Superposition 1: Basics and Implementation

Stephan Schulz

schulz@eprover.org

DHBW

Contents

1. Introduction
2. Basics
3. Implementation
4. Conclusion
5. References

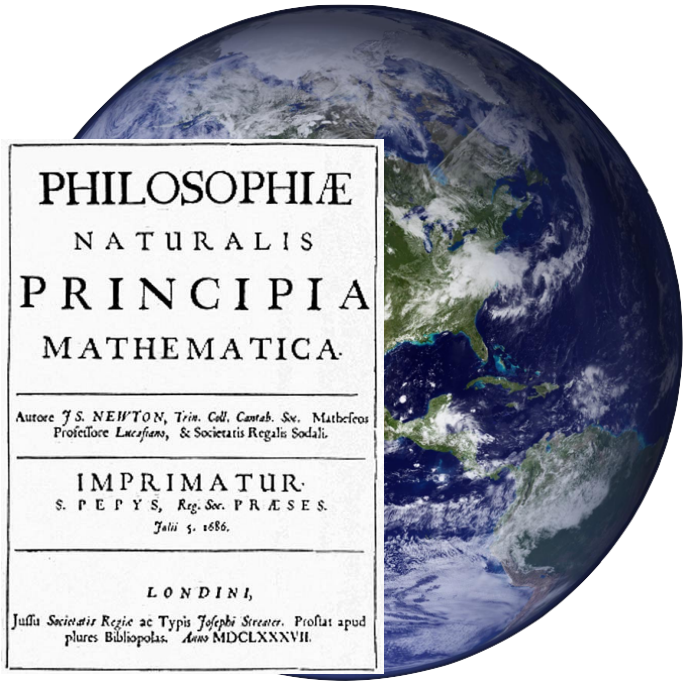
Introduction

The Big Picture

Real World Problem

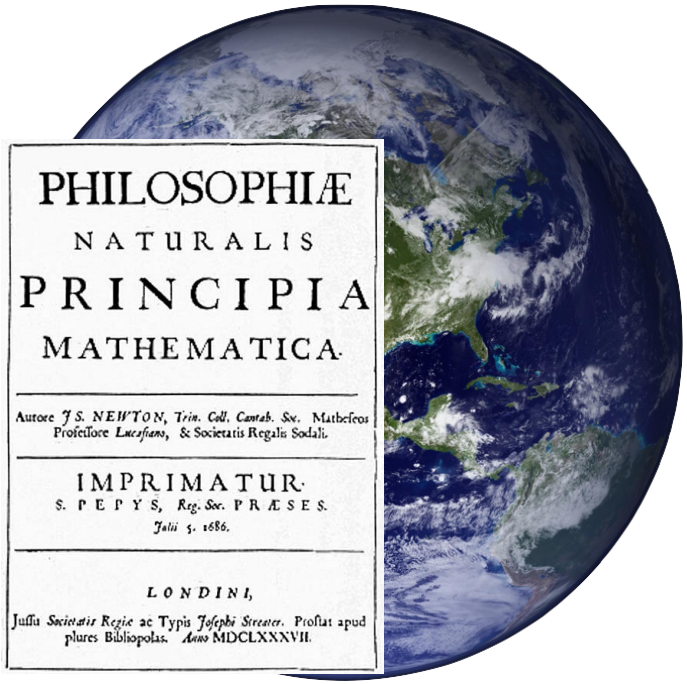
The Big Picture

Real World Problem



The Big Picture

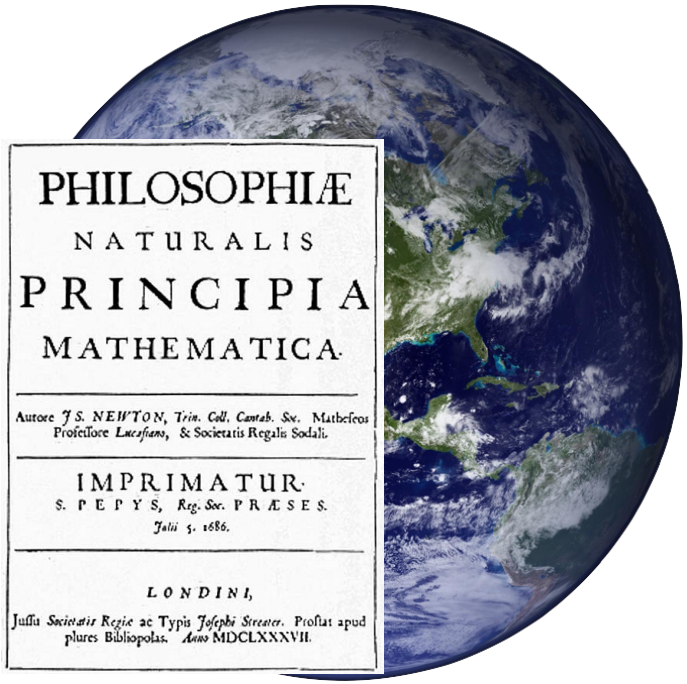
Real World Problem



Formalized Problem

The Big Picture

Real World Problem



Formalized Problem

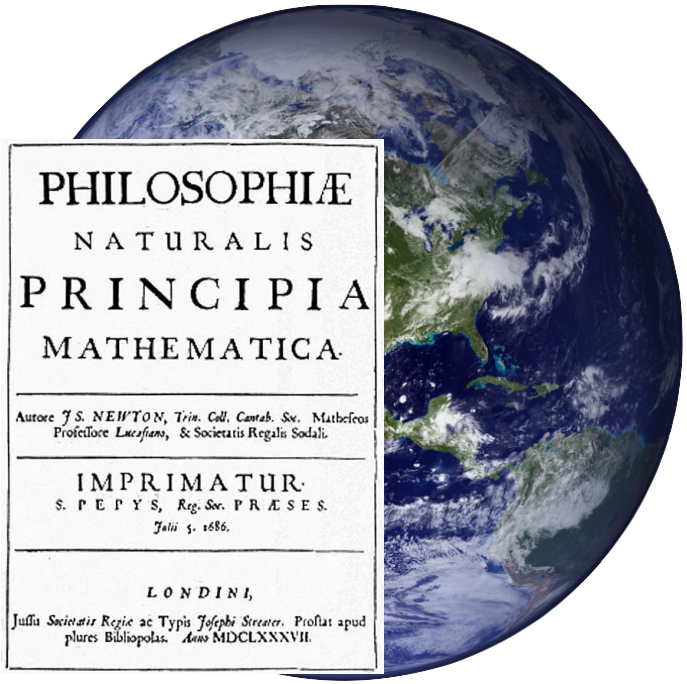
$$\forall X : human(X) \rightarrow mortal(X)$$
$$\forall X : philosopher(X) \rightarrow human(X)$$
$$philosopher(socrates)$$

$$?$$
$$\models$$

$$mortal(socrates)$$

The Big Picture

Real World Problem



Formalized Problem

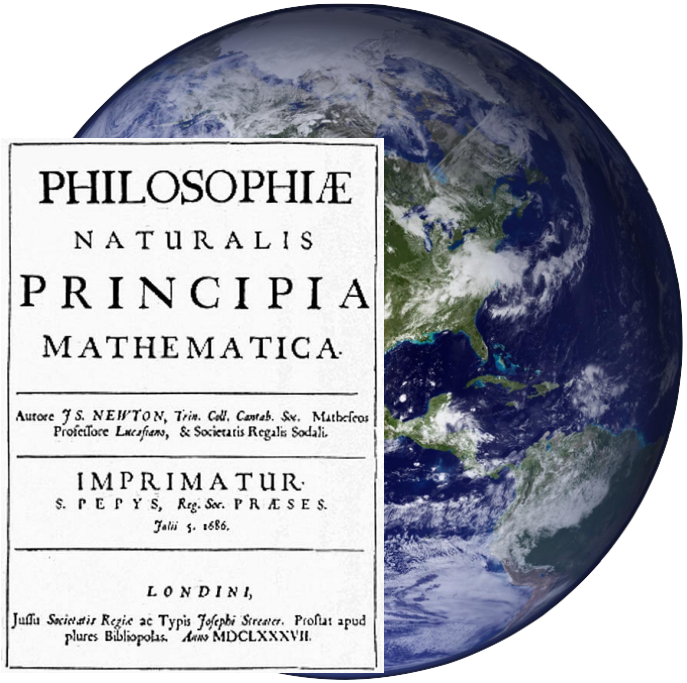
$$\forall X : f(X, 0) \simeq X)$$
$$\forall X : \exists Y : f(X, Y) \simeq 0$$
$$\forall X, Y, Z : f(f(X, Y), Z) \simeq f(X, f(Y, Z))$$

$$?$$
$$\models$$

$$\forall X, Y : (f(X, Y) \simeq 0 \rightarrow f(Y, X) \simeq 0)$$

The Big Picture

Real World Problem



Formalized Problem

$$\forall X : f(X, 0) \simeq X)$$
$$\forall X : \exists Y : f(X, Y) \simeq 0$$
$$\forall X, Y, Z : f(f(X, Y), Z) \simeq f(X, f(Y, Z))$$

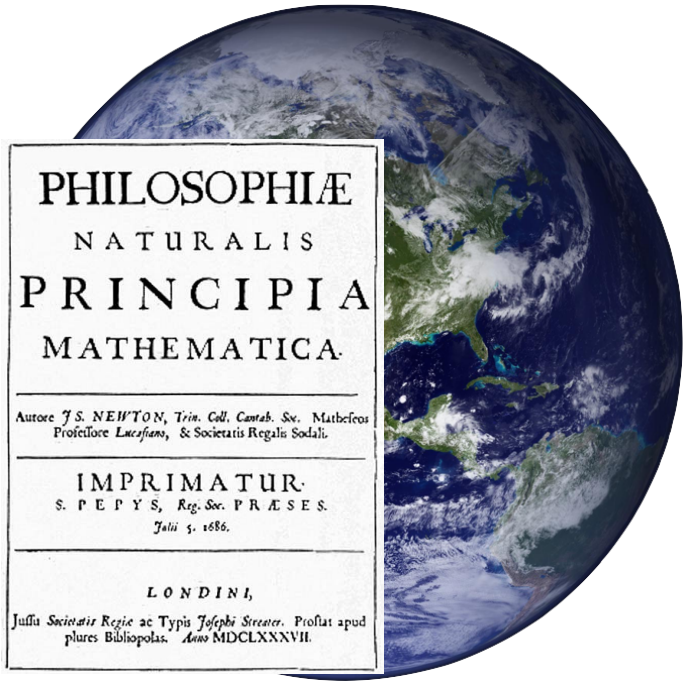
$$?$$
$$\models$$

$$\forall X, Y : (f(X, Y) \simeq 0 \rightarrow f(Y, X) \simeq 0)$$

ATP

The Big Picture

Real World Problem



Formalized Problem

$$\forall X : f(X, 0) \simeq X)$$
$$\forall X : \exists Y : f(X, Y) \simeq 0$$
$$\forall X, Y, Z : f(f(X, Y), Z) \simeq f(X, f(Y, Z))$$

$$?$$
$$\models$$

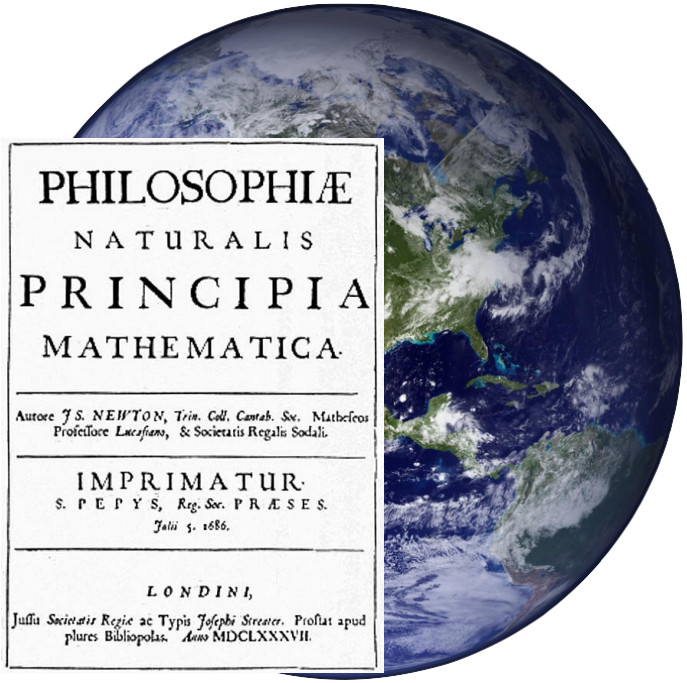
$$\forall X, Y : (f(X, Y) \simeq 0 \rightarrow f(Y, X) \simeq 0)$$

