Logical reasoning and programming

First-order resolution

Karel Chvalovský

CIIRC CTU

Our problem

Let $\Gamma = \{\psi_1, \ldots, \psi_n\}$ be a set of senteces and φ be a sentence. We know that

$$\Gamma \models \varphi$$

iff

$$\Gamma \cup \{\neg\varphi\} \text{ is unsatisfiable}$$

iff

$$\forall\psi_1' \cup \cdots \cup \forall\psi_n' \cup \forall(\neg\varphi)' \text{ is unsatisfiable,}$$

where $\psi_1', \ldots, \psi_n', (\neg\varphi)'$ are $\psi_1, \ldots, \psi_n, \neg\varphi$ in CNF (=clauses) and $\forall\Delta = \{\, \forall\chi \mid \chi \in \Delta \,\}$ for a set of clauses Δ.

We say that Δ is a set of clauses assuming that it is implicitly universally quantified.

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 $p(a) \vee \neg q(a, f(Z))$.

Herbrand models

We can restrict the types of interpretations 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 constant, 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 such that $\mathcal{M} \models \Gamma$.

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, if it has a Herbrand model.

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.

Several early approaches (Gilmore; David and Putnam in 1960) work as follows—generate ground instances and use propositional resolution. If you fail, then produce more instances and repeat.

However, such an approach is generally very inefficient (but it works for ASP).

Lifting lemma

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

For example, we have clauses

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

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

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

Use unification!

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 formulae,
- a (most general) unifier of a set of formulae.

Unification algorithm

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

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.

- $S \cup \{ u \doteq u \} \rightsquigarrow S$,
- $S \cup \{ f(u_1, \ldots, u_k) \doteq f(v_1, \ldots, v_k) \} \rightsquigarrow S \cup \{ u_1 \doteq v_1, \ldots, u_k \doteq v_k \}$,
- $S \cup \{ f(u_1, \ldots, u_k) \doteq g(v_1, \ldots, v_l) \} \rightsquigarrow$ fail, if $f \neq g$ or $k \neq l$,
- $S \cup \{ f(u_1, \ldots, u_k) \doteq X \} \rightsquigarrow S \cup \{ X \doteq f(u_1, \ldots, u_k) \}$,
- $S \cup \{ X \doteq u \} \rightsquigarrow S\{X \mapsto u\} \cup \{ X \doteq u \}$, if $X \notin u$ and $X \in S$,
- $S \cup \{ X \doteq u \} \rightsquigarrow$ 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 a term u for all occurrences of a variable X in S.

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 S,
2. the number of function (and constant) symbols that occur on the left hand sides in S,
3. the number of pairs in S.

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

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 as clauses.

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

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 $\psi_1, \ldots, \psi_n$ 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

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

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

$$\square$$

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

Completeness of resolution calculus

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 $\Gamma \vdash \square$ and this proof is no longer than $\Delta \vdash \square$.

Example

$\{p(X)\} \sqsubseteq \{p(f(a))\}$, $\{p(X)\} \sqsubseteq \{p(Y), q(Y)\}$, 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

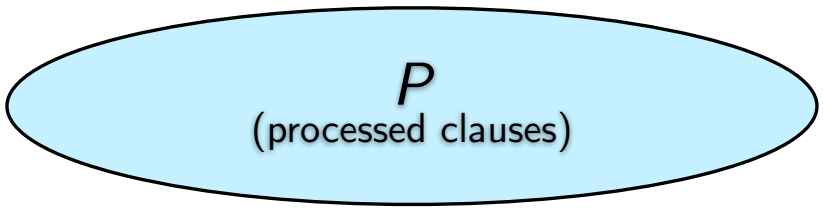
If we derive a clause φ and we already have a clause ψ such that $\varphi \sqsubseteq \psi$, then we can remove ψ, because φ is stronger. We can remove all such ψs.

