Implementing superposition in iProver

(System description)

André Duarte and Konstantin Korovin

{andre.duarte,konstantin.korovin}@manchester.ac.uk

4th July 2020

The University of Manchester

iProver

- iProver is an automated theorem prover for first-order logic.

iProver

- iProver is an automated theorem prover for first-order logic.
- Instantiation calculus, modular architecture.

iProver

- iProver is an automated theorem prover for first-order logic.
- Instantiation calculus, modular architecture.
- Refutationally complete.

iProver

- iProver is an automated theorem prover for first-order logic.
- Instantiation calculus, modular architecture.
- Refutationally complete.
- Powerful redundancy checks.

iProver

- iProver is an automated theorem prover for first-order logic.
- Instantiation calculus, modular architecture.
- Refutationally complete.
- Powerful redundancy checks.
- Decides fragments such as EPR (winner of CASC/EPR almost every year since 2008).

iProver

- iProver is an automated theorem prover for first-order logic.
- Instantiation calculus, modular architecture.
- Refutationally complete.
- Powerful redundancy checks.
- Decides fragments such as EPR (winner of CASC/EPR almost every year since 2008).

Motivation

- No single calculus is optimal.
  - Some problems are solved easily by some techniques and not at all by others.

Motivation

- No single calculus is optimal.
  - Some problems are solved easily by some techniques and not at all by others.
- Performance degrades very fast.
  - Half of problems that are solved in $< 5\,\mathrm{min}$ are solved in $\sim 1\,\mathrm{s}$.
  - $\sim 90\%$ of problems that are solved in $< 5\,\mathrm{min}$ are solved in $< 30\,\mathrm{s}$.

Motivation

- No single calculus is optimal.
  - Some problems are solved easily by some techniques and not at all by others.
- Performance degrades very fast.
  - Half of problems that are solved in $< 5\,\mathrm{min}$ are solved in $\sim 1\,\mathrm{s}$.
  - $\sim 90\%$ of problems that are solved in $< 5\,\mathrm{min}$ are solved in $< 30\,\mathrm{s}$.
- In iProver, clauses are shared for simplifications.

Motivation

- No single calculus is optimal.
  - Some problems are solved easily by some techniques and not at all by others.
- Performance degrades very fast.
  - Half of problems that are solved in $< 5\,\mathrm{min}$ are solved in $\sim 1\,\mathrm{s}$.
  - $\sim 90\%$ of problems that are solved in $< 5\,\mathrm{min}$ are solved in $< 30\,\mathrm{s}$.
- In iProver, clauses are shared for simplifications.

Corollary:

It's better to run many strategies for a shorter time than one strategy for a long time.

Superposition — generating inferences

$$\text{Superposition} \qquad \frac{l = r \vee C \quad t[s] \doteq u \vee D}{(t[s \mapsto r] \doteq u \vee C \vee D)\theta}$$

where $\theta = \mathrm{mgu}(l, s)$, $l\theta \not\preceq r\theta$, $t\theta \not\preceq u\theta$, and s not a variable,

$$\text{Eq. Resolution} \qquad \frac{l \neq r \vee C}{C\theta}$$

where $\theta = \mathrm{mgu}(l, r)$,

$$\text{Eq. Factoring} \qquad \frac{l = r \vee l' = r' \vee C}{(l = r \vee r \neq r' \vee C)\theta}$$

where $\theta = \mathrm{mgu}(l, l')$, $l\theta \not\preceq r\theta$ and $r\theta \not\preceq r'\theta$.