ATP

Proof

The Big Picture

Real World Problem



Formalized Problem

$$\forall X : f(X, 0) \simeq X)$$
$$\forall X : \exists Y : f(X, Y) \simeq 0$$
$$\forall X, Y, Z : f(f(X, Y), Z) \simeq f(X, f(Y, Z))$$

$$?$$
$$\models$$

$$\forall X, Y : (f(X, Y) \simeq 0 \rightarrow f(Y, X) \simeq 0)$$

Proof

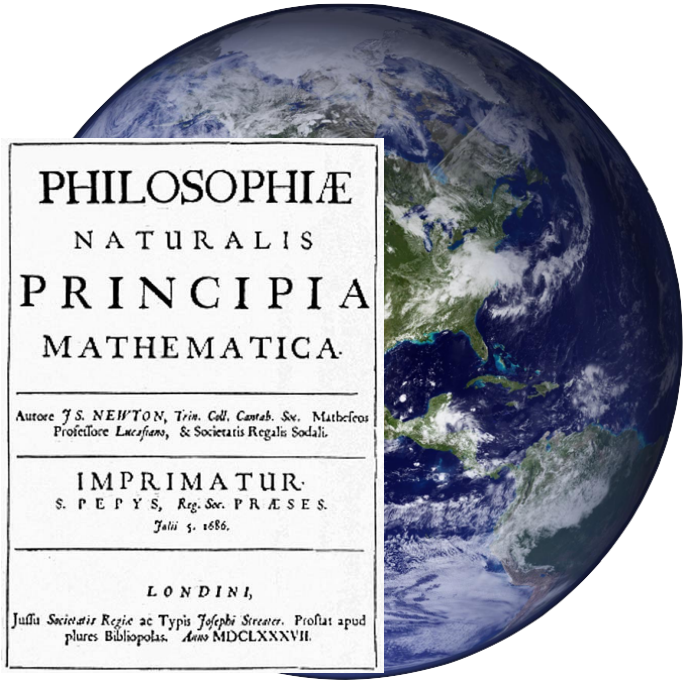
or

Countermodel

ATP

The Big Picture

Real World Problem



Formalized Problem

$$\forall X : f(X, 0) \simeq X)$$
$$\forall X : \exists Y : f(X, Y) \simeq 0$$
$$\forall X, Y, Z : f(f(X, Y), Z) \simeq f(X, f(Y, Z))$$

$$?$$
$$\models$$

$$\forall X, Y : (f(X, Y) \simeq 0 \rightarrow f(Y, X) \simeq 0)$$

Proof

or

Countermodel

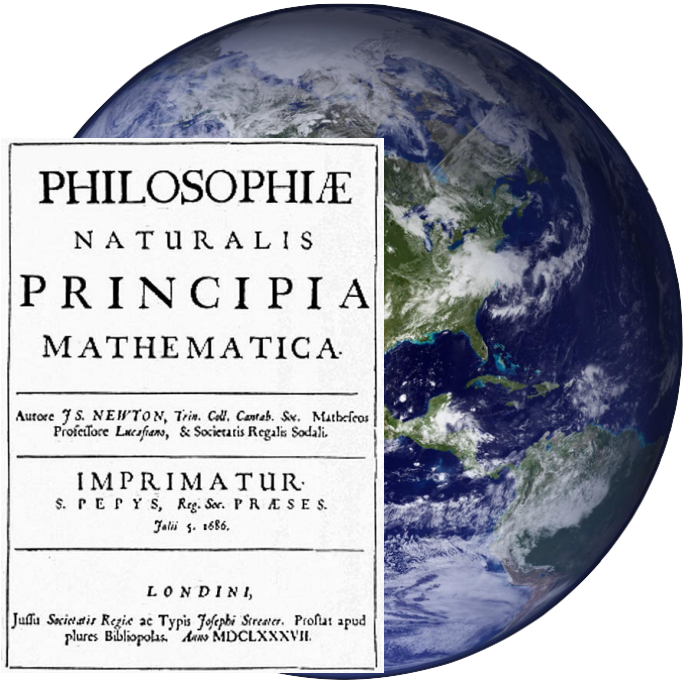
or

Timeout

ATP

The Big Picture

Real World Problem



Formalized Problem

$$\forall X : f(X, 0) \simeq X)$$
$$\forall X : \exists Y : f(X, Y) \simeq 0$$
$$\forall X, Y, Z : f(f(X, Y), Z) \simeq f(X, f(Y, Z))$$

$$?$$
$$\models$$

$$\forall X, Y : (f(X, Y) \simeq 0 \rightarrow f(Y, X) \simeq 0)$$

Proof

or

Countermodel

or

Timeout

ATP

Superposition

Proof by Contradiction

$$\forall X : human(X) \rightarrow mortal(X)$$
$$\forall X : philosopher(X) \rightarrow human(X)$$
$$philosopher(socrates)$$
$$\models$$
$$mortal(socrates)$$

iff

$$\left\{ \begin{array}{c} \forall X : human(X) \rightarrow mortal(X) \\ \forall X : philosopher(X) \rightarrow human(X) \\ philosopher(socrates) \\ \neg mortal(socrates) \end{array} \right\}$$

is unsatisfiable

From Formulas To Clauses

$$\left\{ \begin{array}{c} \forall X : human(X) \rightarrow mortal(X) \\ \forall X : philosopher(X) \rightarrow human(X) \\ philosopher(socrates) \\ \neg mortal(socrates) \end{array} \right\}$$

is unsatisfiable

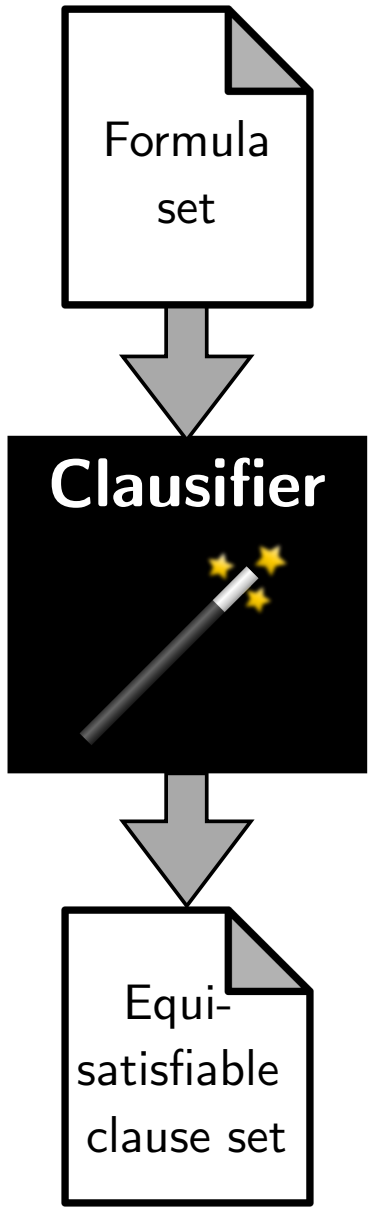
iff

$$\left\{ \begin{array}{c} \neg human(X) \vee mortal(X) \\ \neg philosopher(X) \vee human(X) \\ philosopher(socrates) \\ \neg mortal(socrates) \end{array} \right\}$$

is unsatisfiable

Clausification

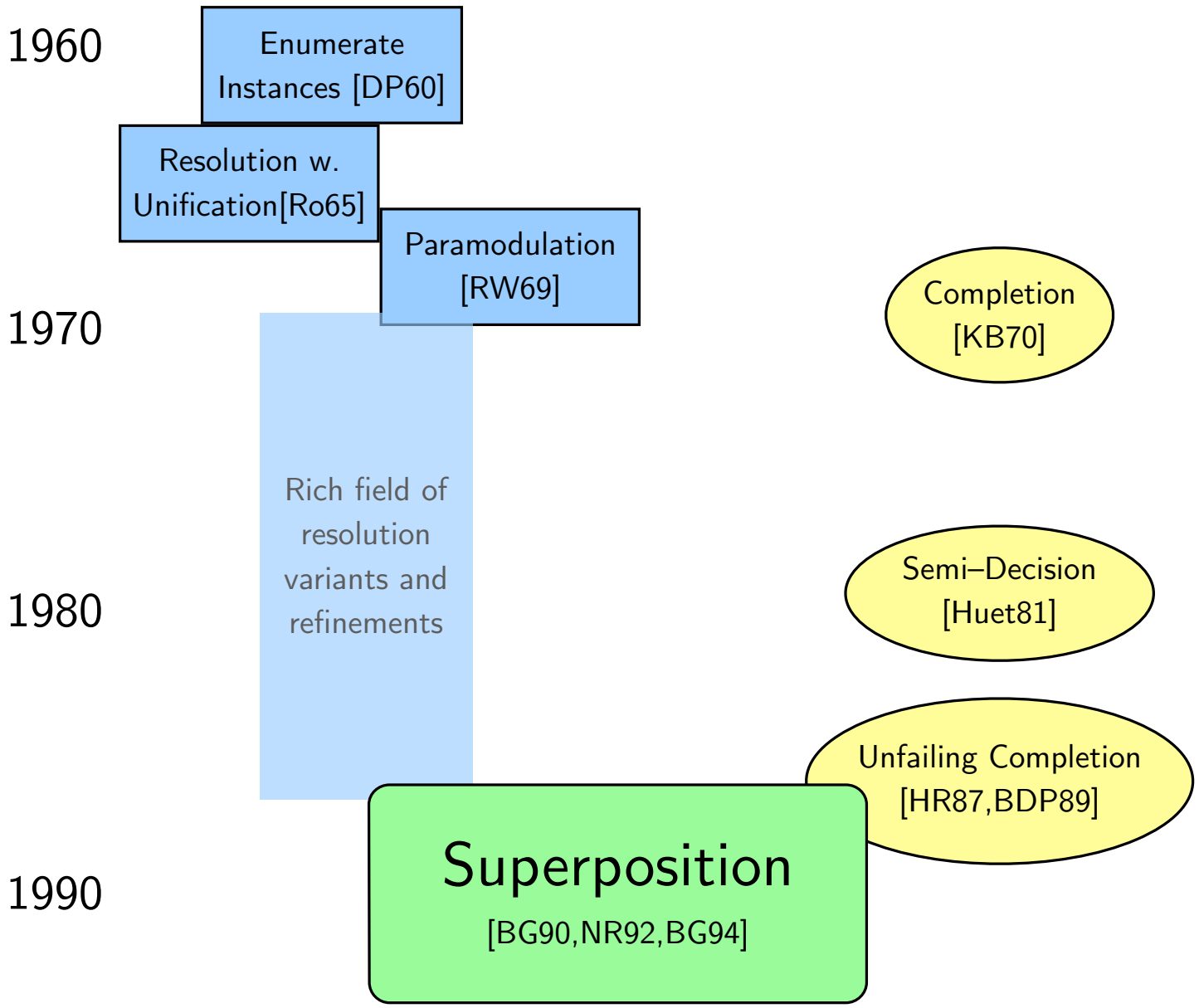
1. Optional: Introduce definitions to avoid exponential explosion
   - Only satisfiability-preserving
2. Eliminate implication and equivalency, simplify
3. Move negations inwards to literals (Negation Normal Form)
4. Optional: Miniscope (move universal quantifiers inwards)
5. Remove existential quantifiers via Skolemization
   - Only satisfiability-preserving
6. Move universal quantifiers outwards
7. Apply distributive law to move disjunctions inwards (Conjunktive Normal Form)
8. Read off disjuncts as clauses



Saturating Theorem Proving

- Goal: Show unsatisfiability of a set of clauses S
- Approach:
  - Systematically enrich S with clauses derived via inferences from clauses in S (Saturation)
  - Optionally: Remove redundant clauses
- Outcome:
  - Derivation of the empty clause $\square$ (explicit witness of unsatisfiability)
  - Successful saturation (up to redundancy): S is satisfiable
  - . . . or infinite sequence of derivations
- Properties:
  - Correctness: Only logical consequences are derived
  - Completeness: Every unsatisfiable S will eventually lead to the derivation of $\square$

A (slightly subjective) historical view



Journal of Logic and Computation (s. u.)

Rewrite-based Equational Theorem Proving with Selection and Simplification

LEO BACHMAIR, *Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794, USA.*
E-mail: leo@sbcs.sunysb.edu

HARALD GANZINGER, *Max-Planck-Institut für Informatik, Im Stadtwald, D-W-66123 Saarbrücken, Germany.*
E-mail: hg@mpi-sb.mpg.de

Abstract

We present various refutationally complete calculi for first-order clauses with equality that allow for arbitrary selection of negative atoms in clauses. Refutational completeness is established via the use of well-founded orderings on clauses for defining a Herbrand model for a consistent set of clauses. We also formulate an abstract notion of redundancy and show that the deletion of redundant clauses during the theorem proving process preserves refutational completeness. It is often possible to compute the closure of non-trivial sets of clauses under application of non-redundant inferences. The refutation of goals for such complete sets of clauses is simpler than for arbitrary sets of clauses, in particular it suffices to consider only those proofs that have support from the goals, without compromising refutational completeness. For certain classes of formulas the search space can be restricted even further, as we demonstrate for so-called quasi-Horn clauses. The results in this paper contain as special cases or generalize many known results about Knuth–Bendix-like completion procedures (for equations, Horn clauses, and Horn clauses with Boolean conditions), completion of first-order clauses by clausal rewriting, and inductive theorem proving for Horn clauses.

Keywords: Theorem proving, first-order logic, equality, paramodulation, rewrite techniques, simplification, saturation.

