An overview of the Superposition Calculus

Simon Cruanes

May. 2021

Imandra Inc., Austin, TX
https://imandra.ai
simon@imandra.ai

Introduction

Superposition is a refutational proof system for First-Order Logic.
In this talk:

- Quick recap of (equational) **classical first-order logic**
- The basic inference rules
- Term orderings, Saturation, and completeness arguments
- Simplifications rules and refinements
- A note on theories
- A few references and pointers

Summary

Classical First-Order Logic

Mathematical formulas with quantifiers and excluded middle.

Example

"Cats are cute, and Felix is a cat; therefore Felix is cute"

$$(\mathrm{isa}(\mathrm{Felix}, \mathrm{cat}) \land (\forall x.\ \mathrm{isa}(x, \mathrm{cat}) \Rightarrow \mathrm{cute}(x))) \Rightarrow \mathrm{cute}(\mathrm{Felix})$$

- $A \land B$ means "A and B"
- $A \lor B$ means "A or B"
- $A \Rightarrow B$ means "if A then B" (note: $a \Rightarrow b$ is $\neg a \lor b$)
- $\forall x.\ F$ means "for all x, F"
- $\exists x.\ F$ means "there exists an x such that F"
- $\neg F$ means "not F" (note: $\neg\neg a \Leftrightarrow a$)

Classical First-Order Logic

Mathematical formulas with quantifiers and excluded middle.

Example

"Cats are cute, and Felix is a cat; therefore Felix is cute"

$$(\mathrm{isa}(\mathrm{Felix}, \mathrm{cat}) \wedge (\forall x.\ \mathrm{isa}(x, \mathrm{cat}) \Rightarrow \mathrm{cute}(x))) \Rightarrow \mathrm{cute}(\mathrm{Felix})$$

- $A \wedge B$ means "A and B"
- $A \vee B$ means "A or B"
- $A \Rightarrow B$ means "if A then B" (note: $a \Rightarrow b$ is $\neg a \vee b$)
- $\forall x.\ F$ means "for all x, F"
- $\exists x.\ F$ means "there exists an x such that F"
- $\neg F$ means "not F" (note: $\neg\neg a \Leftrightarrow a$)

Equational First-Order Logic

Equality

- extension of first-order logic:
  - add predicate $x \simeq y$. ("x equals y")
  - if $x \simeq y$, can replace x by y. (Leibniz equality)
  - $a = b$ is syntactic equality only.
- super useful in practice.

- Superposition (1990): proof system for first-order + equality.
- state of the art for FO (most major provers use it); See the CASC competition.
- recent efforts by Bentkamp et al. to extend to Higher-Order logic (building on top of Zipperposition).

Equational First-Order Logic

Equality

- extension of first-order logic:
  - add predicate $x \simeq y$. ("x equals y")
  - if $x \simeq y$, can replace x by y. (Leibniz equality)
  - $a = b$ is syntactic equality only.
- super useful in practice.

- Superposition (1990): proof system for first-order + equality.
- state of the art for FO (most major provers use it);
  See the CASC competition.
- recent efforts by Bentkamp et al. to extend to Higher-Order logic (building on top of Zipperposition).

Equational First-Order Logic

Equality

- extension of first-order logic:
  - add predicate $x \simeq y$. ("x equals y")
  - if $x \simeq y$, can replace x by y. (Leibniz equality)
  - $a = b$ is syntactic equality only.
- super useful in practice.

- Superposition (1990): proof system for first-order + equality.
- state of the art for FO (most major provers use it);
  See the CASC competition.
- recent efforts by Bentkamp et al. to extend to Higher-Order logic (building on top of Zipperposition).

Superposition Primer : Example

Example

If we learn that "cat" and "chat" are equal (same concept), we can substitute one for the other:

$$\dfrac{\dfrac{\mathrm{isa}(\mathrm{Felix}, \mathrm{chat}) \qquad \mathrm{chat} \simeq \mathrm{cat}}{\mathrm{isa}(\mathrm{Felix}, \mathrm{cat})}\ \text{(Sup)} \qquad \neg\, \mathrm{isa}(x, \mathrm{cat}) \vee \mathrm{cute}(x)}{\mathrm{cute}(\mathrm{Felix})}\ \text{(Res)}$$

Notes:

- x bound to Felix using *unification*. This is what separates Superposition from, e.g., SMT solvers.
- here we have both superposition and resolution; in reality Superposition is sufficient.

