Induction in Saturation-Based Proving

Andrei Voronkov and Petra Hozzová

(based on joint work with Márton Hajdu, Laura Kovács, and Giles Reger)

University of Manchester & TU Wien

July 2nd, 2024

Outline

Proofs of Many Statements Require Induction

- $+$ on $\mathbb{N}$ is associative:

$$\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$$

Proofs of Many Statements Require Induction

- $+$ on $\mathbb{N}$ is associative:

$$\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$$

- closed-form formula for the sum of an integer interval is correct:

$$\forall n, m \in \mathbb{Z}.(n \leq m \to 2 \cdot \Sigma_{i=n}^{m} i = m \cdot (m + 1) - n \cdot (n - 1))$$

Proofs of Many Statements Require Induction

- $+$ on $\mathbb{N}$ is associative:

$$\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$$

- closed-form formula for the sum of an integer interval is correct:

$$\forall n, m \in \mathbb{Z}.(n \leq m \rightarrow 2 \cdot \Sigma_{i=n}^{m} i = m \cdot (m + 1) - n \cdot (n - 1))$$

- there is a function inverse to `half`:

$$\forall x \in \mathbb{N}. \exists y \in \mathbb{N}.\ \mathtt{half}(y) = x$$

Proofs of Many Statements Require Induction

- $+$ on $\mathbb{N}$ is associative:

$$\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$$

- closed-form formula for the sum of an integer interval is correct:

$$\forall n, m \in \mathbb{Z}.(n \leq m \rightarrow 2 \cdot \Sigma_{i=n}^{m} i = m \cdot (m + 1) - n \cdot (n - 1))$$

- there is a function inverse to `half`:

$$\forall x \in \mathbb{N}. \exists y \in \mathbb{N}.\ \mathtt{half}(y) = x$$

- sum of two even natural numbers is even:

$$\forall x, y \in \mathbb{N}.(\mathtt{even}(x) \wedge \mathtt{even}(y) \rightarrow \mathtt{even}(x + y))$$

Our Approach: Induction in Saturation

We extend the saturation-based proving framework for first-order predicate logic by induction.

Outline

First-Order Predicate Logic (FOL) with Theories

Theories: equality

- variables: $x, y, \ldots$
- constants: $a, b, \ldots$
- functions: $f, g, \ldots$
- predicates: $p, =, \ldots$
- logical connectives: $\wedge, \rightarrow, \forall, \ldots$

First-Order Predicate Logic (FOL) with Theories

Theories: equality, integer arithmetic, datatypes

- variables: $x, y, \ldots \in \mathbb{Z}, \mathbb{N}, \mathbb{L}, \ldots$
- constants: $a, b, 0, 1, -1, 2, \ldots$
- functions: $f, g, +, \cdot, \ldots$
- predicates: $p, =, <, \leq, \ldots$
- logical connectives: $\wedge, \rightarrow, \forall, \ldots$
- constructors: $0, \mathtt{s}, \ldots$

First-Order Predicate Logic (FOL) with Theories

Theories: equality, integer arithmetic, datatypes

- variables: $x, y, \ldots \in \mathbb{Z}, \mathbb{N}, \mathbb{L}, \ldots$
- constants: $a, b, 0, 1, -1, 2, \ldots$
- functions: $f, g, +, \cdot, \ldots$
- predicates: $p, =, <, \leq, \ldots$
- logical connectives: $\wedge, \rightarrow, \forall, \ldots$
- constructors: $0, \mathrm{s}, \ldots$

Clause is a disjunction of literals.

We denote $\overline{A} \stackrel{\text{def}}{=} \neg A$ and $\overline{\neg A} \stackrel{\text{def}}{=} A$.

Finding Proofs Automatically

Our prover VAMPIRE:

- implements superposition calculus and saturation algorithms
- supports theories using SMT solvers and AVATAR

Induction and Saturation

Induction can be implemented by reducing goals to subgoals
– so having a goal $\forall x.F[x]$ you can prove it by induction on x.

But. . .

Induction and Saturation

Induction can be implemented by reducing goals to subgoals
– so having a goal $\forall x.F[x]$ you can prove it by induction on x.

But... saturation theorem proving is not about reducing goals to subgoals.

Finding Proofs Automatically: Superposition Calculus

Calculus for reasoning with clauses in FOL with equality.

Finding Proofs Automatically: Superposition Calculus

Calculus for reasoning with clauses in FOL with equality. Selected rules (simplified):

Equality resolution: $\dfrac{s \neq s' \vee C}{C\theta}$ where $\theta := \mathrm{mgu}(s, s')$.

Binary resolution: $\dfrac{L \vee C \quad \overline{L'} \vee C'}{(C \vee C')\theta}$ where $\theta := \mathrm{mgu}(L, L')$.

Superposition: $\dfrac{s = t \vee C \quad L[s'] \vee C'}{(L[t] \vee C \vee C')\theta}$ where $\theta := \mathrm{mgu}(s, s')$

Finding Proofs Automatically: Superposition Calculus

Calculus for reasoning with clauses in FOL with equality. Selected rules (simplified):

Equality resolution: $\dfrac{s \neq s' \lor C}{C\theta}$ where $\theta := \mathtt{mgu}(s, s')$.

Binary resolution: $\dfrac{L \lor C \quad \overline{L'} \lor C'}{(C \lor C')\theta}$ where $\theta := \mathtt{mgu}(L, L')$.

Superposition: $\dfrac{s = t \lor C \quad L[s'] \lor C'}{(L[t] \lor C \lor C')\theta}$ where $\theta := \mathtt{mgu}(s, s')$

The calculus is sound (if $\square$ is derived from F, then F is unsatisfiable) and refutationally complete (if F is unsatisfiable, then $\square$ can be derived from it).

Finding Proofs Automatically: Superposition Calculus

Calculus for reasoning with clauses in FOL with equality. Selected rules (simplified):

Equality resolution: $\dfrac{s \neq s' \vee C}{C\theta}$ where $\theta := \mathtt{mgu}(s, s')$.

Binary resolution: $\dfrac{L \vee C \quad \overline{L'} \vee C'}{(C \vee C')\theta}$ where $\theta := \mathtt{mgu}(L, L')$.

Superposition: $\dfrac{s = t \vee C \quad L[s'] \vee C'}{(L[t] \vee C \vee C')\theta}$ where $\theta := \mathtt{mgu}(s, s')$

The calculus is sound (if $\Box$ is derived from F, then F is unsatisfiable)
and refutationally complete (if F is unsatisfiable, then $\Box$ can be derived from it).

$\Rightarrow$ FOL with equality is semi-decidable.

Time for a Demo!

Time for a Demo!

You can also run VAMPIRE yourself! Download a pre-compiled binary (for Linux) or build it from the source. All available at `https://github.com/vprover/vampire`

Time for a Demo!

You can also run VAMPIRE yourself! Download a pre-compiled binary (for Linux) or build it from the source. All available at `https://github.com/vprover/vampire`

Run it as: `./vampire <input_file> --show_everything on`
Supported input formats are TPTP and SMT-LIB. E.g.:

```
(declare-datatypes ((nat 0)) (((zero) (s (s0 nat)))))
(declare-fun add (nat nat) nat)
(assert (forall ((y nat)) (= (add zero y) y)))
(assert (forall ((x nat) (y nat)) (= (add (s x) y) (s (add x y)))))
(assert (not (forall ((x nat) (y nat) (z nat)) (= (add x (add y z)) (add (add x y) z)))))
```

Time for a Demo!

You can also run Vampire yourself! Download a pre-compiled binary (for Linux) or build it from the source. All available at `https://github.com/vprover/vampire`

Run it as: `./vampire <input_file> --show_everything on`
Supported input formats are TPTP and SMT-LIB. E.g.:

```
(declare-datatypes ((nat 0)) (((zero) (s (s0 nat)))))
(declare-fun add (nat nat) nat)
(assert (forall ((y nat)) (= (add zero y) y)))
(assert (forall ((x nat) (y nat)) (= (add (s x) y) (s (add x y)))))
(assert (not (forall ((x nat) (y nat) (z nat)) (= (add x (add y z)) (add (add x y) z)))))
```

Useful options:

- `--time_limit <seconds>`
- `--induction <struct/int/both>`
- `--mode portfolio` (optionally with `--schedule induction`)

Finding Proofs Automatically: Saturation Algorithms

Suppose that we have an inference system $\mathbb{I}$ (collection of inference rules).

- Take a set of clauses S (the search space), initially $S = S_0$. Repeatedly apply inferences in $\mathbb{I}$ to clauses in S and add their conclusions to S, unless these conclusions are already in S.
- If, at any stage, we obtain $\square$, we terminate and report unsatisfiability of S_0.

Why Saturation?

In a way, we are trying to build a set S such that any inference applied to clauses in S is already a member of S. Any such set of clauses is called saturated (with respect to $\mathbb{I}$).

The process of trying to build a saturated set is referred to as saturation.

There is also a notion of saturation up to redundancy. In practice, saturated sets are normally infinite.

Inference Selection

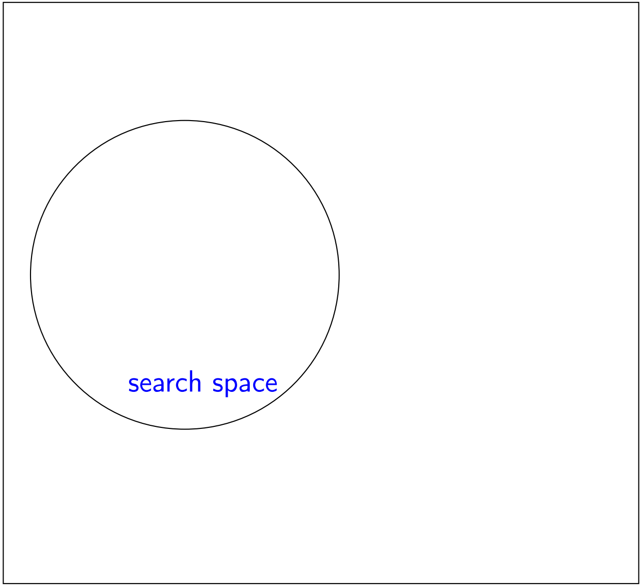
Suppose we have 10^6 clauses in the search space. Then there are potentially 10^{12} inferences with them.

How do we pick up the next inference?