Introduction

Methods for dealing with the equality predicate are of central concern in automated theorem proving. One of the more successful approaches to equational reasoning is the use of equations as one-way rewrite rules, so that a formula can be simplified by repeatedly replacing an instance of the left-hand side of a rule by its right-hand side until a simplest possible form (a normal form) is obtained. The completion procedure [22], for instance, attempts to construct a convergent rewrite system (one for which normal forms exist and are unique) from a given set of universally quantified equational axioms. A convergent rewrite system provides a decision procedure for the underlying equational theory: two terms can be rewritten to identical normal forms if, and only if, they are equal. The completion procedure may fail to find a convergent system, but has been extended to a refutationally complete theorem prover (cf. [25, 19, 3]). The two main components of completion are, first, a deductive inference rule called *superposition* and, secondly, various mechanisms for deleting redundant equations via *simplification by rewriting*. These features have been extended to conditional rewrite rules, that is, Horn clauses in which equations are the only atomic formulas [16, 15, 23]. In this paper we generalize the rewriting approach to general first-order clauses, and combine it with new techniques for equational theorem proving, based on the observation that superposition is a restricted form of paramodulation [33], while demodulation

J. Logic Computat., Vol. 4 No. 3, pp. 217–247 1994

Basics

Equational Clause Logic (0)

Multisets

- Formally: A multiset over a set S is a function $M : S \to \mathbb{N}$.
- Intuitively: A multiset is a set-like collection that allows multiple occurrences of the same element
- We write multisets like sets and generalize most operations in the obvious way:
  - $\forall s \in S : (M_1 \cup M_2)(s) = M_1(s) + M_2(s)$
  - $\forall s \in S (M_1 \cap M_2)(s) = \min(M_1(s), M_2(s)$
  - $M_1 \subseteq M_2$ iff $\forall s \in S : M_1(s) \leq M_s(s)$
  - ...

Equational Clause Logic (1)

Definition: Signature, Terms

We assume:

- Signature F of function symbols with associated arities, e.g. $F = \{f/2, g/1, a/0, n/0\}$
  - We assume at least one constant in F
- Enumerable set of variables $V = \{X, Y, Z, X_1, X_2, \ldots\}$

The set $T(F, V)$ of terms is defined as follows

- $V \subseteq T(F, V)$
- If $f/n \in F, t_1, \ldots, t_n \in T(F, V)$, then $f(t_1, \ldots, t_n) \in T(F, V)$
- Nothing else is a term

- Example terms: $f(X, Y), a, f(g(a), g(b)), X, f(X, f(Y, Z))$
  - We write a instead of $a()$ for constants

Equational Clause Logic (2)

Definition: Equations, literals

- An (positive) equation is an unordered pair of terms
  - We write $s \simeq t$
  - We identify $s \simeq t$ with the multiset $\{\{s\}, \{t\}\}$
- An disequation or negative equation is an unordered pair of terms
  - We write $s \not\simeq t$
  - We identify $s \not\simeq t$ with the multiset $\{\{s, t\}\}$
- A literal is either an equation or a disequation
  - We write $s \mathrel{\dot{\simeq}} t$ if we don't want to specify the polarity
  - ... or we simply write $l_1, l_2, \ldots$ for various literals

Equational Clause Logic (3)

Definition: Clauses

- A clause is a multiset of literals
- We write a clause as a disjunction of literals
  - $\{l_1, l_2, l_3\} = l_1 \vee l_2 \vee l_3$

Examples:

- $g(a) \simeq a \vee g(a) \not\simeq a$ is a (tautological) clause
- $f(X, f(Y, Z)) \simeq f(f(X, Y), Z)$ is a (positive unit) clause
- $a \not\simeq b \vee g(a) \not\simeq g(b)$ is a (negative) clause

Ground Terms and the Herbrand Universe

Definition: Ground Terms

- A ground term is a terms that does not contain any variables
  - Analogously: ground literals, ground clauses
- We write $T(F)$ for the set of all ground terms over the signature F
- We also call $T(F)$ the *Herbrand universe* for F

Ground Terms and the Herbrand Universe

Definition: Ground Terms

- A ground term is a terms that does not contain any variables
  - Analogously: ground literals, ground clauses
- We write $T(F)$ for the set of all ground terms over the signature F
- We also call $T(F)$ the *Herbrand universe* for F

Jaques Herbrand: If we want to show that a set of clauses is unsatisfiable, we only need to consider interpretations where variables range over the Herbrand universe

Herbrand Equality Interpretations

Definition

- A Herbrand Equality Interpretation I is a congruence relation on ground terms, i.e. a relation for which the following holds:
  - I is reflexive, symmetric and transitive
  - If $(s, t) \in I$, then $(u[p \leftarrow s], u[p \leftarrow t]) \in I$
    - ... where $u[p \leftarrow s]$ denotes the term u with the subterm at position p replaced by s
- A ground equation $s \simeq t$ is true under I, if $(s, t) \in I$
  - We write $I \models s \simeq t$
- A ground disequation $s \not\simeq t$ is true under I, if $(s, t) \notin I$
  - We write $I \models s \not\simeq t$
- A ground clause C is true under I, if one of its literals is true
  - We write $I \models s \not\simeq t$

Exercise: Herbrand Equality Interpretations

- Assume $F = \{a/0, b/0, c/0, g/1\}$
- What is the smallest Herbrand Equality Interpretation I_1 in which $a \simeq b$ is true?
- What is the smallest Herbrand Equality Interpretation I_2 in which both $a \simeq b$ and $g(a) \simeq g(c)$ are true?
- What is the truth status of the following clauses under I_1 and I_2?
  - $g(a) \not\simeq b \vee a \simeq c$
  - $g(b) \not\simeq g(c) \vee a \simeq c$

Substitutions

Definition: Substitution, Instance

- A substitution is a mapping $\sigma : V \to T(F, V)$ with the property that $\{X \in V | X\sigma \neq X\}$ is finite
- We continue it to a mapping $\sigma : T(F, V) \to T(F, V)$
  - Every variable X in a term t is replaced by the corresponding term $X\sigma$
  - We write $t\sigma$ for the result of applying σ to t
- Substitutions are further continued to literals and clauses in the obvious way
- We call $t\sigma$, $l\sigma$, $C\sigma$ instances of t, l, or C, respectively
- If $t\sigma, l\sigma, C\sigma$ are ground, we call σ a grounding substitution (for t, l, C), and $t\sigma, l\sigma, C\sigma$ ground instances

Substitutions

Definition: Substitution, Instance

- A substitution is a mapping $\sigma : V \to T(F, V)$ with the property that $\{X \in V | X\sigma \neq X\}$ is finite
- We continue it to a mapping $\sigma : T(F, V) \to T(F, V)$
  - Every variable X in a term t is replaced by the corresponding term $X\sigma$
  - We write $t\sigma$ for the result of applying σ to t
- Substitutions are further continued to literals and clauses in the obvious way
- We call $t\sigma$, $l\sigma$, $C\sigma$ instances of t, l, or C, respectively
- If $t\sigma, l\sigma, C\sigma$ are ground, we call σ a grounding substitution (for t, l, C), and $t\sigma, l\sigma, C\sigma$ ground instances

Examples: Consider $\sigma = \{X \leftarrow a, Y \leftarrow g(X)\}$

- $f(X, Y)\sigma = f(a, g(X))$
- σ is grounding for $g(X)$

Semantics of Clauses

- We have defined the truth value of ground clauses under a Herbrand Equality Interpretation I
- What about non-ground clauses?

Semantics of Clauses

- We have defined the truth value of ground clauses under a Herbrand Equality Interpretation I
- What about non-ground clauses?

Semantics of clauses

Let I be a Herbrand Equality Interpretation and C be a clause

- $I \models C$ iff $I \models C\sigma$ for all grounding substitutions σ

Semantics of Clauses

- We have defined the truth value of ground clauses under a Herbrand Equality Interpretation I
- What about non-ground clauses?

Semantics of clauses

Let I be a Herbrand Equality Interpretation and C be a clause

- $I \models C$ iff $I \models C\sigma$ for all grounding substitutions σ

- A clause is true under Herbrand Equality Interpretation if all its ground instances are

Herbrand Equality Models

Definition

Assume a set S of clauses and a Herbrand Equality Interpretation I

- I is called a Herbrand Equality Model iff $\forall C \in S : I \models C$
- S is called satisfiable, if such an I exists
- Otherwise, S is unsatisfiable

Herbrand Equality Models

Definition

Assume a set S of clauses and a Herbrand Equality Interpretation I

- I is called a Herbrand Equality Model iff $\forall C \in S : I \models C$
- S is called satisfiable, if such an I exists
- Otherwise, S is unsatisfiable

Different Interpretations of Clauses

- As a disjunction, a clause is true, if one of its literals is true

$$s_1 \not\simeq t_1 \vee s_2 \not\simeq t_2 \vee s_3 \simeq t_3 \vee s_4 \simeq t_4$$

- Implicational form: If the disequations are false, one of the equations must be true

$$(s_1 \simeq t_1 \wedge s_2 \simeq t_2) \rightarrow (s_3 \simeq t_3 \vee s_4 \simeq t_4)$$

- Even stronger: If all but one literals are false, the remaining one must be true

$$(s_1 \simeq t_1 \wedge s_2 \simeq t_2 \wedge s_3 \not\simeq t_3) \rightarrow s_4 \simeq t_4$$

Different Interpretations of Clauses

- As a disjunction, a clause is true, if one of its literals is true

$$s_1 \not\simeq t_1 \vee s_2 \not\simeq t_2 \vee s_3 \simeq t_3 \vee s_4 \simeq t_4$$

- Implicational form: If the disequations are false, one of the equations must be true

$$(s_1 \simeq t_1 \wedge s_2 \simeq t_2) \rightarrow (s_3 \simeq t_3 \vee s_4 \simeq t_4)$$

- Even stronger: If all but one literals are false, the remaining one must be true

$$(s_1 \simeq t_1 \wedge s_2 \simeq t_2 \wedge s_3 \not\simeq t_3) \rightarrow s_4 \simeq t_4$$

Clauses can be seen as (conditional) rewrite rules

Unification

Definition: Unifier, mgu

- A substitution σ is called a unifier for two terms s,t, if $s\sigma = t\sigma$
- It is called a most general unifier if it is the most general substitution with this property
  - If a unifier exists, the most general unifier is unique (up to variable renamings) and can be computed easily
  - We write $mgu(s, t)$ to denote the most general unifier of s and t

Unification

Definition: Unifier, mgu

- A substitution σ is called a unifier for two terms s,t, if $s\sigma = t\sigma$
- It is called a most general unifier if it is the most general substitution with this property
  - If a unifier exists, the most general unifier is unique (up to variable renamings) and can be computed easily
  - We write $mgu(s, t)$ to denote the most general unifier of s and t

- If a term represents all its ground instances, the *mgu* allows us to find the common instances of two terms
- Thus, it allows us to derive new clauses which talk about these subsets

Exercise/Examples

- Consider the (unit) clause $g(a) \not\simeq g(X)$. Is it satisfiable? Why or why not?
- Consider the set $S = \{a \simeq b, g(a) \not\simeq g(b)\}$. Is S satisfiable? Why or why not?

Exercise/Examples

- Consider the (unit) clause $g(a) \not\simeq g(X)$. Is it satisfiable? Why or why not?
- Consider the set $S = \{a \simeq b, g(a) \not\simeq g(b)\}$. Is S satisfiable? Why or why not?

The fact that Herbrand Equality Interpretations are congruences allow us to apply equational reasoning!

Equality Resolution

- Syntactically identical terms must be equal in I
  - I is a congruence relation and hence reflexive
- Thus, if the two terms of a negative equation have common instances, we can shorten the clause for these instances

$$\text{(ER)}\ \frac{C \vee s \not\approx t}{C\sigma}\ \text{if}\ \sigma = mgu(s,t)$$

- Notes:
  - C is an arbitrary clause (possibly empty)
  - Remember that clauses are multisets, hence unordered
  - Any interpretation that makes $C \vee s \not\approx t$ true also makes $C\sigma$ true, hence $C\sigma$ is a logical consequence of $C \vee s \not\approx t$
  - This is a generating inference: The new clause is added to the current set of clauses

Paramodulation: Replacing Equals with Equals

- Paramodulation interprets a clause as a conditional rewrite rule
  - One positive literal is applied
  - The others are considered as conditions
  - Conditions are not solved, but lazily added to the result

$$\text{(PM)}\ \frac{C \vee s \simeq t \qquad D \vee u \dot{\simeq} v}{(C \vee D \vee u[p \leftarrow t] \dot{\simeq} v)\sigma}\ \text{if}\ \sigma = mgu(u|_p, s)$$

- Notes:
  - The literal used for replacing must be positive
  - The literal in which a term is replaced can have any polarity
  - This is a generating inference: The new clause is added to the current set of clauses

(Positive) Factoring

- Factoring allow us to unify literals with the same polarity

$$\text{(PF)}\ \frac{C \vee s \simeq t \vee u \simeq v}{(C \vee s \simeq t)\sigma}\ \text{if}\ \sigma = mgu(s \simeq t, u \simeq v)$$

- Notes:
  - Note that $s \simeq t$ and $u \simeq v$ are unordered, hence must be tried in both directions for unification
  - This is a generating inference: The new clause is added to the current set of clauses

Exercise: A Proof Attempt

You should be able to prove the unsatisfiability of the following clause set with just the rules (ER), (PM), (EF). Can you?

