Programming Languages

Logical resolution

Brief introduction to Prolog

Resolution for propositional logic

Resolution for first-order logic

Introduction to Prolog

Example — genealogy of the Greek mythological pantheon

```
father(cronos, zeus).
father(zeus, athena).
father(zeus, hephaestus).
father(zeus, ares).

grandfather(X, Y) :- father(X, Z), father(Z, Y).
```

```
?- father(zeus, athena).
>> true.
?- father(zeus, cronos).
>> false.
?- grandfather(X, athena).
>> X = cronos.
?- grandfather(X, zeus).
>> false.
```

```
?- grandfather(cronos, X).
>> X = athena ;
>> X = hephaestus ;
>> X = ares.
?- grandfather(X, Y).
>> X = cronos, Y = athena ;
>> X = cronos, Y = hephaestus ;
>> X = cronos, Y = ares.
```

Introduction to Prolog

Prolog operates with **first-order terms**:

```
X   Y   succ(succ(zero))   bin(I, R, D)   ...
```

Atomic formulas are of the form $\mathtt{pred}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$:

```
father(zeus, athena)     sum(zero, X, X)
```

Introduction to Prolog

A program is a set of **rules**. Each rule is of the form:

$$\sigma \quad \texttt{:-} \quad \tau_1\texttt{,}\ \ldots\texttt{,}\ \tau_n\texttt{.}$$

E.g.: `grandfather(X, Y)  :-  father(X, Z), father(Z, Y).`

where $\sigma, \tau_1, \ldots, \tau_n$ are atomic formulas.

Rules with $n = 0$ are called **facts** and are written as:

$\sigma.$ E.g.: `father(zeus, ares).`

Rules have the following logical interpretation:

$$\forall \mathtt{X}_1 \ldots \forall \mathtt{X}_k.((\tau_1 \wedge \ldots \wedge \tau_n) \Rightarrow \sigma)$$

where $\mathtt{X}_1, \ldots, \mathtt{X}_k$ are all the free variables in the formulas.

E.g.: $\forall \mathtt{X}. \forall \mathtt{Y}. \forall \mathtt{Z}. ((\mathtt{father(X,\ Z)} \wedge \mathtt{father(Z,\ Y)}) \Rightarrow \mathtt{grandfather(X,\ Y)})$

Introduction to Prolog

A **query** is of the form:

$$?\text{-}\ \sigma_1,\ \ldots,\ \sigma_n$$

E.g.: `?- grandfather(X, ares).`

Queries have the following logical interpretation:

$$\exists \mathtt{X}_1 \ldots \exists \mathtt{X}_k.(\sigma_1 \land \ldots \land \sigma_n)$$

where $\mathtt{X}_1, \ldots, \mathtt{X}_k$ are all the free variables in the formulas.

The Prolog environment tries to prove the formula τ of the query.
Actually, it tries to *refute* $\neg\tau$, i.e., to prove $\neg\tau \Rightarrow \bot$
The search for a refutation is based on the **resolution method**.

Brief introduction to Prolog

Resolution for propositional logic

Resolution for first-order logic

Resolution for propositional logic

Input: a formula σ of propositional logic.
Output: a boolean indicating whether σ is valid.

Resolution method

1. Write $\neg\sigma$ as a set $\mathcal{C}$ of **clauses**.
   (Convert to *clausal form*).
2. Search for a **refutation** of $\mathcal{C}$.
   A refutation of $\mathcal{C}$ is a derivation of $\mathcal{C} \vdash \bot$.

If a refutation of $\mathcal{C}$ is found:

Then $\neg\sigma \vdash \bot$ holds. That is, $\neg\sigma$ is unsatisfiable/a contradiction.

Hence $\vdash \sigma$ holds. That is, σ is valid/a tautology.

If no refutation of $\mathcal{C}$ is found:

Then $\neg\sigma \vdash \bot$ does not hold. That is, σ is satisfiable.

Hence $\vdash \sigma$ does not hold. That is, σ is not valid.

