AC simplifications and closure redundancy in the superposition calculus

André Duarte Konstantin Korovin

andrepd@protonmail.com

8th September 2021

The University of Manchester

Introduction

First-order automated theorem provers are powerful tools for general-purpose problem solving, with many applications:

- Mathematics,
- Software verification, hardware verification,
- Knowledge-base reasoning and ontologies,
- Routines in higher-order provers
- Etc.

Introduction

First-order automated theorem provers are powerful tools for general-purpose problem solving, with many applications:

- Mathematics,
- Software verification, hardware verification,
- Knowledge-base reasoning and ontologies,
- Routines in higher-order provers
- Etc.

Main advantage

- Very general and expressive

Disadvantage

- Struggles in certain domains and even in simple problems

Associativity-commutativity

A binary function '$+$' is associative-commutative (AC) if

$$x + y \approx y + x \qquad (x + y) + z \approx x + (y + z)$$

Ubiquitous, contained in important theories such as arithmetic, etc.

Associativity-commutativity

A binary function '$+$' is associative-commutative (AC) if

$$x + y \approx y + x \qquad (x + y) + z \approx x + (y + z)$$

Ubiquitous, contained in important theories such as arithmetic, etc.

Problem: under superposition, these equations recombine to produce an infinite number of consequences

$$x + (y + z) \approx z + (x + y)$$
$$x + (y + z) \approx y + (z + x)$$
$$x + (y + z) \approx (y + z) + x$$
$$x + (y + (z + w)) \approx (y + (z + w)) + x$$
$$\dots$$

Associativity-commutativity

A binary function '$+$' is associative-commutative (AC) if

$$x + y \approx y + x \qquad (x + y) + z \approx x + (y + z)$$

Ubiquitous, contained in important theories such as arithmetic, etc.

Problem: under superposition, these equations recombine to produce an infinite number of consequences

$$x + (y + z) \approx z + (x + y)$$
$$x + (y + z) \approx y + (z + x)$$
$$x + (y + z) \approx (y + z) + x$$
$$x + (y + (z + w)) \approx (y + (z + w)) + x$$
$$\dots$$

(more precisely, there are $n! \times \left(\frac{(2n-2)!}{(n-1)!n!}\right)^2$ equations for n variables)

Ground joinability

Due to its importance, AC has been the object of active research for decades.

Ground joinability

Due to its importance, AC has been the object of active research for decades.

Knuth-Bendix completion–based provers use ground joinability criteria to delete equations $s \approx t$ where s and t are equal modulo AC [Martin *et al* 1990, Avenhaus *et al* 2003]

Examples: Waldmeister, Twee, etc.

Ground joinability

Due to its importance, AC has been the object of active research for decades.

Knuth-Bendix completion–based provers use ground joinability criteria to delete equations $s \approx t$ where s and t are equal modulo AC [Martin *et al* 1990, Avenhaus *et al* 2003]

Examples: Waldmeister, Twee, etc.
Very good performance!

Ground joinability

Due to its importance, AC has been the object of active research for decades.

Knuth-Bendix completion–based provers use ground joinability criteria to delete equations $s \approx t$ where s and t are equal modulo AC [Martin *et al* 1990, Avenhaus *et al* 2003]

Examples: Waldmeister, Twee, etc.
Very good performance!

Limitations: known proofs apply to Knuth-Bendix completion (i.e. unit equality only) and are based on the technique of proof orderings.

There existed no proof for full clausal first-order logic, up to now.

Results

Main results of this paper:

- New notion of *closure redundancy*, which can be used to justify AC simplifications;

Results

Main results of this paper:

- New notion of *closure redundancy*, which can be used to justify AC simplifications;
- Proof that the superposition calculus is refutationally complete wrt. "saturation up to closure redundancy";

Results

Main results of this paper:

- New notion of *closure redundancy*, which can be used to justify AC simplifications;
- Proof that the superposition calculus is refutationally complete wrt. "saturation up to closure redundancy";
- As corollaries, that ground joinability is a redundancy for the superposition calculus, as well as stronger AC normalisation and encompassment demodulation;

Results

Main results of this paper:

- New notion of *closure redundancy*, which can be used to justify AC simplifications;
- Proof that the superposition calculus is refutationally complete wrt. "saturation up to closure redundancy";
- As corollaries, that ground joinability is a redundancy for the superposition calculus, as well as stronger AC normalisation and encompassment demodulation;
- That the proof also opens up the door for further AC simplifications in the future (currently being researched)