Superposition — simplifying inferences

Tautology deletion $\dfrac{\cancel{l \vee \bar{l} \vee C}}{}$ $\dfrac{\cancel{t = t \vee C}}{}$

Syntactic eq. res. $\dfrac{\cancel{t \neq t \vee C}}{C}$

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

Subset subsumption $\dfrac{\cancel{C \vee D} \quad C}{}$

Demodulation $\dfrac{l = r \quad \cancel{C[l\theta]}}{C[l\theta \mapsto r\theta]}, \quad \begin{array}{l} l\theta \succ r\theta \\ \{l\theta = r\theta\} \prec C \end{array}$

Superposition — simplifying inferences

Tautology deletion $\qquad \dfrac{\cancel{l \vee \bar{l} \vee C}}{} \qquad \dfrac{\cancel{t = t \vee C}}{}$

Syntactic eq. res. $\qquad \dfrac{\cancel{t \neq t \vee C}}{C}$

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

Subset subsumption $\qquad \dfrac{\cancel{C \vee D} \quad C}{}$

Demodulation $\qquad \dfrac{l = r \quad \cancel{C[l\theta]}}{C[l\theta \mapsto r\theta]}, \quad \begin{array}{l} l\theta \succ r\theta \\ \{l\theta = r\theta\} \prec C \end{array}$

Superposition — simplifying inferences

Tautology deletion $\dfrac{\cancel{l \vee \bar{l} \vee C}}{}$ $\dfrac{\cancel{t = t \vee C}}{}$

Syntactic eq. res. $\dfrac{\cancel{t \neq t \vee C}}{C}$

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

Subset subsumption $\dfrac{\cancel{C \vee D} \quad C}{}$

Demodulation $\dfrac{l = r \quad \cancel{C[l\theta]}}{C[l\theta \mapsto r\theta]}, \quad \begin{array}{l} l\theta \succ r\theta \\ \{l\theta = r\theta\} \prec C \end{array}$

Light normalisation $\dfrac{l = r \quad \cancel{C[l]}}{C[l \mapsto r]}, \quad l \succ r$

Light normalisation

We introduce the simplification rule

$$\text{Light normalisation} \qquad \frac{l = r \quad \cancel{C[l]}}{C[l \mapsto r]}$$

where $l \succ r$, and l occurs outside a maximal side of an equality literal.

Light normalisation

We introduce the simplification rule

$$\text{Light normalisation} \qquad \frac{l = r \quad \cancel{C[l]}}{C[l \mapsto r]}$$

where $l \succ r$, and l occurs outside a maximal side of an equality literal.

While a restricted case of demodulation, it's also much faster.

Light normalisation

We introduce the simplification rule

$$\text{Light normalisation} \qquad \frac{l = r \quad \cancel{C[l]}}{C[l \mapsto r]}$$

where $l \succ r$, and l occurs outside a maximal side of an equality literal.

While a restricted case of demodulation, it's also much faster.

Can be implemented with a hashtable + index of subterms. To normalise a clause, simply traverse bottom-up and look up subterms in the hashtable. To add a new equation, first use-it to reduce equations already kept, then add to the hashtable.

We can also choose to add e.g. only ground equations.

Light normalisation

We introduce the simplification rule

$$\text{Light normalisation} \qquad \frac{R \quad \cancel{C[l]}}{C[l \mapsto r]}$$

where $l \to r \in R$, and l occurs outside a maximal side of an equality literal.

While a restricted case of demodulation, it's also much faster.

Can be implemented with a hashtable + index of subterms. To normalise a clause, simply traverse bottom-up and look up subterms in the hashtable. To add a new equation, first use-it to reduce equations already kept, then add to the hashtable.

We can also choose to add e.g. only ground equations.

Simplifications

Important: lots of freedom to choose *how* we do simplifications.

Simplifications

Important: lots of freedom to choose *how* we do simplifications:

- which rules to perform,
- in what order,
- when,
- and with respect to what clauses.

Simplifications

Important: lots of freedom to choose *how* we do simplifications:

- which rules to perform,
- in what order,
- when,
- and with respect to what clauses.