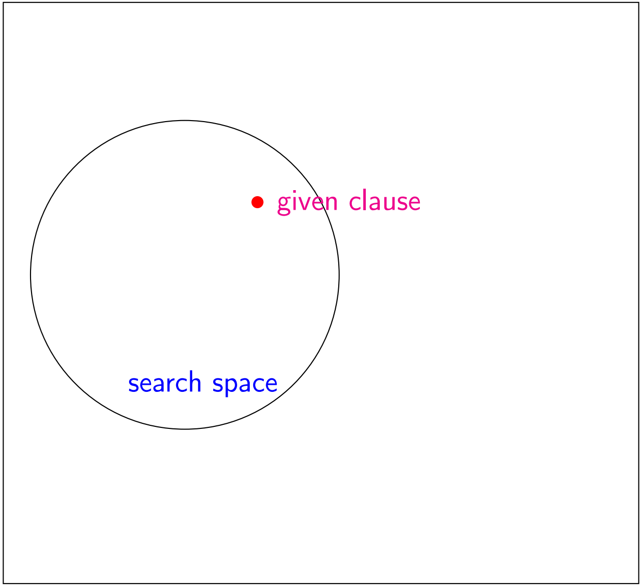
Note that, to build a saturated set, we need the inference selection to be fair: all possible inferences must eventually be performed.

All modern theorem provers use variations of the given clause algorithm.

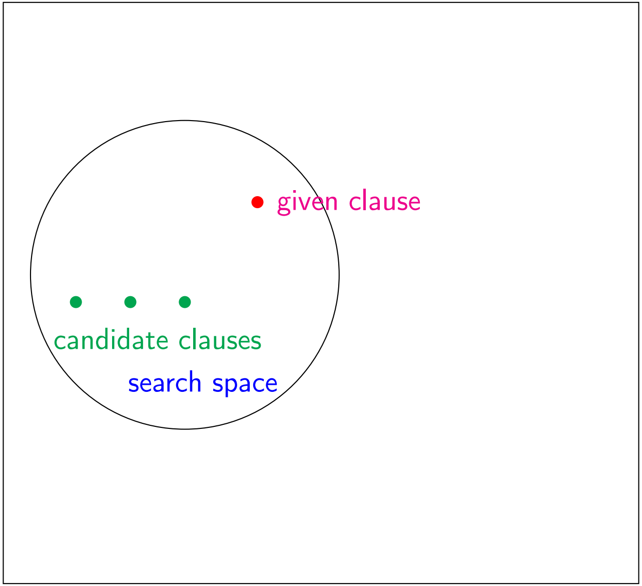
Fair Saturation Algorithms: Inference Selection by Clause Selection



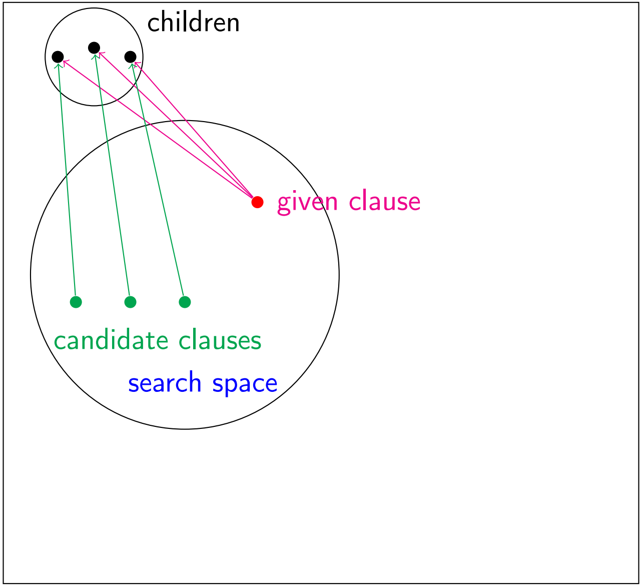
Fair Saturation Algorithms: Inference Selection by Clause Selection



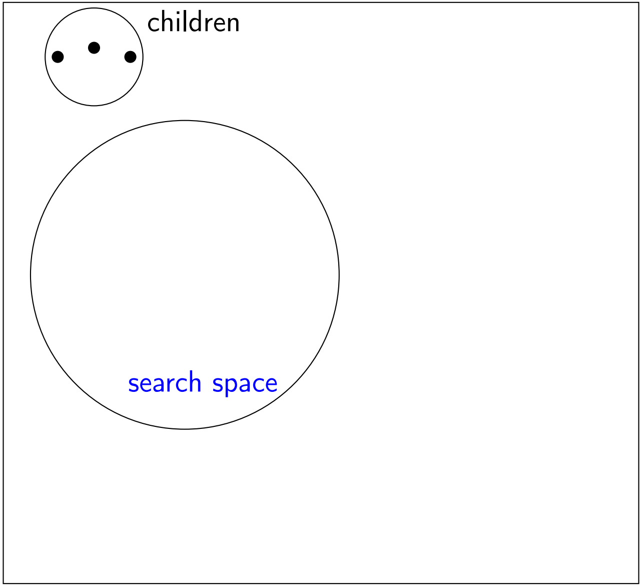
Fair Saturation Algorithms: Inference Selection by Clause Selection



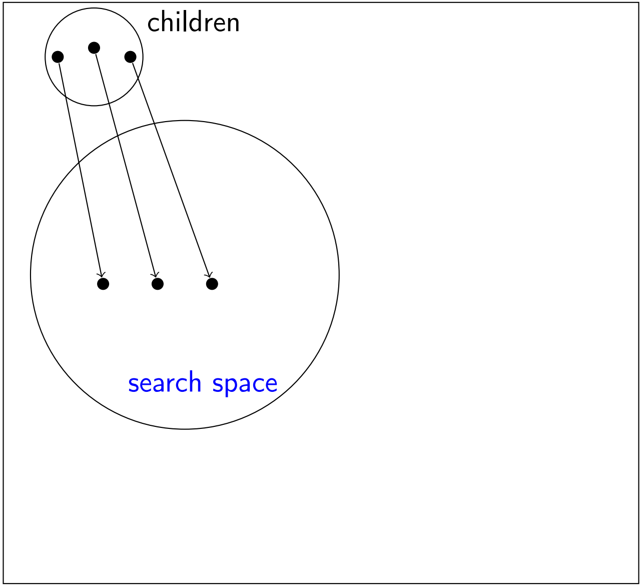
Fair Saturation Algorithms: Inference Selection by Clause Selection



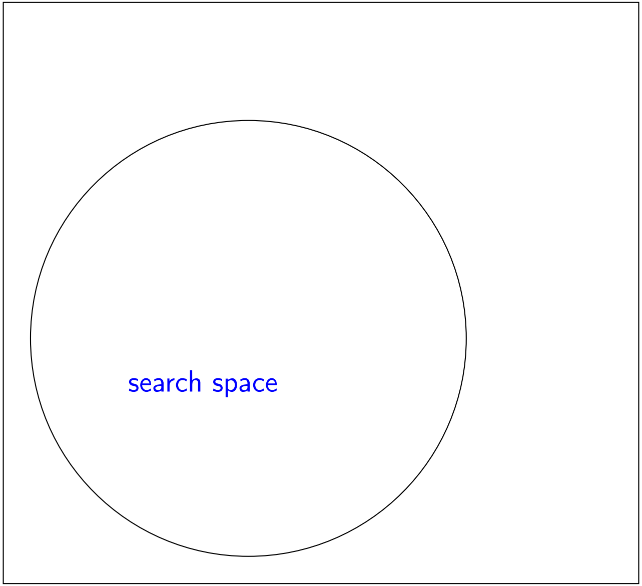
Fair Saturation Algorithms: Inference Selection by Clause Selection



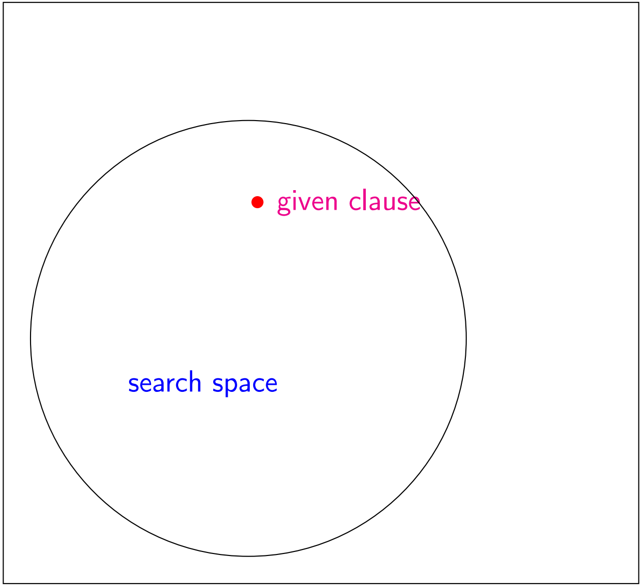
Fair Saturation Algorithms: Inference Selection by Clause Selection



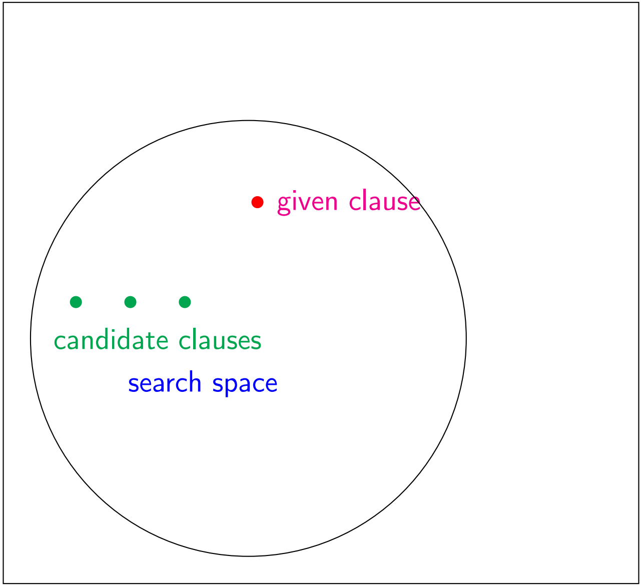
Fair Saturation Algorithms: Inference Selection by Clause Selection



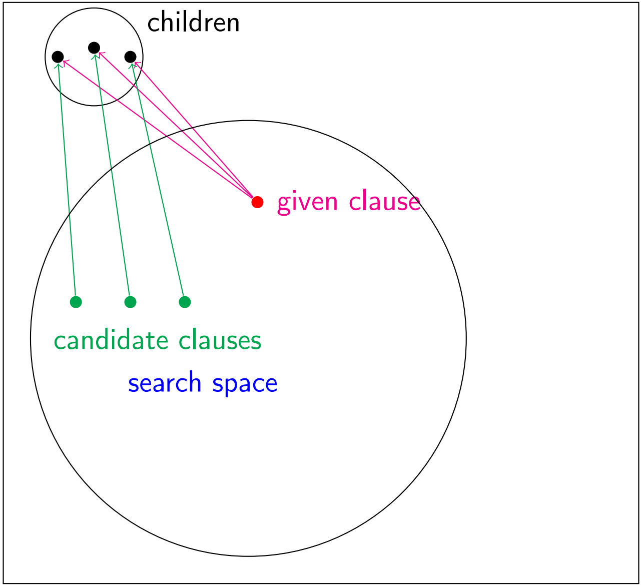
Fair Saturation Algorithms: Inference Selection by Clause Selection



