

Logical reasoning and programming

Lecture 7: First-Order Logic—Resolution, Equality, and Model Finding

Karel Chvalovský

CIIRC CTU

November 3, 2025

Our goal

We want to decide

$$\Gamma \models \varphi$$

in full FOL.

It is undecidable, but we can still use our favorite recipe

1. show that it is sufficient to deal only with a restricted class of formulae by presenting various transformations and

(=clauses)

2. use techniques that worked for less expressive systems

(=resolution)

to obtain a procedure that is useful.

Note that there exist also other approaches!

Instances

Lemma

Let φ be a clause and σ be a substitution, then $\forall\varphi \models \varphi\sigma$.

We say that $\varphi\sigma$ is an *instance* of φ . If an instance contains no variable, then we call it a *ground instance*.

Example

From $\forall X\forall Y(p(X) \vee \neg q(X, Y))$, for example, follows $p(a) \vee \neg q(a, f(Z))$ and $p(b) \vee \neg q(b, f(a))$ (a ground instance).

Herbrand models

We can restrict the types of interpretations (and ground instances) that have to be considered. Let Γ be a set of clauses.

Herbrand universe

The Herbrand universe of Γ , denoted $HU(\Gamma)$, is the set of all ground terms in the language of Γ . If Γ contains no constants, we add a fresh constant c to the language.

Herbrand base

The Herbrand base of Γ , denoted $HB(\Gamma)$, is the set of all ground atomic formulae in the language of Γ , where only terms from $HU(\Gamma)$ are allowed.

Herbrand interpretation

A Herbrand interpretation of Γ is a subset of $HB(\Gamma)$.

Herbrand model

A Herbrand model \mathcal{M} of Γ is a Herbrand interpretation of Γ s.t. $\mathcal{M} \models \Gamma$.

Example

$\Gamma = \{\{\neg p(X, Y), q(f(Y), X)\}\}$. $HU(\Gamma) = \{c, f(c), f(f(c)), \dots\}$ and $HB(\Gamma) = \{p(c, c), p(c, f(c)), \dots, q(f(c), c), \dots\}$. Note that we express what is true.

Herbrand's theorem

Theorem

Let Γ be a set of clauses. The following conditions are equivalent:

- 1. Γ is unsatisfiable,*
- 2. the set of all ground instances of Γ is unsatisfiable,*
- 3. a finite subset of the set of all ground instances of Γ is unsatisfiable.*

Note that Γ has a model iff it has a Herbrand model. However, we should note that clauses are quantifier-free. For example, a formula $p(c) \wedge \exists X \neg p(X)$ is clearly satisfiable, but has no Herbrand model; the Herbrand universe contains only c .

It is even possible to use so called Herbrand semantics, which is common in logic programming, instead of Tarskian semantics, check, for example, The Herbrand Manifesto.

Naïve approach

Herbrand's theorem provides a propositional criterion for unsatisfiability of a set of clauses Γ , because a ground atomic formula can be seen as a propositional atom (like in SMT).

Several early approaches (Gilmore; David and Putnam in 1960) work as follows

- ▶ generate ground instances and use propositional resolution,
- ▶ if it is propositionally satisfiable, then produce more instances (there is usually infinitely many of them) and repeat.

However, such an approach is generally very inefficient.

But variants of it are widely used, for example, in

- ▶ iProver,
 - ▶ EPR (effectively propositional, or Bernays–Schönfinkel–Ramsey class)—no function symbols and a quantifier prefix $\exists^* \forall^*$ hence $|D|$ is bounded by the number of constants occurring in the problem; decidable (NEXPTIME-complete).
- ▶ SMT,
- ▶ Answer Set Programming.

Lifting lemma

A technique to prove completeness theorems for the non-ground case using completeness for a ground instance.

For example, we want to satisfy two clauses

$$\{q(Y, f(X)), p(X, g(a, Y))\} \text{ and } \{\neg p(U, V), r(U, V)\}.$$

We want to represent infinitely many ground instances and possible resolution steps by a single non-ground instance

$$\frac{\{q(Y, f(X)), p(X, g(a, Y))\} \quad \{\neg p(U, V), r(U, V)\}}{\{q(Y, f(X)), r(X, g(a, Y))\}} \quad \{U \mapsto X, V \mapsto g(a, Y)\}$$

Use unification—we want to unify simultaneously X with U and $g(a, Y)$ with V . Here by a substitution $\{U \mapsto X, V \mapsto g(a, Y)\}$.

Remark

$\{q(Y, f(X)), p(X, g(a, Y))\}$ means $\forall X \forall Y (q(Y, f(X)) \vee p(X, g(a, Y)))$ and $\{\neg p(U, V), r(U, V)\}$ means $\forall U \forall V (\neg p(U, V) \vee r(U, V))$.

Unifiers

Let s and t be terms. A *unifier* of s and t is a substitution σ such that $s\sigma$ and $t\sigma$ are identical ($s\sigma = t\sigma$).

A unifier σ of s and t is said to be a *most general unifier* (or *mgu* for short), denoted $\sigma = mgu(s, t)$, if for any unifier θ of s and t there is a substitution η such that $\theta = \sigma\eta$ that is θ is a composition of σ and η .

We can easily extend our definitions to

- ▶ a (most general) unifier of a set of terms,
- ▶ a (most general) unifier of two atomic formulae,
- ▶ a (most general) unifier of a set of atomic formulae.

Example

$\varphi = p(X, g(a, Y))$, $\psi = p(U, V)$, $\sigma = \{X \mapsto a, U \mapsto a, V \mapsto g(a, Y)\}$, $\theta = \{U \mapsto X, V \mapsto g(a, Y)\}$, and $\eta = \{X \mapsto Q, U \mapsto Q, Y \mapsto R, V \mapsto g(a, R)\}$. σ is a unifier of φ and ψ , but $\sigma \neq mgu(\varphi, \psi)$. θ and η are $mgu(\varphi, \psi)$.

Unification algorithm

A set of equations $\{X_1 \doteq t_1, \dots, X_n \doteq t_n\}$ is in *solved form* if X_1, \dots, X_n are distinct variables that do not appear in t_1, \dots, t_n .

Given a finite set of pairs of terms $T = \{s_1 \doteq t_1, \dots, s_n \doteq t_n\}$. The following algorithm either produces a set of equations in solved form that defines an mgu σ such that $s_i\sigma = t_i\sigma$, for $1 \leq i \leq n$, or it fails. If it fails, then there is no unifier for the set.

