resolvent** of two clauses C_1 and C_2 if the following holds:

- ▶ There is a renaming ρ such that
no variable occurs in both C_1 and $C_2 \rho$ and
 ρ is injective on the set of variables in C_2
- ▶ There are literals $L_1, \dots, L_m \in C_1$ ($m \geq 1$)
and literals $L'_1, \dots, L'_n \in C_2 \rho$ ($n \geq 1$) such that

$$L = \{\overline{L_1}, \dots, \overline{L_m}, L'_1, \dots, L'_n\}$$

is unifiable. Let σ be an mgu of L .

- ▶ $R = ((C_1 - \{L_1, \dots, L_m\}) \cup (C_2 \rho - \{L'_1, \dots, L'_n\}))\sigma$

Example

$C_1 = \{ P(x), Q(x), P(g(y)) \}$ and $C_2 = \{ \neg P(x), R(f(x), a) \}$

Exercise

How many resolvents are there?

C_1	C_2	Resolvents
$\{P(x), Q(x, y)\}$	$\{\neg P(f(x))\}$	
$\{Q(g(x)), R(f(x))\}$	$\{\neg Q(f(x))\}$	
$\{P(x), P(f(x))\}$	$\{\neg P(y), Q(y, z)\}$	

Why renaming?

Example

$$\forall x(P(x) \wedge \neg P(f(x)))$$

Resolution for first-order logic

As for propositional logic, $F \vdash_{Res} C$ means that clause C can be derived from a set of clauses F by a sequence of resolution steps, i.e. that there is a sequence of clauses $C_1, \dots, C_m = C$ such that for every C_i

- ▶ either $C_i \in F$
- ▶ or C_i is the resolvent of C_a and C_b where $a, b < i$.

Questions:

Correctness Does $F \vdash_{Res} \square$ imply that F is unsatisfiable?

Completeness Does unsatisfiability of F imply $F \vdash_{Res} \square$?

Exercise

Derive \square from the following clauses:

1. $\{\neg P(x), Q(x), R(x, f(x))\}$
2. $\{\neg P(x), Q(x), S(f(x))\}$
3. $\{T(a)\}$
4. $\{P(a)\}$
5. $\{\neg R(a, z), T(z)\}$
6. $\{\neg T(x), \neg Q(x)\}$
7. $\{\neg T(y), \neg S(y)\}$

Correctness of Resolution for First-Order Logic

Definition

The **universal closure** of a formula H with free variables x_1, \dots, x_n :

$$\forall H = \forall x_1 \forall x_2 \dots \forall x_n H$$

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.

If $F^* \vdash_{Res} \square$ then F is unsatisfiable.

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.

If $F^* \vdash_{Res} \square$ then F is unsatisfiable.

Proof Let C_1, \dots, C_m be the sequence of clauses leading to \square .

By induction on i : if $\forall F^* \models \forall C_i$. Trivial if $C_i \in F^*$.

Let C_i be a resolvent of C_a and C_b ($a, b < i$). We prove

$$\forall C_a, \forall C_b \models \forall C_i \quad (*)$$

Thus $\forall F^* \models \forall C_i$ because $\forall F^* \models \forall C_a$ and $\forall F^* \models \forall C_b$ by IH.

Proof of (*): Assume $\mathcal{A}(\forall C_a) = \mathcal{A}(\forall C_b) = 1$ (**)

$$\begin{aligned} C_i &= ((C_a - \{L_1, \dots\}) \cup (C_b \rho - \{L'_1, \dots\}))\sigma \\ &= (C_a \sigma - \{L\}) \cup (C_b \rho \sigma - \{\bar{L}\}) \end{aligned}$$

Indirect proof of $\mathcal{A}(\forall C_i) = 1$. Assume $\mathcal{A}(\forall C_i) = 0$.

$\Rightarrow \mathcal{A}'(C_i) = 0$ where $\mathcal{A}' = \mathcal{A}[u_1/x_1, \dots]$ for some $u_i \in U_{\mathcal{A}}$

$\Rightarrow \mathcal{A}'(C_a \sigma - \{L\}) = \mathcal{A}'(C_b \rho \sigma - \{\bar{L}\}) = 0$

$\Rightarrow \mathcal{A}'(L) = \mathcal{A}'(\bar{L}) = 1$ becs. $\mathcal{A}(C_a \sigma) = \mathcal{A}(C_b \rho \sigma) = 1$ becs. (**)

