Automating First-Order Logic Proofs Using Resolution

Viktor Kunčak, EPFL

Automating first-order logic (FOL)

First-order logic supports arbitrary (uninterpreted) predicates and functions

- their meaning will be constrained through axioms

FOL can be used in practice to formalize most of mathematics (e.g. through set theory axioms), and thus all of program verification problems.
To prove whether a property holds, we can proceed as follows:

- describe the property using a formula F
- describe the functions and relations in F using a sequence of axioms S

Check if the sequence $(\neg F; S)$ is contradictory. If yes, then F follows from S

Completeness: there is a procedure that will, if F does follow from S, in finite time establish this (we do not know how long it will take, if it does not hold, it loops).
We give one such procedure: resolution for FOL.

Running example of FOL formula

Here is a first-order logic formula:

$$\begin{array}{l}\big[(\forall x.\exists y.\ R(x,y)) \wedge \\ (\forall x.\forall y.\ (R(x,y) \rightarrow \forall z.\ R(x,f(y,z)))) \wedge \\ (\forall x.\ (P(x) \vee P(f(x,a))))\big] \\ \rightarrow \forall x.\exists y.\ (R(x,y) \wedge P(y))\end{array}$$

Note that it contains:

- propositional operations $\wedge, \vee, \neg, \rightarrow$
- variables x, y
- function symbols and constants: f, a
- predicate symbols: P, R
- quantifiers $\forall, \exists$ (only over variables, not function or predicate symbols - FO)

First-order logic syntax and terminology

A first-order *signature* $\mathscr{L}$ (akka language) specifies a countable set of function symbols f (constants c are functions symbols taking no arguments), and predicate symbols p.
Syntax of formulas (F) and terms (t) in first-order logic:

$$F ::= p(t_1, \ldots, t_n) \mid \forall x.F \mid \exists x.F \mid \top \mid \bot \mid \neg F \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid F_1 \to F_2 \mid F_1 \leftrightarrow F_2$$
$$t ::= x \mid c \mid f(t_1, \ldots, t_n)$$

where $x \in Var$ denotes variables, which form a countably infinite fixed set.
ar denotes arity of functions and predicate symbols; e.g. $ar(f) = 2$ means f takes two arguments, so it is allowed to form a term $f(t_1, t_2)$, and also $ar(p) = 2$ for predicate symbol p means that it is allowed to form formula $p(t_1, t_2)$.
We interpret each f as $\overline{f} : D^{ar(f)} \to D$ and each p as $\overline{p} : D^{ar(p)} \to \{0,1\}$ (or: $\subseteq D^{ar(p)}$)

We call $p(t_1, \ldots, t_n)$ an *atomic formula* (contains no logical connectives or quantifiers).
A *literal* is such an atomic formula, or its negation $\neg p(t_1, \ldots, t_n)$
A *clause* is a disjunction of literals, e.g. $\neg p(x, f(y)) \vee q(y) \vee \neg r(x, z)$

Example of a FOL signature

We will look at the signature $\mathscr{L} = \{P, R, a, f\}$ where

- P is a predicate symbol, $ar(P) = 1$
- R is a predicate symbol, $ar(R) = 2$
- a is a constant
- f is a function symbol, $ar(f) = 2$

An **interpretation** for this language is then any structure (D, e) where

- $D \neq \emptyset$ (it can be finite or infinite)
- $e(P) : D \to \{0, 1\}$
- $e(R) : D^2 \to \{0, 1\}$
- $e(a) \in D$
- $e(f) : D^2 \to D$

Example: $D = \{0, 1, 2, \ldots\}$ (non-negative integers) and:

- $e(P)(n) =$ if $(2|n)$ then 1 else 0
- $e(R)(x, y) =$ if $(x \leq y)$ then 1 else 0
- $e(a) = 1$
- $e(f)(x, y) = x + y$

Semantics given an interpretation

Taking that interpretation, $D = \{0, 1, 2, \dots\}$,

- $e(P)(n) =$ if $(2|n)$ then 1 else 0
- $e(R) =$ if $(x \leq y)$ then 1 else 0
- $e(a) = 1$
- $e(f)(x, y) = x + y$

we can talk about the truth value of any closed formula, e.g.:

(text, or tree)

$$
\begin{array}{l}
(\forall x.\exists y.\ R(x,y)) \wedge \\
(\forall x.\forall y.\ (R(x,y) \rightarrow \forall z.\ R(x, f(y,z)))) \wedge \\
(\forall x.\ (P(x) \vee P(f(x,a)))) \\
\rightarrow \forall x.\exists y.\ (R(x,y) \wedge P(y))
\end{array}
$$

Its truth is precisely the truth of this (symbols are replaced with their value in e):

(true or false)

$$
\begin{array}{l}
(\forall x.\exists y.\ x \leq y) \wedge \\
(\forall x.\forall y.\ x \leq y\ \rightarrow \forall z.\ x \leq (y + z)) \wedge \\
(\forall x.\ (2|x) \vee (2|x+1)) \\
\rightarrow \forall x.\exists y.\ (x \leq y \wedge (2|y))
\end{array}
$$

Semantics in general: interpreter written in set theory

A first-order *interpretation* is $I = (D, e)$ where $D \neq \emptyset$ and e maps constants, function and predicate symbols as follows:

- each predicate symbol p with $ar(p) = n$ into $e(p) : D^n \to \{0,1\}$
- each function symbol f with $ar(f) = n$ into a total function of n arguments, $e(f) : D^n \to D$ (and each constant c into element of D, i.e., $e(c) \in D$)
- maps each variable x to element of D, i.e., $e(x) \in D$

We then define $[\![F]\!]_I \in \{0,1\}$ to denote whether F is true (1) or false (0) false in interpretation I. The rules are the expected rules recursive on the formula tree:

$$[\![F_1 \wedge F_2]\!]_I = [\![F_1]\!]_I \wedge [\![F_2]\!]_I \qquad [\![F_1 \vee F_2]\!]_I = [\![F_1]\!]_I \vee [\![F_2]\!]_I \qquad [\![\neg F]\!]_I = \neg [\![F]\!]_I \qquad [\![\bot]\!]_I = 0$$

$$[\![f(t_1,\ldots,t_n)]\!]_I = e(f)([\![t_1]\!]_I,\ldots,[\![t_n]\!]_I) \qquad [\![p(t_1,\ldots,t_n)]\!]_I = e(p)([\![t_1]\!]_I,\ldots,[\![t_n]\!]_I) \qquad [\![x]\!]_I = e(x)$$

$$[\![\forall x.F]\!]_{(D,e)} = \text{ if } (\forall d \in D.\ ([\![F]\!]_{(D,e[x:=d])} = 1)) \text{ then 1 else 0}$$
$$[\![\exists x.F]\!]_{(D,e)} = \text{ if } (\exists d \in D.\ ([\![F]\!]_{(D,e[x:=d])} = 1)) \text{ then 1 else 0}$$

where $e[x := d](y) = (\text{if } x = y \text{ then } d \text{ else } e(y))$

What makes this logic first order

$$[\![\forall x.F]\!]_{(D,e)} = \text{if}(\{d \in D \mid [\![F]\!]_{(D,e[x:=d])} = 1\} = D) \text{ then 1 else 0}$$
$$[\![\exists x.F]\!]_{(D,e)} = \text{if}(\{d \in D \mid [\![F]\!]_{(D,e[x:=d])} = 1\} \neq \emptyset) \text{ then 1 else 0}$$

We *can* quantify over variables $\forall x.F$, $\exists x.F$, which are interpreted over D, and we can nest quantifiers, e.g. $\forall x.\exists y.\ (p(x,y) \wedge q(y,x))$.
We *cannot* write a FOL formula that quantifies over function and relation symbols (that would be *second-order* or, generally, *higher-order*).

The meaning of function and relation symbols is fixed in e of interpretation $I = (D, e)$.

To make statements that do not depend on a particular interpretation, we use concepts of *satisfiability* and *validity*:

- F is **valid** if, **for all interpretations** (D, e) (for arbitrarily small or large sets D and all possible choices of e), $[\![F]\!]_{(D,e)} = 1$
- F is **satisfiable** if **there exists an interpretation** (D, e) such that $[\![F]\!]_{(D,e)} = 1$

Satisfiability and validity: illustration

Take first-order logic (FOL) formula

$$\forall x.\exists y.\ (p(x,y) \wedge q(y,x))$$

Its **satisfiability** is, by definition, equivalent to an informal statement (which itself is not a formula of first-order logic but a (meta)mathematical statement):

$$\exists \mathbf{D} \neq \emptyset.\ \exists \overline{\mathbf{p}}, \overline{\mathbf{q}} : \mathbf{D^2} \to \{\mathbf{0},\mathbf{1}\}.\ \forall x \in D.\exists y \in D.\ ((\overline{p}(x,y) \wedge \overline{q}(y,x)) = 1)$$