- ▶ $S \sqcup \{u \doteq u\} \rightsquigarrow S$,
- ▶ $S \sqcup \{f(u_1, \dots, u_k) \doteq f(v_1, \dots, v_k)\} \rightsquigarrow S \cup \{u_1 \doteq v_1, \dots, u_k \doteq v_k\}$,
- ▶ $S \sqcup \{f(u_1, \dots, u_k) \doteq g(v_1, \dots, v_l)\} \rightsquigarrow \text{fail}$, if $f \neq g$ or $k \neq l$,
- ▶ $S \sqcup \{f(u_1, \dots, u_k) \doteq X\} \rightsquigarrow S \cup \{X \doteq f(u_1, \dots, u_k)\}$,
- ▶ $S \sqcup \{X \doteq u\} \rightsquigarrow S\{X \mapsto u\} \cup \{X \doteq u\}$, if $X \notin u$ and $X \in S$,
- ▶ $S \sqcup \{X \doteq u\} \rightsquigarrow \text{fail}$, if $X \in u$,

where u, u_j, v_j are terms and S is a finite set of pairs of terms. Moreover, $S\{X \mapsto u\}$ means that we substitute u for all occurrences of X in S . $S \sqcup U$ means disjoint union ($S \cup U$ where $S \cap U = \emptyset$) and \in means appears in.

Properties of the unification algorithm

Termination

The algorithm always terminates. Assume the following triplet

1. the number of distinct variables that occur more than once in T ,
2. the number of function (and constant) symbols that occur on the left hand sides in T ,
3. the number of pairs in T .

Clearly, under the lexicographic order, the triple decreases after an application of any rule.

It produces an mgu

A routine induction proof on the number of steps of the algorithm proves that

- ▶ it finds an mgu, if there is a unifier of the set,
- ▶ it fails, if there is no unifier of the set.

Resolution—idea

We want to show that a set of clauses (implicitly universally quantified) is (un)satisfiable. For example, if we want to satisfy

$$\{\{\neg p(X), q(X), r(X)\}, \{p(a), p(b)\}, \{\neg q(Y)\}, \{\neg r(a)\}, \{\neg r(b)\}\},$$

then satisfying the first two clauses means that also clauses

$$\frac{\{\neg p(X), q(X), r(X)\} \quad \{p(a), p(b)\}}{\{q(a), r(a), p(b)\}} \{X \mapsto a\}$$

$$\frac{\{\neg p(X), q(X), r(X)\} \quad \{p(a), p(b)\}}{\{q(b), r(b), p(a)\}} \{X \mapsto b\}$$

must be satisfied. Repeating this process for all clauses, we finally get that \square , the empty clause, must be satisfied; a contradiction.

Remark

$\{\neg p(X), q(X), r(X)\}$ means $\forall X(\neg p(X) \vee q(X) \vee r(X))$ and $\{\neg q(Y)\}$ means $\forall Y(\neg q(Y))$.

Resolution

Let $l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}$ be literals and p and q be atomic formulae.

$$\frac{\{l_1, \dots, l_m, p\} \quad \{\neg q, l_{m+1}, \dots, l_{m+n}\}}{\{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}\}\sigma}$$

where $\sigma = mgu(p, q)$ and $\{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}\}\sigma$ is equal to $\{l_1\sigma, \dots, l_m\sigma, l_{m+1}\sigma, \dots, l_{m+n}\sigma\}$. The clause $\{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}\}\sigma$ produced by the (binary) resolution rule is called the *resolvent* of the two *input* clauses. We assume that **the input clauses do not share variables (renaming away)**.

Theorem (correctness)

$\{l_1, \dots, l_m, p\}, \{\neg q, l_{m+1}, \dots, l_{m+n}\} \models \{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}\}\sigma$, where $\sigma = mgu(p, q)$.

Hence the resolution rule preserves satisfiability.

Factoring

We need to add the factoring rule. Let $l_1, \dots, l_m, l_{m+1}, l, k$ be literals.

$$\frac{\{l_1, \dots, l_m, l, k\}}{\{l_1, \dots, l_m, l\}\sigma}$$

where $\sigma = mgu(l, k)$. Note that l and k are either both positive, or both negative. Moreover, $\{l_1, \dots, l_m, l, k\} \models \{l_1, \dots, l_m, l\}\sigma$.

In propositional logic we avoided this problem completely by using sets (of clauses). However, it is still possible to combine both rules into just one rule.

Example

Using only the binary resolution rule, we cannot derive \square from clauses $\{p(X), p(Y)\}$ and $\{\neg p(U), \neg p(V)\}$.

Resolution calculus (in FOL without $=$)

The resolution calculus has no axioms and the only deduction rules are the binary resolution rule and the factoring.

Resolution proof

A (resolution) proof of clause φ from clauses ψ_1, \dots, ψ_n , in FOL without equality, is a finite sequence of clauses χ_1, \dots, χ_m such that

- ▶ every χ_i is
 - ▶ among ψ_1, \dots, ψ_n , or
 - ▶ derived by the binary resolution rule from input clauses χ_j and χ_k , for $1 \leq j < k < i \leq m$, or
 - ▶ derived by the factoring rule from an input clause χ_j , for $1 \leq j < i \leq m$.
- ▶ $\varphi = \chi_m$.

We say that a clause φ is *provable* (derivable) from a set of clauses $\{\psi_1, \dots, \psi_n\}$, we write $\{\psi_1, \dots, \psi_n\} \vdash \varphi$, if there is a proof of φ from ψ_1, \dots, ψ_n .

Resolution proof

Example

We prove \square from a set of clauses

$$\{\{\neg p(X), q(X), r(X)\}, \{p(a), p(b)\}, \{\neg q(Y)\}, \{\neg r(a)\}, \{\neg r(b)\}\}.$$

$$\frac{\frac{\frac{\neg p(X), \textcolor{red}{q(X)}, r(X)}{\neg p(X), \textcolor{red}{r(X)}} \quad \neg q(Y)}{\textcolor{red}{\neg p(a)} \quad \textcolor{red}{p(a)}, p(b)}}{\textcolor{red}{p(b)}} \quad \frac{\frac{\neg p(X), \textcolor{red}{q(X)}, r(X)}{\neg p(X), \textcolor{red}{r(X)}} \quad \neg q(Y)}{\textcolor{red}{\neg p(b)}}$$

\square

Strictly speaking the presented derivation is not a sequence, but it is easy to produce a sequence from it. For example, $\{\neg p(X), r(X)\}$ is derived only once in the sequence.

More resolvents

Unlike in propositional logic, it is possible to resolve two clauses in multiple ways and still obtain useful resolvents.

Example

From $\{p(a), p(b)\}$ and $\{\neg p(X), q(X)\}$ we can derive two non-tautological clauses

$$\frac{\{p(a), p(b)\} \quad \{\neg p(X), q(X)\}}{\{p(b), q(a)\}}$$

$$\frac{\{p(a), p(b)\} \quad \{\neg p(X), q(X)\}}{\{p(a), q(b)\}}$$

Completeness of resolution calculus (in FOL without $=$)

It is not true that we can derive every valid formula in the resolution calculus, e.g., from the empty set we derive nothing. However, it is so called *refutationally complete*.

Theorem (completeness)

Let Γ be a set of clauses. If Γ is unsatisfiable, then $\Gamma \vdash \square$.

Note that from the correctness theorem we already know the converse implication.

Theorem

Let Γ be a set of clauses. If $\Gamma \vdash \square$, then Γ is unsatisfiable.

Subsumption

A clause φ *subsumes* a clause ψ , denoted $\varphi \sqsubseteq \psi$, if there is a substitution σ such that $\varphi\sigma \subseteq \psi$.