Also, these require indices to implement. Some indices support several simplification rules. We must choose:

Simplifications

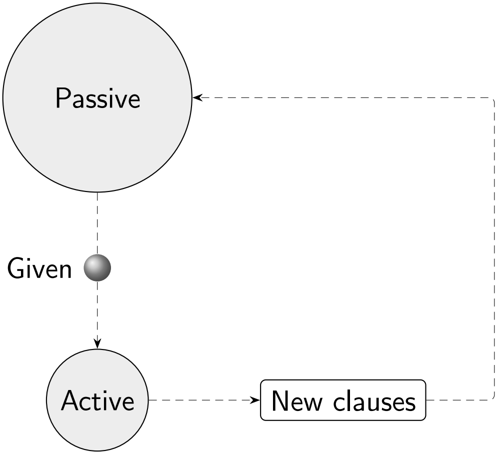
Important: lots of freedom to choose *how* we do simplifications:

- which rules to perform,
- in what order,
- when,
- and with respect to what clauses.

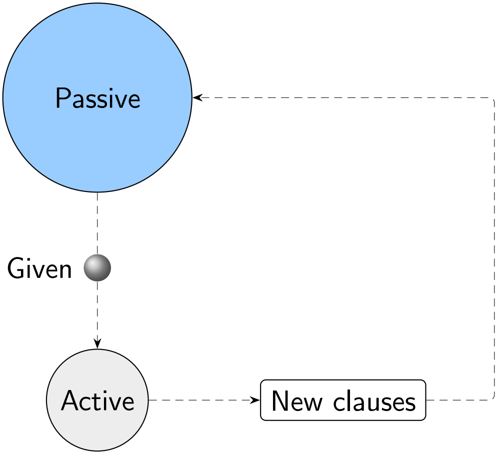
Also, these require indices to implement. Some indices support several simplification rules. We must choose:

- which clauses to add to which indices,
- and when.

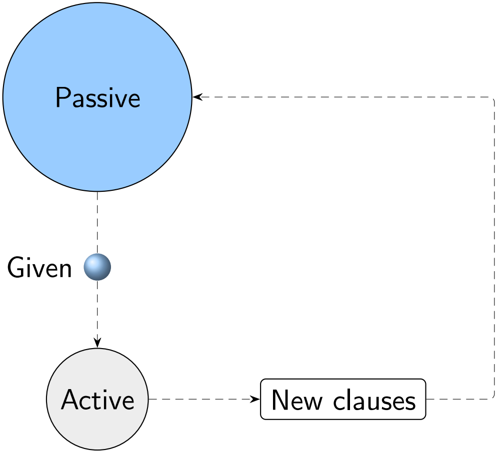
Simplification setup — Given clause loop overview



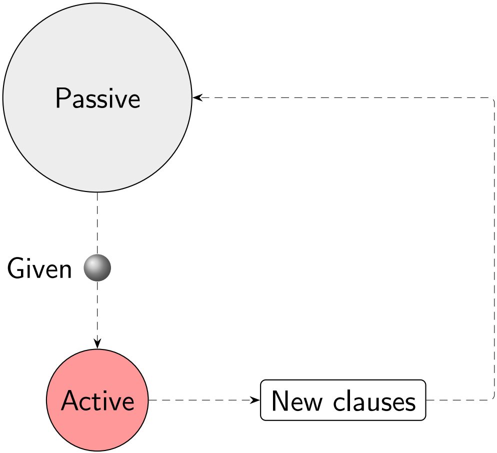
Simplification setup — Given clause loop overview



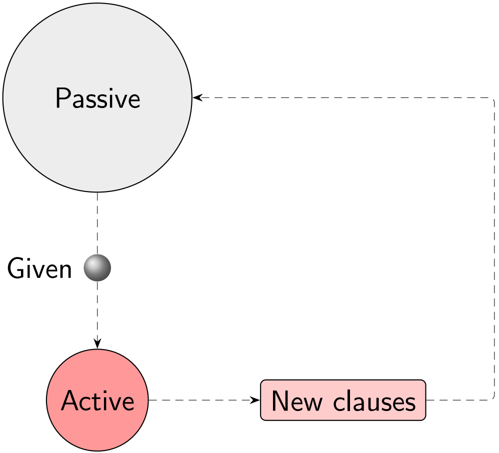
Simplification setup — Given clause loop overview



