Logical reasoning and programming
First-order resolution II.

Karel Chvalovský

CIIRC CTU
Our situation

We have

- the resolution calculus for FOL (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 clauses,
- select only some literals,
- use different term orderings.
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 $\Gamma$ be a set of clauses. We split it into the positive part $\Gamma^+$ and $\Gamma' = \Gamma \setminus \Gamma^+$. 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.
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.

\[
\begin{align*}
\{l_1, \ldots, l_m, p\} \quad & \quad \{\neg q, l_{m+1}, \ldots, l_{m+n}\} \\
\{l_1, \ldots, l_m, l_{m+1}, \ldots, l_{m+n}\} \sigma 
\end{align*}
\]

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

\[
\begin{align*}
\{l_1, \ldots, l_m, l, k\} \\
\{l_1, \ldots, l_m, l\} \sigma
\end{align*}
\]

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

Selection function

We can overrule ordering restrictions in individual clauses by selecting negative literals in clauses.
Saturation procedure

We have already seen a saturated set in the propositional case—we systematically process a set of clauses in such a way that if there is no clause to be processed, then it is impossible to derive \( \square \) from the original set.

The next slides are from the presentation by Stefan Schultz, the author of the E prover, at the SAT/SMT/AR Summer School 2018.

Note that it is sometimes useful to simplify also \( U \) and not to delay such simplifications as in E (DISCOUNT loop).
The Given-Clause Algorithm

We represent the proof state \( S \) by two sets of clauses:

- \( P \) holds the processed clauses (originally empty)
- \( U \) holds the unprocessed clauses (originally all clauses in \( S \))
The Given-Clause Algorithm

Aim: Move everything from $U$ to $P$

$P$ (processed clauses)

$U$ (unprocessed clauses)

Invariant: All generating inferences with premises from $P$ have been performed

Invariant: $P$ is interreduced

Clauses added to $U$ are simplified with respect to $P$
The Given-Clause Algorithm

- **Aim**: Move everything from $U$ to $P$

- **Invariants**:
  - All generating inferences with premises from $P$ have been performed.
  - $P$ is interreduced.
  - Clauses added to $U$ are simplified with respect to $P$.
The Given-Clause Algorithm

Aim: Move everything from $U$ to $P$

Invariant: All generating inferences with premises from $P$ have been performed
The Given-Clause Algorithm

- **Aim:** Move everything from \( U \) to \( P \)
- **Invariant:** All generating inferences with premises from \( P \) have been performed
- **Invariant:** \( P \) is interreduced

\[
P \quad (\text{processed clauses})
\]

\[
U \quad (\text{unprocessed clauses})
\]

- Generate
- Simplify
- \( g \)
- \( g = \square \) ?
- Simplifiable?
The Given-Clause Algorithm

- **Aim**: Move everything from $U$ to $P$
- **Invariant**: All generating inferences with premises from $P$ have been performed
- **Invariant**: $P$ is interreduced
- **Clauses added to $U$ are simplified with respect to $P$**

Diagram:
- $P$ (processed clauses)
- $U$ (unprocessed clauses)
- Generate
- Simplify
- $g$
- $g = \Box$
- Simplifiable?
- Cheap Simplify
The Given-Clause Loop in Fewer Words

while $U \neq \{\}$
  $g = \text{delete\_best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \square$
    SUCCESS, Proof found
  if $g$ is not subsumed by any clause in $P$ (or otherwise redundant w.r.t. $P$)
    $P = P \setminus \{c \in P \mid c \text{ subsumed by (or otherwise redundant w.r.t.) } g\}$
    $T = \{c \in P \mid c \text{ can be simplified with } g\}$
    $P = (P \setminus T) \cup \{g\}$
    $T = T \cup \text{generate}(g, P)$
  foreach $c \in T$
    $c = \text{cheap\_simplify}(c, P)$
    if $c$ is not trivial
      $U = U \cup \{c\}$
  SUCCESS, original $U$ is satisfiable
while $U \neq \emptyset$

$g = \text{delete\_best}(U)$

$g = \text{simplify}(g, P)$

if $g == \square$

SUCCESS, Proof found

if $g$ is not redundant w.r.t. $P$

$P = P \setminus \{c \in P \mid c \text{ redundant w.r.t. } g\}$

$T = \{c \in P \mid c \text{ simplifiable with } g\}$

$P = (P \setminus T) \cup \{g\}$

$T = T \cup \text{generate}(g, P)$

foreach $c \in T$

$c = \text{cheap\_simplify}(c, P)$

if $c$ is not trivial

$U = U \cup \{c\}$

SUCCESS, original $U$ is satisfiable
“You can’t handle the truth!”
Actual problem