If $\varphi \sqsubseteq \psi$, then $\varphi \models \psi$.

Let Γ and Δ be sets of clauses. We write $\Gamma \sqsubseteq \Delta$ if for every clause $\psi \in \Delta$ exists a clause $\varphi \in \Gamma$ such that $\varphi \sqsubseteq \psi$.

Lemma

If $\Delta \vdash \square$ and $\Gamma \sqsubseteq \Delta$, then $\Gamma \vdash \square$. Moreover, for every proof of \square from Δ , there exists a proof of \square from Γ that is not longer.

Example

$\{p(X)\} \sqsubseteq \{p(f(a))\}$, $\{p(X)\} \sqsubseteq \{p(f(a)), p(b)\}$, $\{p(X)\} \sqsubseteq \{p(Y), q(Y)\}$, and $\{p(X), q(Y)\} \sqsubseteq \{p(Z), q(Z)\}$, but $\{p(Z), q(Z)\} \not\sqsubseteq \{p(X), q(Y)\}$.

Subsumption example

Assume we have the following resolution refutation

$$\frac{\frac{\frac{p(f(X)), q(X, Y), r(X)}{q(f(c), Y), r(f(c))} \quad \neg p(f(f(c)))}{r(f(c))} \quad \neg q(U, V)}{\neg r(f(c))} \quad \square$$

Then after deriving $\{p(Y), r(Z)\}$, we can simplify the previous proof into

$$\frac{\frac{p(Y), r(Z)}{r(Z)} \quad \neg p(f(f(c)))}{\neg r(f(c))} \quad \square$$

thanks to $\{p(Y), r(Z)\} \sqsubseteq \{p(f(X)), q(X, Y), r(X)\}$.

Forward and backward subsumptions

Forward subsumption

If we derive a clause ψ and we already have a clause φ such that $\varphi \sqsubseteq \psi$, then we can remove ψ , because φ is stronger.

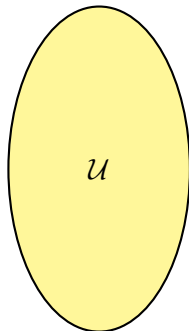
Backward subsumption

If we derive a clause φ and we already have a clause ψ such that $\varphi \sqsubseteq \psi$, then we can remove ψ , because φ is stronger. We can remove all such ψ s.

Saturation procedure (a simplified picture)

A way how to organize proof search (also called ANL loop or the given-clause algorithm). We split the derived clauses into two sets

- ▶ processed clauses \mathcal{P} and
- ▶ unprocessed clauses \mathcal{U} .

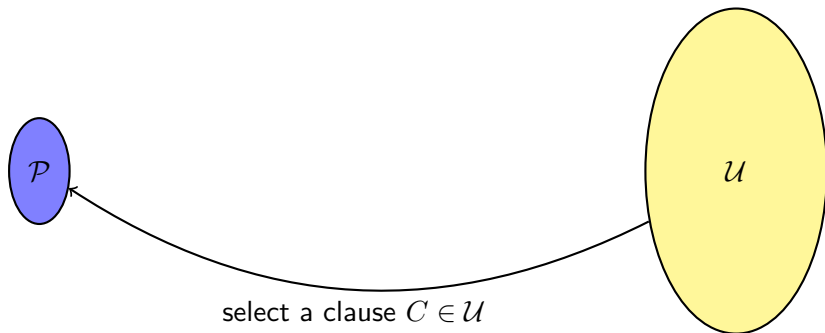


We start with the input clauses in \mathcal{U} and $\mathcal{P} = \emptyset$.

Saturation procedure (a simplified picture)

A way how to organize proof search (also called ANL loop or the given-clause algorithm). We split the derived clauses into two sets

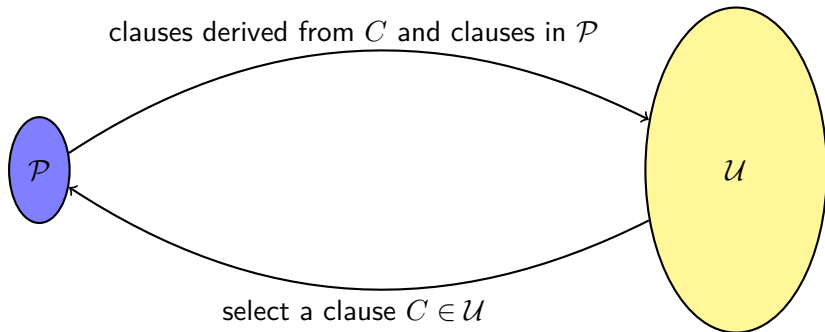
- ▶ processed clauses \mathcal{P} and
- ▶ unprocessed clauses \mathcal{U} .



Saturation procedure (a simplified picture)

A way how to organize proof search (also called ANL loop or the given-clause algorithm). We split the derived clauses into two sets

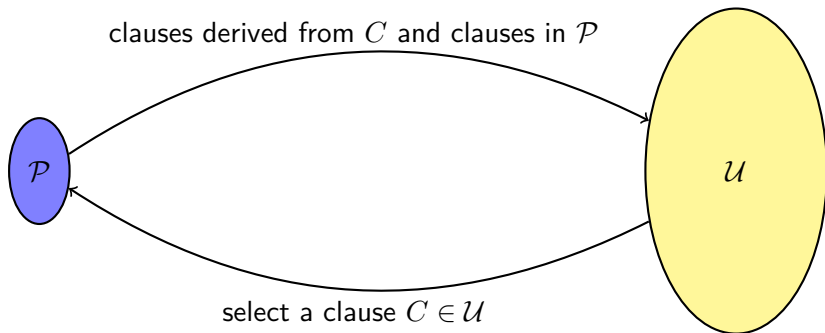
- ▶ processed clauses \mathcal{P} and
- ▶ unprocessed clauses \mathcal{U} .



Saturation procedure (a simplified picture)

A way how to organize proof search (also called ANL loop or the given-clause algorithm). We split the derived clauses into two sets

- ▶ processed clauses \mathcal{P} and
- ▶ unprocessed clauses \mathcal{U} .

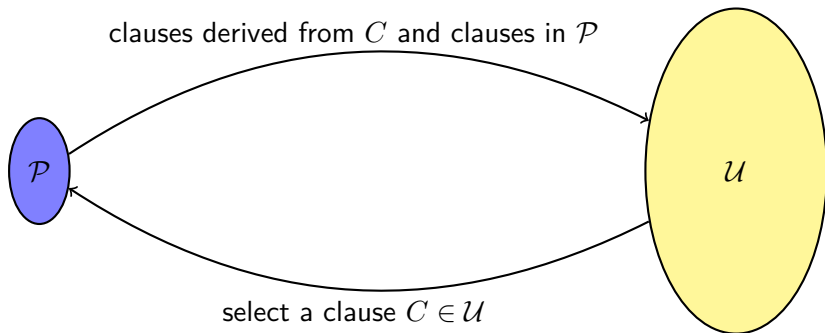


“All” the important consequences of clauses in \mathcal{P} are in $\mathcal{P} \cup \mathcal{U}$.