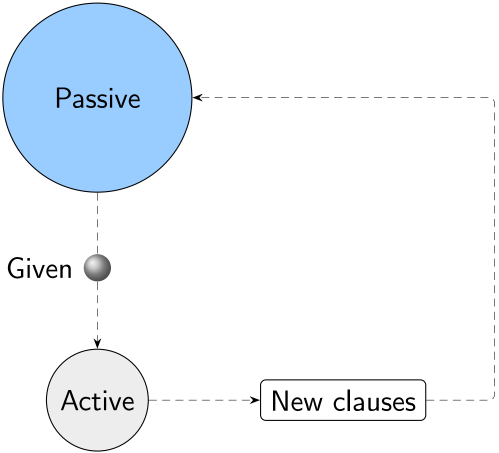
Simplification setup — Given clause loop overview



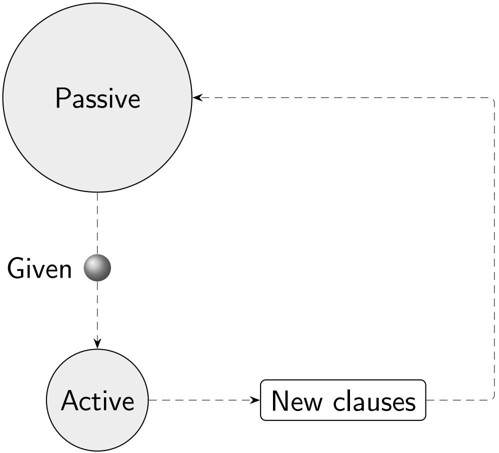
Simplification setup — Given clause loop overview



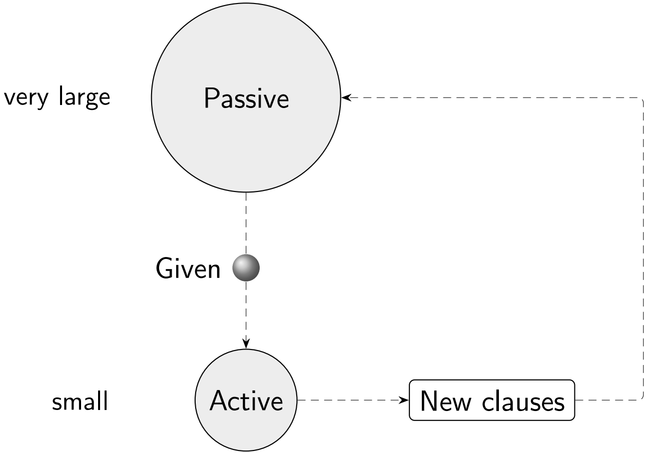
Simplification setup — Given clause loop overview



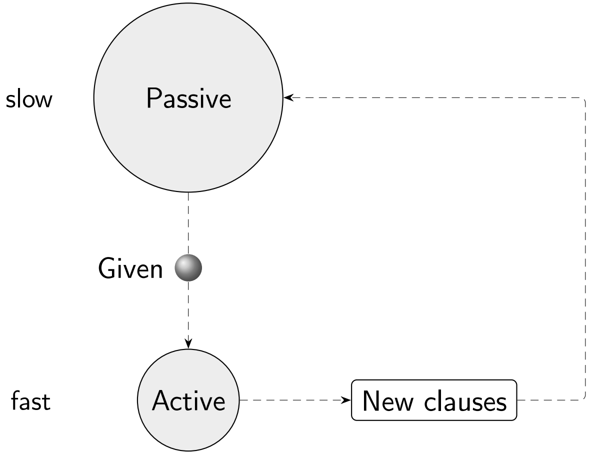
Simplification setup — Given clause loop overview