Its **validity** is, by definition, equivalent a mathematical statement:

$$\forall \mathbf{D} \neq \emptyset.\ \forall \overline{\mathbf{p}}, \overline{\mathbf{q}} : \mathbf{D^2} \to \{\mathbf{0},\mathbf{1}\}.\ \forall x \in D.\exists y \in D.\ ((\overline{p}(x,y) \wedge \overline{q}(y,x)) = 1)$$

The domain, functions, and relations are either all existentially quantified (if we ask about satisfiability) or all universally quantified (if we ask about validity).

Observation: F is valid if and only if $\neg F$ is not satisfiable.
We will be checking satisfiability (aiming for a negative answer).

Back to our example

Consider our example formula F:

$$
\begin{array}{l}
(\forall x.\exists y.\ R(x,y)) \wedge \\
(\forall x.\forall y.\ R(x,y) \rightarrow \forall z.\ R(x,f(y,z))) \wedge \\
(\forall x.\ P(x) \vee P(f(x,a))) \\
\rightarrow \forall x.\exists y.\ R(x,y) \wedge P(y)
\end{array}
$$

We have seen it has an interpretation where it is true, so F is satisfiable.
We are interested in checking its *validity*.
To do that, we will check the satisfiability of $\neg F$.

$$
\begin{array}{ll}
\neg\big(& (\forall x.\exists y.\ R(x,y)) \wedge \\
& (\forall x.\forall y.\ R(x,y) \rightarrow \forall z.\ R(x,f(y,z))) \wedge \\
& (\forall x.\ P(x) \vee P(f(x,a))) \\
& \rightarrow \forall x.\exists y.\ R(x,y) \wedge P(y)\ \big)
\end{array}
$$

In general, we will transform $\neg F$ into a *normal form*.

Negation normal form for FOL

Observation: If $F \longleftrightarrow G$ is a valid FOL formula, then inside any other FOL formula H we can replace a sub-formula F with G without changing the truth value of the formula: $H[F] \rightsquigarrow H[G]$.

We can transform formulas to negation normal using transformations such as these:

$$
\begin{array}{rcl}
F_1 \longleftrightarrow F_2 & \rightsquigarrow & (F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1) \\
F_1 \rightarrow F_2 & \rightsquigarrow & \neg F_1 \vee F_2 \\
\neg\neg F & \rightsquigarrow & F \\
\neg(F_1 \wedge F_2) & \rightsquigarrow & \neg F_1 \vee \neg F_2 \\
\neg(F_1 \vee F_2) & \rightsquigarrow & \neg F_1 \wedge \neg F_2 \\
\neg\forall x.F & \rightsquigarrow & \exists x.\neg F \\
\neg\exists x.F & \rightsquigarrow & \forall x.\neg F \\
\neg\bot & \rightsquigarrow & \top \\
\neg\top & \rightsquigarrow & \bot
\end{array}
$$

In negation normal form, negation applies only to atomic formulas and the only other propositional connectives are $\wedge$, $\vee$.

Compute negation normal form

$$\neg\Big(\big((\forall x.\exists y.\ R(x,y)) \wedge$$
$$\big(\forall x.\forall y.\ (R(x,y) \rightarrow \forall z.\ R(x,f(y,z)))\big) \wedge$$
$$(\forall x.\ P(x) \vee P(f(x,a)))\ \big) \rightarrow \forall x.\exists y.\ R(x,y) \wedge P(y)\Big)$$

becomes:

Compute negation normal form

$$\neg\Big(\big((\forall x.\exists y.\ R(x,y)) \wedge$$
$$\big(\forall x.\forall y.\ (R(x,y) \rightarrow \forall z.\ R(x,f(y,z)))\big) \wedge$$
$$(\forall x.\ P(x) \vee P(f(x,a)))\ \big) \rightarrow \forall x.\exists y.\ R(x,y) \wedge P(y)\Big)$$

becomes:

$$(\forall x.\exists y.\ R(x,y)) \wedge$$
$$\big(\forall x.\forall y.\ (\neg R(x,y) \vee \forall z.\ R(x,f(y,z)))\big) \wedge$$
$$\big(\forall x.\ (P(x) \vee P(f(x,a)))\big) \wedge$$
$$\big(\exists x.\forall y.\ (\neg R(x,y) \vee \neg P(y))\big)$$