Saturation procedure (a simplified picture)

A way how to organize proof search (also called ANL loop or the given-clause algorithm). We split the derived clauses into two sets

- ▶ processed clauses \mathcal{P} and
- ▶ unprocessed clauses \mathcal{U} .



Outcomes: \square (UNSAT), $U = \emptyset$ (SAT), or resources exhausted.

Our situation

We have

- ▶ the resolution calculus for FOL without equality (resolution + factoring),
- ▶ simplifications
 - ▶ pure literal deletion — clauses containing a literal that occurs only positively or negatively can be removed
 - ▶ tautology eliminations — tautologous clauses can be removed
 - ▶ subsumptions

However, the resolution calculus can produce many clauses that are useless or produced in multiple ways. Hence we would like to guide our proof search.

We can restrict our proof search in many ways, for example:

- ▶ select only some literals,
- ▶ select only some clauses,
- ▶ use different term orderings.

Ordered resolution

We know that we can impose an ordering on propositional atoms and still be complete. Hence we can do a similar thing for ground instances, however, for non-ground instances it is much more involved, see later.

$$\frac{\{l_1, \dots, l_m, p\} \quad \{\neg q, l_{m+1}, \dots, l_{m+n}\}}{\{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}\}\sigma}$$

where $\sigma = mgu(p, q)$ and p and $\neg q$ are maximal in their respective clauses.

$$\frac{\{l_1, \dots, l_m, l, k\}}{\{l_1, \dots, l_m, l\}\sigma}$$

where $\sigma = mgu(l, k)$ and l is maximal in the parent clause.

Literal selection function

We can overrule ordering restrictions in individual clauses by selecting non-maximal (negative) literals in clauses. A selection function selects a subset of literals and we compute inferences involving only selected literals (or maximal if no selected).

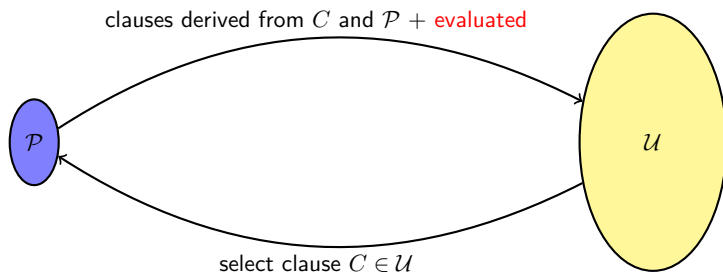
How to select a (given) clause?

For example, all clauses are evaluated when added to \mathcal{U} and we pick clauses, in a given ratio, say 1:10, by

age prefer clauses with a smaller derivational depth, and

weight prefer (shorter) clauses containing fewer symbols.

Small clauses are nice, but selecting only them leads to incompleteness!



Positive resolution

It is possible to restrict resolution in such a way that at least one parent is always a positive clause (=contains no negative literal).

Let Γ be a set of clauses. We split it into the positive part Γ^+ and the rest $\Gamma' = \Gamma \setminus \Gamma^+$.

- ▶ If $\Gamma^+ = \emptyset$, then making all atomic predicates false satisfies $\Gamma' = \Gamma$.
- ▶ If $\Gamma' = \emptyset$, then making all atomic predicates true satisfies $\Gamma^+ = \Gamma$.

Proof is done for the ground case and we use the lifting argument.

Semantic resolution

A generalization of positive resolution, we have a model \mathcal{M} and we always select at least one clause that is not valid in \mathcal{M} .

Set of support

We assume that some clauses must occur in any refutation. For example, if we want to prove $\Gamma \vdash \varphi$, we sometimes assume that Γ is consistent (this is not necessary!) and $\neg\varphi$ is needed to refute $\Gamma \cup \{\neg\varphi\}$. Hence we only allow derivations where a clause obtained from $\neg\varphi$ is involved.

A special case is the input resolution strategy, where at least one clause involved in resolution is always from the input. It is complete for Horn clauses (Prolog) and it is linear; we can see a proof as a linear sequence where every clause is obtained from the previous one.

Clause splitting

If we can split a clause into two (or more) independent parts, which do not share variables, then we can do that and solve two (simpler) cases instead.

If we want to show that

$$\Gamma \cup \{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}\} \vdash \square,$$

where $\{l_1, \dots, l_m\}$ and $\{l_{m+1}, \dots, l_{m+n}\}$ do not share variables, then we can do that by showing both

$$\Gamma \cup \{l_1, \dots, l_m\} \vdash \square \quad \text{and} \quad \Gamma \cup \{l_{m+1}, \dots, l_{m+n}\} \vdash \square.$$

It is possible to go a step further and encode more complex splits into components as various propositional cases. Hence we can employ a SAT solver to help us navigate through these cases. It is called AVATAR in Vampire. Moreover, if theories are involved, we can use an SMT solver.

How to select correct parameters?

See Vampire's CASC mode at GitHub. It is a competition mode.

FOL with equality

We have intentionally ignored the equality predicate. It is the most common predicate used in FOL and hence it deserves a special treatment. There are two main approaches how to deal with it, equality is either

- ▶ a binary predicate and its meaning is given by axioms added to our problem, or
- ▶ a logical symbol interpreted by the identity relation on the domain.

Note that we have already discussed equality in SMT.

Equalities are general

It is possible to express every FOL problem as a problem using only equalities by the following transformation:

$$\begin{aligned} p(t_1, \dots, t_n) &\text{ becomes } f_p(t_1, \dots, t_n) = \top, \\ \neg p(t_1, \dots, t_n) &\text{ becomes } \neg f_p(t_1, \dots, t_n) = \top, \end{aligned}$$

where \top is a new constant and f_p is a new functional symbol for every predicate p in our original language. Note that f_p and \top are not valid arguments of other terms.

Example

$\{p(X), \neg q(X, g(X, Y))\}$ becomes $\{f_p(X) = \top, \neg f_q(X, g(X, Y)) = \top\}$.

Naïve handling of equality

We can handle equality as a new binary predicate symbol \sim defined by the following axioms:

- ▶ $\forall X (X \sim X),$
- ▶ $\forall X \forall Y (X \sim Y \rightarrow Y \sim X),$
- ▶ $\forall X \forall Y \forall Z (X \sim Y \wedge Y \sim Z \rightarrow X \sim Z),$
- ▶ $\forall X_1 \dots \forall X_n \forall Y_1 \dots \forall Y_n (X_1 \sim Y_1 \wedge \dots \wedge X_n \sim Y_n \rightarrow$
 $f(X_1, \dots, X_n) \sim f(Y_1, \dots, Y_n)),$
- ▶ $\forall X_1 \dots \forall X_m \forall Y_1 \dots \forall Y_m (X_1 \sim Y_1 \wedge \dots \wedge X_m \sim Y_m \rightarrow$
 $(p(X_1, \dots, X_m) \rightarrow p(Y_1, \dots, Y_m))).$

for every n -ary function symbol f and m -ary predicate symbol p .

However, this is rarely a feasible approach, mainly thanks to the congruence axioms (for function and predicate symbols) and transitivity.

Axiomatic vs. direct approach

We say that a model is normal if the equality predicate is interpreted as the identity relation on its domain.