Simplification setup — Given clause loop overview

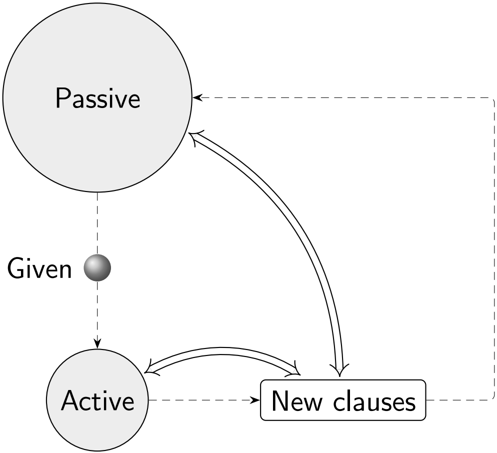


Simplification setup — Given clause loop overview



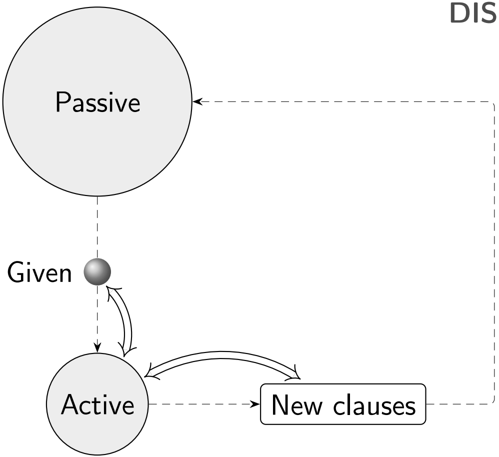
Simplification setup

Otter

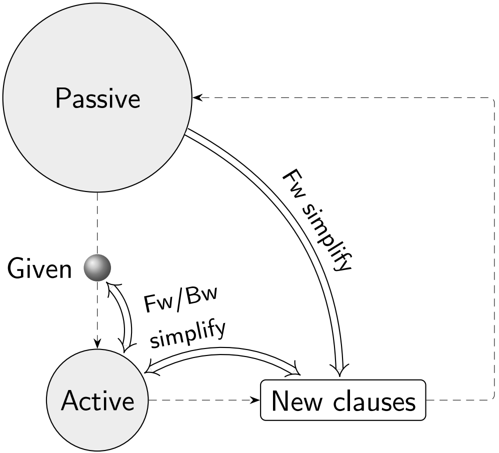


Simplification setup

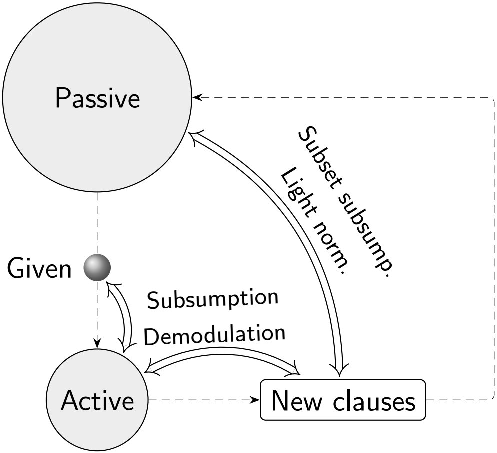
DISCOUNT



Simplification setup



Simplification setup



Immediate simplification

Intuition:

Immediate simplification

Intuition:

- Clauses that are derived in each loop are more "related" to each other and to their parents.

Immediate simplification

Intuition:

- Clauses that are derived in each loop are more "related" to each other and to their parents.
- The passive set grows very large, but the set of new clauses in each loop stays comparatively small.

Immediate simplification

Intuition:

- Clauses that are derived in each loop are more "related" to each other and to their parents.
- The passive set grows very large, but the set of new clauses in each loop stays comparatively small.
- Can check if a new clause deletes a parent clause. If yes, then:
  - we can throw away all its children,
  - and avoid trying to generate any new clauses with it.