Compute negation normal form

$$\neg\Big(\big((\forall x.\exists y.\ R(x,y)) \wedge$$
$$\big(\forall x.\forall y.\ (R(x,y) \rightarrow \forall z.\ R(x,f(y,z)))\big) \wedge$$
$$(\forall x.\ P(x) \vee P(f(x,a)))\ \big) \rightarrow \forall x.\exists y.\ R(x,y) \wedge P(y)\Big)$$

becomes:

$$(\forall x.\exists y.\ R(x,y)) \wedge$$
$$\big(\forall x.\forall y.\ (\neg R(x,y) \vee \forall z.\ R(x,f(y,z)))\big) \wedge$$
$$\big(\forall x.\ (P(x) \vee P(f(x,a)))\big) \wedge$$
$$\big(\exists x.\forall y.\ (\neg R(x,y) \vee \neg P(y))\big)$$

Coming next: get rid of existential quantifiers!

Introducing a Skolem function

Observe that e.g. the following formula is valid: $p(x, s(x)) \rightarrow \exists y.p(x, y)$.
Indeed, fix any interpretation (D, e) and assume $p(x, s(x))$.
To prove $\exists y.p(x, y)$, just let y be $s(x)$.
Consequently, if $\forall x.p(x, s(x))$ is satisfiable, so is $\forall x.\exists y.p(x, y)$.
A converse is also true. Take any interpretation (D, e) in which $\forall x.\exists y.p(x, y)$ holds. Then for any x_d, there exists $y_d \in D$ such that $e(p)(x_d, y_d) = 1$. Construct (by axiom of choice) a function $\bar{s}$ that assigns to every element $x_d \in D$ one such $y_d \in D$ for which $e(p)(x_d, y_d) = 1$. Extend the signature with a **new function symbol** s (Skolem function, according to (W) Thoralf Skolem) that does not appear in the formula. Define a new interpretation $I' = (D, e')$ (with the same domain) by $e' = e[s := \bar{s}]$, that is, e' behaves like e but maps a new symbol s to the function $\bar{s}$. Then $[\![\forall x.p(x, s(x))]\!]_{I'} = 1$.
We can do this transform of $\exists$ to Skolem functions not just for p, but any quantifier-free formula. The transformation preserves satisfiability.

Skolemization Algorithm in General

In a formula that is in negation normal form, replace a subformula

$$\exists y.\ F(x_1,\ldots,x_n,y)$$

with

$$F(x_1,\ldots,x_n,g(x_1,\ldots,x_n))$$

where g is a new function symbol (Skolem function) of arity n and $\{x_1,\ldots,x_n\}$ are the variables free in F, computed like this:

$$\begin{array}{rcl} FV(c) & = & \emptyset,\quad FV(x)=\{x\} \\ FV(f(t_1,\ldots,t_n)) & = & FV(t_1)\cup\ldots\cup FV(t_n)=FV(p(t_1,\ldots,t_n)) \\ FV(F_1\wedge F_2) & = & FV(F_1)\cup FV(F_2) \qquad\qquad FV(\neg F)=FV(F) \\ FV(\forall x.F) & = & FV(F)\setminus\{x\}=FV(\exists x.F) \end{array}$$

If we have quantifiers $Q_1x_1\ldots.Q_nx_n.F$ we start eliminating from outside (left) and then Skolem function arguments for $\exists x_i$ are only $\forall$ quantified variables among $x_1,\ldots,x_{i-1}$.