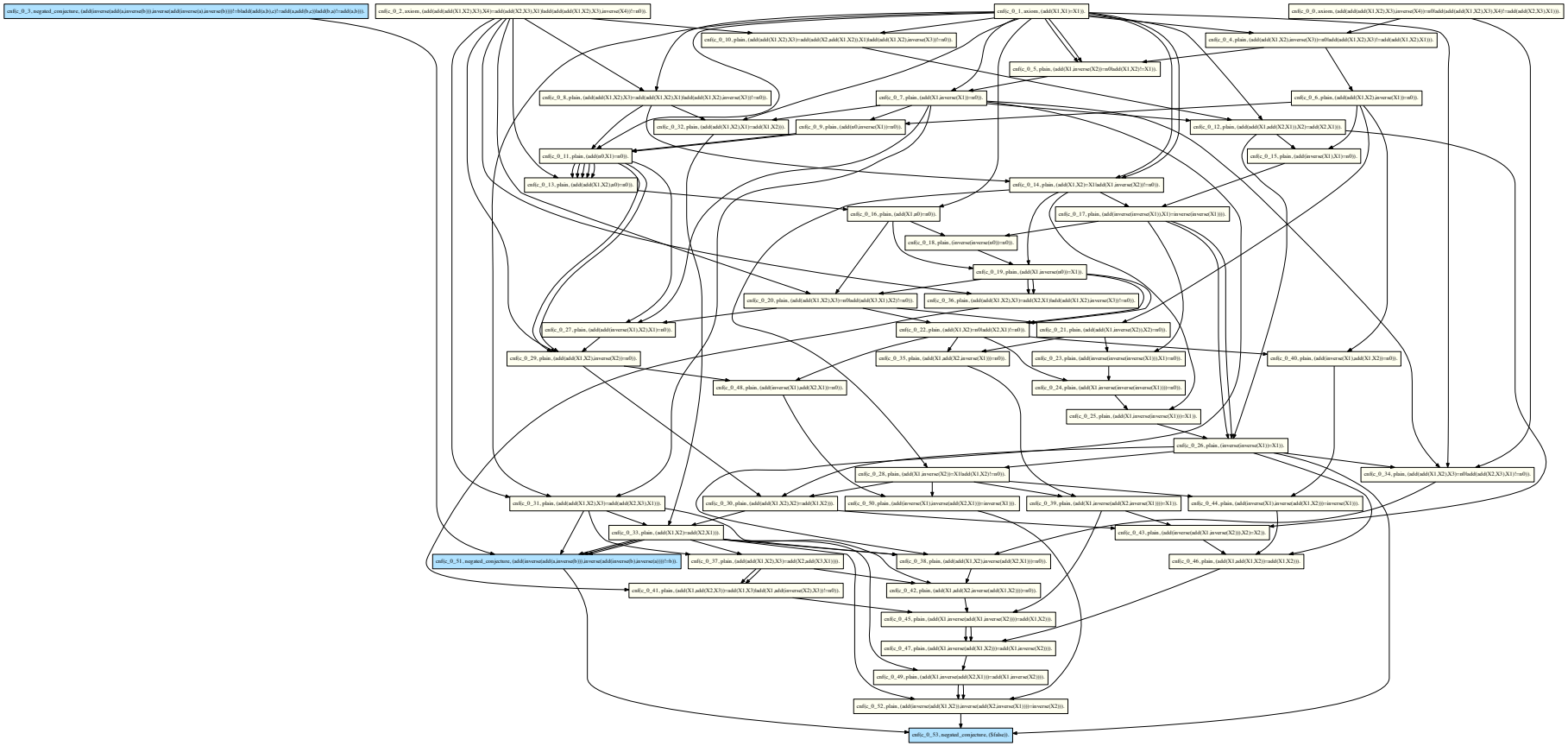
$$p(X, X) \simeq X$$

$$p(p(p(X, Y), Z), U) \not\simeq p(p(Y, Z), X) \vee p(p(p(X, Y), Z), i(U)) \simeq n0))$$

$$p(p(p(X, Y), Z), i(U)) \not\simeq n0 \vee p(p(p(X, Y), Z), U) \simeq p(p(Y, Z), X)))$$

$$p(i(p(a, i(b))), i(p(i(a), i(b)))) \not\simeq b$$
$$\vee p(p(a, b), c) \not\simeq p(a, p(b, c)) \vee p(b, a) \not\simeq p(a, b)$$

The Proof


cnfic_0_3, negated_conjecture, (add(inverse(add(a,inverse(b))),inverse(add(inverse(a),inverse(b))))!=b|add(d,add(a,b),c)!=add(a,add(b,c))|add(b,a)!=add(a,b)))).
cnfic_0_2, axiom, (add(add(add(X1,X2),X3),X4)=add(add(X2,X3),X1)|add(add(add(X1,X2),X3),inverse(X4))!=e0)).
cnfic_0_1, axiom, (add(X1,X1)=X1)).
cnfic_0_0, axiom, (add(add(add(X1,X2),X3),inverse(X4))=e0|add(add(add(X1,X2),X3),X4)!=add(add(X2,X3),X1))).
cnfic_0_10, plain, (add(add(X1,X2),X3)=add(add(X2,add(X1,X2)),X1)|add(add(X1,X2),inverse(X3))!=e0)).
cnfic_0_4, plain, (add(add(X1,X2),inverse(X3))=e0|add(add(X1,X2),X3)!=add(add(X1,X2),X1))).
cnfic_0_5, plain, (add(X1,inverse(X2))=e0|add(X1,X2)!=X1)).
cnfic_0_8, plain, (add(add(X1,X2),X3)=add(add(X1,X2),X1)|add(add(X1,X2),inverse(X3))!=e0)).
cnfic_0_7, plain, (add(X1,inverse(X1))=e0)).
cnfic_0_6, plain, (add(add(X1,X2),inverse(X1))=e0)).
cnfic_0_32, plain, (add(add(X1,X2),X1)=add(X1,X2))).
cnfic_0_9, plain, (add(e0,inverse(X1))=e0)).
cnfic_0_12, plain, (add(add(X1,add(X2,X1)),X2)=add(X2,X1))).
cnfic_0_11, plain, (add(e0,X1)=e0)).
cnfic_0_15, plain, (add(inverse(X1),X1)=e0)).
cnfic_0_13, plain, (add(add(add(X1,X2),e0)=e0)).
cnfic_0_14, plain, (add(X1,X2)=X1|add(X1,inverse(X2))!=e0)).
cnfic_0_16, plain, (add(X1,e0)=e0)).
cnfic_0_17, plain, (add(inverse(inverse(X1)),X1)=inverse(inverse(X1)))).
cnfic_0_18, plain, (inverse(inverse(e0))=e0)).
cnfic_0_19, plain, (add(X1,inverse(e0))=X1)).
cnfic_0_20, plain, (add(add(X1,X2),X3)=e0|add(add(X3,X1),X2)!=e0)).
cnfic_0_36, plain, (add(add(add(X1,X2),X3)=add(X2,X1)|add(add(X1,X2),inverse(X3))!=e0)).
cnfic_0_27, plain, (add(add(inverse(X1),X2),X1)=e0)).
cnfic_0_22, plain, (add(X1,X2)=e0|add(X2,X1)!=e0)).
cnfic_0_21, plain, (add(add(X1,inverse(X2)),X2)=e0)).
cnfic_0_29, plain, (add(add(X1,X2),inverse(X2))=e0)).
cnfic_0_35, plain, (add(X1,add(X2,inverse(X1)))=e0)).
cnfic_0_23, plain, (add(inverse(inverse(X1)),X1)=e0)).
cnfic_0_40, plain, (add(inverse(X1),add(X1,X2))=e0)).
cnfic_0_48, plain, (add(inverse(X1),add(X2,X1))=e0)).
cnfic_0_24, plain, (add(X1,inverse(inverse(inverse(X1))))=e0)).
cnfic_0_25, plain, (add(X1,inverse(inverse(X1)))=X1)).
cnfic_0_26, plain, (inverse(inverse(X1))=X1)).
cnfic_0_28, plain, (add(X1,inverse(X2))=X1|add(X1,X2)!=e0)).
cnfic_0_54, plain, (add(add(X1,X2),X3)=e0|add(add(X2,X3),X1)!=e0)).
cnfic_0_31, plain, (add(add(X1,X2),X3)=add(add(X2,X3),X1))).
cnfic_0_30, plain, (add(add(X1,X2),X2)=add(X1,X2))).
cnfic_0_50, plain, (add(inverse(X1),inverse(add(X2,X1)))=inverse(X1))).
cnfic_0_39, plain, (add(X1,inverse(add(X2,inverse(X1))))=X1)).
cnfic_0_44, plain, (add(inverse(X1),inverse(add(X1,X2)))=inverse(X1))).
cnfic_0_33, plain, (add(X1,X2)=add(X2,X1))).
cnfic_0_43, plain, (add(inverse(add(X1,inverse(X2))),X2)=X2)).
cnfic_0_51, negated_conjecture, (add(inverse(add(a,inverse(b))),inverse(add(inverse(b),inverse(a))))!=b)).
cnfic_0_37, plain, (add(add(X1,X2),X3)=add(X2,add(X3,X1)))).
cnfic_0_38, plain, (add(add(X1,X2),inverse(add(X2,X1)))=e0)).
cnfic_0_46, plain, (add(X1,add(X1,X2))=add(X1,X2))).
cnfic_0_41, plain, (add(X1,add(X2,X3))=add(X1,X3)|add(X1,add(inverse(X2),X3))!=e0)).
cnfic_0_42, plain, (add(X1,add(X2,inverse(add(X1,X2))))=e0)).
cnfic_0_45, plain, (add(X1,inverse(add(X1,inverse(X2))))=add(X1,X2))).
cnfic_0_47, plain, (add(X1,inverse(add(X1,X2)))=add(X1,inverse(X2)))).
cnfic_0_49, plain, (add(X1,inverse(add(X2,X1)))=add(X1,inverse(X2)))).
cnfic_0_52, plain, (add(inverse(add(X1,X2)),inverse(add(X2,inverse(X1))))=inverse(X2))).
cnfic_0_53, negated_conjecture, ($false)).

Paramodulation is Blind

- Fact: (PM), (ER) and (PF) together are a sound and complete inference system for equational reasoning
  - If used exhaustively, the will generate the empty clause from any unsatisfiable clause set
  - They will never generate the empty clause from a satisfiable clause set
- Problem: Paramodulation is too prolific
  - Every positive literal can be used as a rewrite rule
  - . . . in either direction (!)
  - Any literal can be rewritten at any position

Paramodulation is Blind

- Fact: (PM), (ER) and (PF) together are a sound and complete inference system for equational reasoning
  - If used exhaustively, the will generate the empty clause from any unsatisfiable clause set
  - They will never generate the empty clause from a satisfiable clause set
- Problem: Paramodulation is too prolific
  - Every positive literal can be used as a rewrite rule
  - . . . in either direction (!)
  - Any literal can be rewritten at any position

The search space explodes extremely fast!

$\Longrightarrow$

Only very simple proofs can be found

Superposition: Fixing Paramodulation

- Superposition restricts paramodulation by introducing orderings
  - Paramodulation can only be used on maximal terms in maximal literals
  - Only maximal instances of equations can be used as rewrite rules
  - Only potentially reducing rewrite steps can be made
- Side effect: Powerful redundancy criteria
  - In particular: Unconditional rewriting

Simplification Orderings

Definition: Simplification ordering

- A simplification ordering $>$ on $T(F, V)$ is a partial ordering $>$ with the following properties:
  - $>$ is stable under substitutions: $s > t$ implies $s\sigma > t\sigma$
  - $>$ is compatible with the term structure: $s > t$ implies $u[p \leftarrow s] > u[p \leftarrow t]$
  - $>$ is terminating
  - $>$ contains the subterm relation
- $>$ is a ground simplification ordering, if $>$ is total on ground terms

Simplification Orderings

Definition: Simplification ordering

- A simplification ordering $>$ on $T(F, V)$ is a partial ordering $>$ with the following properties:
  - $>$ is stable under substitutions: $s > t$ implies $s\sigma > t\sigma$
  - $>$ is compatible with the term structure: $s > t$ implies $u[p \leftarrow s] > u[p \leftarrow t]$
  - $>$ is terminating
  - $>$ contains the subterm relation
- $>$ is a ground simplification ordering, if $>$ is total on ground terms

Notes:

- $>$ cannot be total on all terms (consider $X > Y$)
- Examples of (families of) simplification orderings are the Knuth-Bendix-Ordering (KBO) and the Lexicographic Path Ordering (LPO)

Lifting Orderings

Let $>$ be a (ground) simplification ordering on terms. $>$ is lifted to literals via the multiset-extension:

- Remember:
  - We identify $s \simeq t$ with $\{\{s\}, \{t\}\}$
  - We identify $s \not\simeq t$ with $\{\{s, t\}\}$
- So $s \dot{\simeq} t > u \dot{\simeq} v$ if their respective multiset representations compare that way under $>>$

From Paramodulation to Superposition

- Term orderings allow us to restrict paramodulation and equality resolution to maximal literals
- Factoring can also be restricted, but needs to be slightly generalized to maintain completeness

(Ordered) Equality Resolution

$$\text{(ER)}\ \frac{C \vee s \not\approx t}{C\sigma}$$

if $\sigma = mgu(s, t)$ and $s \not\approx t$ is maximal in $(C \vee s \not\approx t)\sigma$

- Notes:
  - This restricts the previous version with the added maximality requirement
  - Equality resolution is generally harmless

Superposition

- Superposition restricts paramodulation with term orderings
- It comes in two versions for positive and negative target literals

(SN) $\dfrac{C \vee s \simeq t \qquad D \vee u \not\simeq v}{(C \vee D \vee u[p \leftarrow t] \not\simeq v)\sigma}$ if

  - $u|_p \notin V, \sigma = mgu(u|_p, s)$
  - $s\sigma \not\leq t\sigma$, $u\sigma \not\leq v\sigma$
  - $(s \simeq t)\sigma$ is strictly maximal in $(C \vee s \simeq t)\sigma$
  - $(u \not\simeq v)\sigma$ is maximal in $(D \vee u \not\simeq v)\sigma$