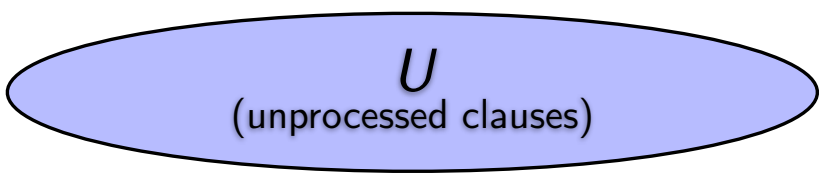
Saturation procedure

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. It is also called ANL loop.

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

The Given-Clause Algorithm

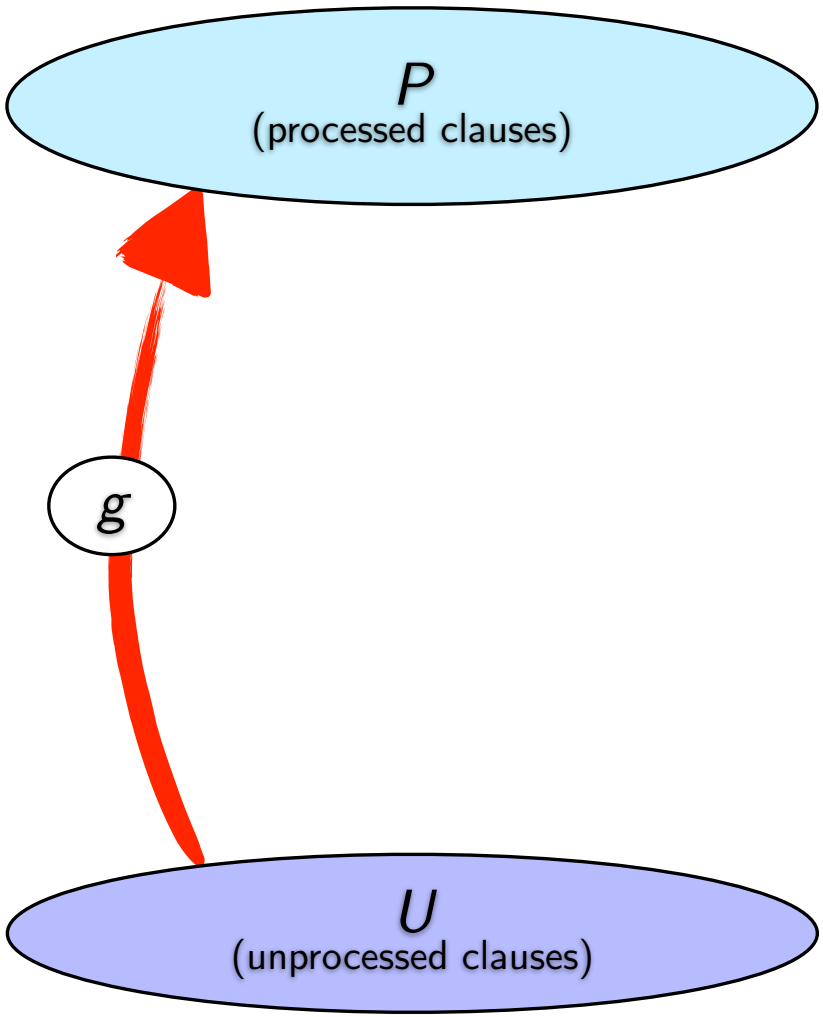




We represent the proof state S by two sets of clauses:

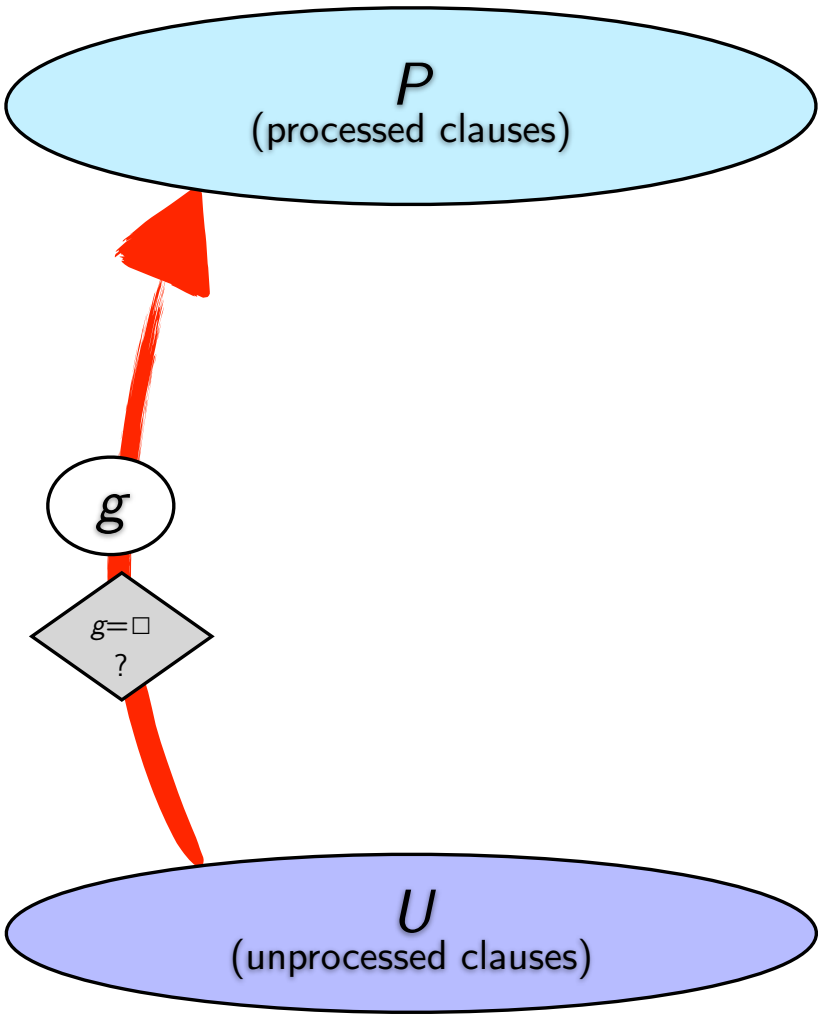
- P holds the processed clauses (originally empty)
- U holds the unprocessed clauses (originally all clauses in S)