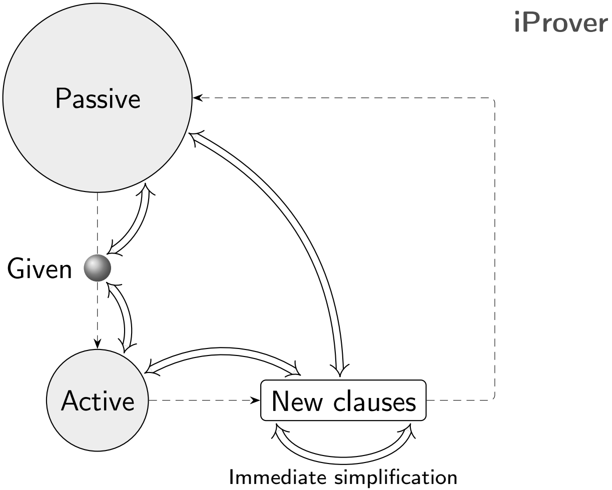
Immediate simplification

Intuition:

- Clauses that are derived in each loop are more "related" to each other and to their parents.
- The passive set grows very large, but the set of new clauses in each loop stays comparatively small.
- Can check if a new clause deletes a parent clause. If yes, then:
  - we can throw away all its children,
  - and avoid trying to generate any new clauses with it.

Hypothesis: it may be useful to keep new clause $\cup$ parents inter-simplified.

Simplification setup



Simplification setup — iProver

```
$ ./iprover --schedule none | grep '--sup'
--sup_indices_passive     [SubsetSubsumption;LightNorm]
--sup_indices_active      [Subsumption;FwDemod;BwDemod]
--sup_indices_immed       [SubsetSubsumption;Subsumption;LightNor...
--sup_indices_input       [SubsetSubsumption;Subsumption;FwDemod;...
--sup_full_triv           [TrivRules;PropSubs]
--sup_full_fw             [FwDemod;ACNormalisation;FwSubsumption;...
--sup_full_bw             [BwDemod]
--sup_immed_triv          [TrivRules]
--sup_immed_fw_main       [FwDemod;ACNormalisation;FwSubsumption]
--sup_immed_bw_main       []
--sup_immed_fw_immed      [FwDemod;FwSubsumption;FwSubsumptionRes]
--sup_immed_bw_immed      [BwDemod;BwSubsumption]
--sup_input_triv          [TrivRules]
--sup_input_fw            [FwDemod;FwSubsumption;FwSubsumptionRes]
--sup_input_bw            [BwDemod;BwSubsumption;BwSubsumptionRes]
```

AC symbols

A symbol f is associative-commutative (AC) iff

$$\forall x, y.\, f(x, y) = f(y, x) \qquad \forall x, y, z.\, f(x, f(y, z)) = f(f(x, y), z)$$

AC symbols

A symbol f is associative-commutative (AC) iff

$$\forall x, y.\, f(x, y) = f(y, x) \qquad \forall x, y, z.\, f(x, f(y, z)) = f(f(x, y), z)$$

AC symbols are notoriously hard to handle by saturation solvers based on ordered rewriting (like superposition).
They are also ubiquitous in a variety of domains.

AC symbols

A symbol f is associative-commutative (AC) iff

$$\forall x, y.\, f(x, y) = f(y, x) \qquad \forall x, y, z.\, f(x, f(y, z)) = f(f(x, y), z)$$

AC symbols are notoriously hard to handle by saturation solvers based on ordered rewriting (like superposition).
They are also ubiquitous in a variety of domains.

$\implies$ Techniques to handle AC are important.

AC symbols

During preprocessing: we have the freedom to transform the problem into any equisatisfiable form, so we can

AC symbols

During preprocessing: we have the freedom to transform the problem into any equisatisfiable form, so we can

- make AC terms right-associative

AC symbols

During preprocessing: we have the freedom to transform the problem into any equisatisfiable form, so we can

- make AC terms right-associative
- sorted wrt. some ordering.

AC symbols

During preprocessing: we have the freedom to transform the problem into any equisatisfiable form, so we can

- make AC terms right-associative
- sorted wrt. some ordering.

During saturation: we are restricted by completeness.

AC symbols

During preprocessing: we have the freedom to transform the problem into any equisatisfiable form, so we can

- make AC terms right-associative
- sorted wrt. some ordering.