Example

We can define $X \sim Y$ over \mathbb{Z} by $X \equiv Y \pmod{n}$ and it clearly satisfies the previous axioms, but the models are non-normal.

In fact, it is impossible to force that all models are normal just by using axioms. For any $\mathcal{M} = (D, i)$, we may add a fresh constant c that will be interpreted exactly as a constant $d \in D$ and we get a new non-normal model.

Theorem

Any set of formulae Γ has a normal model iff Γ has a model satisfying the previous equality axioms.

Proof.

We get a normal model by partitioning the domain into equivalence classes, $[a] = \{b \mid b \sim a\}$, and use them as the new domain.



Paramodulation

We say $s \doteq t$ if $s = t$ or $t = s$, because order will be important for us later on.

Let $l, l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}$ be literals and s, s' and t be terms. Moreover, $l[s']$ means that the literal l contains s'

$$\frac{\{l_1, \dots, l_m, s \doteq t\} \quad \{l[s'], l_{m+1}, \dots, l_{m+n}\}}{\{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}, l[t]\}\sigma}$$

where $\sigma = mgu(s, s')$ and $l[t]$ is the result of replacing an occurrence of s' in $l[s']$ by t . We assume that the input clauses do not share variables (renaming away).

The clause $\{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}, l[t]\}\sigma$ produced by the paramodulation rule is sometimes called the *paramodulant*.

Example

From $\{f(X) = g(X, e)\}$ and $\{p(h(f(0), g(Y, e)), 2)\}$ we can derive $\{p(h(g(0, e), g(Y, e)), 2)\}$ and $\{p(h(f(0), f(Y), 2)\}$ as paramodulants.

Reflexivity resolution

Clearly, we cannot refute

$$\{\neg X = X\}$$

by the paramodulation rule. Hence we have to add also a rule for such cases. Let l_1, \dots, l_m be literals and s and t be terms, then the reflexivity resolution rule is

$$\frac{\{l_1, \dots, l_m, \neg s = t\}}{\{l_1, \dots, l_m\}\sigma}$$

where $\sigma = mgu(s, t)$.

Completeness of paramodulation

We define $\Gamma \vdash_{=} \varphi$ similarly to \vdash , but, moreover, we can use the paramodulation rule and the reflexivity resolution rule.

Theorem

Let Γ be a set of clauses in the FOL language with equality. Γ is unsatisfiable iff $\Gamma \vdash_{=} \square$.

It is possible to produce various restrictions, for example, it is *never necessary to replace a variable by a more complex term* using paramodulation. Loosely speaking, the aim of paramodulation is to make things unifiable, by changing a variable into a complex term we do not improve on this.

However, the most important modification is if we impose orderings on equalities.

Example on equalities (word problem for group theory)

Assume we have axioms

$$1 \cdot X = X$$

$$X^{-1} \cdot X = 1$$

$$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$$

and we want to know whether $X \cdot Y = Y \cdot X^{-1}$ follows from them?

Example on equalities (word problem for group theory)

Assume we have axioms

$$1 \cdot X = X$$

$$X^{-1} \cdot X = 1$$

$$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$$

and we want to know whether $X \cdot Y = Y \cdot X^{-1}$ follows from them?

It would be nice to direct axioms, say left to right (or from bigger to smaller), and use them only in this one direction, where two terms are equal if they reduce to the same term (irreducible word).

Example on equalities (word problem for group theory)

Assume we have axioms

$$1 \cdot X = X$$

$$X^{-1} \cdot X = 1$$

$$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$$

and we want to know whether $X \cdot Y = Y \cdot X^{-1}$ follows from them?

It would be nice to direct axioms, say left to right (or from bigger to smaller), and use them only in this one direction, where two terms are equal if they reduce to the same term (irreducible word).

However, this is not sufficient, because we know that, e.g., $Y \cdot (X \cdot X^{-1}) = Y$ holds. Hence we have to add more rules!

Solution

Using the Knuth–Bendix completion we produce the following set of directed rewriting rules (also called reduction rules):

$$\begin{aligned}1 \cdot X &\succ X \\X^{-1} \cdot X &\succ 1 \\(X \cdot Y) \cdot Z &\succ X \cdot (Y \cdot Z) \\X^{-1} \cdot (X \cdot Y) &\succ Y \\X \cdot 1 &\succ X \\1^{-1} &\succ 1 \\X^{-1-1} &\succ X \\X \cdot X^{-1} &\succ 1 \\X \cdot (X^{-1} \cdot Y) &\succ Y \\(X \cdot Y)^{-1} &\succ Y^{-1} \cdot X^{-1}\end{aligned}$$

Note that the Knuth–Bendix completion may fail.

Term orderings

We say that a binary relation \rightsquigarrow is a *rewrite relation*, if it is

- ▶ stable under contexts: $s \rightsquigarrow t$ implies $u[s] \rightsquigarrow u[t]$, and
- ▶ stable under substitutions: $s \rightsquigarrow t$ implies $s\sigma \rightsquigarrow t\sigma$

where s , t , and u are terms and σ is a substitution.

A *reduction ordering*, denoted \sqsubset , is a rewrite relation where \rightsquigarrow is irreflexive, transitive, and well-founded, i.e., there is no infinite strictly-descending sequence $s_1 \sqsubset s_2 \sqsubset \dots$.

A *simplification ordering*, denoted \succ , is a reduction ordering where a term is larger than all its proper subterms, i.e., $u[s] \succ s$ if $u[s] \neq s$.

Example

$f(X)$ and $g(Y)$ are \sqsubset -incomparable. Let $f(X) \sqsubset g(Y)$ and $\sigma = \{Y \mapsto f(X)\}$, then $f(X) \sqsubset g(f(X))$, and hence $g(f(X)) \sqsubset g(g(f(X)))$, \dots . Therefore, it is possible that all (non-ground) terms and literals are maximal under a \sqsubset .

Useful orderings

Very popular simplification orderings are:

KBO (Knuth–Bendix Ordering)

- ▶ uses function symbols weights and precedence to break ties
- ▶ produces syntactically smaller terms and is more efficient

LPO (Lexicographic Path Ordering)

- ▶ uses function symbols precedence and lexicographic decompositions to break ties
- ▶ produces better directions in many cases, e.g., for distributivity

For further details see for example Baader and Nipkow 1998.

Note that the EQP prover used orderings to prove the Robbins conjecture—are the algebras satisfying a given set of axioms exactly Boolean algebras.

Superposition

Although paramodulation rule

$$\frac{\{l_1, \dots, l_m, s \dot{=} t\} \quad \{l[s'], l_{m+1}, \dots, l_{m+n}\}}{\{l_1, \dots, l_m, l_{m+1}, \dots, l_{m+n}, l[t]\}\sigma}$$

where $\sigma = mgu(s, s')$, is more efficient than the naïve approach, it has to be further improved to be practical.

We want

- ▶ to use only maximal literals (ordered resolution),
- ▶ to use only maximal sides of literals (completion),
- ▶ s' not to be a variable

and the superposition calculus satisfies all this.