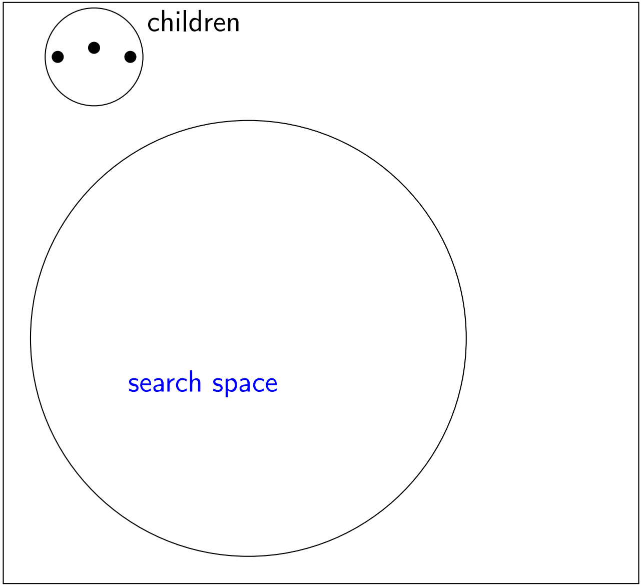
Fair Saturation Algorithms: Inference Selection by Clause Selection



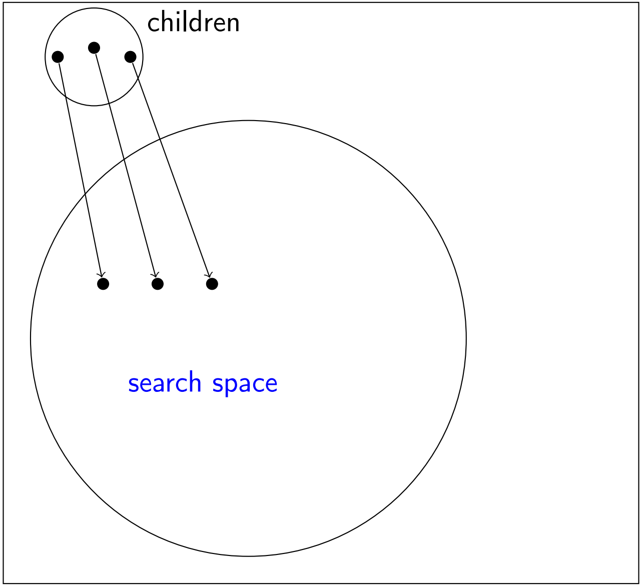
Fair Saturation Algorithms: Inference Selection by Clause Selection



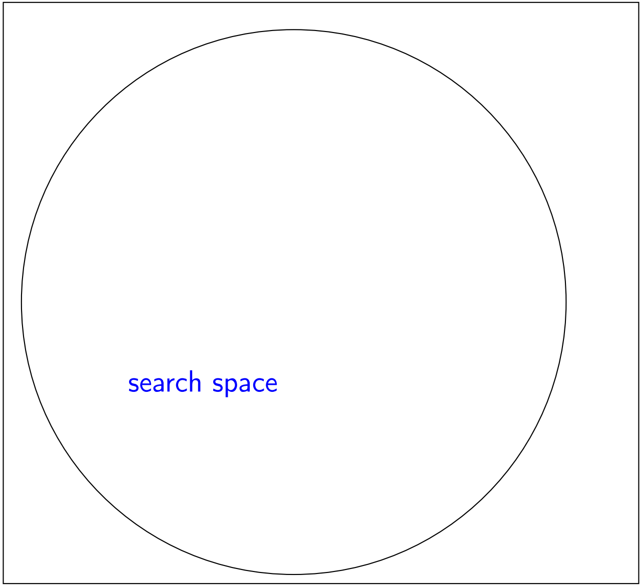
Fair Saturation Algorithms: Inference Selection by Clause Selection



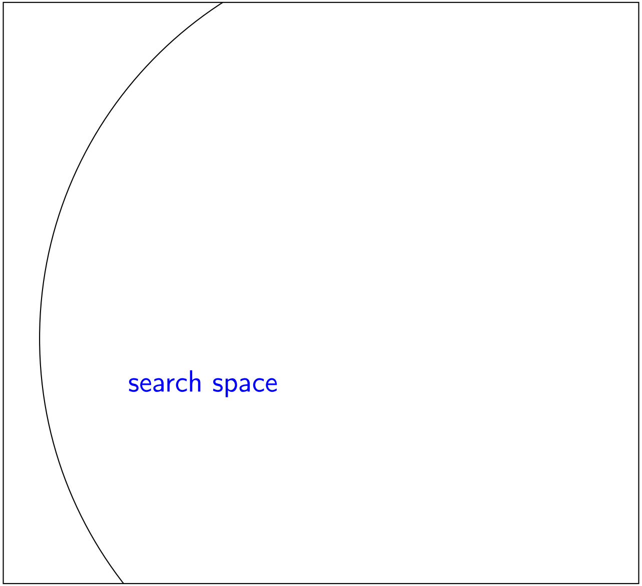
Fair Saturation Algorithms: Inference Selection by Clause Selection



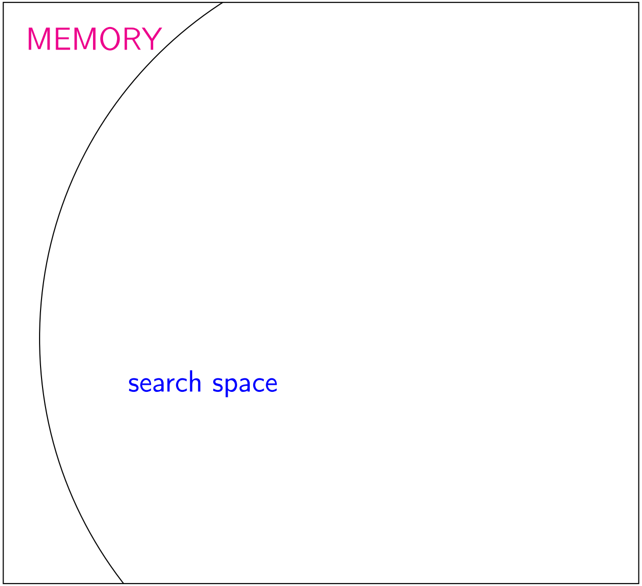
Fair Saturation Algorithms: Inference Selection by Clause Selection



Fair Saturation Algorithms: Inference Selection by Clause Selection



Fair Saturation Algorithms: Inference Selection by Clause Selection



Note on Quantifiers

The prover works with clauses with free variables – without quantifiers.

We preprocess the input I to obtain $CNF(I)$:

1. convert the formulas into prenex normal form ("pull out quantifiers")
2. skolemize existentially quantified variables, for each $\exists y$:
   change $\forall x_1, ..., x_n.\exists y.F[x_1, \ldots, x_n, y]$ to $\forall x_1, ..., x_n.F[x_1, \ldots, x_n, \sigma(x_1, \ldots, x_n)]$
3. clausify $\forall x_1, \ldots, x_n.F$ to produce clauses with implicitly $\forall$-quantified variables

Note on Quantifiers

The prover works with clauses with free variables – without quantifiers.

We preprocess the input I to obtain $CNF(I)$:

1. convert the formulas into prenex normal form ("pull out quantifiers")
2. skolemize existentially quantified variables, for each $\exists y$:
   change $\forall x_1, ..., x_n . \exists y . F[x_1, \ldots, x_n, y]$ to $\forall x_1, ..., x_n . F[x_1, \ldots, x_n, \sigma(x_1, \ldots, x_n)]$
3. clausify $\forall x_1, \ldots, x_n . F$ to produce clauses with implicitly $\forall$-quantified variables

Skolemization produces equisatisfiable formulas & we are proving unsatisfiablity

$\Rightarrow$ we can work with arbitrary $\forall / \exists$ alternations!

Reasoning with Theories

Add axioms and specialized inference rules, e.g.:

$$\forall x, y \in \mathbb{Z}.\ x + y = y + x \qquad\qquad \frac{\mathrm{s}(t) = \mathrm{s}(t') \lor C}{t = t' \lor C}$$

This approach is in general incomplete, but in practice we can prove a lot!

Outline

A Simple Example

Consider the datatype of natural numbers $\mathbb{N}$ and the following definition of $+$:

$$\forall y \in \mathbb{N}.\ 0 + y = y$$
$$\forall x, y \in \mathbb{N}.\ \mathrm{s}(x) + y = \mathrm{s}(x + y)$$

How to prove the associativity of $+$?

$$\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$$

Solving the Example

How to prove the associativity of $+$?

$$\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$$

We use structural induction on $\mathbb{N}$:

$$F[0] \land \forall n \in \mathbb{N}.(F[n] \to F[\mathrm{s}(n)]) \to \forall n \in \mathbb{N}.F[n]$$

Solving the Example

How to prove the associativity of $+$?

$$\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$$

We use structural induction on $\mathbb{N}$:

$$F[0] \wedge \forall n \in \mathbb{N}.(F[n] \rightarrow F[\mathrm{s}(n)]) \rightarrow \forall n \in \mathbb{N}.F[n]$$

We instantiate the axiom by $F[x] := \forall y, z.\ x + (y + z) = (x + y) + z$.

Solving the Example

How to prove the associativity of $+$?

$$\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$$

We use structural induction on $\mathbb{N}$:

$$F[0] \wedge \forall n \in \mathbb{N}.(F[n] \rightarrow F[\mathtt{s}(n)]) \rightarrow \forall n \in \mathbb{N}.F[n]$$

We instantiate the axiom by $F[x] := \forall y, z.\ x + (y + z) = (x + y) + z$.

Proof of the base and step cases using the axioms of $+$ is straightforward:

- $\forall y, z.\ 0 + (y + z) = (0 + y) + z$
- $\forall x.(\forall y, z.\ x + (y + z) = (x + y) + z \rightarrow \forall y, z.\ \mathtt{s}(x) + (y + z) = (\mathtt{s}(x) + y) + z)$

Solving the Example

How to prove the associativity of $+$?

$$\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$$

We use structural induction on $\mathbb{N}$:

$$F[0] \wedge \forall n \in \mathbb{N}.(F[n] \rightarrow F[\mathtt{s}(n)]) \rightarrow \forall n \in \mathbb{N}.F[n]$$

We instantiate the axiom by $F[x] := \forall y, z.\ x + (y + z) = (x + y) + z$.