Superposition

Superposition is comprised of the following inference rules:

$$\text{Superposition} \quad \frac{\underline{l \approx r} \vee C \quad \underline{s[u] \mathrel{\dot{\approx}} t} \vee D}{(s[u \mapsto r] \mathrel{\dot{\approx}} t \vee C \vee D)\theta}, \quad \begin{array}{l}\text{where } \theta = \mathrm{mgu}(l, u),\\ l\theta \not\preceq r\theta,\ s\theta \not\preceq t\theta,\\ \text{and } s \text{ not a variable,}\end{array}$$

$$\text{Eq. Resolution} \quad \frac{\underline{s \not\approx t} \vee C}{C\theta}, \quad \text{where } \theta = \mathrm{mgu}(s, t),$$

$$\text{Eq. Factoring} \quad \frac{\underline{s \approx t} \vee \underline{s' \approx t'} \vee C}{(s \approx t \vee t \not\approx t' \vee C)\theta}, \quad \begin{array}{l}\text{where } \theta = \mathrm{mgu}(s, s'),\\ s\theta \not\preceq t\theta \text{ and } t\theta \not\preceq t'\theta,\end{array}$$

Superposition

Superposition is comprised of the following inference rules:

$$\text{Superposition} \quad \frac{\underline{l \approx r} \vee C \quad \underline{s[u] \mathrel{\dot{\approx}} t} \vee D}{(s[u \mapsto r] \mathrel{\dot{\approx}} t \vee C \vee D)\theta}, \quad \begin{array}{l}\text{where } \theta = \mathrm{mgu}(l, u),\\ l\theta \not\preceq r\theta,\ s\theta \not\preceq t\theta,\\ \text{and } s \text{ not a variable,}\end{array}$$

$$\text{Eq. Resolution} \quad \frac{\underline{s \not\approx t} \vee C}{C\theta}, \quad \text{where } \theta = \mathrm{mgu}(s, t),$$

$$\text{Eq. Factoring} \quad \frac{\underline{s \approx t} \vee \underline{s' \approx t'} \vee C}{(s \approx t \vee t \not\approx t' \vee C)\theta}, \quad \begin{array}{l}\text{where } \theta = \mathrm{mgu}(s, s'),\\ s\theta \not\preceq t\theta \text{ and } t\theta \not\preceq t'\theta,\end{array}$$

These rules, when applied exhaustively, are refutationally complete: if a set of clauses is unsatisfiable then there is a refutation in finite steps, if it is satisfiable, it may stop or loop forever.

Superposition

But we can also show that there are certain simplification rules that do not break completeness.

Superposition

But we can also show that there are certain simplification rules that do not break completeness.

$$\text{Tautology} \qquad \frac{\cancel{s \approx s \vee C}}{}$$

$$\text{Eq. resolution} \qquad \frac{\cancel{s \not\approx s \vee C}}{C}$$

Superposition

But we can also show that there are certain simplification rules that do not break completeness.

Tautology $\qquad \dfrac{\cancel{s \approx s \vee C}}{}$

Eq. resolution $\qquad \dfrac{\cancel{s \not\approx s \vee C}}{C}$

Subsumption $\qquad \dfrac{\cancel{C\theta} \quad C}{} \qquad \dfrac{\cancel{C \vee D} \quad C}{}$

Superposition

But we can also show that there are certain simplification rules that do not break completeness.

Tautology $$\frac{\cancel{s \approx s \vee C}}{}$$

Eq. resolution $$\frac{\cancel{s \not\approx s \vee C}}{C}$$

Subsumption $$\frac{\cancel{C\theta} \quad C}{} \qquad \frac{\cancel{C \vee D} \quad C}{}$$

Demodulation $$\frac{l \approx r \quad \cancel{C[l\theta]}}{C[l\theta \to r\theta]}, \quad \text{where } l\theta \succ r\theta, \text{ and } l\theta \approx r\theta \prec C[l\theta]$$

Superposition

But we can also show that there are certain simplification rules that do not break completeness.