Conversion to clausal form

A formula is converted to clausal form by applying the following rules.
All rules transform the formula into an equivalent one.
Step 1. Eliminate the "$\Rightarrow$" connective:

$$\sigma \Rightarrow \tau \quad \longrightarrow \quad \neg\sigma \lor \tau$$

The resulting formula only uses the connectives $\{\neg, \lor, \land\}$.
Step 2. Push the "$\neg$" connective inward:

$$\begin{array}{lcl} \neg(\sigma \land \tau) & \longrightarrow & \neg\sigma \lor \neg\tau \\ \neg(\sigma \lor \tau) & \longrightarrow & \neg\sigma \land \neg\tau \\ \neg\neg\sigma & \longrightarrow & \sigma \end{array}$$

The resulting formula is in **negation normal form** (NNF):

$$\sigma_{\text{nnf}} ::= \mathbf{P} \mid \neg\mathbf{P} \mid \sigma_{\text{nnf}} \land \sigma_{\text{nnf}} \mid \sigma_{\text{nnf}} \lor \sigma_{\text{nnf}}$$

Conversion to clausal form

Step 3. Distribute $\vee$ over $\wedge$:

$$\begin{array}{lcl} \sigma \vee (\tau \wedge \rho) & \rightarrow & (\sigma \vee \tau) \wedge (\sigma \vee \rho) \\ (\sigma \wedge \tau) \vee \rho & \rightarrow & (\sigma \vee \rho) \wedge (\tau \vee \rho) \end{array}$$

The resulting formula is in **conjunctive normal form** (CNF). A formula in CNF is a conjunction of disjunctions of literals (assuming $\wedge$ and $\vee$ are freely associative):

Formulas in CNF	σ_{cnf}	$::=$	$(\kappa_1 \wedge \kappa_2 \wedge \ldots \wedge \kappa_n)$
Clauses	κ	$::=$	$(\ell_1 \vee \ell_2 \vee \ldots \vee \ell_m)$
Literals	ℓ	$::=$	$\mathbf{P} \mid \neg\mathbf{P}$

Conversion to clausal form

Finally, using the fact that disjunction ($\vee$) is:

associative	$\sigma \vee (\tau \vee \rho) \iff (\sigma \vee \tau) \vee \rho$
commutative	$\sigma \vee \tau \iff \tau \vee \sigma$
idempotent	$\sigma \vee \sigma \iff \sigma$

we denote a clause (disjunction of literals) as a set:

$$(\ell_1 \vee \ell_2 \vee \ldots \vee \ell_n) \quad \text{is denoted} \quad \{\ell_1, \ell_2, \ldots, \ell_n\}$$

Similarly, using the fact that
conjunction ($\wedge$) is associative, commutative, and idempotent
we denote a conjunction of clauses as a set:

$$(\kappa_1 \wedge \kappa_2 \wedge \ldots \wedge \kappa_n) \quad \text{is denoted} \quad \{\kappa_1, \kappa_2, \ldots, \kappa_n\}$$

Conversion to clausal form

Summary — conversion to clausal form

1. Rewrite $\Rightarrow$ using $\neg$ and $\vee$.
2. Convert to negation normal form by pushing $\neg$ inward.
3. Convert to conjunctive normal form by distributing $\vee$ over $\wedge$.

Conversion to clausal form

Example — conversion to clausal form

We want to check if $\sigma \equiv (((\mathbf{P} \Rightarrow (\mathbf{Q} \wedge \mathbf{R})) \wedge \mathbf{P}) \Rightarrow \mathbf{Q})$ is valid.
First we negate it: $\neg\sigma \equiv \neg(((\mathbf{P} \Rightarrow (\mathbf{Q} \wedge \mathbf{R})) \wedge \mathbf{P}) \Rightarrow \mathbf{Q})$.
We convert $\neg\sigma$ to clausal form:

$$
\begin{array}{ll}
 & \neg(((\mathbf{P} \Rightarrow (\mathbf{Q} \wedge \mathbf{R})) \wedge \mathbf{P}) \Rightarrow \mathbf{Q}) \\
\longrightarrow & \neg(\neg((\neg\mathbf{P} \vee (\mathbf{Q} \wedge \mathbf{R})) \wedge \mathbf{P}) \vee \mathbf{Q}) \\
\longrightarrow & (\neg\neg((\neg\mathbf{P} \vee (\mathbf{Q} \wedge \mathbf{R})) \wedge \mathbf{P}) \wedge \neg\mathbf{Q}) \\
\longrightarrow & (((\neg\mathbf{P} \vee (\mathbf{Q} \wedge \mathbf{R})) \wedge \mathbf{P}) \wedge \neg\mathbf{Q}) \\
\longrightarrow & (((\neg\mathbf{P} \vee \mathbf{Q}) \wedge (\neg\mathbf{P} \vee \mathbf{R})) \wedge \mathbf{P}) \wedge \neg\mathbf{Q}) \\
\longrightarrow & (\neg\mathbf{P} \vee \mathbf{Q}) \wedge (\neg\mathbf{P} \vee \mathbf{R}) \wedge \mathbf{P} \wedge \neg\mathbf{Q}
\end{array}
$$

The clausal form is:

$$\mathcal{C} = \{\{\neg\mathbf{P}, \mathbf{Q}\}, \{\neg\mathbf{P}, \mathbf{R}\}, \{\mathbf{P}\}, \{\neg\mathbf{Q}\}\}$$

Refutation

Once a set of clauses $\mathcal{C} = \{\kappa_1, \ldots, \kappa_n\}$ is obtained,
we search for a **refutation**, i.e., a proof of $\mathcal{C} \vdash \bot$.