Superposition Primer : Example

Example

If we learn that "cat" and "chat" are equal (same concept), we can substitute one for the other:

$$\dfrac{\dfrac{\mathrm{isa}(\mathrm{Felix}, \mathrm{chat}) \qquad \mathrm{chat} \simeq \mathrm{cat}}{\mathrm{isa}(\mathrm{Felix}, \mathrm{cat})}\,(\mathsf{Sup}) \qquad \neg\, \mathrm{isa}(x, \mathrm{cat}) \vee \mathrm{cute}(x)}{\mathrm{cute}(\mathrm{Felix})}\,(\mathsf{Res})$$

Notes:

- x bound to Felix using *unification*. This is what separates Superposition from, e.g., SMT solvers.
- here we have both superposition and resolution; in reality Superposition is sufficient.

Another example: a bit of Group Theory

An easy test problem for Zipperposition[1]:

$$\frac{\begin{array}{rcl} \forall x\ y\ z.\ x \odot (y \odot z) & \simeq & (x \odot y) \odot z \\ \forall x.\ e \odot x & \simeq & x \\ \forall x.\ x^{-1} \odot x & \simeq & e \end{array}}{\forall x\ y.\ (x \odot y) \simeq e \quad \Rightarrow \quad (y \odot x) \simeq e}$$

(demo)

Note: not so easy for resolution or Tableaux!

[1]From the classic "Pelletier problems"

Substitutions and Unification

Substitution

- noted σ
- maps *variables* to *terms*

Unification (Robinson, 1960)

- central operation (millions done during proof)
- unify terms s and t means finding σ such that $s\sigma = t\sigma$

Example

- $\mathrm{isa}(x, \mathrm{cat})$ and $\mathrm{isa}(\mathrm{Felix}, \mathrm{cat})$ unified by $\sigma = \{x \mapsto \mathrm{Felix}\}$
- $f(f(x, b), y)$ and $f(y, f(a, z))$ unified by
  $\sigma = \{x \mapsto a, y \mapsto f(a, b), z \mapsto b\}$

Summary

Notations

Superposition is only three rules!

Let $\succeq$ be a term ordering.

(next section)

- σ is a substitution
- C, D are clauses (disjunctions)
- $u\,[t]_p$ puts t at position p in term u
- we omit some *eligibility constraints*, which apply to literals.

Notations

Superposition is only three rules!

Let $\succeq$ be a term ordering.

(next section)

- σ is a substitution
- C, D are clauses (disjunctions)
- $u[t]_p$ puts t at position p in term u
- we omit some *eligibility constraints*, which apply to literals.

Superposition Rules (1)

Equality Resolution (EqRes)

$$\frac{C \vee s \not\approx t}{C\sigma}$$

where $s\sigma = t\sigma$

Superposition Rules (2)

Superposition (Sup)

$$\frac{C \vee \boxed{s} \simeq t \qquad D \vee u\left[\boxed{s_2}\right]_p \overset{?}{\simeq} v}{(C \vee D \vee u\left[t\right]_p \overset{?}{\simeq} v)\sigma}$$

where $s\sigma = s_2\sigma$, $\overset{?}{\simeq} \in \{\simeq, \not\simeq\}$, $s\sigma \not\preceq t\sigma$,
$u[s_2]\sigma \not\preceq v\sigma$

Note: we use $s\sigma \not\preceq t\sigma$ rather than $s\sigma \succeq t\sigma$ as a computable approximation.

Superposition Rules (2)

Superposition (Sup)

$$\frac{C \vee \boxed{s} \simeq t \qquad D \vee u\left[\boxed{s_2}\right]_p \overset{?}{\simeq} v}{(C \vee D \vee u\,[t]_p \overset{?}{\simeq} v)\sigma}$$

where $s\sigma = s_2\sigma$, $\overset{?}{\simeq} \in \{\simeq, \not\simeq\}$, $s\sigma \not\prec t\sigma$,
$u[s_2]\sigma \not\prec v\sigma$

Note: we use $s\sigma \not\prec t\sigma$ rather than $s\sigma \succeq t\sigma$ as a computable approximation.

