Logical reasoning and programming

First-Order Logic—Resolution

Karel Chvalovský

CIIRC CTU

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 quite 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 $HB(\Gamma) = \{p(c, c), p(c, f(c)), \dots, q(f(c), c), \dots\}$.

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 proposional, 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 ($\mathrm{NEXPTIME}$-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

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

Use unification!

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

Given a finite set of pairs of terms $T = \{ s_1 \doteq t_1, \ldots, s_n \doteq t_n \}$. 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.

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

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

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

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

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

where $\sigma = mgu(p, q)$ and $\{l_1, \ldots, l_m, l_{m+1}, \ldots, l_{m+n}\}\sigma$ is equal to $\{l_1\sigma, \ldots, l_m\sigma, l_{m+1}\sigma, \ldots, l_{m+n}\sigma\}$. The clause $\{l_1, \ldots, l_m, l_{m+1}, \ldots, l_{m+n}\}\sigma$ 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)

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

Hence the resolution rule preserves satisfiability.

Factoring

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

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

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

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 $\psi_1, \ldots, \psi_n$, in FOL without equality, is a finite sequence of clauses $\chi_1, \ldots, \chi_m$ such that

- every χ_i is
  - among $\psi_1, \ldots, \psi_n$, 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 $\{\psi_1, \ldots, \psi_n\}$, we write $\{\psi_1, \ldots, \psi_n\} \vdash \varphi$, if there is a proof of φ from $\psi_1, \ldots, \psi_n$.

Resolution proof

Example

We prove $\square$ from a set of clauses

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

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

$$\square$$

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 both $\{p(b), q(a)\}$ and $\{p(a), q(b)\}$.

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

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

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

$$\dfrac{\dfrac{p(Y), r(Z) \qquad \neg p(f(f(c)))}{r(Z)} \qquad \neg r(f(c))}{\square}$$

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 ψs.

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

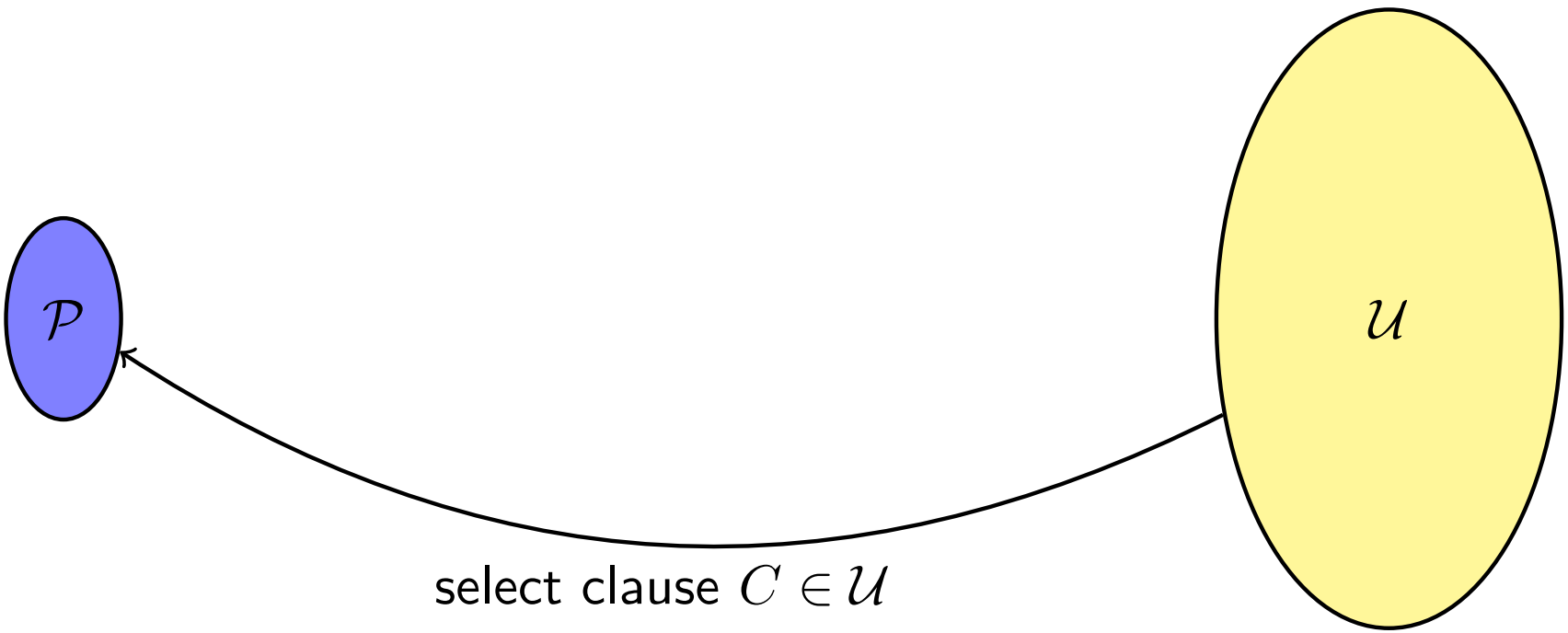
- processed clauses $\mathcal{P}$ and
- unprocessed clauses $\mathcal{U}$.