Skolemized form for the example

$$
\begin{array}{ll}
(\forall x.\exists y.\ R(x,y)) \wedge & y \rightsquigarrow s_1(x) \\
\big(\forall x.\forall y.\ (\neg R(x,y) \vee \forall z.\ R(x, f(y,z)))\big) \wedge & \\
\big(\forall x.\ (P(x) \vee P(f(x,a)))\big) \wedge & \\
\big(\exists x.\forall y.\ (\neg R(x,y) \vee \neg P(y))\big) & x \rightsquigarrow s_2
\end{array}
$$

becomes:

$$
\begin{array}{l}
(\forall x.\ R(x, s_1(x))) \wedge \\
\big(\forall x.\forall y.\ (\neg R(x,y) \vee \forall z.\ R(x, f(y,z)))\big) \wedge \\
\big(\forall x.\ (P(x) \vee P(f(x,a)))\big) \wedge \\
\big(\forall y.\ (\neg R(s_2,y) \vee \neg P(y))\big)
\end{array}
$$

Moving quantifiers and conjunctive normal form

Note that all bound quantified variables can be renamed while preserving equivalence and that quantifier can be pulled in or out of formula where it does not occur:

- $(\forall x.F) \leftrightarrow \forall y.F[x := y]$, where y is fresh
- $(\forall x.F) \vee G \leftrightarrow \forall x.(F \vee G)$
- $(\forall x.F) \wedge G \leftrightarrow \forall x.(F \wedge G)$

We can thus put formulas outwards. If we choose, we can obtain: *prenex normal form*, a formula of the form

$$Q_1 x_1 . Q_2 x_2 . \ldots Q_n x_n . G$$

where $Q_i \in \{\forall, \exists\}$ and G has no quantifiers.
We can then transform G to conjunctive normal form, as for propositional logic:

- $(F \wedge G) \vee H \;\leftrightarrow\; (F \vee H) \wedge (G \vee H)$

We can also introduce fresh predicate symbols to avoid exponential blowup.

Conjunction of clauses

Theorem: each formula F can be transformed into an equisatisfiable formula of the form:

$$(\forall x_1, \ldots, x_n. C_1) \wedge \ldots \wedge (\forall x_1, \ldots, x_n. C_m)$$

where each C_i is quantifier free and, moreover, a disjunction of first-order literals.
Note: a first-order literal is either an atomic formula, that is $p(t_1, \ldots, t_k)$ where p is a predicate symbol, or a negation of an atomic formula, $\neg p(t_1, \ldots, t_k)$.
Indeed, we have seen how to move $\neg$ to predicates (NNF), eliminate $\exists$, and pull $\forall$ and $\wedge$ to the outside, leaving $\vee$ applied to literals.
Each C_i is a (first-order) clause. We can also view it as a set, because the order of disjuncts does not matter up to equivalence. Clause that is an empty set means $\bot$.
For a given formula F, denote the set of obtained clauses $\{C_1, \ldots, C_m\}$ by $\mathit{clauses}(F)$.
We omit universal quantifiers because all variables are universally quantified. We use a convention to denote variables by $x, y, z, \ldots$ and constants by $a, b, c, \ldots$.

Sets of formulas when checking satisfiability

We define that a set of formulas S is true in an interpretation I if *every* formula is true in that interpretation:

$$[\![S]\!]_I = \text{ if } (\forall F \in S.[\![F]\!]_I = 1) \text{ then } 1 \text{ else } 0$$

If S is finite then $[\![S]\!]_I = [\![\bigwedge_{F \in S} S]\!]_I$.

We say two sets are *equivalent* iff the set of interpretations in which they are true are equal.
We say that S is satisfiable iff there exists I such that $[\![S]\!]_I = 1$.

Thus, F is satisfiable iff $\mathit{clauses}(F)$ is satisfiable.

Clauses for the example

$$
\begin{array}{l}
(\forall x.\ R(x, s_1(x))) \wedge \\
\big(\forall x. \forall y.\ (\neg R(x,y) \vee \forall z.\ R(x, f(y,z)))\big) \wedge \\
\big(\forall x.\ (P(x) \vee P(f(x,a)))\big) \wedge \\
\big(\forall y.\ (\neg R(s_2, y) \vee \neg P(y))\big)
\end{array}
$$

gives the set of clauses:

$$
\begin{array}{l}
\big\{R(x, s_1(x)), \\
\ \neg R(x,y) \vee R(x, f(y,z)), \\
\ P(x) \vee P(f(x,a)), \\
\ \neg R(s_2, y) \vee \neg P(y)\big\}
\end{array}
$$