It is used in almost all state-of-the-art automated theorem provers.

How do we show that a formula is not provable?

We have seen several methods that can be used to prove a formula φ from a set of formulae Γ and hence $\Gamma \models \varphi$. However, can we show that

$$\Gamma \not\models \varphi$$

by these methods? Sometimes we can, but it is quite rare, e.g., if we obtain a saturated set.

Note that $\Gamma \not\models \varphi$ is not equivalent to $\Gamma \models \neg\varphi$! For example, $\not\models p(a)$ and $\not\models \neg p(a)$.

A general method is to provide a counterexample. A model of Γ where φ is false, for simplicity assume that φ is a closed formula.

How do we find a counterexample?

We have to check all possible models.

Finite models

For a finite language and a given size of domain, it is possible to check all possible models exhaustively (up to trivial isomorphisms).

Infinite models

Clearly, there are many sets of formulae with only infinite models, for example,

$$\begin{aligned}\forall X \neg (X < X), \\ \forall X \forall Y \forall Z (X < Y \wedge Y < Z \rightarrow X < Z), \\ \forall X \exists Y (X < Y).\end{aligned}$$

However, the problem how to generate useful infinite models is widely open. Moreover, the necessity to consider infinite models makes first-order logic algorithmically undecidable (precisely semi-decidable). Fortunately, for many problems finite counterexamples are sufficient.

MACE-style approach

We attempt to generate a finite counterexample iteratively. We try to produce a model of size 1, 2, 3, ...

The main idea is to produce a grounding of the problem assuming a given cardinality of our model and encode such a grounding as a SAT problem. Using a clever encoding we can significantly reduce the search space; no need to go through all possible models of the given size.

We present some basic techniques used in a model finder called Paradox, see Claessen and Sörensson 2003.

Our example

There is a counterexample for the problem that from

$$\begin{aligned}\forall X(e \cdot X &= X), \\ \forall X(X^{-1} \cdot X &= e), \\ \forall X \forall Y \forall Z(X \cdot (Y \cdot Z) &= (X \cdot Y) \cdot Z)\end{aligned}$$

follows

$$\forall X(X \cdot (X \cdot X) = X).$$

Or equivalently.

Our example

There is a model for

$$\begin{aligned}\forall X(e \cdot X &= X), \\ \forall X(X^{-1} \cdot X &= e), \\ \forall X \forall Y \forall Z(X \cdot (Y \cdot Z) &= (X \cdot Y) \cdot Z), \\ \neg(a \cdot (a \cdot a) &= a).\end{aligned}$$

We have $\mathcal{M} = (D, i)$, where $D = \{1, 2, 3\}$, $i(e) = 1$, $i(a) = 2$, and

$i(-^1)$	
1	1
2	3
3	2

$i(\cdot)$	1	2	3
1	1	2	3
2	2	3	1
3	3	1	2

You may check the following slides for the details, but we have seen some of these tricks before.

Propositional encoding

We are looking for a model $\mathcal{M} = (D, i)$ of a given cardinality, say n , that satisfies a set of clauses Γ . Assume without loss of generality that $D = \{1, \dots, n\}$. Hence it only remains to generate a function i .

We want to describe i using propositional variables (atoms). For every k -ary

- ▶ predicate symbol p in Γ , there is a prop. variable for every

$$p(d_1, \dots, d_k)$$

where $d_1, \dots, d_k \in D$.

- ▶ function symbol f in Γ , there is a prop. variable for every

$$f(d_1, \dots, d_k) = d$$

where $d_1, \dots, d_k, d \in D$.

This is all we need to describe a model.

Our example

Note that our example is a bit confusing—we have only one predicate symbol ($=$), which is very special, because it has the fixed meaning. Still we have atoms for all

$$1 = 1, \quad 1 = 2, \quad 1 = 3, \quad 2 = 1, \quad \dots, \quad 3 = 2, \quad 3 = 3$$

We also have propositional variables for all

$$e = 1, \quad e = 2, \quad e = 3$$

$$a = 1, \quad a = 2, \quad a = 3$$

$$1^{-1} = 1, \quad 1^{-1} = 2, \quad 1^{-1} = 3, \quad 2^{-1} = 1, \quad \dots, \quad 3^{-1} = 3$$

$$1 \cdot 1 = 1, \quad 1 \cdot 1 = 2, \quad 1 \cdot 1 = 3, \quad 1 \cdot 2 = 1, \quad \dots, \quad 3 \cdot 3 = 3$$

Flattening

However, it is impossible to express complex terms like $X \cdot (Y \cdot Z)$ directly in our language. We can only express so called shallow literals:

- ▶ $p(X_1, \dots, X_k)$, or $\neg p(X_1, \dots, X_k)$,
- ▶ $f(X_1, \dots, X_l) = Y$, or $f(X_1, \dots, X_l) \neq Y$,
- ▶ $X = Y$.

Note that $s \neq t$ is a shortcut for $\neg s = t$.

We do not want $X \neq Y$, because we can transform a clause

$$\{X \neq Y, \varphi(X, Y)\}$$

into

$$\{\varphi(X, X)\}.$$

Note that a clause $\{X \neq Y\}$ is unsatisfiable.

Flattening complex terms

If we have a clause

$$\varphi(t),$$

it is equivalent to

$$\forall X (X = t \rightarrow \varphi(X)),$$

which is

$$X \neq t \vee \varphi(X)$$

where X is fresh in $\varphi(t)$. $\varphi(X)$ is produced from $\varphi(t)$ by replacing all (free) occurrences of t by X .

We can repeat this process as long as necessary.

Example

$$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z \rightsquigarrow$$

$$(X \cdot Y) \cdot Z \neq W \vee X \cdot (Y \cdot Z) = W \rightsquigarrow$$

$$X \cdot Y \neq V \vee V \cdot Z \neq W \vee X \cdot (Y \cdot Z) = W \rightsquigarrow$$

$$X \cdot Y \neq V \vee V \cdot Z \neq W \vee Y \cdot Z \neq U \vee X \cdot U = W.$$

Instantiating

For every flattened clause we create three sets of propositional clauses

1. instances — we generate all possible groundings, where we can immediately simplify all groundings containing $d_1 = d_2$ or $d_1 \neq d_2$, for $d_1, d_2 \in D$, based on whether it is true (discard the clause), or not (discard the literal)
2. function definitions — for each k -ary function f and $d, d' \in D$ such that $d \neq d'$, we add

$$\{f(d_1, \dots, d_k) \neq d, f(d_1, \dots, d_k) \neq d'\}$$

for every $d_1, \dots, d_k \in D$.

3. totality definitions — for each k -ary function f , we add

$$\{f(d_1, \dots, d_k) = 1, f(d_1, \dots, d_k) = 2, \dots, f(d_1, \dots, d_k) = n\}$$

for every $d_1, \dots, d_k \in D$.

Reducing the number of distinct variables

The number of instances is exponential in the number of distinct variables in a flattened clause.

Term definitions

It is possible to decrease the number of newly introduced variables during flattening for deep terms by using definitions based on constants. From $a \cdot (a \cdot a)$ we can obtain $a \cdot b$, where b is a fresh constant, and define $b = a \cdot a$. It is also possible to introduce definitions for non-ground terms and use definitions across clauses.