The refutation method is based on the following deduction rule:

Resolution rule

$$\frac{\mathbf{P} \vee \ell_1 \vee \ldots \vee \ell_n \qquad \neg\mathbf{P} \vee \ell'_1 \vee \ldots \vee \ell'_m}{\ell_1 \vee \ldots \vee \ell_n \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Written in clause notation:

$$\frac{\{\mathbf{P}, \ell_1, \ldots, \ell_n\} \qquad \{\neg\mathbf{P}, \ell'_1, \ldots, \ell'_m\}}{\{\ell_1, \ldots, \ell_n, \ell'_1, \ldots, \ell'_m\}}$$

The conclusion is called the **resolvent** of the premises.

Refutation

Input: a set of clauses $\mathcal{C}_0 = \{\kappa_1, \ldots, \kappa_n\}$.
Output: `SAT`/`UNSAT` indicating whether $\mathcal{C}_0$ is unsatisfiable ($\mathcal{C}_0 \vdash \bot$).

Refutation algorithm

Let $\mathcal{C} := \mathcal{C}_0$. Repeat as long as possible:

1. If $\{\,\} \in \mathcal{C}$, return `UNSAT`.
2. Choose two clauses $\kappa, \kappa' \in \mathcal{C}$, such that:

   $\kappa = \{\mathbf{P}, \ell_1, \ldots, \ell_n\}$

   $\kappa' = \{\neg\mathbf{P}, \ell'_1, \ldots, \ell'_m\}$

   The resolvent $\rho = \{\ell_1, \ldots, \ell_n, \ell'_1, \ldots, \ell'_m\}$ is not in $\mathcal{C}$.

   If not possible, return `SAT`.
3. Set $\mathcal{C} := \mathcal{C} \cup \{\rho\}$ and go back to step 1.

Refutation

Example — resolution method

We want to prove $\sigma \equiv (((\mathbf{P} \Rightarrow (\mathbf{Q} \wedge \mathbf{R})) \wedge \mathbf{P}) \Rightarrow \mathbf{Q})$.
Equivalently, we show that $\neg\sigma \vdash \bot$.
The clausal form of $\neg\sigma$ was:

$$\mathcal{C} = \{\underbrace{\{\neg\mathbf{P}, \mathbf{Q}\}}_{\boxed{1}}, \underbrace{\{\neg\mathbf{P}, \mathbf{R}\}}_{\boxed{2}}, \underbrace{\{\mathbf{P}\}}_{\boxed{3}}, \underbrace{\{\neg\mathbf{Q}\}}_{\boxed{4}}\}$$

- From $\boxed{1}$ and $\boxed{3}$ we obtain the resolvent $\boxed{5} = \{\mathbf{Q}\}$.
- From $\boxed{4}$ and $\boxed{5}$ we obtain the resolvent $\{\,\}$.
- Hence $\mathcal{C} \vdash \bot$.
  Hence $\neg\sigma \vdash \bot$.
  Hence $\vdash \sigma$.

Correctness of the propositional resolution method

Theorem (correctness of conversion to clausal form)

Given a formula σ:

1. Conversion to clausal form terminates.
2. The resulting clause set $\mathcal{C}$ is equivalent to σ.
   That is, $\vdash \sigma \iff \mathcal{C}$.

Correctness of the propositional resolution method

Theorem (correctness of the refutation algorithm)

Given a set of clauses $\mathcal{C}_0$:

1. The refutation algorithm terminates.
2. The algorithm returns `UNSAT` if and only if $\mathcal{C}_0 \vdash \bot$.

Ideas for the proof:

1. If $\mathcal{C}_0$ contains n distinct literals, 2^n possible clauses can be formed. Each step adds a clause. Hence the algorithm cannot take more than 2^n steps.

2.($\Rightarrow$). The algorithm preserves the invariant that for every clause $\kappa \in \mathcal{C}$ we have $\mathcal{C}_0 \vdash \kappa$. The key observation is that if $\kappa, \kappa' \in \mathcal{C}$ and ρ is the resolvent, then $\kappa, \kappa' \vdash \rho$.