Tautology $\dfrac{\cancel{s \approx s \vee C}}{}$

Eq. resolution $\dfrac{\cancel{s \not\approx s \vee C}}{C}$

Subsumption $\dfrac{\cancel{C\theta} \quad C}{} \qquad \dfrac{\cancel{C \vee D} \quad C}{}$

Demodulation $\dfrac{l \approx r \quad \cancel{C[l\theta]}}{C[l\theta \to r\theta]}$, where $l\theta \succ r\theta$, and $l\theta \approx r\theta \prec C[l\theta]$

...

Superposition — Model construction

Proof of completeness: inductive model construction [Bachmair Ganzinger].

Superposition — Model construction

Proof of completeness: inductive model construction [Bachmair Ganzinger].

Sketch (given a saturated set of clauses S):

- Fix a simplification ordering $\succ$ on clauses,

Superposition — Model construction

Proof of completeness: inductive model construction [Bachmair Ganzinger].

Sketch (given a saturated set of clauses S):

- Fix a simplification ordering $\succ$ on clauses,
- Take the set G of all ground instances of clauses in S,

Superposition — Model construction

Proof of completeness: inductive model construction [Bachmair Ganzinger].

Sketch (given a saturated set of clauses S):

- Fix a simplification ordering $\succ$ on clauses,
- Take the set G of all ground instances of clauses in S,
- Recursively define an interpretation for G,

Superposition — Model construction

Proof of completeness: inductive model construction [Bachmair Ganzinger].

Sketch (given a saturated set of clauses S):

- Fix a simplification ordering $\succ$ on clauses,
- Take the set G of all ground instances of clauses in S,
- Recursively define an interpretation for G,
- Prove by induction: this interpretation is a model for G.

Superposition — Model construction

Proof of completeness: inductive model construction [Bachmair Ganzinger].

Sketch (given a saturated set of clauses S):

- Fix a simplification ordering $\succ$ on clauses,
- Take the set G of all ground instances of clauses in S,
- Recursively define an interpretation for G,
- Prove by induction: this interpretation is a model for G.

Define saturation thus:

- A set is saturated if there are no inferences with premises in the set whose conclusion is not also in the set.

Superposition — Model construction

Proof of completeness: inductive model construction [Bachmair Ganzinger].

Sketch (given a saturated-up-to-redundancy set of clauses S):

- Fix a simplification ordering $\succ$ on clauses,
- Take the set G of all ground instances of clauses in S,
- Recursively define an interpretation for G,
- Prove by induction: this interpretation is a model for G.

Define saturation up to redundancy thus:

- A set is saturated if there are no non-redundant inferences with non-redundant premises in the set.

Superposition — Model construction

Proof of completeness: inductive model construction [Bachmair Ganzinger].

Sketch (given a saturated-up-to-redundancy set of clauses S):

- Fix a simplification ordering $\succ$ on clauses,
- Take the set G of all ground instances of clauses in S,
- Recursively define an interpretation for G,
- Prove by induction: this interpretation is a model for G.

Define saturation up to redundancy thus:

- A set is saturated if there are no non-redundant inferences with non-redundant premises in the set.

Redundant clause (in S)

All ground instances follow from smaller clauses in G.

Redundant inference (in S)

For all ground instances, conclusion follows from clauses in G smaller than the maximal premise.

Superposition — Closures

Problem: standard notion of redundancy doesn't support ground joinability.

Superposition — Closures

Problem: standard notion of redundancy doesn't support ground joinability.

Example: rewriting $f(b) + (a + c) \approx c$
to $a + (f(b) + c) \approx c$
via $x + (y + z) \approx y + (x + z)$

Superposition — Closures

Problem: standard notion of redundancy doesn't support ground joinability.

Example: rewriting $f(b) + (a + c) \approx c$
to $a + (f(b) + c) \approx c$
via $x + (y + z) \approx y + (x + z)$

Must be $\prec_c$ than deleted clause

Superposition — Closures

Problem: standard notion of redundancy doesn't support ground joinability.

Example: rewriting $f(b) + (a + c) \approx c$
to $a + (f(b) + c) \approx c$
via $x + (y + z) \approx y + (x + z)$

Must be $\prec_c$ than deleted clause

$$f(b) + (a + c) \approx c \quad \prec_c \quad f(b) + (a + c) \approx a + (f(b) + c)$$

Superposition — Closures

Problem: standard notion of redundancy doesn't support ground joinability.