The Given-Clause Algorithm



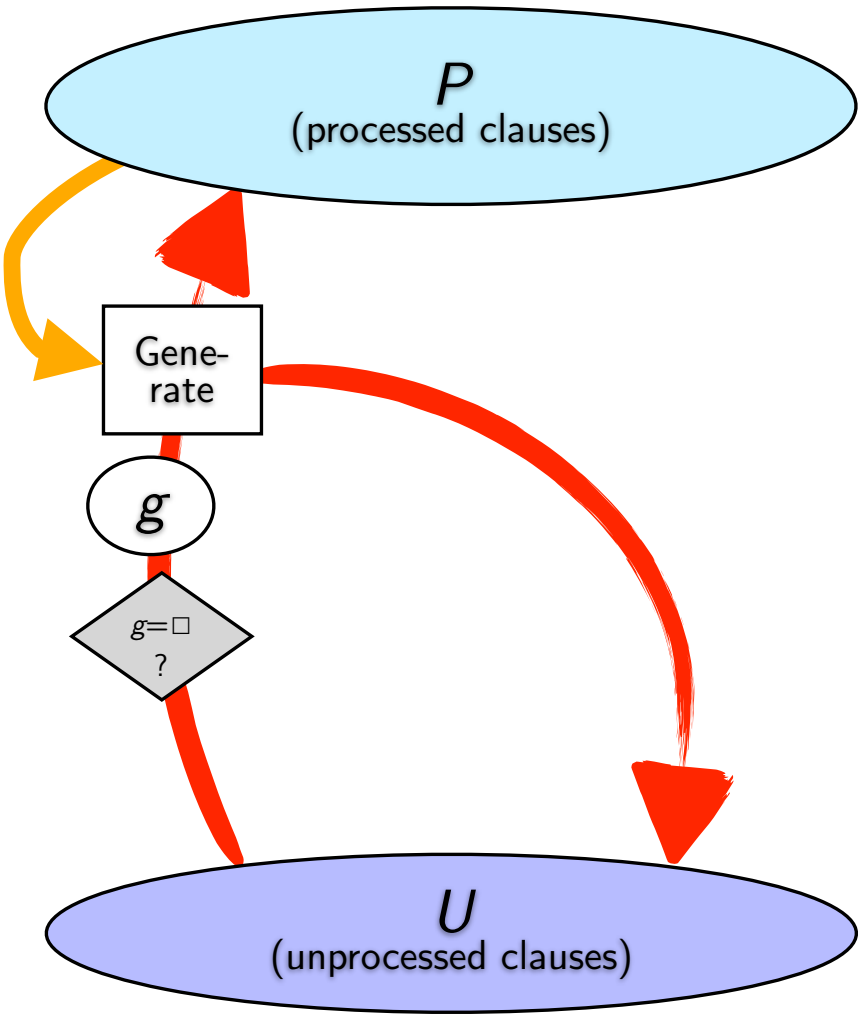
- Aim: Move everything from U to P

The Given-Clause Algorithm



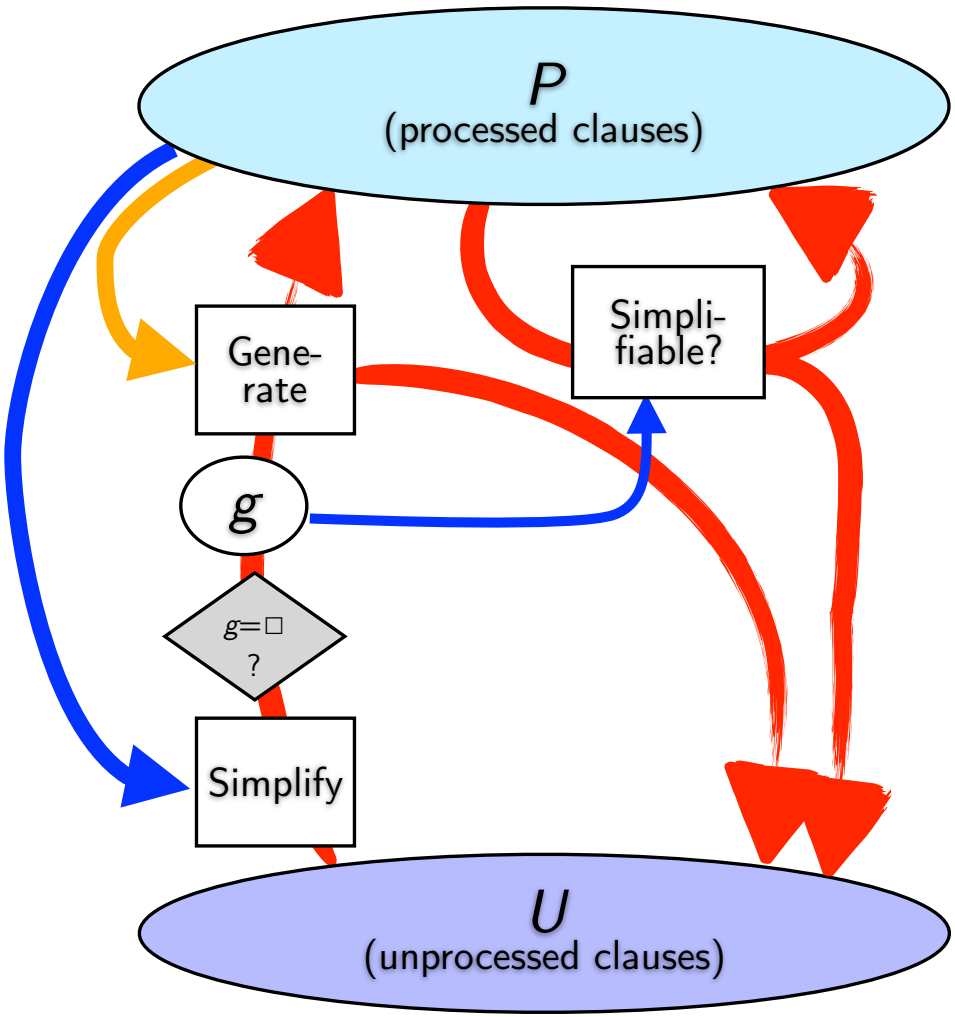