2.($\Leftarrow$). More difficult. It can be proved by induction on the number of propositional variables appearing in $\mathcal{C}_0$.

See *Handbook of Proof Theory*. Samuel R. Buss (editor). Elsevier, 1998. Section 2.6.

Brief introduction to Prolog

Resolution for propositional logic

Resolution for first-order logic

Resolution for first-order logic

Input: a closed formula σ of first-order logic.
Output: a boolean indicating whether σ is valid.
If σ is valid, the method always terminates.
If σ is not valid, the method may not terminate.

First-order resolution method
(Semi-decision procedure)

1. Write $\neg\sigma$ as a set $\mathcal{C}$ of **clauses**.
2. Search for a **refutation** of $\mathcal{C}$.
   If a refutation exists, the method will find one.
   If no refutation exists, the method may "hang".

Conversion to clausal form in first-order logic

A formula is converted to clausal form by applying the following rules.

Step 1. Eliminate the "⇒" connective:

$$\sigma \Rightarrow \tau \quad \longrightarrow \quad \neg\sigma \vee \tau$$

The resulting formula only uses the connectives $\{\neg, \vee, \wedge, \forall, \exists\}$.

Step 2. Push the "¬" connective inward:

$$\begin{array}{lcl}
\neg(\sigma \wedge \tau) & \longrightarrow & \neg\sigma \vee \neg\tau \\
\neg(\sigma \vee \tau) & \longrightarrow & \neg\sigma \wedge \neg\tau \\
\neg\neg\sigma & \longrightarrow & \sigma \\
\neg\forall \mathtt{X}.\, \sigma & \longrightarrow & \exists \mathtt{X}.\, \neg\sigma \\
\neg\exists \mathtt{X}.\, \sigma & \longrightarrow & \forall \mathtt{X}.\, \neg\sigma
\end{array}$$

The resulting formula is in **negation normal form** (NNF):

$$\begin{array}{rcl}
\sigma_{\mathrm{nnf}} & ::= & \mathbf{P}(t_1, \ldots t_n) \mid \neg\mathbf{P}(t_1, \ldots t_n) \mid \sigma_{\mathrm{nnf}} \wedge \sigma_{\mathrm{nnf}} \mid \sigma_{\mathrm{nnf}} \vee \sigma_{\mathrm{nnf}} \\
 & \mid & \forall \mathtt{X}.\, \sigma_{\mathrm{nnf}} \mid \exists \mathtt{X}.\, \sigma_{\mathrm{nnf}}
\end{array}$$

Conversion to clausal form in first-order logic

Step 3. Extract quantifiers ("$\forall/\exists$") outward.
We always assume $\mathtt{X} \notin \mathrm{fv}(\tau)$:

$$
\begin{array}{llll}
(\forall \mathtt{X}.\, \sigma) \wedge \tau \longrightarrow \forall \mathtt{X}.\, (\sigma \wedge \tau) & & \tau \wedge (\forall \mathtt{X}.\, \sigma) \longrightarrow \forall \mathtt{X}.\, (\tau \wedge \sigma) \\
(\forall \mathtt{X}.\, \sigma) \vee \tau \longrightarrow \forall \mathtt{X}.\, (\sigma \vee \tau) & & \tau \vee (\forall \mathtt{X}.\, \sigma) \longrightarrow \forall \mathtt{X}.\, (\tau \vee \sigma) \\
(\exists \mathtt{X}.\, \sigma) \wedge \tau \longrightarrow \exists \mathtt{X}.\, (\sigma \wedge \tau) & & \tau \wedge (\exists \mathtt{X}.\, \sigma) \longrightarrow \exists \mathtt{X}.\, (\tau \wedge \sigma) \\
(\exists \mathtt{X}.\, \sigma) \vee \tau \longrightarrow \exists \mathtt{X}.\, (\sigma \vee \tau) & & \tau \vee (\exists \mathtt{X}.\, \sigma) \longrightarrow \exists \mathtt{X}.\, (\tau \vee \sigma)
\end{array}
$$

All rules transform the formula into an equivalent one.

The resulting formula is in **prenex normal form**:

$$\sigma_{\mathrm{pre}} ::= \mathcal{Q}_1 \mathtt{X}_1.\, \mathcal{Q}_2 \mathtt{X}_2.\, \ldots\, \mathcal{Q}_n \mathtt{X}_n.\, \tau$$

where each $\mathcal{Q}_i$ is a quantifier $\{\forall, \exists\}$
and τ represents a quantifier-free formula in NNF.