Proof of the base and step cases using the axioms of $+$ is straightforward:

- $\forall y, z.\ 0 + (y + z) = (0 + y) + z$
- $\forall x.(\forall y, z.\ x + (y + z) = (x + y) + z \rightarrow \forall y, z.\ \mathtt{s}(x) + (y + z) = (\mathtt{s}(x) + y) + z)$

But how to automate this type of reasoning?

How to Use Induction in First-Order Automated Proving?

We want to use induction principle/axioms in proving.
E.g., structural induction axiom for $\mathbb{N}$:

$$F[0] \wedge \forall y \in \mathbb{N}.(F[y] \rightarrow F[\mathrm{s}(y)]) \rightarrow \forall x \in \mathbb{N}.F[x]$$

How to Use Induction in First-Order Automated Proving?

We want to use induction principle/axioms in proving.
E.g., structural induction axiom for $\mathbb{N}$:

$$F[0] \land \forall y \in \mathbb{N}.(F[y] \to F[\mathtt{s}(y)]) \to \forall x \in \mathbb{N}.F[x]$$

Challenges:

1. How to find suitable instances of the induction axioms?
2. How to use these axioms within the reasoning framework?

How to Use Induction in First-Order Automated Proving?

We want to use induction principle/axioms in proving.
E.g., structural induction axiom for $\mathbb{N}$:

$$F[0] \wedge \forall y \in \mathbb{N}.(F[y] \rightarrow F[\mathrm{s}(y)]) \rightarrow \forall x \in \mathbb{N}.F[x]$$

Challenges:

1. How to find suitable instances of the induction axioms?
2. How to use these axioms within the reasoning framework?

Different answers for different solvers:
inductive provers, SMT solvers, saturation-based provers

Induction in ACL2

ACL2 is based on term rewriting, uses goal/subgoal architecture, and a waterfall proving scheme:

- simplification
- eliminate destructors by introducing constructors
- generalize
- eliminate irrelevant hypothesis
- use induction to derive new subgoals, and repeat.

Induction schemes: based on recursive function definitions, for both datatypes and integers.

Limited expressive power: no explicit quantification, and only terminating recursive functions.

Induction in CVC5

CVC5 is an SMT solver: it combines decision procedures for theories using DPLL(T).

It implements inductive strengthening of goals to be proven.

When skolemizing a variable as σ, it uses an assumption "σ is the smallest such value".

Works for both datatypes and integers.

Induction in CVC5

CVC5 is an SMT solver: it combines decision procedures for theories using DPLL(T).

It implements inductive strengthening of goals to be proven.

When skolemizing a variable as σ, it uses an assumption "σ is the smallest such value".

Works for both datatypes and integers.

Additional subgoal discovery:

- hypothesize a subgoal $\forall x. t = s$ based on terms in existing goals
- check if $t = s$ uses canonical terms and does not contradict ground facts
- if the check passes, split on $(\neg \forall x. t = s \vee \forall x. t = s)$.
  Try refuting the first disjunct with induction, if that works, use the second disjunct.

Induction in ZIPPERPOSITION

ZIPPERPOSITION uses interleaving of induction and regular first-order superposition.

Induction axioms are instantiated using recursive function definitions only for datatypes. AVATAR handles splitting for:

- induction cases, where it splits on the constructor of the datatype,
- hypothesized lemmas, where it splits on whether the lemma holds.

Lemmas are hypothesized based on subgoals occuring in the search space and their generalizations.

The VAMPIRE Approach

To prove $L[t]$, we can use a valid induction axiom *premise* $\rightarrow \forall x.L[x]$, e.g.:

$$L[0] \land \forall y \in \mathbb{N}.(L[y] \rightarrow L[\mathrm{s}(y)]) \;\rightarrow\; \forall x \in \mathbb{N}.L[x]$$

$$\forall x.(\overline{L}[x] \rightarrow \exists y.(x \succ y \land \overline{L}[y])) \;\rightarrow\; \forall x.L[x]$$

The VAMPIRE Approach

To refute $\overline{L}[t]$, we can use a valid induction axiom $premise \rightarrow \forall x.L[x]$, e.g.:

$$L[0] \land \forall y \in \mathbb{N}.(L[y] \rightarrow L[\mathrm{s}(y)]) \ \rightarrow \ \forall x \in \mathbb{N}.L[x]$$

$$\forall x.(\overline{L}[x] \rightarrow \exists y.(x \succ y \land \overline{L}[y])) \ \rightarrow \ \forall x.L[x]$$

Further, $premise \rightarrow \forall x.L[x] \ \equiv \ CNF(\neg premise) \lor L[x]$.

The VAMPIRE Approach

To refute $\overline{L}[t]$, we can use a valid induction axiom $premise \rightarrow \forall x.L[x]$, e.g.:

$$L[0] \wedge \forall y \in \mathbb{N}.(L[y] \rightarrow L[\mathrm{s}(y)]) \;\rightarrow\; \forall x \in \mathbb{N}.L[x]$$
$$\forall x.(\overline{L}[x] \rightarrow \exists y.(x \succ y \wedge \overline{L}[y])) \;\rightarrow\; \forall x.L[x]$$

Further, $premise \rightarrow \forall x.L[x] \;\equiv\; CNF(\neg premise) \vee L[x]$.

Our approach [Reger & Voronkov CADE'19]:

1. New rule which uses $\overline{L}[t]$ to instantiate an induction axiom: $\dfrac{\overline{L}[t] \vee C}{premise \rightarrow \forall x.L[x]}$

The VAMPIRE Approach

To refute $\overline{L}[t]$, we can use a valid induction axiom $premise \rightarrow \forall x.L[x]$, e.g.:

$$L[0] \wedge \forall y \in \mathbb{N}.(L[y] \rightarrow L[\mathrm{s}(y)]) \;\rightarrow\; \forall x \in \mathbb{N}.L[x]$$

$$\forall x.(\overline{L}[x] \rightarrow \exists y.(x \succ y \wedge \overline{L}[y])) \;\rightarrow\; \forall x.L[x]$$

Further, $premise \rightarrow \forall x.L[x] \;\equiv\; CNF(\neg premise) \vee L[x]$.

Our approach [Reger & Voronkov CADE'19]:

1. New rule which uses $\overline{L}[t]$ to instantiate an induction axiom: $\dfrac{\overline{L}[t] \vee C}{premise \rightarrow \forall x.L[x]}$
2. Clausify $premise \rightarrow \forall x.L[x]$ to $CNF(\neg premise) \vee L[x]$, then apply binary resolution with $\overline{L}[t] \vee C$.

The VAMPIRE Approach

To refute $\overline{L}[t]$, we can use a valid induction axiom *premise* $\rightarrow \forall x.L[x]$, e.g.:

$$L[0] \land \forall y \in \mathbb{N}.(L[y] \rightarrow L[\mathsf{s}(y)]) \;\rightarrow\; \forall x \in \mathbb{N}.L[x]$$

$$\forall x.(\overline{L}[x] \rightarrow \exists y.(x \succ y \land \overline{L}[y])) \;\rightarrow\; \forall x.L[x]$$

Further, *premise* $\rightarrow \forall x.L[x] \;\equiv\; \mathit{CNF}(\neg \mathit{premise}) \lor L[x]$.

Our approach [Reger & Voronkov CADE'19]:

1. New rule which uses $\overline{L}[t]$ to instantiate an induction axiom: $\dfrac{\overline{L}[t] \lor C}{\mathit{premise} \rightarrow \forall x.L[x]}$
2. Clausify *premise* $\rightarrow \forall x.L[x]$ to $\mathit{CNF}(\neg \mathit{premise}) \lor L[x]$, then apply binary resolution with $\overline{L}[t] \lor C$.

Result: $\mathit{CNF}(\neg \mathit{premise}) \lor C$, corresponding to "negated base and step cases or C".

Time for a demo!

```
% Refutation found. Thanks to Tanya!
1. ! [X0:nat] : add(zero,X0) = X0 [input]
2. ! [X1:nat,X0:nat] : s(add(X0,X1)) = add(s(X0),X1) [input]
3. ~! [X1:nat,X0:nat,X2:nat] : add(add(X0,X1),X2) = add(X0,add(X1,X2)) [input]
7. ! [X0:nat,X1:nat] : s(add(X1,X0)) = add(s(X1),X0) [rectify 2]
8. ~! [X0:nat,X1:nat,X2:nat] : add(add(X1,X0),X2) = add(X1,add(X0,X2)) [rectify 3]
9. ? [X0:nat,X1:nat,X2:nat] : add(add(X1,X0),X2) != add(X1,add(X0,X2)) [ennf transformation 8]
10. ? [X0:nat,X1:nat,X2:nat] : add(add(X1,X0),X2) != add(X1,add(X0,X2)) => add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [choice ax.]
11. add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [skolemisation 9,10]
12. add(zero,X0) = X0 [cnf transformation 1]
13. s(add(X1,X0)) = add(s(X1),X0) [cnf transformation 7]
14. add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [cnf transformation 11]
26. ! [X0:nat] : (add(add(zero,sK0),sK2) = add(zero,add(sK0,sK2)) & (add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)) =>
add(add(s(X0),sK0),sK2) = add(s(X0),add(sK0,sK2)))) => ! [X1:nat] : add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) [struct. ind. ax.]
27. ! [X1:nat] : add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) | ? [X0:nat] : (add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) |
(add(add(s(X0),sK0),sK2) != add(s(X0),add(sK0,sK2)) & add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)))) [ennf transformation 26]
28. add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
29. add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
30. add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [ind. hyperresolution 14,29]
31. add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [induction hyperresolution 14,28]
55. add(sK0,sK2) != add(add(zero,sK0),sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 31,12]
56. add(sK0,sK2) != add(sK0,sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 55,12]
57. add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [trivial inequality removal 56]
58. add(add(s(sK5),sK0),sK2) != s(add(sK5,add(sK0,sK2))) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 30,13]
59. s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 58,13]
60. add(sK0,sK2) != add(add(zero,sK0),sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 59,12]
61. add(sK0,sK2) != add(sK0,sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 60,12]
62. s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [trivial inequality removal 61]
176. s(add(sK5,add(sK0,sK2))) != s(add(add(sK5,sK0),sK2)) [superposition 62,13]
195. add(add(sK5,sK0),sK2) != add(sK5,add(sK0,sK2)) [term algebras injectivity 176]
196. $false [subsumption resolution 195,57]
```