(SP) $\dfrac{C \vee s \simeq t \qquad D \vee u \simeq v}{(C \vee D \vee u[p \leftarrow t] \simeq v)\sigma}$ if

  - $u|_p \notin V, \sigma = mgu(u|_p, s)$
  - $s\sigma \not\leq t\sigma$, $u\sigma \not\leq v\sigma$
  - $(s \simeq t)\sigma$ is strictly maximal in $(C \vee s \simeq t)\sigma$
  - $(u \simeq v)\sigma$ is strictly maximal in $(D \vee u \not\simeq v)\sigma$

Equality Factoring

- Equality factoring generalizes positive factoring

  (EF) $$\frac{C \vee s \simeq t \vee u \simeq v}{(C \vee t \not\simeq v \vee u \simeq v)\sigma}$$

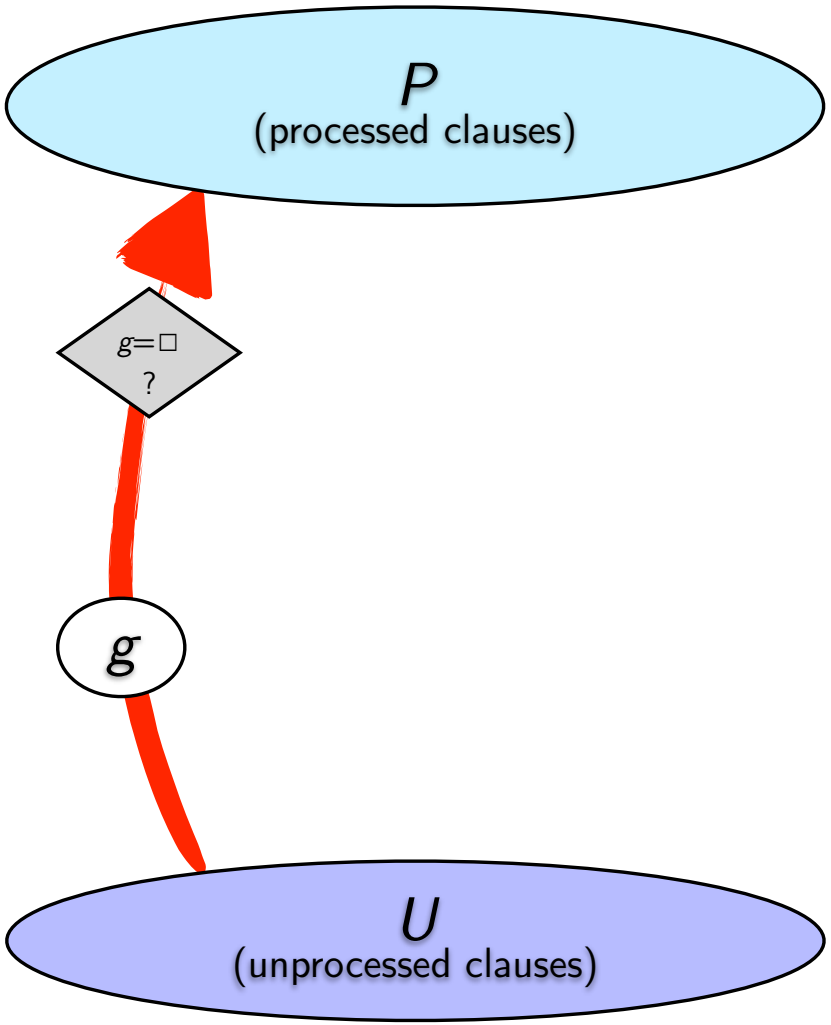
  if
  - $\sigma = mgu(s, u)$
  - $s\sigma \not\leq t\sigma$
  - $(s \simeq t)\sigma$ is maximal in $(C \vee s \simeq t \vee u \simeq v\sigma)$

Correctness

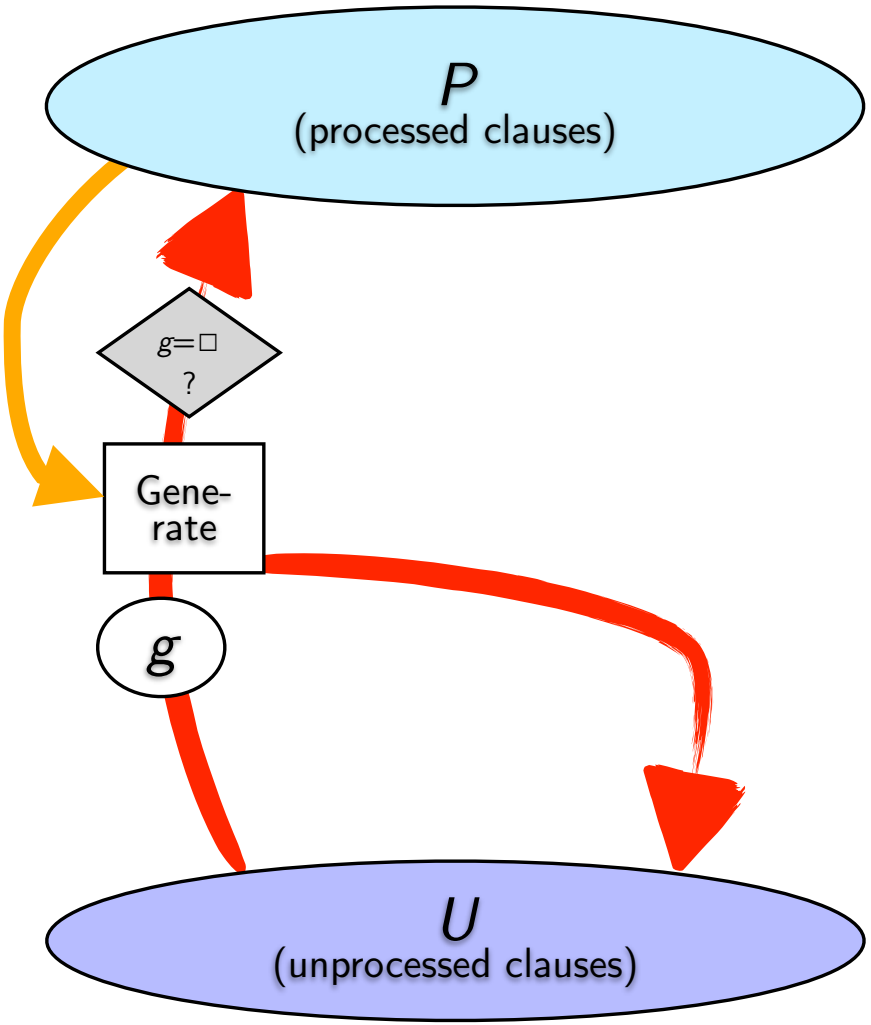
An inference system that contains just (ER), (SN), (SP), (EF) is sound and complete!

- For completeness, every possible inference must eventually be performed
- Implementation e.g.:
  - Level saturation
  - Given-clause algorithm
- In practice, redundancy elimination is critical!

A Basic Proof Procedure



A Basic Proof Procedure



Redundancy

- Superposition comes with a powerful redundancy concept:
  - A clause is redundant, if every ground instance is implied by smaller ground instances of other clauses
    - The ordering lifts the literal ordering via the multiset-construction (again!) to the clause level
  - Redundant clauses can be removed without affecting completeness

Redundancy

- Superposition comes with a powerful redundancy concept:
  - A clause is redundant, if every ground instance is implied by smaller ground instances of other clauses
    - The ordering lifts the literal ordering via the multiset-construction (again!) to the clause level
  - Redundant clauses can be removed without affecting completeness
- Note: Simplification can add new, smaller clauses to make a previous clause redundant!

Tautologies

- Tautologies are clauses that are true under any implementation
  - Example: $a \not\simeq b \vee a \simeq b$
  - Example: $a \not\simeq b \vee a \simeq a$
- Tautologies are implied by the empty set of clauses
  - Hence they can be removed per the general redundancy criterion

Intra-Clause Simplifications

- A clause that is a subset of another clause is smaller
  - Any inference that removes a literal replaces a clause by a smaller one that makes it redundant
- Examples:
  - Removal of duplicate literals
  - Removal of trivially false literals ($s \not\simeq s$)

Rewriting

- Rewriting replaces a term with a smaller term
  - The original clause is implied by the new one and the rewriting one
  - Crucial for efficient proof procedures!

$$\text{(RP)}\ \frac{s \simeq t \qquad u \simeq v \vee C}{s \simeq t \qquad u[p \leftarrow t\sigma] \simeq v \vee C}$$

if

- $u|_p = s\sigma$, $s\sigma > t\sigma$
- $u \simeq v$ is not maximal or $u \not> v$ or $p \neq \lambda$ or σ is not a variable renaming

$$\text{(RN)}\ \frac{s \simeq t \qquad u \not\simeq v \vee C}{s \simeq t \qquad u[p \leftarrow t\sigma] \not\simeq v \vee C}$$

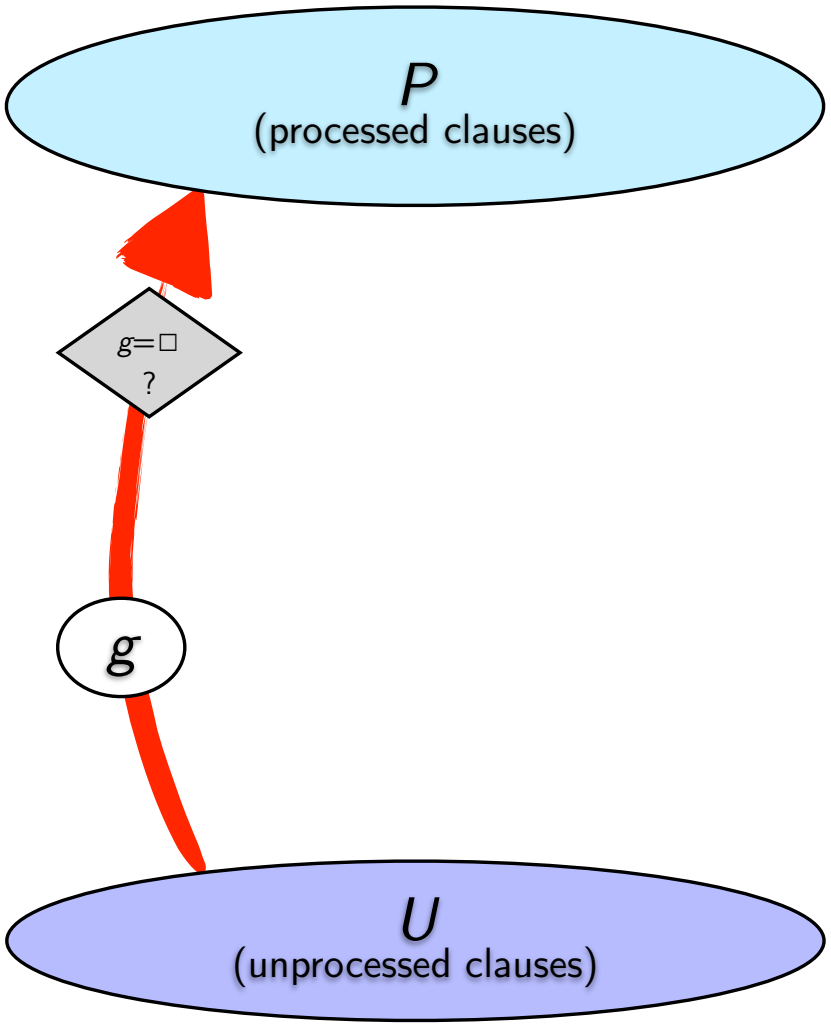
if $u|_p = s\sigma$ and $s\sigma > t\sigma$

Additional Simplifications

- Subsumption
- Contextual literal cutting a.k.a clause simplification a.k.a. subsumption resolution
- Condensation
- AC redundancy elimination