Conversion to clausal form in first-order logic

Step 4. Eliminate existential quantifiers ($\exists$).
For this, the following technique by Herbrand and Skolem is used:

Lemma (Skolemization)

$$\begin{array}{rcl}
\forall \mathtt{X}.\, \exists \mathtt{Y}.\, \sigma(\mathtt{X}, \mathtt{Y}) \text{ is sat.} & \text{iff} & \forall \mathtt{X}.\, \sigma(\mathtt{X}, \mathtt{f}(\mathtt{X})) \text{ is sat.} \\
\forall \mathtt{X}_1 \mathtt{X}_2.\, \exists \mathtt{Y}.\, \sigma(\mathtt{X}_1, \mathtt{X}_2, \mathtt{Y}) \text{ is sat.} & \text{iff} & \forall \mathtt{X}_1 \mathtt{X}_2.\, \sigma(\mathtt{X}_1, \mathtt{X}_2, \mathtt{f}(\mathtt{X}_1, \mathtt{X}_2)) \text{ is sat.} \\
 & \vdots & \\
\forall \vec{\mathtt{X}}.\, \exists \mathtt{Y}.\, \sigma(\vec{\mathtt{X}}, \mathtt{Y}) \text{ is sat.} & \text{iff} & \forall \vec{\mathtt{X}}.\, \sigma(\vec{\mathtt{X}}, \mathtt{f}(\vec{\mathtt{X}})) \text{ is sat.}
\end{array}$$

The left-hand side is a formula in the language $\mathcal{L}$.
The right-hand side is a formula in the language $\mathcal{L} \cup \{\mathtt{f}\}$.

Special case when $|\vec{\mathtt{X}}| = 0$

$$\exists \mathtt{Y}.\, \sigma(\mathtt{Y}) \text{ is sat.} \quad \text{iff} \quad \sigma(\mathtt{c}) \text{ is sat.}$$

The language is extended with a new constant c.

Conversion to clausal form in first-order logic

Skolemization preserves **satisfiability**.
But it does not always produce equivalent formulas.
That is, **it does not preserve validity**.

Example — Skolemization does not preserve validity

$$\underbrace{\exists \mathtt{X}.\, (\mathbf{P}(0) \Rightarrow \mathbf{P}(\mathtt{X}))}_{\text{valid}} \qquad \underbrace{\mathbf{P}(0) \Rightarrow \mathbf{P}(\mathtt{c})}_{\text{invalid}}$$

Conversion to clausal form in first-order logic

Given a formula in prenex normal form, the rule is applied:

$$\forall \mathtt{X}_1. \ldots \forall \mathtt{X}_n. \exists \mathtt{Y}. \sigma \quad \longrightarrow \quad \forall \mathtt{X}_1. \ldots \forall \mathtt{X}_n. \sigma\{\mathtt{Y} := \mathtt{f}(\mathtt{X}_1, \ldots, \mathtt{X}_n)\}$$

where `f` is a new function symbol of arity $n \geq 0$.

The resulting formula is in **Skolem normal form**:

$$\sigma_{\mathrm{Sk}} ::= \forall \mathtt{X}_1 \mathtt{X}_2 \ldots \mathtt{X}_n. \tau$$

where τ represents a quantifier-free formula in NNF.

Conversion to clausal form in first-order logic

Step 5. Given a formula in Skolem normal form:

$$\forall \mathtt{X}_1 \mathtt{X}_2 \ldots \mathtt{X}_n . \tau \quad (\tau \text{ quantifier-free})$$

τ is converted to conjunctive normal form using the already seen rules:

$$\begin{array}{rcl} \sigma \vee (\tau \wedge \rho) & \longrightarrow & (\sigma \vee \tau) \wedge (\sigma \vee \rho) \\ (\sigma \wedge \tau) \vee \rho & \longrightarrow & (\sigma \vee \rho) \wedge (\tau \vee \rho) \end{array}$$

The result is a formula of the form:

$$\forall \mathtt{X}_1 \ldots \mathtt{X}_n . \left(\begin{array}{ll} & (\ell_1^{(1)} \vee \ldots \vee \ell_{m_1}^{(1)}) \\ \wedge & (\ell_1^{(2)} \vee \ldots \vee \ell_{m_2}^{(2)}) \\ \ldots & \\ \wedge & (\ell_1^{(k)} \vee \ldots \vee \ell_{m_k}^{(k)}) \end{array} \right)$$