Example: rewriting $f(b) + (a + c) \approx c$
to $a + (f(b) + c) \approx c$
via $x + (y + z) \approx y + (x + z)$

Must be $\prec_c$ than deleted clause

$$f(b) + (a + c) \approx c \quad \prec_c \quad f(b) + (a + c) \approx a + (f(b) + c)$$

Solution: refine the notion of ground instance to ground closure, and define an ordering where more general is smaller.

Superposition — Closures

Problem: standard notion of redundancy doesn't support ground joinability.

Example: rewriting $f(b) + (a + c) \approx c$
to $a + (f(b) + c) \approx c$
via $x + (y + z) \approx y + (x + z)$

Must be $\prec_c$ than deleted clause

$$f(b) + (a + c) \approx c \quad \prec_c \quad f(b) + (a + c) \approx a + (f(b) + c)$$

Solution: refine the notion of ground instance to ground closure, and define an ordering where more general is smaller.

Closure:

Pair of term/literal/clause and grounding substitution: $t \cdot \theta$.
Example: of $f(x, b)$, instance $f(a, b)$ becomes $f(x, b) \cdot x/a$.
Ordering: $s \cdot \sigma \succ_{tc} t \cdot \rho$ iff either $s\sigma \succ_t t\rho$ or else $s\sigma = t\rho$ and $s \sqsupset t$

Superposition — Closures

Problem: standard notion of redundancy doesn't support ground joinability.

Example: rewriting $f(b) + (a + c) \approx c$
to $a + (f(b) + c) \approx c$
via $x + (y + z) \approx y + (x + z)$

Must be $\prec_{cc}$ than deleted clause

$$(f(b)+(a+c) \approx c) \cdot id \quad \succ_{cc} \quad (x+(y+z) \approx y+(x+z)) \cdot [x/f(b), y/a, z/c]$$

Solution: refine the notion of ground instance to ground closure, and define an ordering where more general is smaller.

Closure:

Pair of term/literal/clause and grounding substitution: $t \cdot \theta$.
Example: of $f(x, b)$, instance $f(a, b)$ becomes $f(x, b) \cdot x/a$.
Ordering: $s \cdot \sigma \succ_{tc} t \cdot \rho$ iff either $s\sigma \succ_t t\rho$ or else $s\sigma = t\rho$ and $s \sqsupset t$

Superposition — Model construction

Proof of completeness: inductive model construction [Bachmair Ganzinger].

Sketch (given a saturated-up-to-closure-redundancy set of clauses S):

- Fix an ordering $\succ$ on clauses,
- Take the set G of all ground closures of clauses in S,
- Recursively define an interpretation for G,
- Prove by induction: this interpretation is a model for G.

Define saturation up to closure redundancy thus:

- A set is saturated if there are no non-redundant inferences with non-redundant premises in the set.

Closure redundant clause (in S)

All ground closures follow from smaller closures in G.

Closure redundant inference (in S)

All ground closures of the conclusion follow from closures in G smaller than the maximal ground closure of the premises.

Superposition — Model construction

Theorem

The superposition inference system is refutationally complete up to closure redundancy.

Superposition — Model construction

Theorem

The superposition inference system is refutationally complete up to closure redundancy.

This is nontrivial!

Simplifications — AC joinability

We can now justify the following AC redundancies. Let AC_f be

$$xy \approx yx \qquad (xy)z \approx x(yz) \qquad x(yz) \approx y(xz)$$

Simplifications — AC joinability

We can now justify the following AC redundancies. Let AC_f be

$$xy \approx yx \qquad (xy)z \approx x(yz) \qquad x(yz) \approx y(xz)$$

then

$$\text{AC joinability (pos)} \quad \frac{\cancel{s \approx t \vee C} \quad AC_f}{}, \quad \begin{array}{l}\text{where } s \downarrow_{AC_f} t \\ \text{and } s \approx t \vee C \notin AC_f\end{array}$$

$$\text{AC joinability (neg)} \quad \frac{\cancel{s \not\approx t \vee C} \quad AC_f}{C}, \quad \text{where } s \downarrow_{AC_f} t$$

Simplifications — AC joinability

We can now justify the following AC redundancies. Let AC_f be

$$xy \approx yx \qquad (xy)z \approx x(yz) \qquad x(yz) \approx y(xz)$$

then

$$\text{AC joinability (pos)} \quad \frac{\cancel{s \approx t \vee C} \quad AC_f}{}, \quad \begin{array}{l}\text{where } s \downarrow_{AC_f} t \\ \text{and } s \approx t \vee C \notin AC_f\end{array}$$

$$\text{AC joinability (neg)} \quad \frac{\cancel{s \not\approx t \vee C} \quad AC_f}{C}, \quad \text{where } s \downarrow_{AC_f} t$$

Corollary 1

AC joinability is a simplification rule in the superposition calculus.

that is, we can delete/simplify any equation/inequation where both sides are equal modulo AC.