Proof by refutation, inferences from previously derived formulas, theory reasoning, preprocessing, induction, superposition calculus, unused formulas

```
% Refutation found.  Thanks to Tanya!
1.  !  [X0:nat] :  add(zero,X0) = X0 [input]
2.  !  [X1:nat,X0:nat] :  s(add(X0,X1)) = add(s(X0),X1) [input]
3.  ~!  [X1:nat,X0:nat,X2:nat] :  add(add(X0,X1),X2) = add(X0,add(X1,X2)) [input]
7.  !  [X0:nat,X1:nat] :  s(add(X1,X0)) = add(s(X1),X0) [rectify 2]
8.  ~!  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) = add(X1,add(X0,X2)) [rectify 3]
9.  ?  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) != add(X1,add(X0,X2)) [ennf transformation 8]
10.  ?  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) != add(X1,add(X0,X2)) => add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [choice ax.]
11.  add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [skolemisation 9,10]
12.  add(zero,X0) = X0 [cnf transformation 1]
13.  s(add(X1,X0)) = add(s(X1),X0) [cnf transformation 7]
14.  add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [cnf transformation 11]
26.  !  [X0:nat] :  (add(add(zero,sK0),sK2) = add(zero,add(sK0,sK2)) & (add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)) =>
add(add(s(X0),sK0),sK2) = add(s(X0),add(sK0,sK2)))) => !  [X1:nat] :  add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) [struct.  ind.  ax.]
27.  !  [X1:nat] :  add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) | ?  [X0:nat] :  (add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) |
(add(add(s(X0),sK0),sK2) != add(s(X0),add(sK0,sK2)) & add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)))) [ennf transformation 26]
28.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
29.  add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
30.  add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [ind.  hyperresolution 14,29]
31.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [induction hyperresolution 14,28]
55.  add(sK0,sK2) != add(add(zero,sK0),sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 31,12]
56.  add(sK0,sK2) != add(sK0,sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 55,12]
57.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [trivial inequality removal 56]
58.  add(add(s(sK5),sK0),sK2) != s(add(sK5,add(sK0,sK2))) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 30,13]
59.  s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 58,13]
60.  add(sK0,sK2) != add(add(zero,sK0),sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 59,12]
61.  add(sK0,sK2) != add(sK0,sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 60,12]
62.  s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [trivial inequality removal 61]
176.  s(add(sK5,add(sK0,sK2))) != s(add(add(sK5,sK0),sK2)) [superposition 62,13]
195.  add(add(sK5,sK0),sK2) != add(sK5,add(sK0,sK2)) [term algebras injectivity 176]
196.  $false [subsumption resolution 195,57]
```

Proof by refutation, inferences from previously derived formulas, theory reasoning, preprocessing, induction, superposition calculus, unused formulas

```
% Refutation found. Thanks to Tanya!
1. ! [X0:nat] : add(zero,X0) = X0 [input]
2. ! [X1:nat,X0:nat] : s(add(X0,X1)) = add(s(X0),X1) [input]
3. ~! [X1:nat,X0:nat,X2:nat] : add(add(X0,X1),X2) = add(X0,add(X1,X2)) [input]
7. ! [X0:nat,X1:nat] : s(add(X1,X0)) = add(s(X1),X0) [rectify 2]
8. ~! [X0:nat,X1:nat,X2:nat] : add(add(X1,X0),X2) = add(X1,add(X0,X2)) [rectify 3]
9. ? [X0:nat,X1:nat,X2:nat] : add(add(X1,X0),X2) != add(X1,add(X0,X2)) [ennf transformation 8]
10. ? [X0:nat,X1:nat,X2:nat] : add(add(X1,X0),X2) != add(X1,add(X0,X2)) => add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [choice ax.]
11. add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [skolemisation 9,10]
12. add(zero,X0) = X0 [cnf transformation 1]
13. s(add(X1,X0)) = add(s(X1),X0) [cnf transformation 7]
14. add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [cnf transformation 11]
26. ! [X0:nat] : (add(add(zero,sK0),sK2) = add(zero,add(sK0,sK2)) & (add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)) =>
add(add(s(X0),sK0),sK2) = add(s(X0),add(sK0,sK2)))) => ! [X1:nat] : add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) [struct. ind. ax.]
27. ! [X1:nat] : add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) | ? [X0:nat] : (add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) |
(add(add(s(X0),sK0),sK2) != add(s(X0),add(sK0,sK2)) & add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)))) [ennf transformation 26]
28. add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
29. add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
30. add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [ind. hyperresolution 14,29]
31. add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [induction hyperresolution 14,28]
55. add(sK0,sK2) != add(add(zero,sK0),sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 31,12]
56. add(sK0,sK2) != add(sK0,sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 55,12]
57. add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [trivial inequality removal 56]
58. add(add(s(sK5),sK0),sK2) != s(add(sK5,add(sK0,sK2))) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 30,13]
59. s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 58,13]
60. add(sK0,sK2) != add(add(zero,sK0),sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 59,12]
61. add(sK0,sK2) != add(sK0,sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 60,12]
62. s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [trivial inequality removal 61]
176. s(add(sK5,add(sK0,sK2))) != s(add(add(sK5,sK0),sK2)) [superposition 62,13]
195. add(add(sK5,sK0),sK2) != add(sK5,add(sK0,sK2)) [term algebras injectivity 176]
196. $false [subsumption resolution 195,57]
```

Proof by refutation, inferences from previously derived formulas, theory reasoning, preprocessing, induction, superposition calculus, unused formulas

```
% Refutation found.  Thanks to Tanya!
1.  !  [X0:nat] :  add(zero,X0) = X0 [input]
2.  !  [X1:nat,X0:nat] :  s(add(X0,X1)) = add(s(X0),X1) [input]
3.  ~!  [X1:nat,X0:nat,X2:nat] :  add(add(X0,X1),X2) = add(X0,add(X1,X2)) [input]
7.  !  [X0:nat,X1:nat] :  s(add(X1,X0)) = add(s(X1),X0) [rectify 2]
8.  ~!  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) = add(X1,add(X0,X2)) [rectify 3]
9.  ?  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) != add(X1,add(X0,X2)) [ennf transformation 8]
10.  ?  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) != add(X1,add(X0,X2)) => add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [choice ax.]
11.  add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [skolemisation 9,10]
12.  add(zero,X0) = X0 [cnf transformation 1]
13.  s(add(X1,X0)) = add(s(X1),X0) [cnf transformation 7]
14.  add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [cnf transformation 11]
26.  !  [X0:nat] :  (add(add(zero,sK0),sK2) = add(zero,add(sK0,sK2)) & (add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)) =>
add(add(s(X0),sK0),sK2) = add(s(X0),add(sK0,sK2)))) => !  [X1:nat] :  add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) [struct.  ind.  ax.]
27.  !  [X1:nat] :  add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) | ?  [X0:nat] :  (add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) |
(add(add(s(X0),sK0),sK2) != add(s(X0),add(sK0,sK2)) & add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)))) [ennf transformation 26]
28.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
29.  add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
30.  add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [ind.  hyperresolution 14,29]
31.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [induction hyperresolution 14,28]
55.  add(sK0,sK2) != add(add(zero,sK0),sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 31,12]
56.  add(sK0,sK2) != add(sK0,sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 55,12]
57.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [trivial inequality removal 56]
58.  add(add(s(sK5),sK0),sK2) != s(add(sK5,add(sK0,sK2))) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 30,13]
59.  s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 58,13]
60.  add(sK0,sK2) != add(add(zero,sK0),sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 59,12]
61.  add(sK0,sK2) != add(sK0,sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 60,12]
62.  s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [trivial inequality removal 61]
176.  s(add(sK5,add(sK0,sK2))) != s(add(add(sK5,sK0),sK2)) [superposition 62,13]
195.  add(add(sK5,sK0),sK2) != add(sK5,add(sK0,sK2)) [term algebras injectivity 176]
196.  $false [subsumption resolution 195,57]
```

Proof by refutation, inferences from previously derived formulas, theory reasoning, preprocessing, induction, superposition calculus, unused formulas

```
% Refutation found.  Thanks to Tanya!
1.  !  [X0:nat] :  add(zero,X0) = X0 [input]
2.  !  [X1:nat,X0:nat] :  s(add(X0,X1)) = add(s(X0),X1) [input]
3.  ~!  [X1:nat,X0:nat,X2:nat] :  add(add(X0,X1),X2) = add(X0,add(X1,X2)) [input]
7.  !  [X0:nat,X1:nat] :  s(add(X1,X0)) = add(s(X1),X0) [rectify 2]
8.  ~!  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) = add(X1,add(X0,X2)) [rectify 3]
9.  ?  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) != add(X1,add(X0,X2)) [ennf transformation 8]
10.  ?  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) != add(X1,add(X0,X2)) => add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [choice ax.]
11.  add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [skolemisation 9,10]
12.  add(zero,X0) = X0 [cnf transformation 1]
13.  s(add(X1,X0)) = add(s(X1),X0) [cnf transformation 7]
14.  add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [cnf transformation 11]
26.  !  [X0:nat] :  (add(add(zero,sK0),sK2) = add(zero,add(sK0,sK2)) & (add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)) =>
add(add(s(X0),sK0),sK2) = add(s(X0),add(sK0,sK2)))) => !  [X1:nat] :  add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) [struct.  ind.  ax.]
27.  !  [X1:nat] :  add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) | ?  [X0:nat] :  (add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) |
(add(add(s(X0),sK0),sK2) != add(s(X0),add(sK0,sK2)) & add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)))) [ennf transformation 26]
28.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
29.  add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
30.  add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [ind.  hyperresolution 14,29]
31.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [induction hyperresolution 14,28]
55.  add(sK0,sK2) != add(add(zero,sK0),sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 31,12]
56.  add(sK0,sK2) != add(sK0,sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 55,12]
57.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [trivial inequality removal 56]
58.  add(add(s(sK5),sK0),sK2) != s(add(sK5,add(sK0,sK2))) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 30,13]
59.  s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 58,13]
60.  add(sK0,sK2) != add(add(zero,sK0),sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 59,12]
61.  add(sK0,sK2) != add(sK0,sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 60,12]
62.  s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [trivial inequality removal 61]
176.  s(add(sK5,add(sK0,sK2))) != s(add(add(sK5,sK0),sK2)) [superposition 62,13]
195.  add(add(sK5,sK0),sK2) != add(sK5,add(sK0,sK2)) [term algebras injectivity 176]
196.  $false [subsumption resolution 195,57]
```