Martin Suda’s PUZ001+1.p example.
Clause selection heuristics

We usually sort clauses by some criteria:

- **clause age** — the maximal age of its parents + 1 (breadth-first search)
- **clause weight** — different types of weighting (best-first search)
  - different types of symbols can have different weights (predicates, functions, variables)
  - symbols occurring in the conjecture can be prioritized

In E there are 5 priority queues and you can count different things, in Vampire (LRS) you have just age and weight. Otter’s approach is to combine the best-first and breadth-first search in a given ratio (e.g., 10:1).
Semantic resolution

Set of support
We assume that some clauses must occur in any refutation. For example, if we want to prove $\Gamma \vdash \varphi$, we can assume that $\Gamma$ is consistent and to refute $\Gamma \cup \{\neg \varphi\}$, it is necessary to use $\neg \varphi$. 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 was 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.
Watchlist

It is a set of clauses we feed into the prover that can be used for guiding the proof search (lemmata).

In E, whenever a clause is generated, it is tested against the watchlist by subsumption. Every clause on the watchlist that is subsumed by a generated clause is removed and we can use this fact in our strategy.
Clause splitting

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

If we want to refute $\Gamma \cup \{l_1, \ldots, l_m, l_{m+1}, \ldots, l_{m+n}\}$, where $\{l_1, \ldots, l_m\}$ and $\{l_{m+1}, \ldots, l_{m+n}\}$ do not share variables, then we can do that by refuting both $\Gamma \cup \{l_1, \ldots, l_m\}$ and $\Gamma \cup \{l_{m+1}, \ldots, l_{m+n}\}$.

It is possible to go a step further and assume that these splits are like propositional cases and we can solve them by a SAT solver. 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, where the standard timelimit is 300s.
Naïve handling of equality

Let $\approx$ be the equality symbol, we can handle it as a new binary predicate symbol $\sim$ defined by the following axioms:

- $\forall X (X \sim X)$,
- $\forall X \forall Y (X \sim Y \rightarrow Y \sim X)$,
- $\forall X \forall Y \forall Z (X \sim Y \land Y \sim Z \rightarrow X \sim Z)$,
- $\forall X_1 \ldots \forall X_n \forall Y_1 \ldots \forall Y_n (X_1 \sim Y_1 \land \cdots \land X_n \sim Y_n \rightarrow f(X_1, \ldots, X_n) \sim f(Y_1, \ldots, Y_n))$,
- $\forall X_1 \ldots \forall X_m \forall Y_1 \ldots \forall Y_m (X_1 \sim Y_1 \land \cdots \land X_m \sim Y_m \rightarrow (p(X_1, \ldots, X_m) \rightarrow p(Y_m, \ldots, Y_m)))$.

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

However, this is not a feasible approach mainly thanks to the axioms for function and predicate symbols and transitivity.
We say $s \approx t$ if $s \approx t$ or $t \approx s$.

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

$$\frac{\{l_1, \ldots, l_m, s \approx t\}}{\{l_1, \ldots, l_m, l_{m+1}, \ldots, l_{m+n}, l[t]\}} \sigma$$

where $\sigma = mgu(s, s')$ and $l[t]$ is the result of replacing an occurrence of $s'$ in $l[s']$ by $t$.

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

Clearly, we cannot refute

\[ \{ \neg X \approx X \} \]

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

\[
\begin{array}{c}
\{ l_1, \ldots, l_m, \neg s \approx t \} \\
\{ l_1, \ldots, l_m \} \sigma
\end{array}
\]

where \( \sigma = \text{mgu}(s, t) \).
Completeness of paramodulation

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

**Theorem**

*Let $\Gamma$ be a set of clauses in the FOL language with equality. $\Gamma$ is unsatisfiable iff $\Gamma \vdash \approx \Box$.*

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.
Assume we have axioms

\[
\begin{align*}
1 \cdot X &\approx X \\
X^{-1} \cdot X &\approx 1 \\
(X \cdot Y) \cdot Z &\approx X \cdot (Y \cdot Z)
\end{align*}
\]

and we want to know whether \(X \cdot Y \approx Y \cdot X^{-1}\) follows from them?
Example on equalities (word problem for group theory)

Assume we have axioms

\[
1 \cdot X \approx X
\]

\[
X^{-1} \cdot X \approx 1
\]

\[
(X \cdot Y) \cdot Z \approx X \cdot (Y \cdot Z)
\]

and we want to know whether \( X \cdot Y \approx Y \cdot X^{-1} \) follows from them?