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

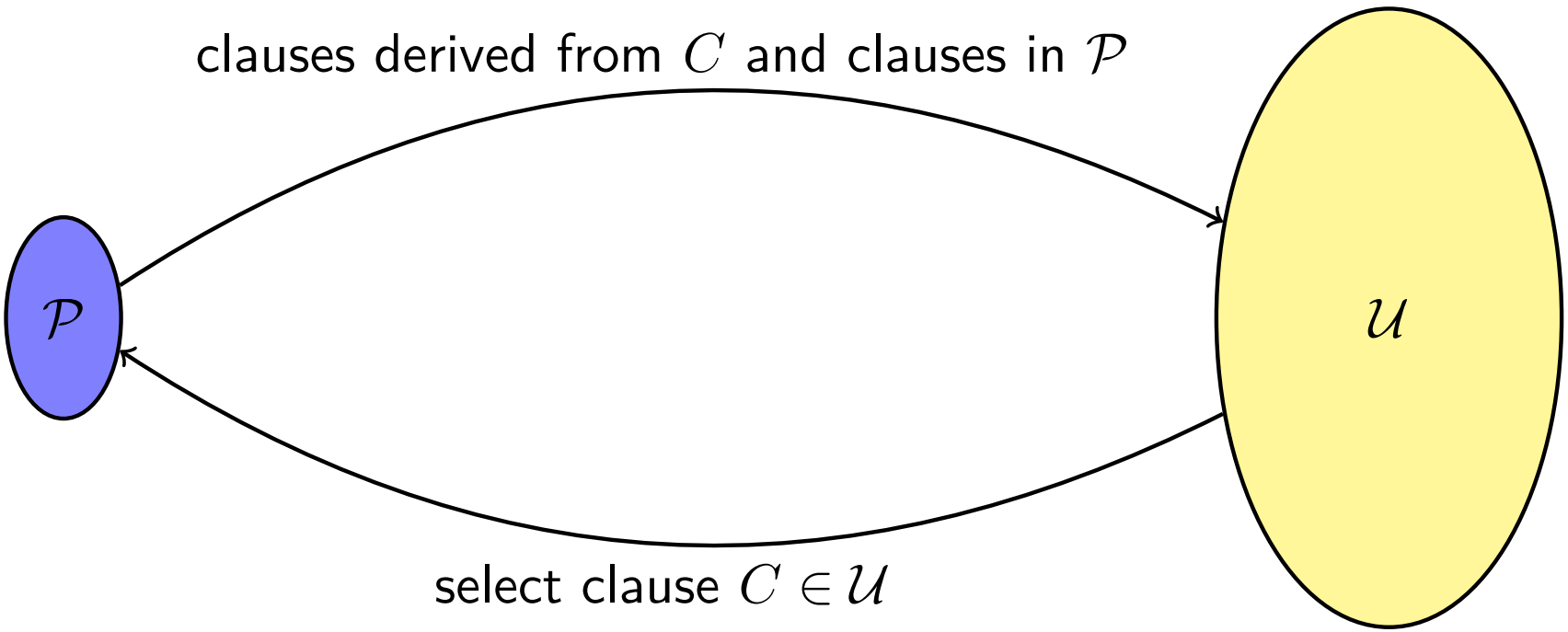
- processed clauses $\mathcal{P}$ and
- unprocessed clauses $\mathcal{U}$.