Proof by refutation, inferences from previously derived formulas, theory reasoning, preprocessing, induction, superposition calculus, unused formulas

```
% Refutation found.  Thanks to Tanya!
1.  !  [X0:nat] :  add(zero,X0) = X0 [input]
2.  !  [X1:nat,X0:nat] :  s(add(X0,X1)) = add(s(X0),X1) [input]
3.  ~!  [X1:nat,X0:nat,X2:nat] :  add(add(X0,X1),X2) = add(X0,add(X1,X2)) [input]
7.  !  [X0:nat,X1:nat] :  s(add(X1,X0)) = add(s(X1),X0) [rectify 2]
8.  ~!  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) = add(X1,add(X0,X2)) [rectify 3]
9.  ?  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) != add(X1,add(X0,X2)) [ennf transformation 8]
10.  ?  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) != add(X1,add(X0,X2)) => add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [choice ax.]
11.  add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [skolemisation 9,10]
12.  add(zero,X0) = X0 [cnf transformation 1]
13.  s(add(X1,X0)) = add(s(X1),X0) [cnf transformation 7]
14.  add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [cnf transformation 11]
26.  !  [X0:nat] :  (add(add(zero,sK0),sK2) = add(zero,add(sK0,sK2)) & (add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)) =>
add(add(s(X0),sK0),sK2) = add(s(X0),add(sK0,sK2)))) => !  [X1:nat] :  add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) [struct.  ind.  ax.]
27.  !  [X1:nat] :  add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) | ?  [X0:nat] :  (add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) |
(add(add(s(X0),sK0),sK2) != add(s(X0),add(sK0,sK2)) & add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)))) [ennf transformation 26]
28.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
29.  add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
30.  add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [ind.  hyperresolution 14,29]
31.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [induction hyperresolution 14,28]
55.  add(sK0,sK2) != add(add(zero,sK0),sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 31,12]
56.  add(sK0,sK2) != add(sK0,sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 55,12]
57.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [trivial inequality removal 56]
58.  add(add(s(sK5),sK0),sK2) != s(add(sK5,add(sK0,sK2))) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 30,13]
59.  s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 58,13]
60.  add(sK0,sK2) != add(add(zero,sK0),sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 59,12]
61.  add(sK0,sK2) != add(sK0,sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 60,12]
62.  s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [trivial inequality removal 61]
176.  s(add(sK5,add(sK0,sK2))) != s(add(add(sK5,sK0),sK2)) [superposition 62,13]
195.  add(add(sK5,sK0),sK2) != add(sK5,add(sK0,sK2)) [term algebras injectivity 176]
196.  $false [subsumption resolution 195,57]
```

Proof by refutation, inferences from previously derived formulas, theory reasoning, preprocessing, induction, superposition calculus, unused formulas

```
% Refutation found.  Thanks to Tanya!
1.  !  [X0:nat] :  add(zero,X0) = X0 [input]
2.  !  [X1:nat,X0:nat] :  s(add(X0,X1)) = add(s(X0),X1) [input]
3.  ~!  [X1:nat,X0:nat,X2:nat] :  add(add(X0,X1),X2) = add(X0,add(X1,X2)) [input]
7.  !  [X0:nat,X1:nat] :  s(add(X1,X0)) = add(s(X1),X0) [rectify 2]
8.  ~!  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) = add(X1,add(X0,X2)) [rectify 3]
9.  ?  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) != add(X1,add(X0,X2)) [ennf transformation 8]
10.  ?  [X0:nat,X1:nat,X2:nat] :  add(add(X1,X0),X2) != add(X1,add(X0,X2)) => add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [choice ax.]
11.  add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [skolemisation 9,10]
12.  add(zero,X0) = X0 [cnf transformation 1]
13.  s(add(X1,X0)) = add(s(X1),X0) [cnf transformation 7]
14.  add(add(sK1,sK0),sK2) != add(sK1,add(sK0,sK2)) [cnf transformation 11]
26.  !  [X0:nat] :  (add(add(zero,sK0),sK2) = add(zero,add(sK0,sK2)) & (add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)) =>
add(add(s(X0),sK0),sK2) = add(s(X0),add(sK0,sK2)))) => !  [X1:nat] :  add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) [struct.  ind.  ax.]
27.  !  [X1:nat] :  add(add(X1,sK0),sK2) = add(X1,add(sK0,sK2)) | ?  [X0:nat] :  (add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) |
(add(add(s(X0),sK0),sK2) != add(s(X0),add(sK0,sK2)) & add(add(X0,sK0),sK2) = add(X0,add(sK0,sK2)))) [ennf transformation 26]
28.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
29.  add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) | add(add(X1,sK0),sK2) =
add(X1,add(sK0,sK2)) [cnf transformation 27]
30.  add(add(s(sK5),sK0),sK2) != add(s(sK5),add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [ind.  hyperresolution 14,29]
31.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [induction hyperresolution 14,28]
55.  add(sK0,sK2) != add(add(zero,sK0),sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 31,12]
56.  add(sK0,sK2) != add(sK0,sK2) | add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [forward demodulation 55,12]
57.  add(add(sK5,sK0),sK2) = add(sK5,add(sK0,sK2)) [trivial inequality removal 56]
58.  add(add(s(sK5),sK0),sK2) != s(add(sK5,add(sK0,sK2))) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 30,13]
59.  s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) | add(add(zero,sK0),sK2) != add(zero,add(sK0,sK2)) [forward demodulation 58,13]
60.  add(sK0,sK2) != add(add(zero,sK0),sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 59,12]
61.  add(sK0,sK2) != add(sK0,sK2) | s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [forward demodulation 60,12]
62.  s(add(sK5,add(sK0,sK2))) != add(s(add(sK5,sK0)),sK2) [trivial inequality removal 61]
176.  s(add(sK5,add(sK0,sK2))) != s(add(add(sK5,sK0),sK2)) [superposition 62,13]
195.  add(add(sK5,sK0),sK2) != add(sK5,add(sK0,sK2)) [term algebras injectivity 176]
196.  $false [subsumption resolution 195,57]
```

Proof by refutation, inferences from previously derived formulas, theory reasoning, preprocessing, induction, superposition calculus, unused formulas

Proof in Details

Conjecture: $\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$

Axioms: $\forall y \in \mathbb{N}.\ 0 + y = y, \quad \forall x, y \in \mathbb{N}.\ \mathrm{s}(x) + y = \mathrm{s}(x + y)$

1. $\sigma_1 + (\sigma_2 + \sigma_3) \neq (\sigma_1 + \sigma_2) + \sigma_3$ [negated and skolemized input]
2. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3 \vee$
   $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
3. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3 \vee$
   $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
4. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3$ [BR 1, 2]
5. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3$ [BR 1, 3]
6. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma + (\sigma_2 + \sigma_3)) \neq \mathrm{s}((\sigma + \sigma_2) + \sigma_3)$ [5, axiom]
7. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) \neq (\sigma + \sigma_2) + \sigma_3$ [s injectivity 6]
8. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3$ [BR 4, 7]
9. $\sigma_2 + \sigma_3 \neq \sigma_2 + \sigma_3$ [8, axiom]
10. $\square$ [trivial inequality removal 9]

Proof in Details

Conjecture: $\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$

Axioms: $\forall y \in \mathbb{N}.\ 0 + y = y, \quad \forall x, y \in \mathbb{N}.\ \mathrm{s}(x) + y = \mathrm{s}(x + y)$

1. $\sigma_1 + (\sigma_2 + \sigma_3) \neq (\sigma_1 + \sigma_2) + \sigma_3$ [negated and skolemized input]
2. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3 \vee$
   $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
3. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3 \vee$
   $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
4. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3$ [BR 1, 2]
5. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3$ [BR 1, 3]
6. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma + (\sigma_2 + \sigma_3)) \neq \mathrm{s}((\sigma + \sigma_2) + \sigma_3)$ [5, axiom]
7. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) \neq (\sigma + \sigma_2) + \sigma_3$ [s injectivity 6]
8. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3$ [BR 4, 7]
9. $\sigma_2 + \sigma_3 \neq \sigma_2 + \sigma_3$ [8, axiom]
10. $\square$ [trivial inequality removal 9]

Proof in Details

Conjecture: $\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$

Axioms: $\forall y \in \mathbb{N}.\ 0 + y = y, \quad \forall x, y \in \mathbb{N}.\ \mathrm{s}(x) + y = \mathrm{s}(x + y)$

1. $\underline{\sigma_1 + (\sigma_2 + \sigma_3) \neq (\sigma_1 + \sigma_2) + \sigma_3}$ [negated and skolemized input]
2. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3 \vee$ $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
3. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3 \vee$ $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
4. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3$ [BR 1, 2]
5. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3$ [BR 1, 3]
6. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma + (\sigma_2 + \sigma_3)) \neq \mathrm{s}((\sigma + \sigma_2) + \sigma_3)$ [5, axiom]
7. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) \neq (\sigma + \sigma_2) + \sigma_3$ [s injectivity 6]
8. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3$ [BR 4, 7]
9. $\sigma_2 + \sigma_3 \neq \sigma_2 + \sigma_3$ [8, axiom]
10. $\square$ [trivial inequality removal 9]

Proof in Details

Conjecture: $\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$

Axioms: $\forall y \in \mathbb{N}.\ 0 + y = y,\quad \forall x, y \in \mathbb{N}.\ \mathrm{s}(x) + y = \mathrm{s}(x + y)$

1. $\underline{\sigma_1 + (\sigma_2 + \sigma_3) \neq (\sigma_1 + \sigma_2) + \sigma_3}$ [negated and skolemized input]
2. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3 \vee$ $\underline{x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3}$ [Ind 1]
3. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3 \vee$ $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
4. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3$ [BR 1, 2]
5. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3$ [BR 1, 3]
6. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma + (\sigma_2 + \sigma_3)) \neq \mathrm{s}((\sigma + \sigma_2) + \sigma_3)$ [5, axiom]
7. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) \neq (\sigma + \sigma_2) + \sigma_3$ [s injectivity 6]
8. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3$ [BR 4, 7]
9. $\sigma_2 + \sigma_3 \neq \sigma_2 + \sigma_3$ [8, axiom]
10. $\square$ [trivial inequality removal 9]

Proof in Details

Conjecture: $\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$

Axioms: $\forall y \in \mathbb{N}.\ 0 + y = y, \quad \forall x, y \in \mathbb{N}.\ \mathrm{s}(x) + y = \mathrm{s}(x + y)$

1. $\underline{\sigma_1 + (\sigma_2 + \sigma_3) \neq (\sigma_1 + \sigma_2) + \sigma_3}$ [negated and skolemized input]
2. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3 \vee$
   $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
3. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3 \vee$
   $\underline{x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3}$ [Ind 1]
4. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3$ [BR 1, 2]
5. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3$ [BR 1, 3]
6. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma + (\sigma_2 + \sigma_3)) \neq \mathrm{s}((\sigma + \sigma_2) + \sigma_3)$ [5, axiom]
7. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) \neq (\sigma + \sigma_2) + \sigma_3$ [s injectivity 6]
8. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3$ [BR 4, 7]
9. $\sigma_2 + \sigma_3 \neq \sigma_2 + \sigma_3$ [8, axiom]
10. $\square$ [trivial inequality removal 9]