Contradiction

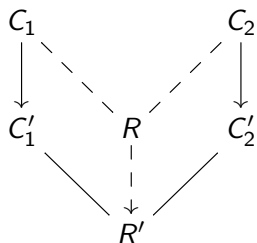
Completeness: The idea

Simulate ground resolution because that is complete

Lift the resolution proof from the ground resolution proof

Lifting Lemma

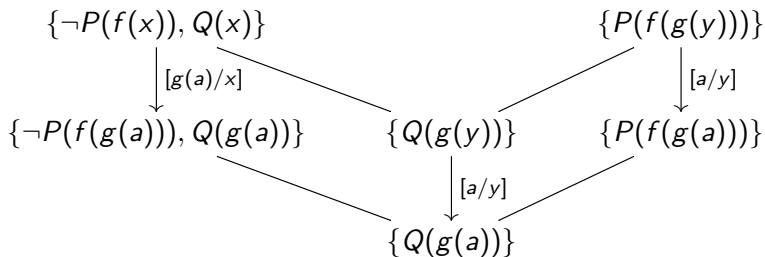
Let C_1, C_2 be two clauses and
let C'_1, C'_2 be two ground instances
with (propositional) resolvent R' .
Then there is a resolvent R of C_1, C_2
such that R' is a ground instance of R .



\rightarrow : Substitution

— : Resolution

Lifting Lemma: example



Proof of Lifting Lemma.

(1) C'_1, C'_2 are ground instances of C_1, C_2

(2) R' is propositional resolvent of C'_1 and C'_2

We prove that R' is an instance of a resolvent of C_1 and C_2

(3) Let ρ be a renaming s.t. C_1 and $C_2\rho$ have no common variables

(1) $\Rightarrow C'_2$ is a ground instance of $C_2\rho$. Thus there are σ_1, σ_2 s.t.

$C'_1 = C_1\sigma_1$ and $C'_2 = C_2\rho\sigma_2$ and $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$

$\Rightarrow C'_1 = C_1\sigma$ and $C'_2 = C_2\rho\sigma$ where $\sigma = \sigma_1 \cup \sigma_2$

(2) $\Rightarrow R' = (C'_1 - \{L\}) \cup (C'_2 - \{\bar{L}\})$ where $L \in C'_1$ and $\bar{L} \in C'_2$

\Rightarrow there are $\{L_1, \dots\} \subseteq C_1$ and $\{L'_1, \dots\} \subseteq C_2\rho$

s.t. σ is a unifier of $\{\bar{L}_1, \dots, L'_1, \dots\} =: M$.

Let σ_0 be an mgu of M and let $\sigma = \sigma_0\delta$ for some δ

\Rightarrow A resolvent of C_1 and C_2 :

$$R := ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L'_1, \dots\}))\sigma_0$$

$$R\delta = ((C_1 - \{L_1, \dots\}) \cup (C_2\rho - \{L'_1, \dots\}))\sigma$$

$$= (C_1\sigma - \{L\}) \cup (C_2\rho\sigma - \{\bar{L}\})$$

$$= (C'_1 - \{L\}) \cup (C'_2 - \{\bar{L}\})$$

$$= R'$$

Completeness of Resolution for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^* in CNF.
If F is unsatisfiable then $F^* \vdash_{Res} \square$.

Proof If F is unsatisfiable, there is a ground resolution proof $C'_1, \dots, C'_n = \square$. We transform this step by step into a resolution proof $C_1, \dots, C_n = \square$ such that C'_i is a ground instance of C_i .

If C'_i is a ground instance of some clause $C \in F^*$:

Set $C_i = C$

If C'_i is a resolvent of C'_a, C'_b ($a, b < i$):

C'_a, C'_b have been transformed already into C_a, C_b s.t. C'_a, C'_b are ground instances of C_a, C_b . By the Lifting Lemma there is a resolvent R of C_a, C_b s.t. C'_i is a ground instance of R .

Set $C_i = R$.

Resolution Theorem for First-Order Logic

Theorem

Let F be a closed formula in Skolem form with matrix F^ in CNF.*

Then F is unsatisfiable iff $F^ \vdash_{Res} \square$.*

A resolution algorithm

Input: A closed formula F in Skolem form with matrix S in CNF,
i.e. S is a finite set of clauses

```
while  $\square \notin S$  and  
    there are clauses  $C_a, C_b \in S$  and resolvent  $R$  of  $C_a$  and  $C_b$   
    such that  $R \notin S$  (modulo renaming)  
do  $S := S \cup \{R\}$ 
```

The selection of resolvents must be *fair*:
every resolvent is added eventually

Three possible behaviours:

- ▶ The algorithm terminates and $\square \in S$
 $\Rightarrow F$ is unsatisfiable
- ▶ The algorithm terminates and $\square \notin S$
 $\Rightarrow F$ is satisfiable
- ▶ The algorithm does not terminate
($\Rightarrow F$ is satisfiable)

Refinements of resolution

Problems of resolution:

- ▶ Branching degree of the search space too large
- ▶ Too many dead ends
- ▶ Combinatorial explosion of the search space

Solution:

Strategies and **heuristics**: forbid certain resolution steps, which narrows the search space.

But: Completeness must be preserved!