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

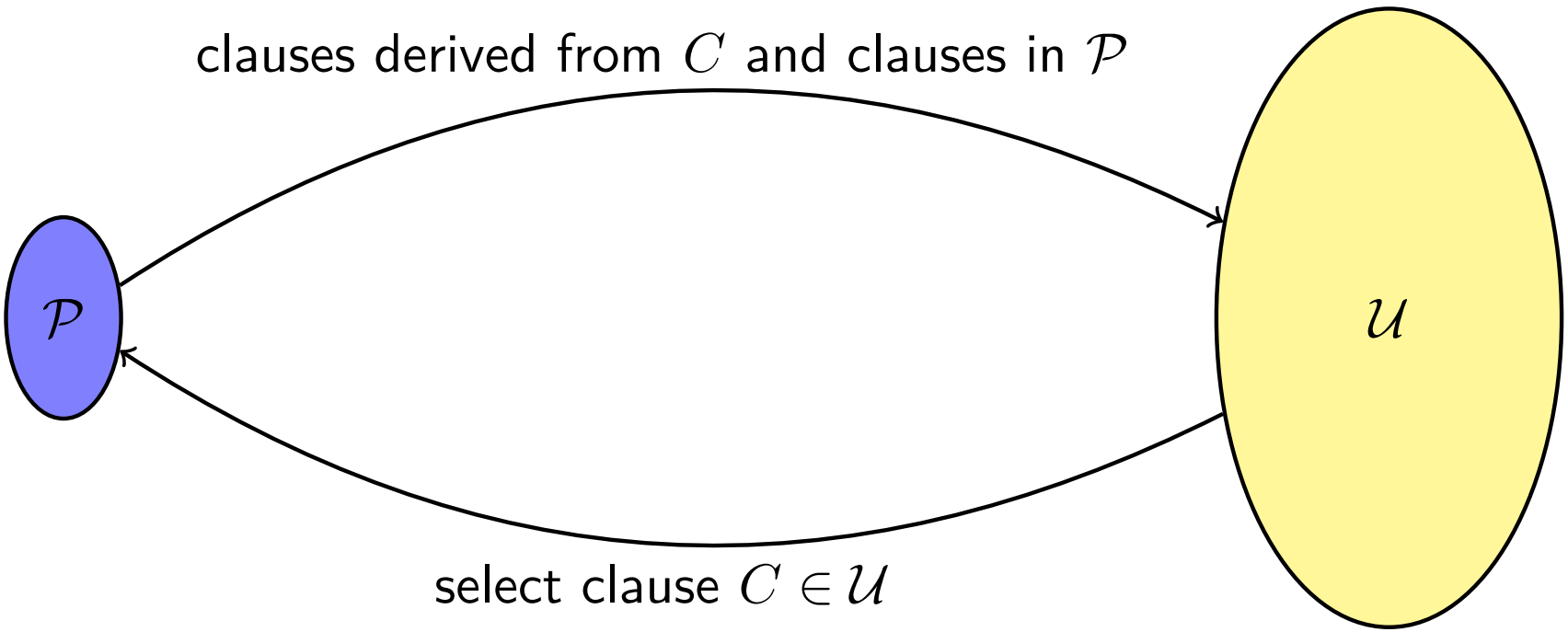
- processed clauses $\mathcal{P}$ and
- unprocessed clauses $\mathcal{U}$.



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 $\mathcal{U}$.