Superposition Rules (3)

Equality Factoring (EqFact)

$$\frac{C \vee \boxed{s} \simeq s' \vee \boxed{t} \simeq t'}{(C \vee s' \not\simeq t' \vee t \simeq t')\sigma}$$

where $s\sigma = t\sigma$, $s\sigma \not\prec s'\sigma$

(Rarely useful, but necessary for completeness)

Other Rules

- in practice, a real prover will have many more rules
- most are simplification rules (more later)
- some (e.g. Vampire) also keep resolution rules

For further reference

- "E: A Brainiac Theorem Prover", S. Schulz (link), one of my favorite papers, excellent overview
- Handbook of Automated Reasoning, (chapter "paramodulation", Nieuwenhuis & Rubio) for the theory

Summary

Term Ordering

Main idea: if we rewrite using $a \simeq b$, avoid rewriting with $b \simeq a$ (if it can be avoided)

- how: $a \succeq b$ required to rewrite using $a \simeq b$ left-to-right
- on ground terms: **terms totally ordered**
- on non-ground terms: computable over-approximation (such that $\exists \sigma. a\sigma \succeq b\sigma$ implies $a \not\prec b$)
- also: inferences operate only on maximal literals in clauses[2]
- prunes search space significantly
- ensures termination on ground problems

[2] actual criterion is more complicated (see "eligibility" in refs).

Term Ordering

Main idea: if we rewrite using $a \simeq b$, avoid rewriting with $b \simeq a$ (if it can be avoided)

- how: $a \succeq b$ required to rewrite using $a \simeq b$ left-to-right
- on ground terms: **terms totally ordered**
- on non-ground terms: computable over-approximation (such that $\exists \sigma . a\sigma \succeq b\sigma$ implies $a \not\prec b$)
- also: inferences operate only on maximal literals in clauses[2]
- prunes search space significantly
- ensures termination on ground problems

[2] actual criterion is more complicated (see "eligibility" in refs).

Term Ordering

Main idea: if we rewrite using $a \simeq b$, avoid rewriting with $b \simeq a$
(if it can be avoided)

- how: $a \succeq b$ required to rewrite using $a \simeq b$ left-to-right
- on ground terms: **terms totally ordered**
- on non-ground terms: computable over-approximation
  (such that $\exists \sigma . a\sigma \succeq b\sigma$ implies $a \not\prec b$)
- also: inferences operate only on maximal literals in clauses[2]
- prunes search space significantly
- ensures termination on ground problems

[2] actual criterion is more complicated (see "eligibility" in refs).

Term Ordering

Main idea: if we rewrite using $a \simeq b$, avoid rewriting with $b \simeq a$ (if it can be avoided)

- how: $a \succeq b$ required to rewrite using $a \simeq b$ left-to-right
- on ground terms: **terms totally ordered**
- on non-ground terms: computable over-approximation (such that $\exists \sigma . a\sigma \succeq b\sigma$ implies $a \not\prec b$)
- also: inferences operate only on maximal literals in clauses[2]
- prunes search space significantly
- ensures termination on ground problems

[2] actual criterion is more complicated (see "eligibility" in refs).

Completeness Argument for Superposition

The term ordering is crucial to the completeness argument.

Very rough sketch:[3]

- extend ordering to literals, then clauses (multiset extension)
- consider a saturated clause set[4] without $\perp$
- model is a set of oriented, orthogonal, ground rewrite rules [5]
- sort ground instances of clauses in ascending order
- for each clause:
  - if satisfied by model, continue
  - else if it can contribute a rule, add the rule
  - otherwise: absurd

    assume it's the smallest such clause

    show there is a smaller one using one of the inference rules

[3] actual proof in the handbook of AR.

[4] not extensible with new inferences (more later.)

[5] model can be infinite!

Completeness Argument for Superposition

The term ordering is crucial to the completeness argument.

Very rough sketch:[3]

- extend ordering to literals, then clauses (multiset extension)
- consider a saturated clause set[4] without $\bot$
- model is a set of oriented, orthogonal, ground rewrite rules [5]
- sort ground instances of clauses in ascending order
- for each clause:
  - if satisfied by model, continue
  - else if it can contribute a rule, add the rule
  - otherwise: absurd

    assume it's the smallest such clause

    show there is a smaller one using one of the inference rules

[3] actual proof in the handbook of AR.