Clauses for the example

$$
\begin{array}{l}
(\forall x.\ R(x, s_1(x))) \wedge \\
\big(\forall x. \forall y.\ (\neg R(x,y) \vee \forall z.\ R(x, f(y,z)))\big) \wedge \\
\big(\forall x.\ (P(x) \vee P(f(x,a)))\big) \wedge \\
\big(\forall y.\ (\neg R(s_2, y) \vee \neg P(y))\big)
\end{array}
$$

gives the set of clauses:

$$
\begin{array}{l}
\big\{R(x, s_1(x)), \\
\ \neg R(x,y) \vee R(x, f(y,z)), \\
\ P(x) \vee P(f(x,a)), \\
\ \neg R(s_2, y) \vee \neg P(y)\big\}
\end{array}
$$

or, if we represent clauses themselves as sets:

$$
\begin{array}{l}
\big\{\{R(x, s_1(x))\}, \\
\ \{\neg R(x,y),\ R(x, f(y,z))\}, \\
\ \{P(x), P(f(x,a))\}, \\
\ \{\neg R(s_2, y), \neg P(y)\}\ \big\}
\end{array}
$$

Applying sound inference rules

We say that an inference rule

$$\frac{F_1 \ \ldots F_n}{G} \qquad (*)$$

is **sound** iff the formula $(F_1 \wedge \ldots \wedge F_n) \rightarrow G$ is valid.

Observation. Let $(*)$ be a sound inference rule and $F_1, \ldots, F_n \in S$. Then S and $S \cup \{G\}$ are equivalent.

Example: using convention that variables are $\forall$ quantified, the **instantiation** rule

$$\frac{C}{C[\overline{x} := \overline{t}]}$$

is sound, where $C[\overline{x} := \overline{t}]$ denotes substituting variables of C by some terms.

In general, $\overline{t}$ may contain other variables. Special cases of instantiation are:

- renaming: replace variables with fresh variables
- ground instantiation: replace variables with *ground terms*, that have no variables (built only from constants and function symbols)

Resolution for FOL clauses

The following formula (transitivity of universal implication) is easily proven to be valid for all formulas F, G, H, where $\overline{x}$ denotes the finite list of all free variables in F, G, H:

$$\big((\forall \overline{x}.\ (F \to G)) \ \wedge\ (\forall \overline{x}.\ (G \to H))\big) \quad \to \quad (\forall \overline{x}.(F \to H))$$

Write it down as an inference rule, assume that all formulas universally quantified:

$$\frac{F \to G \qquad G \to H}{F \to H}$$

Expressing $\to$ using $\vee$ and $\neg$:

$$\frac{\neg F \vee G \qquad \neg G \vee H}{\neg F \vee H}$$

Let G be a literal L, let $\neg F$ be equivalent to clause C_1 and H to clause C_2. We get:

$$\frac{C_1 \vee L \qquad \neg L \vee C_2}{C_1 \vee C_2}$$

resolution (simple, without unification)

Resolution with instantiation

We call literals L and $\neg L$ *complementary*.
If literals are not complementary, we may still be able to apply substitution rule to make them complementary:

$$\frac{\dfrac{C_1' \vee L'}{C_1 \vee L} \qquad \dfrac{\neg L'' \vee C_2''}{\neg L \vee C_2}}{C_1 \vee C_2}$$

To check whether we can find a substitution to make L' and L'' identical ($\equiv$), we can use (syntactic, first-order) **unification**, which has a linear-time algorithm.

(W) Unification (computer science)

We rename variables in two clauses so that they are disjoint, the instantiate them by computing *most general unifier* computation so that literals become complementary (one is an atomic formula and the other its negation).

Applying resolution

Apply resolution and instantiation to clauses in our example:

1. $R(x, s_1(x))$
2. $\neg R(x, y) \vee R(x, f(y, z))$
3. $P(x) \vee P(f(x, a))$
4. $\neg R(s_2, y) \vee \neg P(y)$

Applying resolution

Apply resolution and instantiation to clauses in our example:

1. $R(x, s_1(x))$
2. $\neg R(x, y) \vee R(x, f(y, z))$
3. $P(x) \vee P(f(x, a))$
4. $\neg R(s_2, y) \vee \neg P(y)$
5. (1,2): $R(x, f(s_1(x), z))$ $\quad y \rightsquigarrow s_1(x)$

Applying resolution

Apply resolution and instantiation to clauses in our example:

1. $R(x, s_1(x))$
2. $\neg R(x, y) \vee R(x, f(y, z))$
3. $P(x) \vee P(f(x, a))$
4. $\neg R(s_2, y) \vee \neg P(y)$
5. (1,2): $R(x, f(s_1(x), z))$ $\quad y \rightsquigarrow s_1(x)$
6. (1,4): $\neg P(s_1(s_2))$ $\quad x \rightsquigarrow s_2, y \rightsquigarrow s_1(x)$