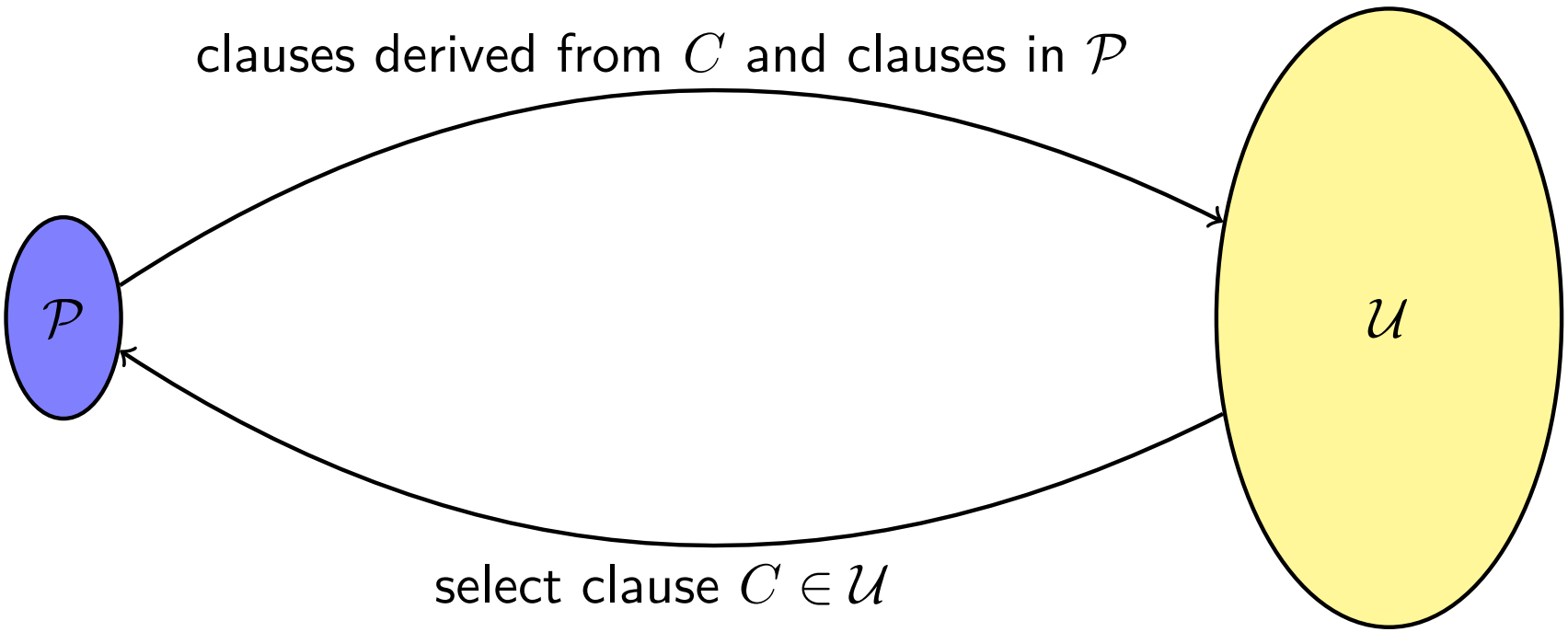


"All" the important consequences of clauses in $\mathcal{P}$ are in $\mathcal{P} \cup \mathcal{U}$.

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.