It would be nice to direct axioms, say left to right, and use them only in this one direction.
Example on equalities (word problem for group theory)

Assume we have axioms

\[
1 \cdot X \approx X \\
X^{-1} \cdot X \approx 1 \\
(X \cdot Y) \cdot Z \approx X \cdot (Y \cdot Z)
\]

and we want to know whether \( X \cdot Y \approx Y \cdot X^{-1} \) follows from them?

It would be nice to direct axioms, say left to right, and use them only in this one direction.

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

Using the Knuth–Bendix completion we produce the following set of directed rewriting rules:

\[
\begin{align*}
1 \cdot X & \succ X \\
X^{-1} \cdot X & \succ 1 \\
(X \cdot Y) \cdot Z & \succ X \cdot (Y \cdot Z) \\
X^{-1} \cdot (X \cdot Y) & \succ Y \\
X \cdot 1 & \succ X \\
1^{-1} & \succ 1 \\
X^{-1}^{-1} & \succ X \\
X \cdot X^{-1} & \succ 1 \\
X \cdot (X^{-1} \cdot Y) & \succ Y \\
(X \cdot Y)^{-1} & \succ Y^{-1} \cdot X^{-1}
\end{align*}
\]

Note that the Knuth–Bendix completion may fail.
Equalities are general

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

\[ p(t_1, \ldots, t_n) \text{ becomes } f_p(t_1, \ldots, t_n) \approx \top, \]
\[ \neg p(t_1, \ldots, t_n) \text{ becomes } \neg f_p(t_1, \ldots, t_n) \approx \top, \]

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**

\[ \{ p(X), \neg q(X, g(X, Y)) \} \text{ becomes } \]
\[ \{ f_p(X) \approx \top, \neg f_q(X, g(X, Y)) \approx \top \}. \]
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 $s\delta \rightsquigarrow t\delta$

where $s$, $t$, and $u$ are terms and $\delta$ is a substitution.

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

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

Example

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

Very popular simplifications orderings are:

**KBO (Knuth–Bendix Ordering)**
- uses 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
Although paramodulation is more efficient than the naïve approach, it has to be further improved to be practical.

We want

- to use only maximal literals,
- to use only maximal sides of literals,
- $s'$ not to be a variable,
- no explicit equality axioms

and the superposition calculus satisfies all this.
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.
- **hypotheses** 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 is an interface for preprocessing systems.

\begin{verbatim}
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) )).
\end{verbatim}
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', some_beautiful_crime)).
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))&has_crime(X1)), file('/tmp/SystemOnTPTPFormReply111578/SOT_mQKyc4', country_big_city)).
fof(usa_capital_axiom, axiom, ![X1]:((city(X1)&X1=capital_of(usa))&hs_city(X1)&beautiful(X1)), 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)).
fof(c_0_6, negated_conjecture, ![X6]:(~city(X6)|~beautiful(X6)|~has_crime(X6)), inference(variable_rename)).
fof(c_0_7, plain, ![X4]:(~big_city(X4)|has_crime(X4)), inference(variable_rename)).
fof(c_0_8, plain, ![X2]:((big_city(capital_of(X2))|~country(X2))&(beautiful(capital_of(X2))|~country(X2))), inference(variable_rename)).
fof(c_0_9, plain, (city(esk1_0)&esk1_0=capital_of(usa)), inference(skolemize)).
fof(c_0_10, negated_conjecture, (~city(X1)|~beautiful(X1)|~has_crime(X1)), inference(variable_rename)).
fof(c_0_11, plain, (has_crime(X1)|~big_city(X1)), inference(variable_rename)).
fof(c_0_12, plain, (beautiful(capital_of(X1))|~country(X1)), inference(variable_rename)).
fof(c_0_13, plain, (esk1_0=capital_of(usa)), inference(variable_rename)).
fof(c_0_14, plain, (country(usa)), inference(variable_rename)).
fof(c_0_15, plain, (big_city(capital_of(X1))|~country(X1)), inference(variable_rename)).
fof(c_0_16, negated_conjecture, (~city(X1)|~beautiful(X1)|~big_city(X1)), inference(variable_rename)).
fof(c_0_17, plain, (city(esk1_0)), inference(variable_rename)).
fof(c_0_18, plain, (beautiful(esk1_0)), inference(variable_rename)).
fof(c_0_19, plain, (big_city(esk1_0)), inference(variable_rename)).
fof(c_0_20, negated_conjecture, ($false), inference(variable_rename)).