Clause splitting

If a clause can be split into parts, where each part contains less distinct variables than the whole clause, then we can decrease the number of distinct variables by introducing a new predicate.

Example

From $\{p(X, Y), q(Y, Z)\}$, we can produce $\{p(X, Y), r(Y)\}, \{\neg r(Y), q(Y, Z)\}$, where r is a fresh predicate.

Isomorphic models

We have $\mathcal{M} = (D, i)$, where $D = \{1, 2, 3\}$, $i(e) = 1$, $i(a) = 2$, and

$i(-^1)$	
1	1
2	3
3	2

$i(\cdot)$	1	2	3
1	1	2	3
2	2	3	1
3	3	1	2

Note that any permutation on elements of D produces an isomorphic model. It makes no sense to look for all of them.

Static symmetry reduction

It is possible to avoid many isomorphic models using the following symmetry reduction technique. If we start to build a model by interpreting a constant c_1 , then we can safely assign $i(c_1) = 1$, because no element of D has an assigned meaning. Hence we have

$$\{c_1 = 1\} \text{ instead of } \{c_1 = 1, c_1 = 2, \dots, c_1 = n\}.$$

Then we can assume that $i(c_2) \in \{1, 2\}$ and $i(c_3) \in \{1, 2, 3\}$, because it has to be interpreted by an element with a meaning, or the first fresh element (if available). However, if $i(c_2) = 1$, then $i(c_3) \in \{1, 2\}$. Or more generally

$$\{c_i \neq k, c_1 = k - 1, c_2 = k - 1, \dots, c_{i-1} = k - 1\}.$$

This can be used also for functions, however, we have to take into account the meaning assigned to elements of D by constants.

Example

Hence $i(e) = 1$ and $i(a) = 2$ in our example, although $i(a) = 3$ works as well.

Other techniques

It is possible to use other techniques like

- ▶ pre-processing in SAT—variable and clause elimination, which is incompatible with an incremental search
- ▶ finding bounds for $|D|$
 - ▶ look for cardinality axioms
 - ▶ EPR (effectively propositional), also called Bernays–Schönfinkel–Ramsey class—no function symbols and a quantifier prefix $\exists^*\forall^*$ hence $|D|$ is bounded by the number of constants occurring in the problem; decidable (NEXPTIME-complete)
- ▶ use sorts
 - ▶ some problems are expressed in a many-sorted language,
 - ▶ other problems can be reformulated in a many-sorted language, if we have parts that can be defined independently

Proving in first-order logic (quick summary)

We are interested in the problem $\Gamma \models \varphi$ in FOL, which is an algorithmically undecidable (precisely semi-decidable) problem. Still we can solve many instances by using the following methods:

- ▶ we classify formulae (skolemization, . . .),
- ▶ we extend the resolution calculus to FOL using unification,
- ▶ we add direct equality handling,
- ▶ we use term orderings.

The current most powerful solvers are based on superposition calculus and we also showed how to use SAT to find small counter-examples.

Note that all these methods are machine-oriented and not particularly suitable for humans (tableaux systems are better). We will use them in the next lecture on proof assistants.

TPTP and TSTP

The TPTP (Thousands of Problems for Theorem Provers) is a library of test problems for ATP systems. The TSTP (Thousands of Solutions from Theorem Provers) is a library of solutions to TPTP problems.

Language

Prolog like language both for input (problems) and output (solutions). For details see TPTP and TSTP Quick Guide.

TPTP example

```
fof(usa,axiom,( country(usa) )).
```

```
fof(country_big_city,axiom,( ! [C] : ( country(C)  
=> ( big_city(capital_of(C))  
    & beautiful(capital_of(C)) ) ) ) ).
```

```
fof(usa_capital_axiom,axiom,( ? [C] : ( city(C)  
    & C = capital_of(usa) ) ) ).
```

```
fof(crime_axiom,axiom,( ! [C] : ( big_city(C)  
=> has_crime(C) ) ) ).
```

```
fof(big_city_city,axiom,( ! [C] : ( big_city(C)  
=> city(C) ) ) ).
```

```
fof(some_beautiful_crime,conjecture,( ? [C] : ( city(C)  
    & beautiful(C) & has_crime(C) ) ) ).
```

TPTP roles (official definitions)

- ▶ axioms are accepted, without proof. There is no guarantee that the axioms of a problem are consistent.
- ▶ hypothesiss are assumed to be true for a particular problem, and are used like axioms.
- ▶ definitions are intended to define symbols. They are either universally quantified equations, or universally quantified equivalences with an atomic lefthand side. They can be treated like axioms.
- ▶ assumptions can be used like axioms, but must be discharged before a derivation is complete.
- ▶ lemmas and theorems have been proven from the axioms. They can be used like axioms in problems, and a problem containing a non-redundant lemma or theorem is ill-formed. They can also appear in derivations. theorems are more important than lemmas from the user perspective.
- ▶ conjectures are to be proven from the axiom(-like) formulae. A problem is solved only when all conjectures are proven.
- ▶ negated_conjectures are formed from negation of a conjecture (usually in a FOF to CNF conversion).
- ▶ plains have no specified user semantics.

Moreover, there are fi_domain, fi_functors, fi_predicates, type, and unknown roles.

System before TPTP

System before TPTP is an interface for preprocessing systems.






```
cnf(i_0_4,plain, ( capital_of(usa) = esk1_0 ) ).
cnf(i_0_1,plain, ( country(usa) ) ).
cnf(i_0_5,plain, ( city(esk1_0) ) ).
cnf(i_0_7,plain, ( city(X1) | ~ big_city(X1) ) ).
cnf(i_0_6,plain, ( has_crime(X1) | ~ big_city(X1) ) ).
cnf(i_0_3,plain,
    ( big_city(capital_of(X1)) | ~ country(X1) ) ).
cnf(i_0_2,plain,
    ( beautiful(capital_of(X1)) | ~ country(X1) ) ).
cnf(i_0_8,negated_conjecture,
    ( ~ beautiful(X1) | ~ city(X1) | ~ has_crime(X1) ) ).
```