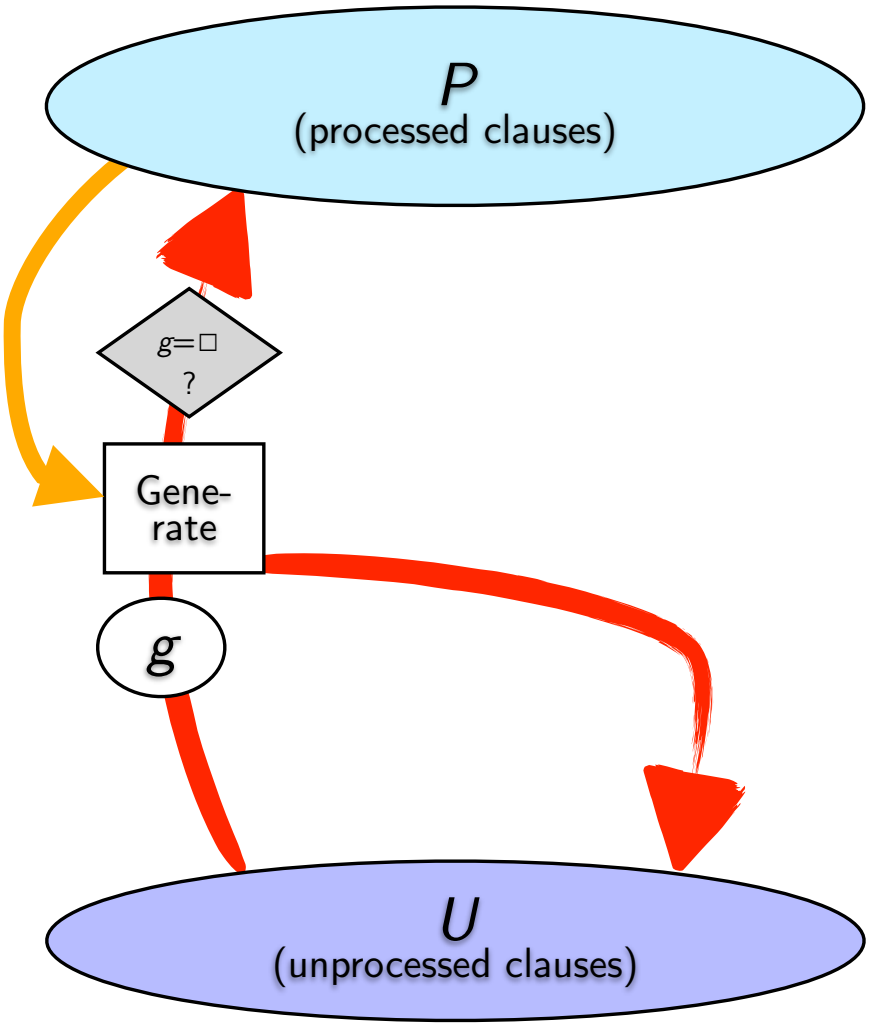
Implementation

The Given-Clause Algorithm



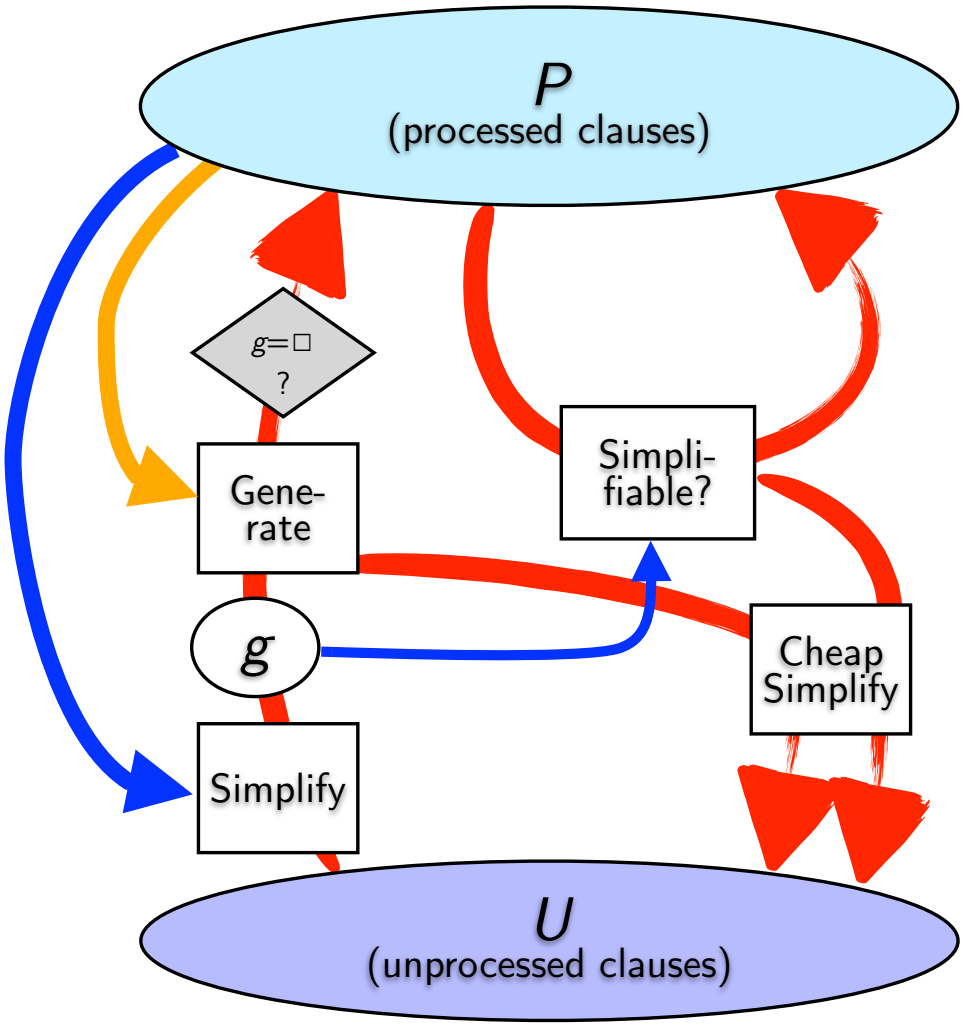
- Aim: Move everything from U to P

The Given-Clause Algorithm



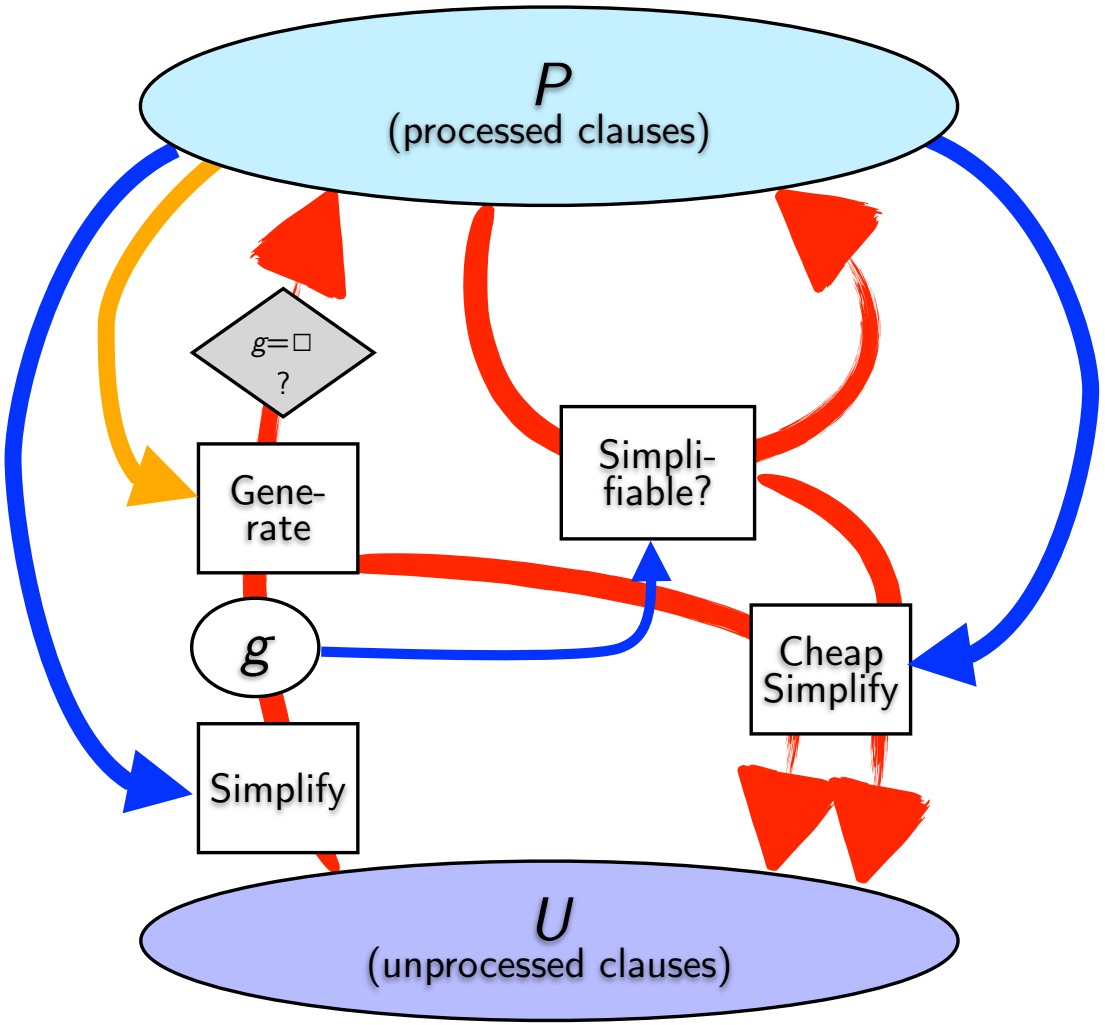
- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed

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

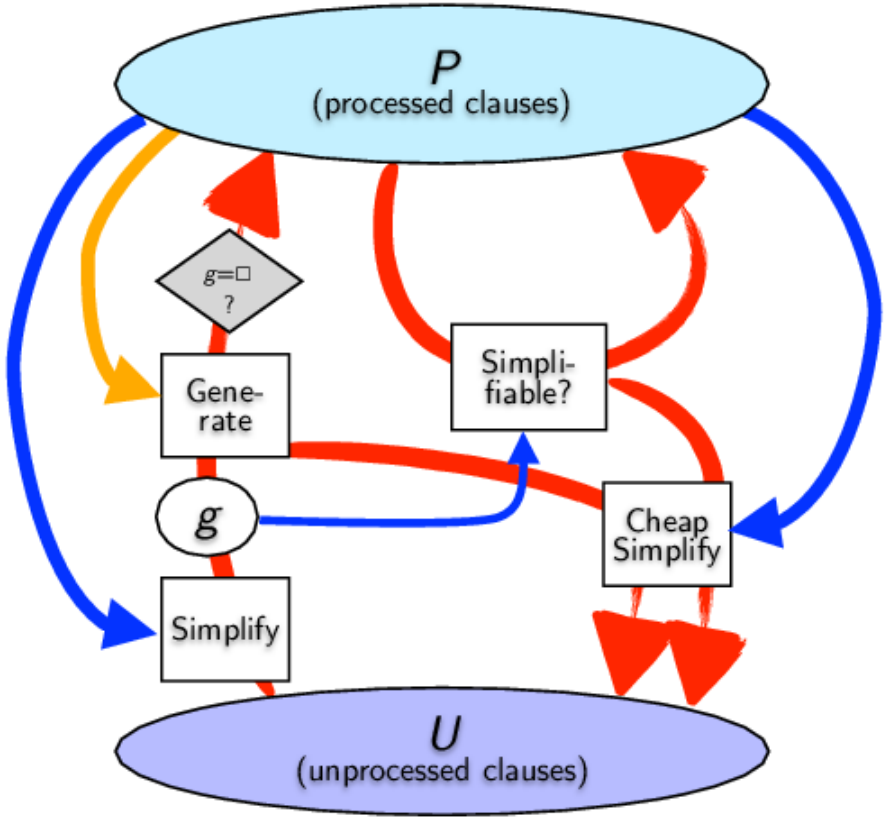


- Aim: Move everything from U to P
- Invariant: All generating inferences with premises from P have been performed
- Invariant: P is interreduced
- Clauses added to U are simplified with respect to P