Simplifications — AC joinability

We can now justify the following AC redundancies. Let AC_f be

$$xy \approx yx \qquad (xy)z \approx x(yz) \qquad x(yz) \approx y(xz)$$

then

$$\text{AC joinability (pos)} \quad \frac{\cancel{s \approx t \vee C} \quad AC_f}{}, \quad \begin{array}{l}\text{where } s \downarrow_{AC_f} t \\ \text{and } s \approx t \vee C \notin AC_f\end{array}$$

$$\text{AC joinability (neg)} \quad \frac{\cancel{s \not\approx t \vee C} \quad AC_f}{C}, \quad \text{where } s \downarrow_{AC_f} t$$

Corollary 1

AC joinability is a simplification rule in the superposition calculus.

that is, we can delete/simplify any equation/inequation where both sides are equal modulo AC. Includes all inferences between AC_f!

Simplifications — AC normalisation

Sometimes we can simplify AC terms by demodulation:
$a + (c + b) \rightarrow a + (b + c)$.

Simplifications — AC normalisation

Sometimes we can simplify AC terms by demodulation:
$a + (c + b) \to a + (b + c)$.

Sometimes we can't: $b + (x + a) \to a + (x + b)$.

Simplifications — AC normalisation

Sometimes we can simplify AC terms by demodulation:
$a + (c + b) \to a + (b + c)$.

Sometimes we can't: $b + (x + a) \to a + (x + b)$.

$$\text{AC normalisation} \quad \frac{\cancel{C[t_1(\cdots t_n)]} \quad AC_f}{C[t'_1(\cdots t'_n)]}, \quad \begin{array}{l}\text{where } t_1, \ldots \succ_{\text{lex}} t'_1, \ldots \\ \text{and } \{t_1, \ldots\} = \{t'_1, \ldots\}\end{array}$$

Corollary 2

AC normalisation is a simplification rule in the superposition calculus.

Advantages: more redundant clauses discarded vs demodulation, faster implementation since we don't need to store prolific AC axioms in indices.

Simplifications — Encompassment demodulation

We have also improved the constraints for demodulation:

$$\text{Demodulation} \qquad \frac{l \approx r \quad \cancel{C[l\theta]}}{C[l\theta \to r\theta]}, \quad \begin{array}{l} \text{where } l\theta \succ r\theta \\ \text{and } l\theta \approx r\theta \prec C[l\theta]. \end{array}$$

Simplifications — Encompassment demodulation

We have also improved the constraints for demodulation:

Encompassment Demodulation $\dfrac{l \approx r \quad \cancel{C[l\theta]}}{C[l\theta \to r\theta]}$, where $l\theta \succ r\theta$, and either $l\theta \approx r\theta \prec C[l\theta]$, or else θ not a renaming.

Simplifications — Encompassment demodulation

We have also improved the constraints for demodulation:

Encompassment Demodulation $\dfrac{l \approx r \quad \cancel{C[l\theta]}}{C[l\theta \to r\theta]}$, where $l\theta \succ r\theta$, and either $l\theta \approx r\theta \prec C[l\theta]$, or else θ not a renaming.

Corollary 3

Encompassment demodulation is a simplification rule in the superposition calculus.

This enables demodulation at more places than before (irrespective of AC), and also faster implementation.

Simplifications — Further work

More rules are under investigation, enabled by the theoretical proof of completeness up to closure redundancy.

- Extensions of AC (AC + inverses, AC + idempotence, etc.)
- AC demodulation
- ...

Implementation

iProver is an automated, first-order theorem prover.

- Implements several calculi (Inst-Gen, superposition) and several strategies for running them, many advanced techniques
- Can run in auto mode, but can also be extensively customised
- Free software (GPL), written in OCaml
- Good performance (2nd place in FOF, FNT, LTB at CASC 2021, winner in parallel single-query at SMTCOMP 2021)

Thank you