[4] not extensible with new inferences (more later.)

[5] model can be infinite!

Completeness Argument for Superposition

The term ordering is crucial to the completeness argument.

Very rough sketch:[3]

- extend ordering to literals, then clauses (multiset extension)
- consider a saturated clause set[4] without $\perp$
- model is a set of oriented, orthogonal, ground rewrite rules [5]
- sort ground instances of clauses in ascending order
- for each clause:
  - if satisfied by model, continue
  - else if it can contribute a rule, add the rule
  - otherwise: absurd

    assume it's the smallest such clause

    show there is a smaller one using one of the inference rules

[3] actual proof in the handbook of AR.

[4] not extensible with new inferences (more later.)

[5] model can be infinite!

Completeness Argument for Superposition

The term ordering is crucial to the completeness argument.

Very rough sketch:[3]

- extend ordering to literals, then clauses (multiset extension)
- consider a saturated clause set[4] without $\perp$
- model is a set of oriented, orthogonal, ground rewrite rules [5]
- sort ground instances of clauses in ascending order
- for each clause:
  - if satisfied by model, continue
  - else if it can contribute a rule, add the rule
  - otherwise: absurd

    assume it's the smallest such clause

    show there is a smaller one using one of the inference rules

[3] actual proof in the handbook of AR.

[4] not extensible with new inferences (more later.)

[5] model can be infinite!

Completeness Argument for Superposition

The term ordering is crucial to the completeness argument.

Very rough sketch:[3]

- extend ordering to literals, then clauses (multiset extension)
- consider a saturated clause set[4] without $\perp$
- model is a set of oriented, orthogonal, ground rewrite rules [5]
- sort ground instances of clauses in ascending order
- for each clause:
    - if satisfied by model, continue
    - else if it can contribute a rule, add the rule
    - otherwise: absurd

      assume it's the smallest such clause

      show there is a smaller one using one of the inference rules

[3] actual proof in the handbook of AR.

[4] not extensible with new inferences (more later.)

[5] model can be infinite!

Completeness Argument for Superposition

The term ordering is crucial to the completeness argument.

Very rough sketch:[3]

- extend ordering to literals, then clauses (multiset extension)
- consider a saturated clause set[4] without $\bot$
- model is a set of oriented, orthogonal, ground rewrite rules [5]
- sort ground instances of clauses in ascending order
- for each clause:
    - if satisfied by model, continue
    - else if it can contribute a rule, add the rule
    - otherwise: absurd

      assume it's the smallest such clause

      show there is a smaller one using one of the inference rules

[3] actual proof in the handbook of AR.

[4] not extensible with new inferences (more later.)

[5] model can be infinite!

Completeness Argument for Superposition

The term ordering is crucial to the completeness argument.

Very rough sketch:[3]

- extend ordering to literals, then clauses (multiset extension)
- consider a saturated clause set[4] without $\perp$
- model is a set of oriented, orthogonal, ground rewrite rules [5]
- sort ground instances of clauses in ascending order
- for each clause:
  - if satisfied by model, continue
  - else if it can contribute a rule, add the rule
  - otherwise: absurd

    assume it's the smallest such clause

    show there is a smaller one using one of the inference rules

[3] actual proof in the handbook of AR.

[4] not extensible with new inferences (more later.)

[5] model can be infinite!

Term Ordering (cont'd)

Term orderings used in real provers:

KBO: (Knuth-Bendix Ordering) fast, orders by weight + lexicographic recursion.

RPO: can be stronger, but slower. Less often used.[6]

But also sometimes:

- more exotic orderings from the termination world.
- variations compatible with AC, higher-order, etc.

[6] Portfolio strategies mean it's still useful.

Term Ordering (cont'd)

Term orderings used in real provers:

KBO: (Knuth-Bendix Ordering) fast, orders by weight + lexicographic recursion.

RPO: can be stronger, but slower. Less often used.[6]

But also sometimes:

- more exotic orderings from the termination world.
- variations compatible with AC, higher-order, etc.

[6] Portfolio strategies mean it's still useful.