- Aim: Move everything from U to P

The Given-Clause Algorithm



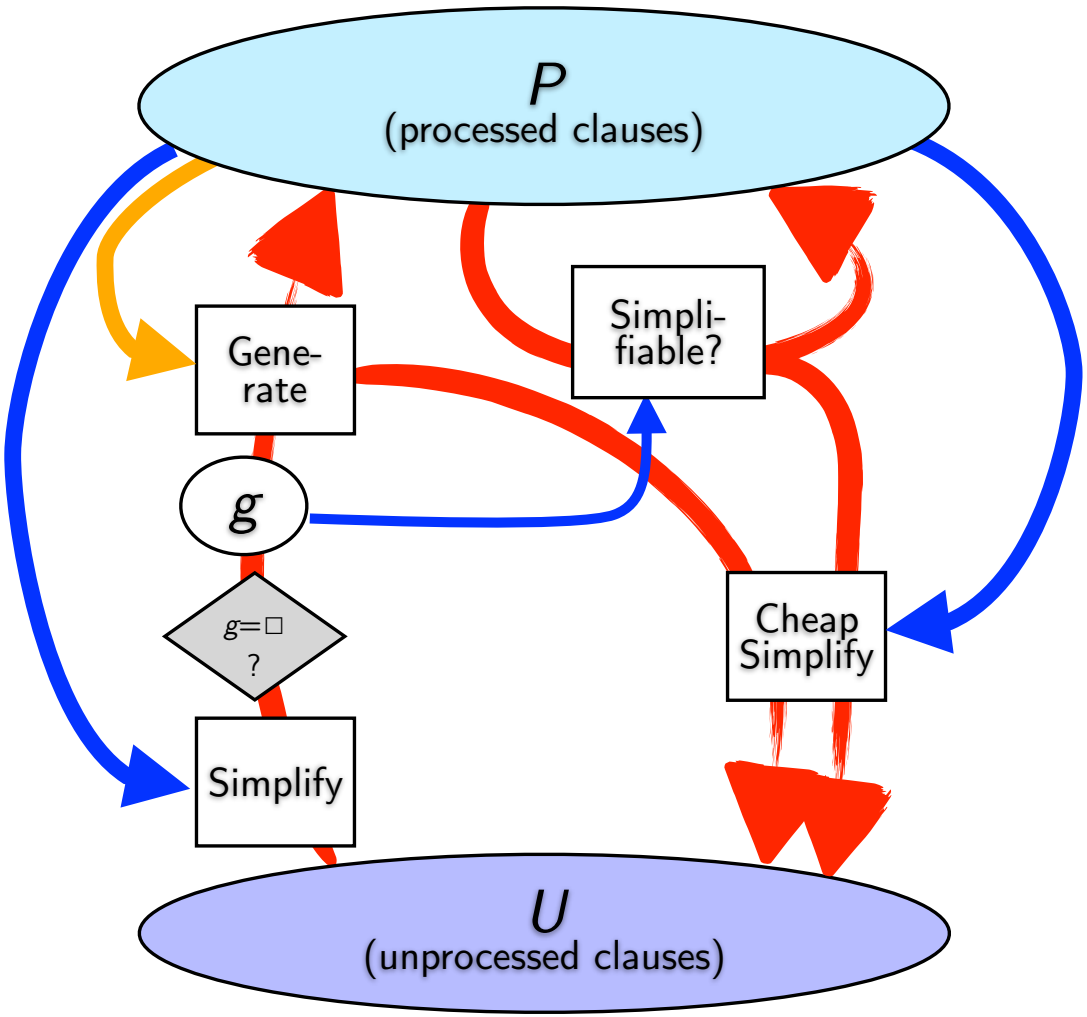
- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed

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

The Given-Clause Algorithm

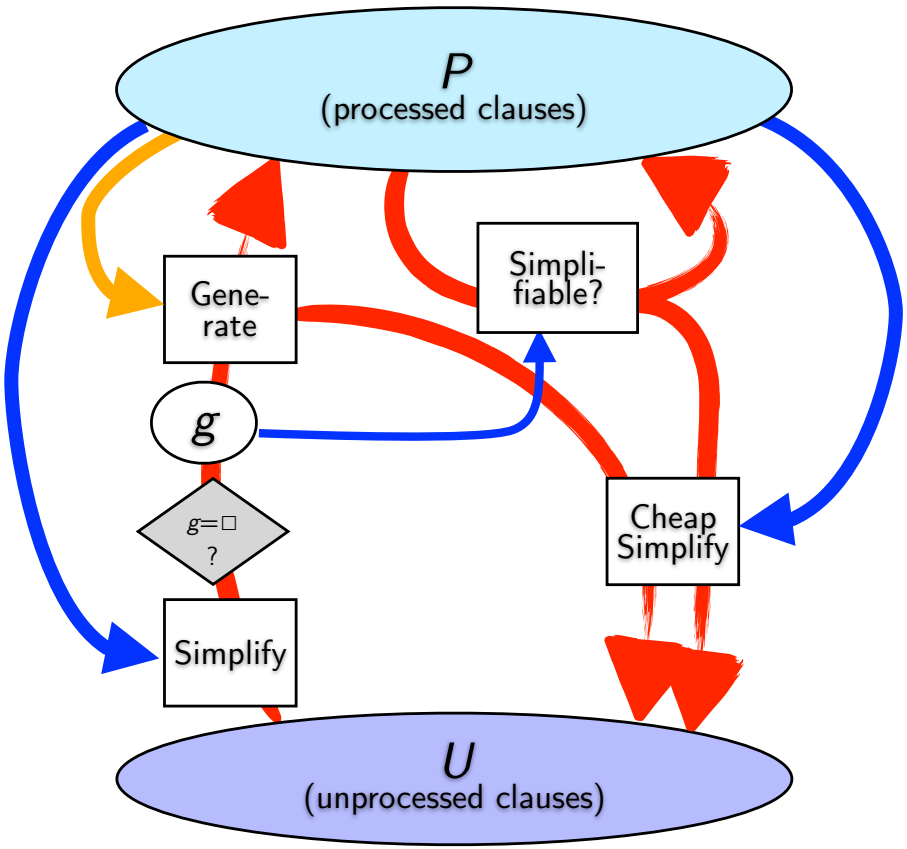


- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed
- Invariant: P is interreduced
- Clauses added to U are simplified with respect to P