Conversion to clausal form in first-order logic

Step 6. Push the universal quantifiers inward:

$$\forall \mathtt{X}_1 \dots \mathtt{X}_n . \left(\begin{array}{ll} & (\ell_1^{(1)} \vee \dots \vee \ell_{m_1}^{(1)}) \\ \wedge & (\ell_1^{(2)} \vee \dots \vee \ell_{m_2}^{(2)}) \\ \dots & \\ \wedge & (\ell_1^{(k)} \vee \dots \vee \ell_{m_k}^{(k)}) \end{array} \right) \longrightarrow \left(\begin{array}{ll} & \forall \mathtt{X}_1 \dots \mathtt{X}_n . (\ell_1^{(1)} \vee \dots \vee \ell_{m_1}^{(1)}) \\ \wedge & \forall \mathtt{X}_1 \dots \mathtt{X}_n . (\ell_1^{(2)} \vee \dots \vee \ell_{m_2}^{(2)}) \\ \dots & \\ \wedge & \forall \mathtt{X}_1 \dots \mathtt{X}_n . (\ell_1^{(k)} \vee \dots \vee \ell_{m_k}^{(k)}) \end{array} \right)$$

Finally the **clausal form** is:

$$\left\{ \begin{array}{l} \{\ell_1^{(1)}, \dots, \ell_{m_1}^{(1)}\}, \\ \{\ell_1^{(2)}, \dots, \ell_{m_2}^{(2)}\}, \\ \vdots \\ \{\ell_1^{(k)}, \dots, \ell_{m_k}^{(k)}\} \end{array} \right\}$$

Conversion to clausal form in first-order logic

Summary — conversion to clausal form in first-order logic

1. Rewrite $\Rightarrow$ using $\neg$ and $\vee$.
2. Convert to negation normal form, pushing $\neg$ inward.
3. Convert to prenex normal form, extracting $\forall, \exists$ outward.
4. Convert to Skolem normal form, Skolemizing existentials.
5. Convert to conjunctive normal form, distributing $\vee$ over $\wedge$.
6. Push quantifiers inward over conjunctions.

Each step produces an equivalent formula,
except Skolemization which only preserves satisfiability.

Conversion to clausal form in first-order logic

Example — conversion to clausal form

We want to check if $\sigma \equiv \exists \mathtt{X}.\,(\forall \mathtt{Y}.\,\mathbf{P}(\mathtt{X},\mathtt{Y}) \Rightarrow \forall \mathtt{Y}.\,\mathbf{P}(\mathtt{Y},\mathtt{X}))$ is valid.
First we negate it: $\neg\sigma \equiv \neg\exists \mathtt{X}.\,(\forall \mathtt{Y}.\,\mathbf{P}(\mathtt{X},\mathtt{Y}) \Rightarrow \forall \mathtt{Y}.\,\mathbf{P}(\mathtt{Y},\mathtt{X}))$.
We convert $\neg\sigma$ to clausal form:

$$
\begin{array}{ll}
 & \neg\exists \mathtt{X}.\,(\forall \mathtt{Y}.\,\mathbf{P}(\mathtt{X},\mathtt{Y}) \Rightarrow \forall \mathtt{Y}.\,\mathbf{P}(\mathtt{Y},\mathtt{X})) \\
\longrightarrow & \neg\exists \mathtt{X}.\,(\neg\forall \mathtt{Y}.\,\mathbf{P}(\mathtt{X},\mathtt{Y}) \vee \forall \mathtt{Y}.\,\mathbf{P}(\mathtt{Y},\mathtt{X})) \\
\longrightarrow & \forall \mathtt{X}.\,\neg(\neg\forall \mathtt{Y}.\,\mathbf{P}(\mathtt{X},\mathtt{Y}) \vee \forall \mathtt{Y}.\,\mathbf{P}(\mathtt{Y},\mathtt{X})) \\
\longrightarrow & \forall \mathtt{X}.\,(\neg\neg\forall \mathtt{Y}.\,\mathbf{P}(\mathtt{X},\mathtt{Y}) \wedge \neg\forall \mathtt{Y}.\,\mathbf{P}(\mathtt{Y},\mathtt{X})) \\
\longrightarrow & \forall \mathtt{X}.\,(\forall \mathtt{Y}.\,\mathbf{P}(\mathtt{X},\mathtt{Y}) \wedge \exists \mathtt{Y}.\,\neg\mathbf{P}(\mathtt{Y},\mathtt{X})) \\
\longrightarrow & \forall \mathtt{X}.\,\exists \mathtt{Y}.\,(\forall \mathtt{Y}.\,\mathbf{P}(\mathtt{X},\mathtt{Y}) \wedge \neg\mathbf{P}(\mathtt{Y},\mathtt{X})) \\
\longrightarrow & \forall \mathtt{X}.\,\exists \mathtt{Y}.\,\forall \mathtt{Z}.\,(\mathbf{P}(\mathtt{X},\mathtt{Z}) \wedge \neg\mathbf{P}(\mathtt{Y},\mathtt{X})) \\
\longrightarrow & \forall \mathtt{X}.\,\forall \mathtt{Z}.\,(\mathbf{P}(\mathtt{X},\mathtt{Z}) \wedge \neg\mathbf{P}(\mathtt{f}(\mathtt{X}),\mathtt{X})) \\
\longrightarrow & \forall \mathtt{X}.\,\forall \mathtt{Z}.\,\mathbf{P}(\mathtt{X},\mathtt{Z}) \wedge \forall \mathtt{X}.\,\forall \mathtt{Z}.\,\neg\mathbf{P}(\mathtt{f}(\mathtt{X}),\mathtt{X})
\end{array}
$$