System on TPTP

System on TPTP is an interface for solvers.

```
# Proof found!
# SZS status Theorem
# SZS output start CNFRefutation
fof(some_beautiful_crime, conjecture, ?[X1]:((city(X1)&beautiful(X1))&has_crime(X1)), file('/tmp/SystemOnTPTPFormReply111578/SOT_mQKyc4', crime_axiom)).
fof(crime_axiom, axiom, ![X1]:(big_city(X1)=>has_crime(X1)), file('/tmp/SystemOnTPTPFormReply111578/SOT_mQKyc4', crime_axiom)).
fof(country_big_city, axiom, ![X1]:(country(X1)=>(big_city(capital_of(X1))&beautiful(capital_of(X1)))), file('/tmp/SystemOnTPTPFormReply111578/SOT_mQKyc4', usa_capital_axiom)).
fof(usa_capital_axiom, axiom, ?[X1]:(city(X1)&X1=capital_of(usa)), file('/tmp/SystemOnTPTPFormReply111578/SOT_mQKyc4', usa_capital_axiom)).
fof(usa, axiom, country(usa), file('/tmp/SystemOnTPTPFormReply111578/SOT_mQKyc4', usa)).
fof(c_0_5, negated_conjecture, ~(?[X1]:((city(X1)&beautiful(X1))&has_crime(X1))), inference(assume_negation,[status(cth)], [some_beautiful_crime])).
fof(c_0_6, negated_conjecture, ![X6]:(~city(X6)|~beautiful(X6)|~has_crime(X6)), inference(variable_rename,[status(thm)], [inference(fof(c_0_5, negated_conjecture, ~(?[X1]:((city(X1)&beautiful(X1))&has_crime(X1))), inference(assume_negation,[status(cth)], [some_beautiful_crime]))])).
fof(c_0_7, plain, ![X4]:(~big_city(X4)|has_crime(X4)), inference(variable_rename,[status(thm)], [inference(fof_nnf,[status(thm)], [crime_axiom])])).
fof(c_0_8, plain, ![X2]:((big_city(capital_of(X2))|~country(X2))&(beautiful(capital_of(X2))|~country(X2))), inference(distribute,[status(thm)], [inference(fof(c_0_7, plain, ![X4]:(~big_city(X4)|has_crime(X4)), inference(variable_rename,[status(thm)], [inference(fof_nnf,[status(thm)], [crime_axiom])]))])).
fof(c_0_9, plain, (city(esk1_0)&esk1_0=capital_of(usa)), inference(skolemize,[status(esa)], [inference(variable_rename,[status(thm)], [usa_capital_axiom])])).
cnf(c_0_10, negated_conjecture, (~city(X1)|~beautiful(X1)|~has_crime(X1)), inference(split_conjunct,[status(thm)], [c_0_6])).
cnf(c_0_11, plain, (has_crime(X1)|~big_city(X1)), inference(split_conjunct,[status(thm)], [c_0_7])).
cnf(c_0_12, plain, (beautiful(capital_of(X1))|~country(X1)), inference(split_conjunct,[status(thm)], [c_0_8])).
cnf(c_0_13, plain, (esk1_0=capital_of(usa)), inference(split_conjunct,[status(thm)], [c_0_9])).
cnf(c_0_14, plain, (country(usa)), inference(split_conjunct,[status(thm)], [usa])).
cnf(c_0_15, plain, (big_city(capital_of(X1))|~country(X1)), inference(split_conjunct,[status(thm)], [c_0_8])).
cnf(c_0_16, negated_conjecture, (~city(X1)|~beautiful(X1)|~big_city(X1)), inference(spm,[status(thm)], [c_0_10, c_0_11])).
cnf(c_0_17, plain, (city(esk1_0)), inference(split_conjunct,[status(thm)], [c_0_9])).
cnf(c_0_18, plain, (beautiful(esk1_0)), inference(cn,[status(thm)], [inference(rw,[status(thm)], [inference(spm,[status(thm)], [c_0_12, c_0_13])])])).
cnf(c_0_19, plain, (big_city(esk1_0)), inference(cn,[status(thm)], [inference(rw,[status(thm)], [inference(spm,[status(thm)], [c_0_15, c_0_16])])])).
cnf(c_0_20, negated_conjecture, ($false), inference(cn,[status(thm)], [inference(rw,[status(thm)], [inference(rw,[status(thm)], [inference(spm,[status(thm)], [c_0_17, c_0_18])])])])])).
```

Bibliography I

-  Baader, Franz and Tobias Nipkow (1998). *Term Rewriting and All That*. Cambridge University Press.
-  Claessen, Koen and Niklas Sörensson (2003). "New Techniques that Improve MACE-style Finite Model Finding". In: *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*. Ed. by Peter Baumgartner and Chris Fermüller.
-  D'Agostino, Marcello et al., eds. (1999). *Handbook of Tableau Methods*. Springer Netherlands. DOI: 10.1007/978-94-017-1754-0.
-  Harrison, John (Mar. 2009). *Handbook of Practical Logic and Automated Reasoning*. New York: Cambridge University Press, p. 702. URL: <http://www.cambridge.org/9780521899574>.
-  Robinson, John Alan and Andrei Voronkov, eds. (2001). *Handbook of Automated Reasoning*. Vol. 1. Elsevier Science.

Tableaux systems

There are many other approaches used to prove formulae in FOL. For example, we have (semantic) tableaux. There are many simple implementations of tableaux in Prolog, e.g., `leanTAP` or `leanCoP`, available.

Tableaux systems are also popular in non-classical logics, because

- ▶ there is no need for special normal forms like CNF,
 - ▶ can be complicated, or
 - ▶ even impossible to obtain
- ▶ given a semantic meaning of a connective we can usually produce a rule (or rules) in a straightforward way.

Generally, they are relatively easy to produce, in most cases, and still suitable for automated theorem proving. However, the handling of equality is as tricky as in resolution (superposition).

Moreover, they are similar to other proof systems like natural deduction and sequent calculi.

Example

We want to prove $\forall X(\neg p(X)) \rightarrow (\neg p(a) \wedge \neg p(b))$. We can prove it by showing that $\forall X(\neg p(X))$ and $p(a) \vee p(b)$ are together unsatisfiable.

$$\begin{array}{c}
 \frac{\forall X(\neg p(X)) \wedge (p(a) \vee p(b))}{\forall X(\neg p(X))} (\wedge) \\
 \begin{array}{cc}
 \frac{p(a) \vee p(b)}{p(a)} (\vee) & \frac{p(a) \vee p(b)}{p(b)} (\vee) \\
 \frac{p(a)}{\neg p(X_1)} (\forall) & \frac{p(b)}{\neg p(X_2)} (\forall) \\
 \hline \hline \sigma_1 = \{X_1 \mapsto a\} & \hline \hline \sigma_2 = \{X_2 \mapsto b\}
 \end{array}
 \end{array}$$

Note that we have to use $\forall X(\neg p(X))$ twice.

```

prove((A,B),UnExp,Lits,FreeV,VarLim) :- !,
  prove(A,[B|UnExp],Lits,FreeV,VarLim).
prove((A;B),UnExp,Lits,FreeV,VarLim) :- !,
  prove(A,UnExp,Lits,FreeV,VarLim),
  prove(B,UnExp,Lits,FreeV,VarLim).
prove(all(X,Fml),UnExp,Lits,FreeV,VarLim) :- !,
  \+ length(FreeV,VarLim),
  copy_term((X,Fml,FreeV),(X1,Fml1,FreeV)),
  append(UnExp,[all(X,Fml)],UnExp1),
  prove(Fml1,UnExp1,Lits,[X1|FreeV],VarLim).
prove(Lit,_,[L|Lits],_,_) :-
  (Lit = -Neg; -Lit = Neg) ->
  (unify(Neg,L); prove(Lit,[],Lits,_,_)).
prove(Lit,[Next|UnExp],Lits,FreeV,VarLim) :-
  prove(Next,UnExp,[Lit|Lits],FreeV,VarLim).

```