Proof in Details

Conjecture: $\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$

Axioms: $\forall y \in \mathbb{N}.\ 0 + y = y$, $\quad \forall x, y \in \mathbb{N}.\ \mathrm{s}(x) + y = \mathrm{s}(x + y)$

1. $\sigma_1 + (\sigma_2 + \sigma_3) \neq (\sigma_1 + \sigma_2) + \sigma_3$ [negated and skolemized input]
2. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \lor \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3 \lor$
   $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
3. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \lor \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3 \lor$
   $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
4. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \lor \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3$ [BR 1, 2]
5. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \lor \underline{\mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3}$ [BR 1, 3]
6. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \lor \mathrm{s}(\sigma + (\sigma_2 + \sigma_3)) \neq \mathrm{s}((\sigma + \sigma_2) + \sigma_3)$ [5, axiom]
7. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \lor \sigma + (\sigma_2 + \sigma_3) \neq (\sigma + \sigma_2) + \sigma_3$ [s injectivity 6]
8. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3$ [BR 4, 7]
9. $\sigma_2 + \sigma_3 \neq \sigma_2 + \sigma_3$ [8, axiom]
10. $\square$ [trivial inequality removal 9]

Proof in Details

Conjecture: $\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$

Axioms: $\forall y \in \mathbb{N}.\ 0 + y = y,\quad \forall x, y \in \mathbb{N}.\ \mathrm{s}(x) + y = \mathrm{s}(x + y)$

1. $\sigma_1 + (\sigma_2 + \sigma_3) \neq (\sigma_1 + \sigma_2) + \sigma_3$ [negated and skolemized input]
2. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3 \vee$
   $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
3. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3 \vee$
   $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
4. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3$ [BR 1, 2]
5. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3$ [BR 1, 3]
6. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \underline{\mathrm{s}(\sigma + (\sigma_2 + \sigma_3)) \neq \mathrm{s}((\sigma + \sigma_2) + \sigma_3)}$ [5, axiom]
7. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) \neq (\sigma + \sigma_2) + \sigma_3$ [s injectivity 6]
8. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3$ [BR 4, 7]
9. $\sigma_2 + \sigma_3 \neq \sigma_2 + \sigma_3$ [8, axiom]
10. $\square$ [trivial inequality removal 9]

Proof in Details

Conjecture: $\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$

Axioms: $\forall y \in \mathbb{N}.\ 0 + y = y, \quad \forall x, y \in \mathbb{N}.\ \mathrm{s}(x) + y = \mathrm{s}(x + y)$

1. $\sigma_1 + (\sigma_2 + \sigma_3) \neq (\sigma_1 + \sigma_2) + \sigma_3$ [negated and skolemized input]
2. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3 \vee$
   $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
3. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3 \vee$
   $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
4. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \underline{\sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3}$ [BR 1, 2]
5. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3$ [BR 1, 3]
6. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma + (\sigma_2 + \sigma_3)) \neq \mathrm{s}((\sigma + \sigma_2) + \sigma_3)$ [5, axiom]
7. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \underline{\sigma + (\sigma_2 + \sigma_3) \neq (\sigma + \sigma_2) + \sigma_3}$ [s injectivity 6]
8. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3$ [BR 4, 7]
9. $\sigma_2 + \sigma_3 \neq \sigma_2 + \sigma_3$ [8, axiom]
10. $\square$ [trivial inequality removal 9]

Proof in Details

Conjecture: $\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$

Axioms: $\forall y \in \mathbb{N}.\ 0 + y = y$, $\forall x, y \in \mathbb{N}.\ \mathrm{s}(x) + y = \mathrm{s}(x + y)$

1. $\sigma_1 + (\sigma_2 + \sigma_3) \neq (\sigma_1 + \sigma_2) + \sigma_3$ [negated and skolemized input]
2. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3 \vee$ $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
3. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3 \vee$ $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
4. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3$ [BR 1, 2]
5. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3$ [BR 1, 3]
6. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma + (\sigma_2 + \sigma_3)) \neq \mathrm{s}((\sigma + \sigma_2) + \sigma_3)$ [5, axiom]
7. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) \neq (\sigma + \sigma_2) + \sigma_3$ [s injectivity 6]
8. $\underline{0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3}$ [BR 4, 7]
9. $\sigma_2 + \sigma_3 \neq \sigma_2 + \sigma_3$ [8, axiom]
10. $\square$ [trivial inequality removal 9]

Proof in Details

Conjecture: $\forall x, y, z \in \mathbb{N}.\ x + (y + z) = (x + y) + z$

Axioms: $\forall y \in \mathbb{N}.\ 0 + y = y, \quad \forall x, y \in \mathbb{N}.\ \mathrm{s}(x) + y = \mathrm{s}(x + y)$

1. $\sigma_1 + (\sigma_2 + \sigma_3) \neq (\sigma_1 + \sigma_2) + \sigma_3$ [negated and skolemized input]
2. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3 \vee$ $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
3. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3 \vee$ $x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3$ [Ind 1]
4. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3$ [BR 1, 2]
5. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma) + (\sigma_2 + \sigma_3) \neq (\mathrm{s}(\sigma) + \sigma_2) + \sigma_3$ [BR 1, 3]
6. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \mathrm{s}(\sigma + (\sigma_2 + \sigma_3)) \neq \mathrm{s}((\sigma + \sigma_2) + \sigma_3)$ [5, axiom]
7. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) \neq (\sigma + \sigma_2) + \sigma_3$ [s injectivity 6]
8. $0 + (\sigma_2 + \sigma_3) \neq (0 + \sigma_2) + \sigma_3$ [BR 4, 7]
9. $\underline{\sigma_2 + \sigma_3 \neq \sigma_2 + \sigma_3}$ [8, axiom]
10. $\square$ [trivial inequality removal 9]

Note on Efficiency of Structural Induction in Saturation

Consider Ind instantiated with the structural induction axiom for $\mathbb{N}$:

$$\frac{\overline{L}[t] \vee C}{L[0] \wedge \forall y \in \mathbb{N}.(L[y] \rightarrow L[\mathrm{s}(y)]) \rightarrow \forall x \in \mathbb{N}.L[x]} \text{ (Ind)}$$

The CNF of the axiom:

$$\overline{L}[0] \vee L[\sigma_y] \vee L[x]$$
$$\overline{L}[0] \vee \overline{L}[\mathrm{s}(\sigma_y)] \vee L[x]$$

Note on Efficiency of Structural Induction in Saturation

Consider Ind instantiated with the structural induction axiom for $\mathbb{N}$:

$$\frac{\overline{L}[t] \vee C}{L[0] \wedge \forall y \in \mathbb{N}.(L[y] \rightarrow L[\mathrm{s}(y)]) \rightarrow \forall x \in \mathbb{N}.L[x]} \text{ (Ind)}$$

The CNF of the axiom:

$$\overline{L}[0] \vee L[\sigma_y] \vee L[x]$$
$$\overline{L}[0] \vee \overline{L}[\mathrm{s}(\sigma_y)] \vee L[x]$$

After resolution with the premise:

$$\overline{L}[0] \vee L[\sigma_y] \vee C$$
$$\overline{L}[0] \vee \overline{L}[\mathrm{s}(\sigma_y)] \vee C$$

The first two literals of both clauses are ground $\Rightarrow$ friendly to saturation-based provers!

Implementation Concerns

Options controlling:

- which induction schemas are used (well-founded, structural, . . .)
- choosing which literals and terms to apply induction on (ground, negative, complex terms, uninterpreted constants from the conjectures, . . .)
- limit on induction depth

Experiments: Induction Inferences and Depth

Statistics from 165 successful problems in the SMT-LIB benchmark set UFDTLIA:

Number of induction inferences	Count
0	44
1	82
2	16
3	6
5	2
10-50	7
50-145	4

Max induction depth	Count
1	84
2	25
3	4
4	3
6	1

Experiments: Vampire vs. CVC4

Logic	Size	No Induction		With Induction	
		CVC4	Vampire	CVC4	Vampire
UFDT	4376	2270	2226	2275	2294
UFDTLIA	303	69	76	224	165

Experiments: Vampire vs. CVC4

Unique problems:

Logic	Size	No Induction		With Induction	
		CVC4	Vampire	CVC4	Vampire
UFDT	4376	2270	2226 (2)	2275 (5)	2294 (37)
UFDTLIA	303	69	76	224 (69)	165 (9)

Experiments: Vampire vs. CVC4

Unique problems:

Logic	Size	No Induction		With Induction	
		CVC4	Vampire	CVC4	Vampire
UFDT	4376	2270	2226 (2)	2275 (5)	2294 (37)
UFDTLIA	303	69	76	224 (69)	165 (9)

Conclusion: Vampire commonly uses induction to solve a problem more quickly even when no induction is required.

Outline

Why Integer Induction?

Before we only applied induction on algebraic dataypes: natural numbers, lists, . . .

Integers are ubiquitous in programming $\Rightarrow$ verification needs reasoning with $\mathbb{Z}$.

Why Integer Induction?

Before we only applied induction on algebraic dataypes: natural numbers, lists, ...

Integers are ubiquitous in programming $\Rightarrow$ verification needs reasoning with $\mathbb{Z}$.

$\mathbb{Z}$ with the standard ordering are not well-founded – but any set of integers with a lower or an upper bound is.