The clausal form is:

$$\{\{\mathbf{P}(\mathtt{X},\mathtt{Z})\},\{\neg\mathbf{P}(\mathtt{f}(\mathtt{X}),\mathtt{X})\}\} \equiv \{\{\mathbf{P}(\mathtt{X},\mathtt{Y})\},\{\neg\mathbf{P}(\mathtt{f}(\mathtt{Z}),\mathtt{Z})\}\}$$

Refutation in first-order logic

Once a set of clauses $\mathcal{C} = \{\kappa_1, \ldots, \kappa_n\}$ is obtained, we search for a **refutation**, i.e., a proof of $\mathcal{C} \vdash \bot$.

Recall the propositional resolution rule:

$$\frac{\{\mathbf{P}, \ell_1, \ldots, \ell_n\} \quad \{\neg\mathbf{P}, \ell'_1, \ldots, \ell'_m\}}{\{\ell_1, \ldots, \ell_n, \ell'_1, \ldots, \ell'_m\}}$$

We want to adapt it to first-order logic.

Instead of a propositional variable **P** we will have an atomic formula $\mathbf{P}(t_1, \ldots, t_n)$.
Can we write the rule like this?:

$$\frac{\{\mathbf{P}(t_1, \ldots, t_n), \ell_1, \ldots, \ell_n\} \quad \{\neg\mathbf{P}(t_1, \ldots, t_n), \ell'_1, \ldots, \ell'_m\}}{\{\ell_1, \ldots, \ell_n, \ell'_1, \ldots, \ell'_m\}}$$

Refutation in first-order logic

Consider the formula:

$$\forall \mathtt{X}.\, \mathbf{P}(\mathtt{X}) \wedge \neg \mathbf{P}(0)$$

It should be refutable, since it is unsatisfiable.
Its clausal form consists of two clauses:

$$\{\mathbf{P}(\mathtt{X})\} \quad \{\neg \mathbf{P}(0)\}$$

The proposed resolution rule does not apply because $\mathbf{P}(\mathtt{X}) \neq \mathbf{P}(0)$.

The terms do not necessarily have to be equal.
We relax the rule to allow them to be **unifiable**.

Refutation in first-order logic

The first-order resolution rule is:

$$\frac{\{\sigma_1, \ldots, \sigma_p, \ell_1, \ldots, \ell_n\} \quad \{\neg\tau_1, \ldots, \neg\tau_q, \ell'_1, \ldots, \ell'_m\} \\ \mathbf{S} = \mathrm{mgu}(\{\sigma_1 \stackrel{?}{=} \sigma_2 \stackrel{?}{=} \ldots \stackrel{?}{=} \sigma_p \stackrel{?}{=} \tau_1 \stackrel{?}{=} \tau_2 \stackrel{?}{=} \ldots \stackrel{?}{=} \tau_q\})}{\mathbf{S}(\{\ell_1, \ldots, \ell_n, \ell'_1, \ldots, \ell'_m\})}$$

with $p > 0$ and $q > 0$.

It is implicitly assumed that the clauses are renamed so that $\{\sigma_1, \ldots, \sigma_p, \ell_1, \ldots, \ell_n\}$ and $\{\neg\tau_1, \ldots, \neg\tau_q, \ell'_1, \ldots, \ell'_m\}$ have no variables in common.