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

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

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

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

Selection function

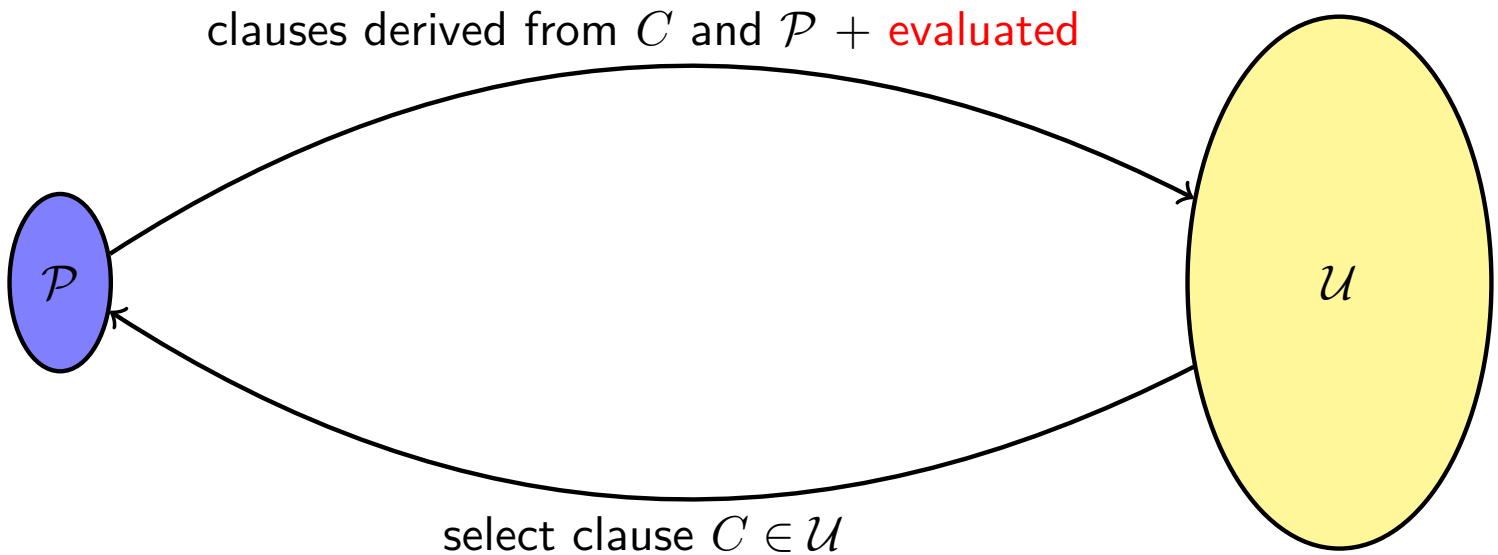
We can overrule ordering restrictions in individual clauses by selecting non-maximal (negative) literals in clauses. A selection function selects a subset of literals and we compute inferences involving only selected literals (or maximal if no selected).

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.



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)
  - small clauses are nice, but selecting only them leads to incompleteness
  - 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 you have just age and weight. Otter's approach is to combine the breadth-first and best-first search in a given ratio (e.g., 1:5 or 1:10).

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.

Watchlist

It is a set of clauses we feed into the prover that can be used for guiding the proof search (lemmata, hints). For example, clauses that led to proofs in previous similar problems.

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 (optional) and we can use this fact in our strategy (boost the generated clause).

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

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

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

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

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, where the standard timelimit is 300s.

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.

Morever, 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/SystemOnT
fof(crime_axiom, axiom, ![X1]:(big_city(X1)=>has_crime(X1)), file('/tmp/SystemOnTPTPFormReply111578/SOT_m0
fof(country_big_city, axiom, ![X1]:(country(X1)=>(big_city(capital_of(X1))&beautiful(capital_of(X1)))), fi
fof(usa_capital_axiom, axiom, ?[X1]:(city(X1)&X1=capital_of(usa)), file('/tmp/SystemOnTPTPFormReply111578/
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_negati
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,[status(thm)],[inference
fof(c_0_8, plain, ![X2]:((big_city(capital_of(X2))|~country(X2))&(beautiful(capital_of(X2))|~country(X2))
fof(c_0_9, plain, (city(esk1_0)&esk1_0=capital_of(usa)), inference(skolemize,[status(esa)],[inference(vari
cnf(c_0_10, negated_conjecture, (~city(X1)|~beautiful(X1)|~has_crime(X1)), inference(split_conjunct,[statu
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
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(
cnf(c_0_19, plain, (big_city(esk1_0)), inference(cn,[status(thm)],[inference(rw,[status(thm)],[inference(s
cnf(c_0_20, negated_conjecture, ($false), inference(cn,[status(thm)],[inference(rw,[status(thm)],[inferenc
```

Bibliography I


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.