During saturation: we are restricted by completeness.

- can still make AC terms right-associative (but not at the top of a maximal side of an equation)

AC symbols

During preprocessing: we have the freedom to transform the problem into any equisatisfiable form, so we can

- make AC terms right-associative
- sorted wrt. some ordering.

During saturation: we are restricted by completeness.

- can still make AC terms right-associative (but not at the top of a maximal side of an equation)
- (stably) sorted wrt. the reduction ordering being used.

AC joinability

Theorem. Let R_{AC} be

$$f(x, y) = f(y, x) \tag{1}$$
$$f(x, f(y, z)) = f(f(x, y), z) \tag{2}$$
$$f(x, f(y, z)) = f(y, f(x, z)) \tag{3}$$

if $l = r$ is not an instance of an eq. in R_{AC} and cannot be simplified via a rule in R_{AC}, then if l and r are equal modulo AC then $l = r \vee C$ is a tautology and $l \neq r \vee C$ simplifies to C.

AC joinability

Theorem. Let R_{AC} be

$$f(x,y) = f(y,x) \tag{1}$$
$$f(x,f(y,z)) = f(f(x,y),z) \tag{2}$$
$$f(x,f(y,z)) = f(y,f(x,z)) \tag{3}$$

if $l = r$ is not an instance of an eq. in R_{AC} and cannot be simplified via a rule in R_{AC}, then if l and r are equal modulo AC then $l = r \vee C$ is a tautology and $l \neq r \vee C$ simplifies to C.

Tests for AC joinability are cheap!

Semantic AC detection

In general, f is AC if the input problem implies the AC axioms.

Usually it's only checked if they are verbatim in the input (syntactical detection).

Semantic AC detection

In general, f is AC if the input problem implies the AC axioms.

Usually it's only checked if they are verbatim in the input (syntactical detection).

$$\text{Input} \models AC \impliedby AC \in \text{Input}$$

Semantic AC detection

In general, f is AC if the input problem implies the AC axioms.

Usually it's only checked if they are verbatim in the input (syntactical detection).

$$\mathrm{Input} \models AC \iff AC \in \mathrm{Input}$$

The axioms may also be implied by the input problem but not be present right away.

Semantic AC detection

In general, f is AC if the input problem implies the AC axioms.

Usually it's only checked if they are verbatim in the input (syntactical detection).

$$\text{Input} \models AC \impliedby AC \in \text{Input}$$

The axioms may also be implied by the input problem but not be present right away.

Important to detect AC symbols as soon as possible so we can apply AC reasoning early. We do this in two more ways:

Semantic AC detection

In general, f is AC if the input problem implies the AC axioms.

Usually it's only checked if they are verbatim in the input (syntactical detection).

$$\text{Input} \models AC \impliedby AC \in \text{Input}$$

The axioms may also be implied by the input problem but not be present right away.

Important to detect AC symbols as soon as possible so we can apply AC reasoning early. We do this in two more ways:

- Detect if AC axioms are derived normally in saturation,

Semantic AC detection

In general, f is AC if the input problem implies the AC axioms.

Usually it's only checked if they are verbatim in the input (syntactical detection).

$$\text{Input} \models AC \impliedby AC \in \text{Input}$$

The axioms may also be implied by the input problem but not be present right away.

Important to detect AC symbols as soon as possible so we can apply AC reasoning early. We do this in two more ways:

- Detect if AC axioms are derived normally in saturation,
- Check in preprocessing if axioms are implied via fast approximations like ground implication checking using an SMT solver.

Summary

- It is generally better to combine many strategies/options than to run just one.
  - In particular, instantiation + superposition performs better than just instantiation or just superposition.
- Huge freedom in choosing how to do simplifications, but no clear path.
  - Work on hyperparameter optimisation may help here.
- "Immediate simplification" can block many redundant generating inferences, and is relatively inexpensive.
- AC reasoning speeds up many problems (axioms found in 12% of TPTP (incl. by semantic detection)).