Refutation in first-order logic

The refutation algorithm adapts without major changes.
The new resolution rule is used to compute the resolvent.

Refutation in first-order logic

Example — resolution method

We want to prove $\sigma \equiv \exists \mathtt{X}.\,(\forall \mathtt{Y}.\,\mathbf{P}(\mathtt{X},\mathtt{Y}) \Rightarrow \forall \mathtt{Y}.\,\mathbf{P}(\mathtt{Y},\mathtt{X}))$.
Equivalently, we show that $\neg\sigma \vdash \bot$.
The clausal form of $\neg\sigma$ was:

$$\mathcal{C} = \{\underbrace{\{\mathbf{P}(\mathtt{X},\mathtt{Y})\}}_{\boxed{1}}, \underbrace{\{\neg\mathbf{P}(\mathtt{f}(\mathtt{Z}),\mathtt{Z})\}}_{\boxed{2}}\}$$

- From $\boxed{1}$ and $\boxed{2}$ we compute $\mathbf{mgu}(\mathbf{P}(\mathtt{X},\mathtt{Y}) \stackrel{?}{=} \mathbf{P}(\mathtt{f}(\mathtt{Z}),\mathtt{Z})) = \{\mathtt{X} := \mathtt{f}(\mathtt{Z}), \mathtt{Y} := \mathtt{Z}\}$ and obtain the resolvent $\{\,\}$.

Refutation in first-order logic

Binary resolution

Consider the following variant of the resolution rule:

$$\frac{\{\sigma, \ell_1, \ldots, \ell_n\} \quad \{\neg\tau, \ell'_1, \ldots, \ell'_m\} \quad \mathbf{S} = \mathrm{mgu}(\{\sigma \stackrel{?}{=} \tau\})}{\mathbf{S}(\{\ell_1, \ldots, \ell_n, \ell'_1, \ldots, \ell'_m\})}$$

It is not complete.

Example

$\{\{\mathbf{P}(\mathtt{X}), \mathbf{P}(\mathtt{Y})\}, \{\neg\mathbf{P}(\mathtt{Z}), \neg\mathbf{P}(\mathtt{W})\}\}$ is unsatisfiable.
It is not possible to reach the empty clause $\{\,\}$ with binary resolution.

Correctness of the first-order resolution method

Theorem (correctness of conversion to clausal form)

Given a formula σ:

1. Conversion to clausal form terminates.
2. The resulting clause set $\mathcal{C}$ is equisatisfiable with σ.
   That is, σ is sat. if and only if $\mathcal{C}$ is sat..

Correctness of the first-order resolution method

Theorem (correctness of the refutation algorithm)

Given a set of clauses $\mathcal{C}_0$:

1. If $\mathcal{C}_0 \vdash \bot$, there is a way to choose the clauses such that the refutation algorithm terminates.
2. The algorithm returns `UNSAT` if and only if $\mathcal{C}_0 \vdash \bot$.

If $\mathcal{C}_0 \nvdash \bot$, termination is not guaranteed.

First-order resolution

Example — non-termination

The following formula σ is not valid:

$$\forall \mathtt{X}.\, (\mathbf{P}(\mathtt{succ}(\mathtt{X})) \Rightarrow \mathbf{P}(\mathtt{X})) \Rightarrow \mathbf{P}(0)$$

Let's try to prove its validity using the resolution method.
To do so, we convert $\neg\sigma$ to clausal form:

$$\{\underbrace{\{\neg\mathbf{P}(\mathtt{succ}(\mathtt{X})), \mathbf{P}(\mathtt{X})\}}_{\boxed{1}}, \underbrace{\{\neg\mathbf{P}(0)\}}_{\boxed{2}}\}$$

- From $\boxed{1}$ and $\boxed{2}$ we obtain $\boxed{3} = \{\neg\mathbf{P}(\mathtt{succ}(0))\}$.
- From $\boxed{1}$ and $\boxed{3}$ we obtain $\boxed{4} = \{\neg\mathbf{P}(\mathtt{succ}(\mathtt{succ}(0)))\}$.
- From $\boxed{1}$ and $\boxed{4}$ we obtain $\boxed{5} = \{\neg\mathbf{P}(\mathtt{succ}(\mathtt{succ}(\mathtt{succ}(0))))\}$.

  ...

¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ? ? ? ? ? ? ? ? ?

Recommended reading

Robinson's original article.

J. A. Robinson. *A Machine-Oriented Logic Based on the Resolution Principle*.
Journal of the Association for Computing Machinery, Vol. 12, No. 1 (January 1965), pp. 23-41.