The Given-Clause Loop in Fewer Words

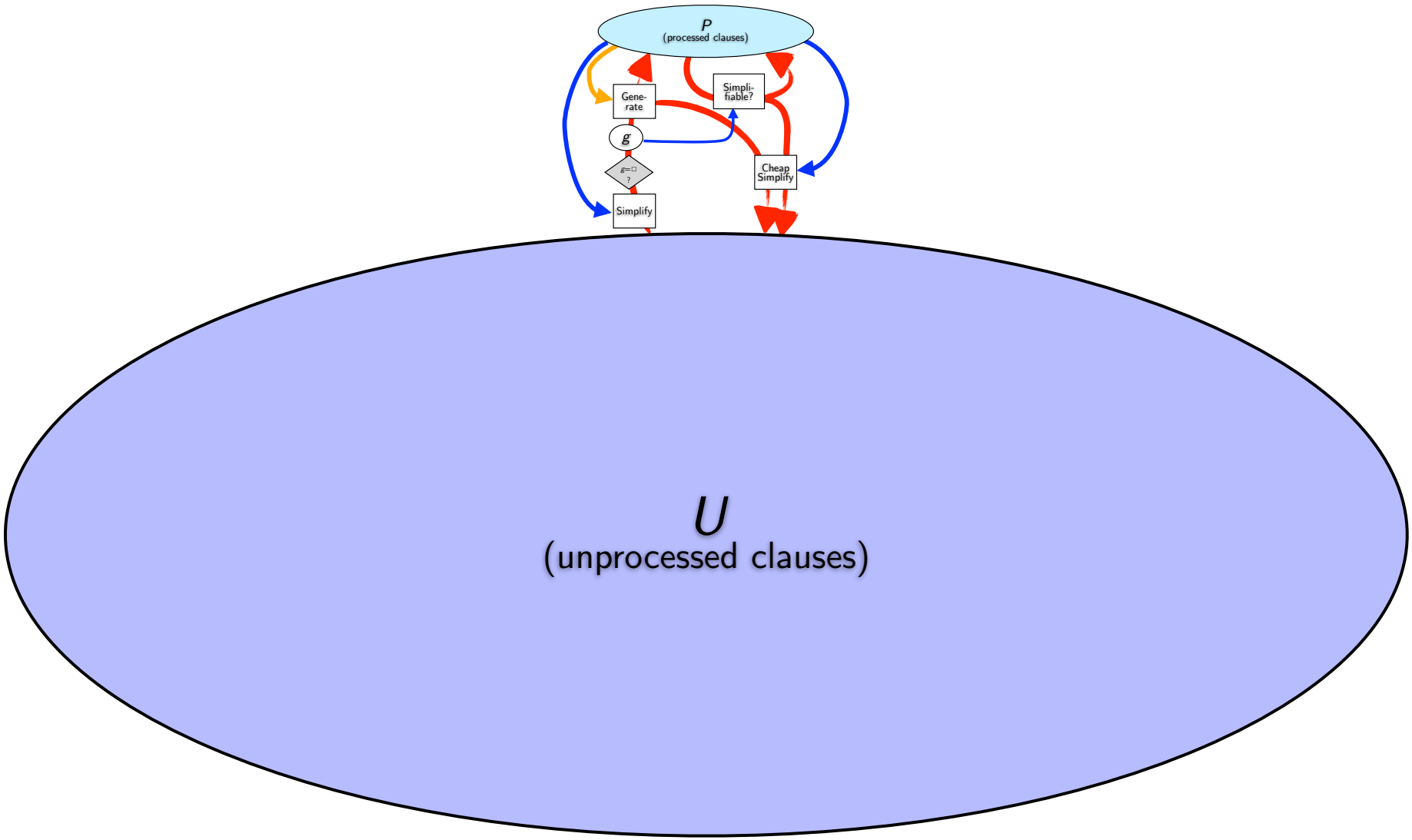
while $U \neq \{\}$
  $g = \text{delete\_best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \square$
    SUCCESS, Proof found
  if g is not subsumed by any clause in P (or otherwise redundant w.r.t. P)
    $P = P \backslash \{c \in P \mid c \text{ subsumed by (or otherwise redundant w.r.t.) } g\}$
 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

Compare and Contrast



while $U \neq \{\}$
  $g = \text{delete\_best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \square$
    SUCCESS, Proof found
  if g is not redundant w.r.t. P
    $P = P \backslash \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \backslash T) \cup \{g\}$
    $T = T \cup \text{generate}(g, P)$
    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!”



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/SystemOn
fof(crime_axiom, axiom, ![X1]:(big_city(X1)=>has_crime(X1)), file('/tmp/SystemOnTPTPFormReply111578/SOT_m
fof(country_big_city, axiom, ![X1]:(country(X1)=>(big_city(capital_of(X1))&beautiful(capital_of(X1)))), f
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(var
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_
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


Robinson, John Alan and Andrei Voronkov, eds. (2001). *Handbook of Automated Reasoning*. Vol. 1. Elsevier Science.