Term Orderings (cont'd)

- important choice point for heuristics

  "We believe that the selection and generation of term orderings is the weakest part of E's heuristic control component."

- can incur non-trivial computational costs
- ordering needs strong properties ("simplification ordering")

  (cannot just use any order)

Summary

Superposition: the basics

Superposition: more details

Crucial Restriction: Term Ordering

Crucial Restriction: Redundancy

Applications and Theories

Search Space Explosion

The Superposition rules are very prolific.[7]

- redundancy criterion says we can throw away some clauses
- C redundant wrt a set of clauses S (means $S^{\preceq C} \models C$)
- also enables simplifications (e.g. rewriting)
- in practice, many clauses are/become redundant

Example of redundancy rule: Subsumption
C subsumes D if $\exists \sigma.\ C\sigma \subseteq D$

[7] even with the term ordering.

Search Space Explosion

The Superposition rules are very prolific.[7]

- redundancy criterion says we can throw away some clauses
- C redundant wrt a set of clauses S (means $S^{\preceq C} \models C$)
- also enables simplifications (e.g. rewriting)
- in practice, many clauses are/become redundant

Example of redundancy rule: Subsumption
C subsumes D if $\exists \sigma.\ C\sigma \subseteq D$

[7] even with the term ordering.

Search Space Explosion

The Superposition rules are very prolific.[7]

- redundancy criterion says we can throw away some clauses
- C redundant wrt a set of clauses S (means $S^{\preceq C} \models C$)
- also enables simplifications (e.g. rewriting)
- in practice, many clauses are/become redundant

Example of redundancy rule: Subsumption
C subsumes D if $\exists \sigma.\ C\sigma \subseteq D$

[7] even with the term ordering.

Search Space Explosion

The Superposition rules are very prolific.[7]

- redundancy criterion says we can throw away some clauses
- C redundant wrt a set of clauses S (means $S^{\preceq C} \models C$)
- also enables simplifications (e.g. rewriting)
- in practice, many clauses are/become redundant

Example of redundancy rule: Subsumption
C subsumes D if $\exists \sigma.\ C\sigma \subseteq D$

[7] even with the term ordering.

Search Space Explosion

The Superposition rules are very prolific.[7]

- redundancy criterion says we can throw away some clauses
- C redundant wrt a set of clauses S (means $S^{\preceq C} \vDash C$)
- also enables simplifications (e.g. rewriting)
- in practice, many clauses are/become redundant

Example of redundancy rule: Subsumption
C subsumes D if $\exists \sigma.\ C\sigma \subseteq D$

[7] even with the term ordering.

Saturation Process

Saturation operates on a set of clauses S_i, for $i \in \mathbb{N}$

- S_0 is the CNF of (negated) problem
- at each step, either:
  - add inferred clause: $S_{i+1} = S_i \cup \{C\}$, or
  - remove redundant clause: $S_{i+1} = S_i \setminus \{C\}$
- if $\bot \in S_i$, return UNSAT
- if all inferred clauses are in S_i or redundant wrt S_i:
  S_i is saturated, return SAT
- (not explained: notion of fairness)

Note: a satisfiable problem might never saturate.

Saturation Process

Saturation operates on a set of clauses S_i, for $i \in \mathbb{N}$

- S_0 is the CNF of (negated) problem
- at each step, either:
  - add inferred clause: $S_{i+1} = S_i \cup \{C\}$, or
  - remove redundant clause: $S_{i+1} = S_i \setminus \{C\}$
- if $\bot \in S_i$, return UNSAT
- if all inferred clauses are in S_i or redundant wrt S_i: S_i is saturated, return SAT
- (not explained: notion of fairness)

Note: a satisfiable problem might never saturate.

Saturation Process

Saturation operates on a set of clauses S_i, for $i \in \mathbb{N}$

- S_0 is the CNF of (negated) problem
- at each step, either:
  - add inferred clause: $S_{i+1} = S_i \cup \{C\}$, or
  - remove redundant clause: $S_{i+1} = S_i \setminus \{C\}$
- if $\bot \in S_i$, return UNSAT
- if all inferred clauses are in S_i or redundant wrt S_i:
  S_i is saturated, return SAT
- (not explained: notion of fairness)

Note: a satisfiable problem might never saturate.