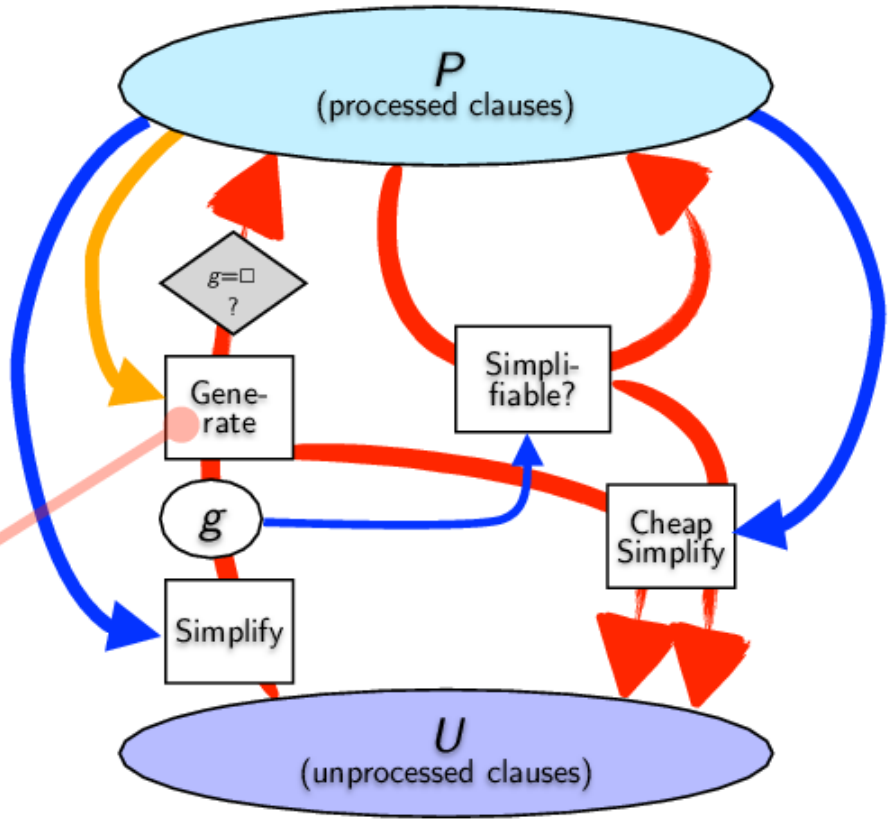
The Given-Clause Loop

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

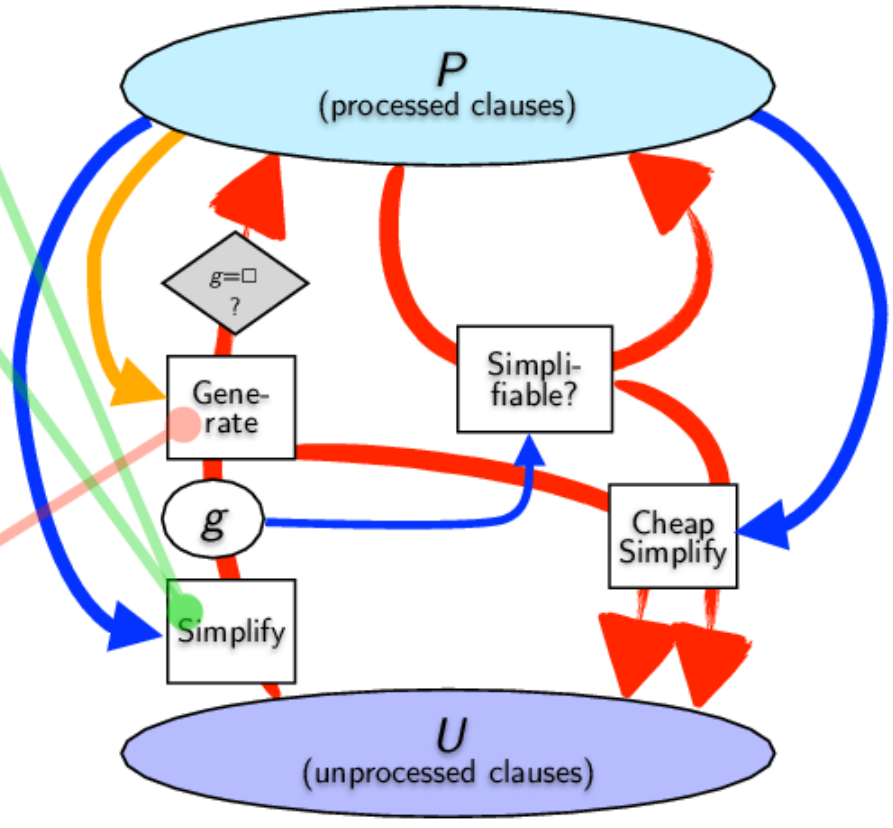
while $U \neq \{\}$
  $g = \text{delete\_best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \square$
    SUCCESS, Proof found
  if g is not redundant w.r.t. P
    $P = P \backslash \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \backslash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



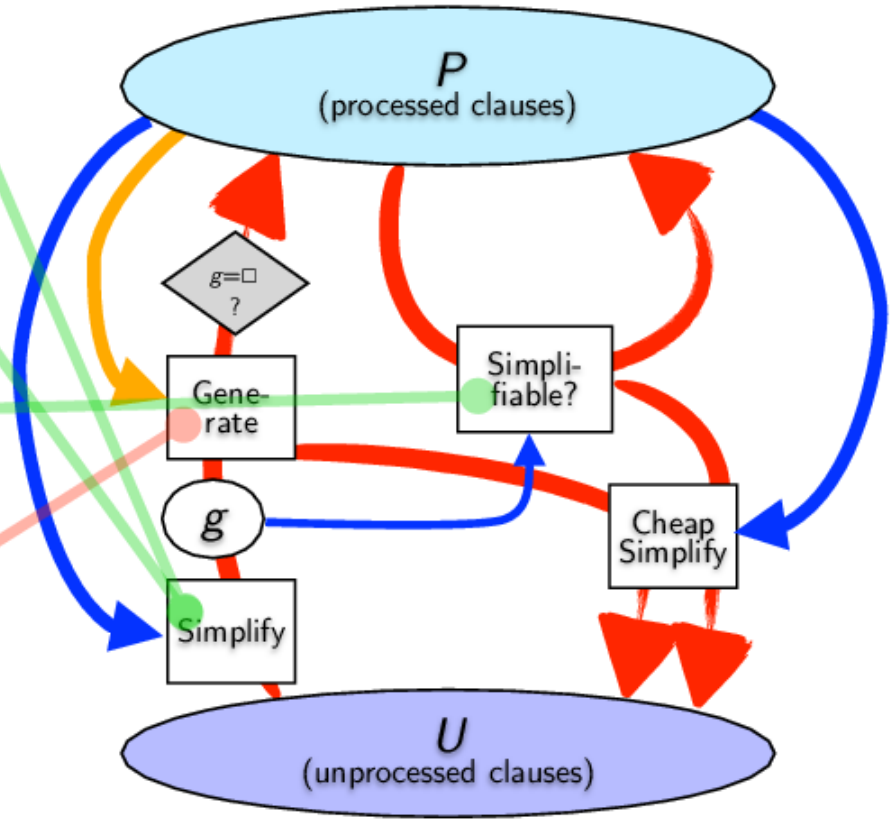
while $U \neq \{\}$
  $g = \text{delete\_best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \square$
    SUCCESS, Proof found
  if g is not redundant w.r.t. P
    $P = P \backslash \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \backslash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



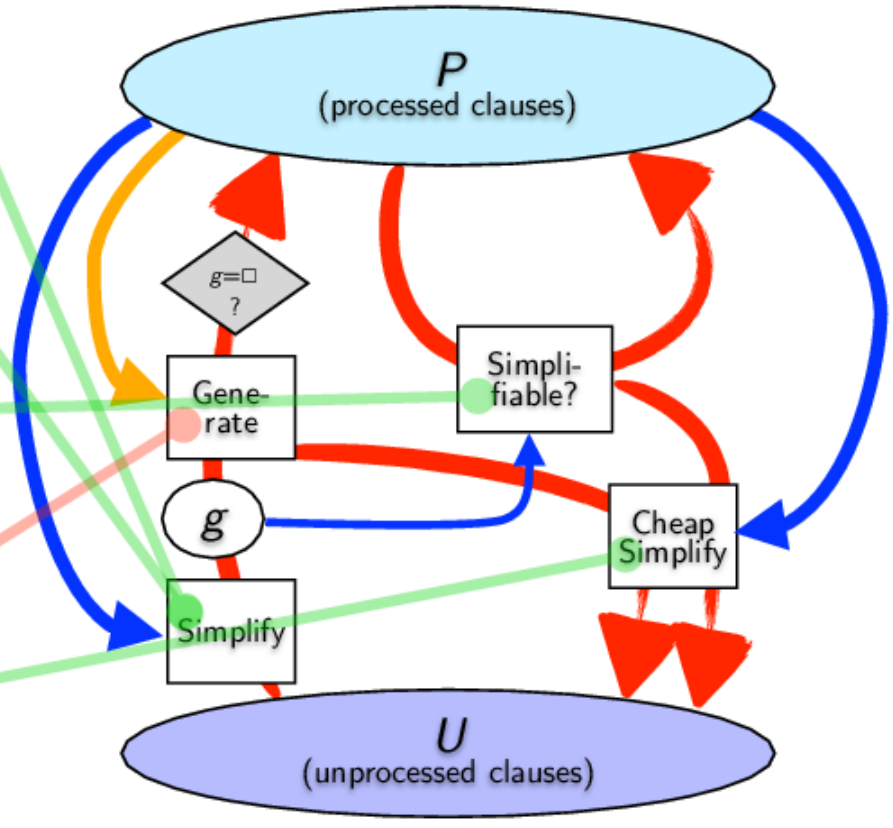
while $U \neq \{\}$
  $g = \text{delete\_best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \square$
    SUCCESS, Proof found
  if g is not redundant w.r.t. P
    $P = P \backslash \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \backslash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



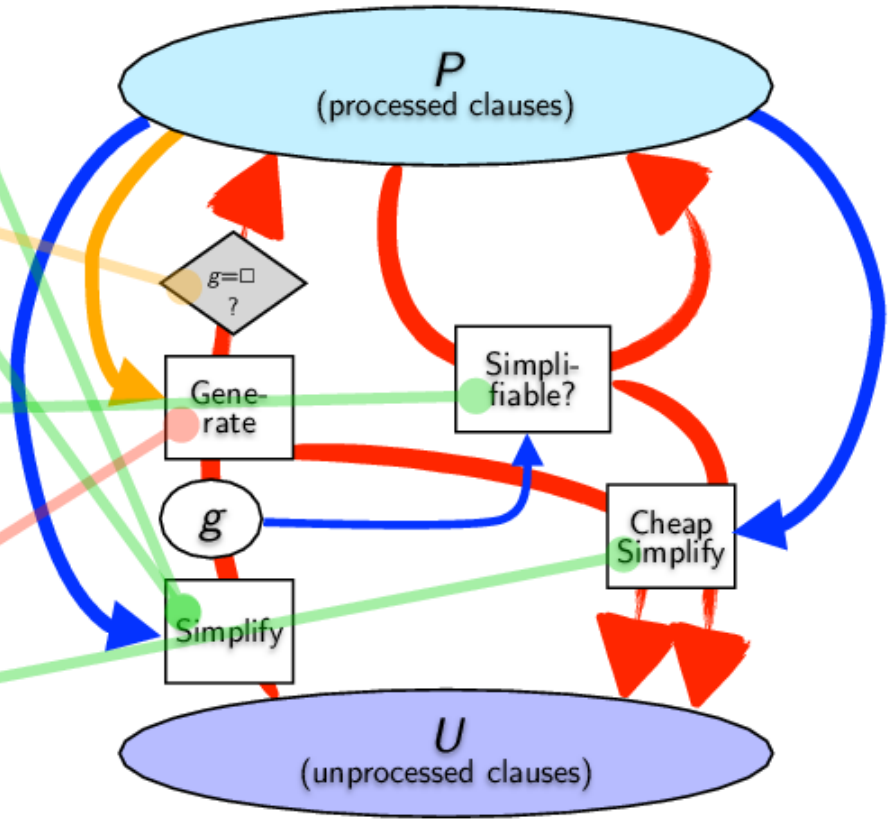
while $U \neq \{\}$
  $g = \text{delete\_best}(U)$
  $g = \text{simplify}(g, P)$
  if $g == \square$
    SUCCESS, Proof found
  if g is not redundant w.r.t. P
    $P = P \backslash \{c \in P \mid c \text{ redundant w.r.t. } g\}$
    $T = \{c \in P \mid c \text{ simplifiable with } g\}$
    $P = (P \backslash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