Applying resolution

Apply resolution and instantiation to clauses in our example:

1. $R(x, s_1(x))$
2. $\neg R(x, y) \vee R(x, f(y, z))$
3. $P(x) \vee P(f(x, a))$
4. $\neg R(s_2, y) \vee \neg P(y)$
5. (1,2): $R(x, f(s_1(x), z))$ $\quad y \rightsquigarrow s_1(x)$
6. (1,4): $\neg P(s_1(s_2))$ $\quad x \rightsquigarrow s_2, y \rightsquigarrow s_1(x)$
7. (3,6): $P(f(s_1(s_2), a))$ $\quad x \rightsquigarrow s_1(s_2)$

Applying resolution

Apply resolution and instantiation to clauses in our example:

1. $R(x, s_1(x))$
2. $\neg R(x, y) \vee R(x, f(y, z))$
3. $P(x) \vee P(f(x, a))$
4. $\neg R(s_2, y) \vee \neg P(y)$
5. (1,2): $R(x, f(s_1(x), z))$ $\quad y \rightsquigarrow s_1(x)$
6. (1,4): $\neg P(s_1(s_2))$ $\quad x \rightsquigarrow s_2, y \rightsquigarrow s_1(x)$
7. (3,6): $P(f(s_1(s_2), a))$ $\quad x \rightsquigarrow s_1(s_2)$
8. (4,7): $\neg R(s_2, f(s_1(s_2), a))$ $\quad y \rightsquigarrow f(s_1(s_2), a)$

Applying resolution

Apply resolution and instantiation to clauses in our example:

1. $R(x, s_1(x))$
2. $\neg R(x, y) \vee R(x, f(y, z))$
3. $P(x) \vee P(f(x, a))$
4. $\neg R(s_2, y) \vee \neg P(y)$
5. (1,2): $R(x, f(s_1(x), z))$ $\quad y \rightsquigarrow s_1(x)$
6. (1,4): $\neg P(s_1(s_2))$ $\quad x \rightsquigarrow s_2, y \rightsquigarrow s_1(x)$
7. (3,6): $P(f(s_1(s_2), a))$ $\quad x \rightsquigarrow s_1(s_2)$
8. (4,7): $\neg R(s_2, f(s_1(s_2), a))$ $\quad y \rightsquigarrow f(s_1(s_2), a)$
9. (5,8): $\bot$ $\quad x \rightsquigarrow s_2$

Applying resolution

Apply resolution and instantiation to clauses in our example:

1. $R(x, s_1(x))$
2. $\neg R(x, y) \vee R(x, f(y, z))$
3. $P(x) \vee P(f(x, a))$
4. $\neg R(s_2, y) \vee \neg P(y)$
5. (1,2): $R(x, f(s_1(x), z))$ $\quad y \rightsquigarrow s_1(x)$
6. (1,4): $\neg P(s_1(s_2))$ $\quad x \rightsquigarrow s_2, y \rightsquigarrow s_1(x)$
7. (3,6): $P(f(s_1(s_2), a))$ $\quad x \rightsquigarrow s_1(s_2)$
8. (4,7): $\neg R(s_2, f(s_1(s_2), a))$ $\quad y \rightsquigarrow f(s_1(s_2), a)$
9. (5,8): $\bot$ $\quad x \rightsquigarrow s_2$

We can derive $\bot$ from the set of clauses.
Thus, the set of clauses is unsatisfiable. The formula we started from as well.

Conclusion of the procedure on this example

Thus, the formula before negation:

$$\begin{array}{l}\big[(\forall x.\exists y.\ R(x,y)) \wedge \\ (\forall x.\forall y.\ R(x,y) \rightarrow \forall z.\ R(x,f(y,z))) \wedge \\ (\forall x.\ P(x) \vee P(f(x,a)))\big] \\ \rightarrow \forall x.\exists y.\ R(x,y) \wedge P(y)\end{array}$$

was **valid**. We have proven that this formula holds in all possible models (no matter how small, large, or complex they might be).

Remarkably, resolution with instantiation is refutationally complete: if the formula is valid, then from the clauses equisatisfiable to its negation we will find a contradiction.