Saturation Process

Saturation operates on a set of clauses S_i, for $i \in \mathbb{N}$

- S_0 is the CNF of (negated) problem
- at each step, either:
  - add inferred clause: $S_{i+1} = S_i \cup \{C\}$, or
  - remove redundant clause: $S_{i+1} = S_i \setminus \{C\}$
- if $\bot \in S_i$, return UNSAT
- if all inferred clauses are in S_i or redundant wrt S_i: S_i is saturated, return SAT
- (not explained: notion of fairness)

Note: a satisfiable problem might never saturate.

Saturation Process

Saturation operates on a set of clauses S_i, for $i \in \mathbb{N}$

- S_0 is the CNF of (negated) problem
- at each step, either:
    - add inferred clause: $S_{i+1} = S_i \cup \{C\}$, or
    - remove redundant clause: $S_{i+1} = S_i \setminus \{C\}$
- if $\bot \in S_i$, return UNSAT
- if all inferred clauses are in S_i or redundant wrt S_i:
  S_i is saturated, return SAT
- (not explained: notion of fairness)

Note: a satisfiable problem might never saturate.

Example of Saturation

(demo)

Note that some clauses in the saturated set are non-ground.

Main loop

Sketch of a prover's main loop ("given clause algorithm"):

```
unprocessed, processed = set(cnf(problem)), set()
while unprocessed:
  c = unprocessed.pick()  # given clause
  unprocessed.remove(c)
  if c.is_empty(): return 'unsat'

  if c.redundant_in(processed): continue
  processed.remove_clauses_subsumed_by(c)

  new = infer(c, processed)
  new = [c in new if not c.redundant(processed)]
  unprocessed.append(new)
  processed.append(c)
return 'sat'
```

(this considers subsumption, not simplifications)

Summary

Current Applications

Hammers for ITPs and verification tools:

- SledgeHammer in Isabelle/HOL
- HOLyHammer in HOL light
- Why3 (TPTP bridge)

A potential issue is unpredictability of results.

(depends on several input-sensitive heuristics)

Current Applications

Hammers for ITPs and verification tools:

- SledgeHammer in Isabelle/HOL
- HOLyHammer in HOL light
- Why3 (TPTP bridge)

A potential issue is unpredictability of results.

(depends on several input-sensitive heuristics)

Support for theories

Arithmetic: hierarchic superposition to interface with SMT/Cooper ($\sim$ 2013)

Datatypes: a recent extension (in Vampire). ($\sim$ 2018)

Induction: some experimental results ($\sim$ 2017), but undecidable.

Arrays: can be expressed as rewriting ($\sim$ 2009).

Bitvectors: n/a. Best hope is interface with SMT.

Higher Order: recent, very promising developments ($\sim$ 2019).

Overall: much weaker than SMT. Best hope is to delegate to SMT.

Some pointers for theories

(non-exhaustive!!)

arrays: Bonacina et al.: "New results on rewrite-based satisfiability procedures" (arxiv)

datatypes: Blanchette et al. "Superposition with datatypes and codatatypes"

arith: Baumgartner et al. "Beagle – a hierarchic superposition theorem prover"

induction: my own work (Frocos'17), also papers on Vampire

HO: See the Matryoshka project (my friends in Amsterdam)

Quick Digression on Implementation

- a competitive prover is a lot of work
- no single bottleneck (well. . . simplifications?)
- best FO provers: E prover[8], Vampire, SPASS
- my own PhD work: Zipperposition (in OCaml)[9]
- further material: "implementation of saturating theorem provers", S. Schulz (slides)

[8] https://www.eprover.org/
[9] https://github.com/sneeuwballen/zipperposition

Conclusion

- Superposition excels on FO reasoning.
- Combination of strong theory, and good implementations (Vampire, E, SPASS).
- Good on algebra, ITP problems.
- Taking over the world of HO reasoning as well.
- Theories: still early days, not really there yet.

Questions?

Conclusion

- Superposition excels on FO reasoning.
- Combination of strong theory, and good implementations (Vampire, E, SPASS).
- Good on algebra, ITP problems.
- Taking over the world of HO reasoning as well.
- Theories: still early days, not really there yet.

Questions?