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



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



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

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

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

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

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

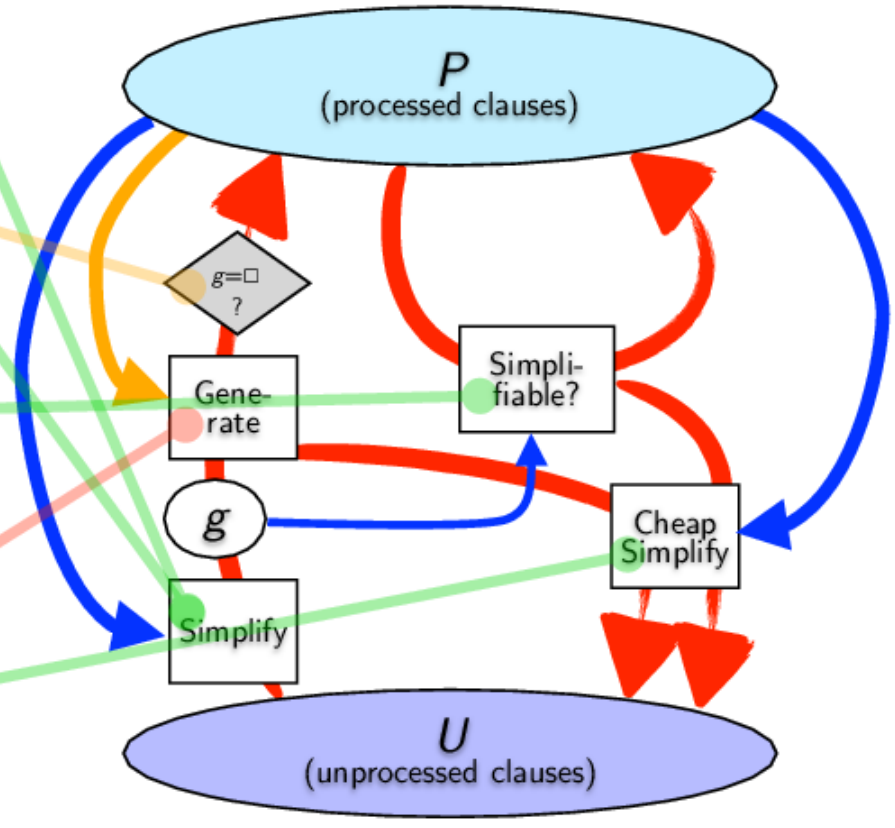
foreach $c \in T$

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

if c is not trivial

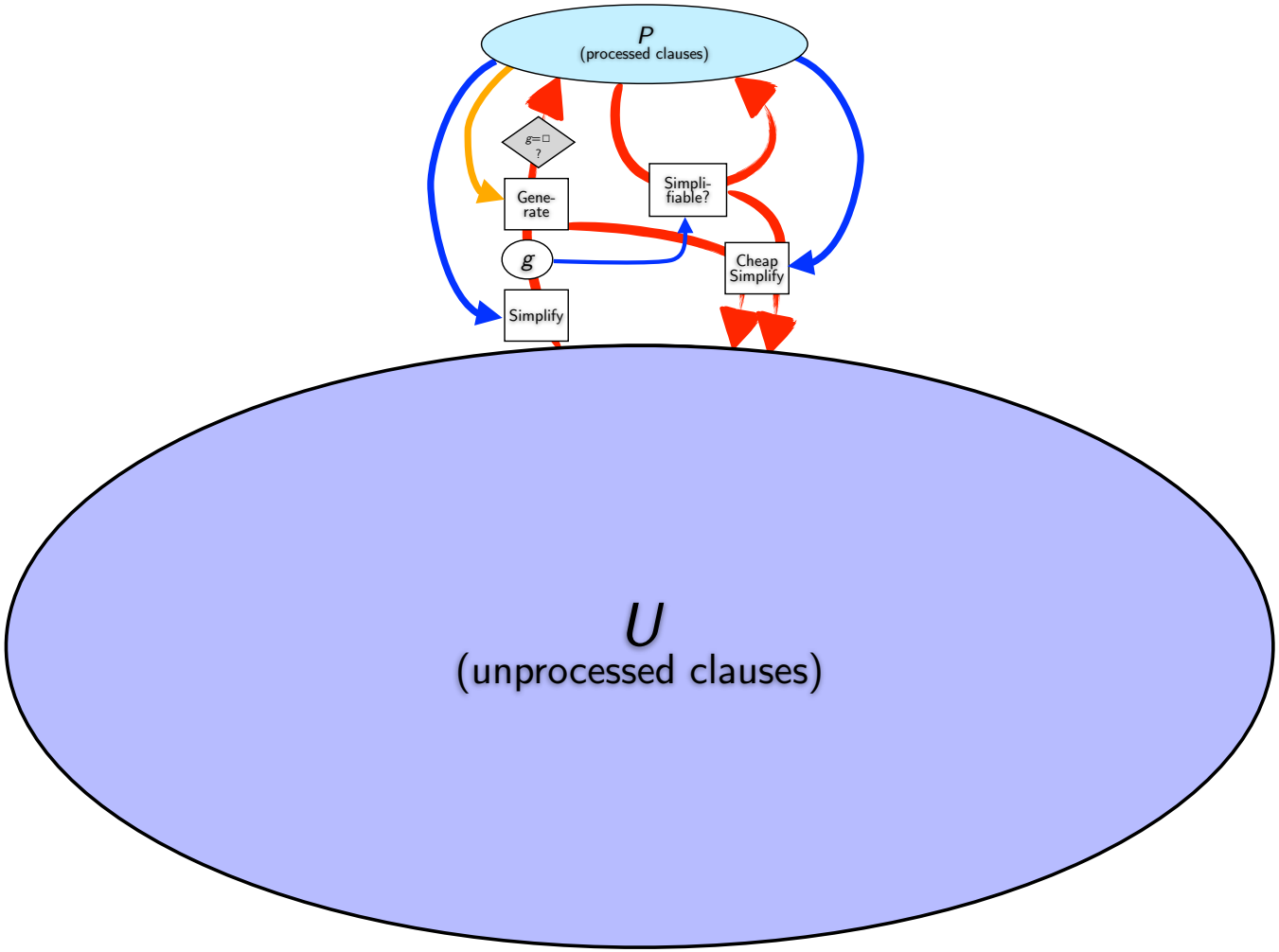
$U = U \cup \{c\}$

SUCCESS, original U is satisfiable



Some Numbers

```
# Initial clauses in saturation          : 4
# Processed clauses                      : 4808
# ...of these trivial                    : 385
# ...subsumed                            : 3976
# ...remaining for further processing    : 447
# Other redundant clauses eliminated     : 604
# Generated clauses                      : 46595
# ...of the previous two non-trivial     : 33866
# Paramodulations                        : 45989
# Factorizations                         : 0
# Equation resolutions                   : 606
# Current number of processed clauses    : 306
# Current number of unprocessed clauses: 25454
```


P
(processed clauses)
g=□ ?
Gene-
rate
g
Simplify
Simpli-
fiable?
Cheap
Simplify
U
(unprocessed clauses)

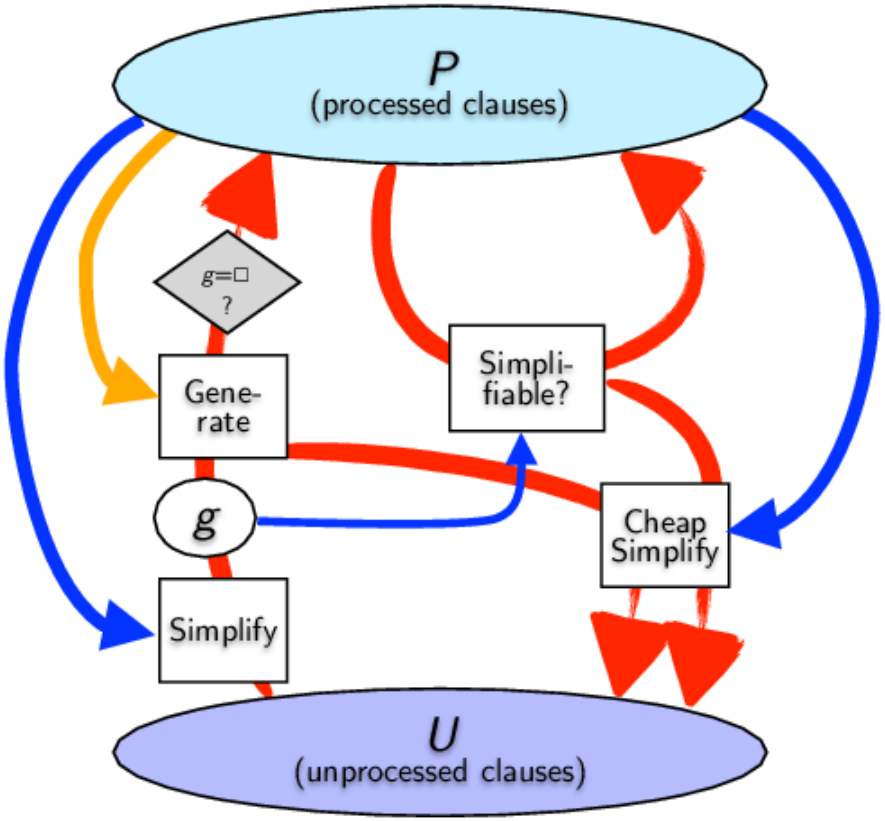
Expensive operations

- Paramodulation/Superposition: Find partners for g
  - Indexing with e.g. Fingerprint Indexing or Discrimination Tree Indexing
- Forward rewriting: Find matching equations for g and newly generated clauses
  - Biggest single cost in many provers!
  - Use Discrimination Tree Indexing
- Backward rewriting: Find clauses rewritable with g
  - Use Path Indexing or Fingerprint Indexing
- Subsumption: Find clauses subsuming or subsumed by g
  - Use Feature Vector Indexing

Search control

Clause selection

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



Basic Approaches

- Symbol counting
  - Pick smallest clause in P
  - $|\{f(X) \neq a, P(a) \neq \$true, g(Y) = f(a)\}| = 10$
- FIFO
  - Always pick oldest clause in P
- Flexible weighting
  - Symbol counting, but give different weight to different symbols
  - E.g. lower weight to symbols from goal!
- Combinations
  - Interleave different schemes

Influences on E

- DISCOUNT
  - Different *experts* (heuristic evaluation functions)
  - Only one *expert* per saturation phase
- Otter
  - Interleaves size/age selection
  - Larry Wos: *"The optimal pick-given ration is 5"*
- Waldmeister
  - Larry is right in general, wrong in detail

The Second System Effect

The general tendency is to over-design the second system, using all the ideas and frills that were cautiously sidetracked on the first one. The result, as Ovid says, is a "big pile."

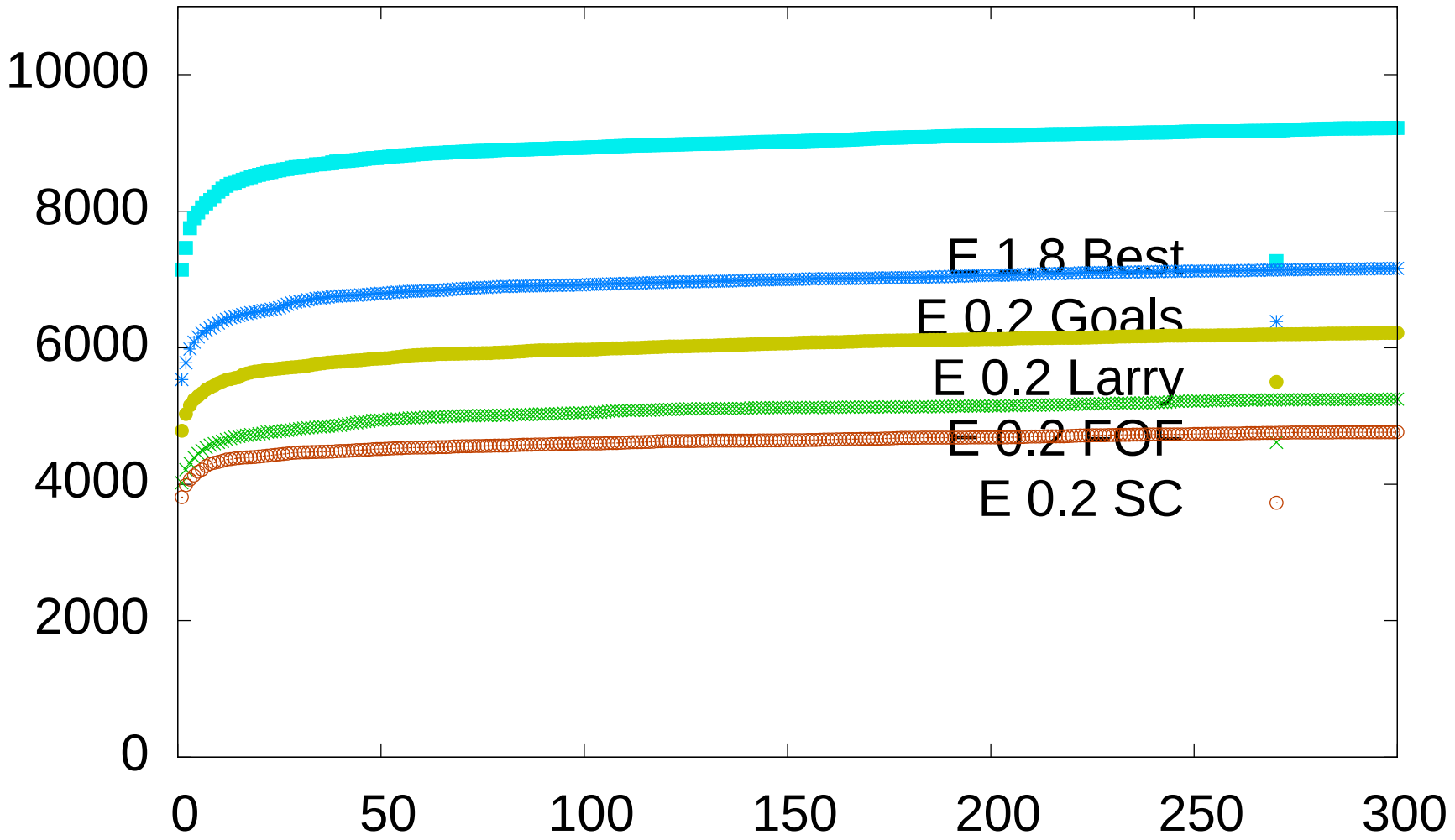
— Frederick P. Brooks, Jr.

Given-Clause Selection in E

- Domain Specific Language (DSL) for clause selection scheme
- Arbitrary number of queues
- Each queue ordered by:
  - Unparameterized priority function
  - Parameterized heuristic evaluation function
- Clauses picked using weighted round-robbin scheme
  - Example:
    - 4 clauses from queue 1
    - 2 clauses from queue 2
    - 2 clauses from queue 3
    - Start over at queue 1

Second-system effect gone wild

The Influence of Clause Selection



Exercise: Playing with E

- Download and install E from `http://www.eprover.org`
- Run it some of the provided examples
- Play with the parameters
- Take a guided tour through the source

Conclusion

Conclusion

- After 25 years, superposition is still the best general-purpose calculus for first-order logic with equality
- Good implementations are available
  - E
  - SPASS
  - Prover 9
  - Vampire
- . . . but further improvements are possible
- Search control heuristics are still crucial

Conclusion

- After 25 years, superposition is still the best general-purpose calculus for first-order logic with equality
- Good implementations are available
  - E
  - SPASS
  - Prover 9
  - Vampire
- . . . but further improvements are possible
- Search control heuristics are still crucial

And part of the future will be part of the afternoon session!

References

References

L. Bachmair and H. Ganzinger.

On Restrictions of Ordered Paramodulation with Simplification.

In M.E. Stickel, editor, *Proc. of the 10th CADE, Kaiserslautern*, volume 449 of *LNAI*, pages 427—441. Springer, 1990.

L. Bachmair and H. Ganzinger.

Rewrite-Based Equational Theorem Proving with Selection and Simplification.

Journal of Logic and Computation, 3(4):217–247, 1994.

R. Nieuwenhuis and A. Rubio.

Paramodulation-Based Theorem Proving.

In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science and MIT Press, 2001.

A. Nonnengart and C. Weidenbach.

Computing Small Clause Normal Forms.

In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 5, pages 335–367. Elsevier Science and MIT Press, 2001.

Stephan Schulz.

System Description: E 1.8.

In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*. Springer, 2013.

Image Sources

- Jaques Herbrand: Public Domain via Wikimedia
- Clipart: `http://openclipart.org`