Example

$$\textbf{fun } \texttt{sum}(n, m) = \textbf{if } n = m \textbf{ then } n$$
$$\textbf{else } n + \texttt{sum}(n+1, m);$$

$\underline{\textbf{assert}}$ $\forall n, m \in \mathbb{Z}.(n \leq m \rightarrow 2 \cdot \texttt{sum}(n, m) = m \cdot (m+1) - n \cdot (n-1))$

Example

$$\textbf{fun}\ \texttt{sum}(n,m) = \textbf{if}\ n{=}m\ \textbf{then}\ n$$
$$\textbf{else}\ n + \texttt{sum}(n+1,m);$$

$\underline{\textbf{assert}}$ $\forall n,m \in \mathbb{Z}.(n \leq m \to 2 \cdot \texttt{sum}(n,m) = m \cdot (m+1) - n \cdot (n-1))$

Translated into FOL with theories:

$$\forall n \in \mathbb{Z}.\texttt{sum}(n,n) = n \ \wedge \ \forall n,m \in \mathbb{Z}.(n \neq m \to \texttt{sum}(n,m) = n + \texttt{sum}(n+1,m))$$
$$\to \ \forall n,m \in \mathbb{Z}.(n \leq m \to 2 \cdot \texttt{sum}(n,m) = m \cdot (m+1) - n \cdot (n-1))$$

Example

$$\textbf{fun}\ \mathtt{sum}(n, m) = \textbf{if}\ n = m\ \textbf{then}\ n$$
$$\textbf{else}\ n + \mathtt{sum}(n+1, m);$$

$$\underline{\textbf{assert}}\ \ \forall n, m \in \mathbb{Z}.(n \leq m \to 2 \cdot \mathtt{sum}(n, m) = m \cdot (m+1) - n \cdot (n-1))$$

Translated into FOL with theories:

$$\forall n \in \mathbb{Z}.\mathtt{sum}(n, n) = n \ \wedge \ \forall n, m \in \mathbb{Z}.(n \neq m \to \mathtt{sum}(n, m) = n + \mathtt{sum}(n+1, m))$$
$$\to \ \forall n, m \in \mathbb{Z}.(n \leq m \to 2 \cdot \mathtt{sum}(n, m) = m \cdot (m+1) - n \cdot (n-1))$$

Negated conjecture for proof by refutation:

$$\sigma_n \leq \sigma_m \wedge 2 \cdot \mathtt{sum}(\sigma_n, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_n \cdot (\sigma_n - 1)$$

Integer Induction Axioms

New induction axioms for integers with symbolic bounds.
[Hozzová, Kovács, Voronkov CADE'21]

For finite intervals:

$$L[b_1] \wedge \forall y.(b_1 \leq y < b_2 \wedge L[y] \rightarrow L[y+1]) \rightarrow \forall x.(b_1 \leq x \leq b_2 \rightarrow L[x]) \qquad \textit{(upward)}$$

$$L[b_2] \wedge \forall y.(b_1 < y \leq b_2 \wedge L[y] \rightarrow L[y-1]) \rightarrow \forall x.(b_1 \leq x \leq b_2 \rightarrow L[x]) \qquad \textit{(downward)}$$

For infinite intervals:

$$L[b] \wedge \forall y.(y \geq b \wedge L[y] \rightarrow L[y+1]) \rightarrow \forall x.(x \geq b \rightarrow L[x]) \qquad \textit{(upward)}$$

$$L[b] \wedge \forall y.(y \leq b \wedge L[y] \rightarrow L[y-1]) \rightarrow \forall x.(x \leq b \rightarrow L[x]) \qquad \textit{(downward)}$$

Integer Induction Axioms

New induction axioms for integers with symbolic bounds.
[Hozzová, Kovács, Voronkov CADE'21]

For finite intervals:

$$L[b_1] \land \forall y.(b_1 \leq y < b_2 \land L[y] \rightarrow L[y+1]) \rightarrow \forall x.(b_1 \leq x \leq b_2 \rightarrow L[x]) \qquad \textit{(upward)}$$

$$L[b_2] \land \forall y.(b_1 < y \leq b_2 \land L[y] \rightarrow L[y-1]) \rightarrow \forall x.(b_1 \leq x \leq b_2 \rightarrow L[x]) \qquad \textit{(downward)}$$

For infinite intervals:

$$L[b] \land \forall y.(y \geq b \land L[y] \rightarrow L[y+1]) \rightarrow \forall x.(x \geq b \rightarrow L[x]) \qquad \textit{(upward)}$$

$$L[b] \land \forall y.(y \leq b \land L[y] \rightarrow L[y-1]) \rightarrow \forall x.(x \leq b \rightarrow L[x]) \qquad \textit{(downward)}$$

Integer Induction Axioms

New induction axioms for integers with symbolic bounds.
[Hozzová, Kovács, Voronkov CADE'21]

For finite intervals:

$$L[b_1] \land \forall y.(b_1 \leq y < b_2 \land L[y] \rightarrow L[y+1]) \rightarrow \forall x.(b_1 \leq x \leq b_2 \rightarrow L[x]) \qquad \textit{(upward)}$$

$$L[b_2] \land \forall y.(b_1 < y \leq b_2 \land L[y] \rightarrow L[y-1]) \rightarrow \forall x.(b_1 \leq x \leq b_2 \rightarrow L[x]) \qquad \textit{(downward)}$$

For infinite intervals:

$$L[b] \land \forall y.(y \geq b \land L[y] \rightarrow L[y+1]) \rightarrow \forall x.(x \geq b \rightarrow L[x]) \qquad \textit{(upward)}$$

$$L[b] \land \forall y.(y \leq b \land L[y] \rightarrow L[y-1]) \rightarrow \forall x.(x \leq b \rightarrow L[x]) \qquad \textit{(downward)}$$

Symbolic means that b is not necessarily a concrete integer, it can be any term.

Integer Induction Rule

We introduce new variants of the induction rule, e.g.:

$$\frac{\overline{L}[t] \vee C}{L[b] \wedge \forall y.(y \leq b \wedge L[y] \rightarrow L[y-1]) \rightarrow \forall x.(x \leq b \rightarrow L[x])} \;(\mathsf{IntInd}_{\leq})$$

Integer Induction Rule

We introduce new variants of the induction rule, e.g.:

$$\frac{\overline{L}[t] \vee C}{L[b] \wedge \forall y.(y \leq b \wedge L[y] \rightarrow L[y-1]) \rightarrow \forall x.(x \leq b \rightarrow L[x])}\ (\mathsf{IntInd}_{\leq})$$

1. *How to find suitable b to instantiate the axiom?*

Integer Induction Rule

We introduce new variants of the induction rule, e.g.:

$$\frac{\overline{L}[t] \vee C \quad t \leq b}{L[b] \wedge \forall y.(y \leq b \wedge L[y] \rightarrow L[y-1]) \rightarrow \forall x.(x \leq b \rightarrow L[x])} \ (\mathsf{IntInd}_{\leq})$$

1. *How to find suitable b to instantiate the axiom?*

   We use as bounds terms already occurring in the search space.

Integer Induction Rule

We introduce new variants of the induction rule, e.g.:

$$\frac{\overline{L}[t] \vee C \quad t \leq b}{L[b] \wedge \forall y.(y \leq b \wedge L[y] \rightarrow L[y-1]) \rightarrow \forall x.(x \leq b \rightarrow L[x])} \ (\mathsf{IntInd}_{\leq})$$

1. *How to find suitable b to instantiate the axiom?*

   We use as bounds terms already occurring in the search space.

   Recall our example negated and skolemized conjecture:
   $\sigma_n \leq \sigma_m \wedge 2 \cdot \mathtt{sum}(\sigma_n, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_n \cdot (\sigma_n - 1)$

Integer Induction Rule

We introduce new variants of the induction rule, e.g.:

$$\frac{\overline{L}[t] \lor C \quad t \leq b}{L[b] \land \forall y.(y \leq b \land L[y] \rightarrow L[y-1]) \rightarrow \forall x.(x \leq b \rightarrow L[x])} \ (\mathrm{IntInd}_{\leq})$$

1. *How to find suitable b to instantiate the axiom?*

   We use as bounds terms already occurring in the search space.

   Recall our example negated and skolemized conjecture:
   $\sigma_n \leq \sigma_m \land 2 \cdot \mathtt{sum}(\sigma_n, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_n \cdot (\sigma_n - 1)$

2. *How to use IntInd rules within saturation?*

   Resolve the conclusion against all the premises.
   Also restrict which literals $L[t]$ are eligible for induction.

Solving the Example

After resolving the axiom with the premises, we are left with the following three clauses:

$$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee \sigma \leq \sigma_m$$
$$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee 2 \cdot \mathtt{sum}(\sigma, \sigma_m) = \sigma_m \cdot (\sigma_m + 1) - \sigma \cdot (\sigma - 1)$$
$$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee 2 \cdot \mathtt{sum}(\sigma - 1, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - (\sigma - 1) \cdot ((\sigma - 1) - 1)$$

Solving the Example

After resolving the axiom with the premises, we are left with the following three clauses:

$$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee \sigma \leq \sigma_m$$
$$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee 2 \cdot \mathtt{sum}(\sigma, \sigma_m) = \sigma_m \cdot (\sigma_m + 1) - \sigma \cdot (\sigma - 1)$$
$$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee 2 \cdot \mathtt{sum}(\sigma - 1, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - (\sigma - 1) \cdot ((\sigma - 1) - 1)$$

We resolve the base case using the definition $\forall n \in \mathbb{Z}.\mathtt{sum}(n, n) = n$.

Solving the Example

After resolving the axiom with the premises, we are left with the following three clauses:

$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee \sigma \leq \sigma_m$
$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee 2 \cdot \mathtt{sum}(\sigma, \sigma_m) = \sigma_m \cdot (\sigma_m + 1) - \sigma \cdot (\sigma - 1)$
$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee 2 \cdot \mathtt{sum}(\sigma - 1, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - (\sigma - 1) \cdot ((\sigma - 1) - 1)$

We resolve the base case using the definition $\forall n \in \mathbb{Z}.\mathtt{sum}(n, n) = n$.

For the step case we use that $\sigma \leq \sigma_m \rightarrow \sigma - 1 \neq \sigma_m$, and then the second part of the definition:

$$\forall n, m \in \mathbb{Z}.(n \neq m \rightarrow \mathtt{sum}(n, m) = n + \mathtt{sum}(n + 1, m))$$

...and combine it with arithmetic reasoning.

Solving the Example

After resolving the axiom with the premises, we are left with the following three clauses:

$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee \sigma \leq \sigma_m$
$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee 2 \cdot \mathtt{sum}(\sigma, \sigma_m) = \sigma_m \cdot (\sigma_m + 1) - \sigma \cdot (\sigma - 1)$
$2 \cdot \mathtt{sum}(\sigma_m, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - \sigma_m \cdot (\sigma_m - 1) \vee 2 \cdot \mathtt{sum}(\sigma - 1, \sigma_m) \neq \sigma_m \cdot (\sigma_m + 1) - (\sigma - 1) \cdot ((\sigma - 1) - 1)$

We resolve the base case using the definition $\forall n \in \mathbb{Z}.\mathtt{sum}(n, n) = n$.

For the step case we use that $\sigma \leq \sigma_m \rightarrow \sigma - 1 \neq \sigma_m$, and then the second part of the definition:

$$\forall n, m \in \mathbb{Z}.(n \neq m \rightarrow \mathtt{sum}(n, m) = n + \mathtt{sum}(n + 1, m))$$

. . . and combine it with arithmetic reasoning.

Time for a demo!

Outline

Induction on Complex Formulas

Induction on literals is not always sufficient.

However, we can plug in arbitrary formula into induction axioms:

$$L[0] \wedge \forall y \in \mathbb{N}.(L[y] \rightarrow L[\mathrm{s}(y)]) \;\rightarrow\; \forall x \in \mathbb{N}.L[x]$$

$$\forall x.(\overline{L}[x] \rightarrow \exists y.(x \succ y \wedge \overline{L}[y])) \;\rightarrow\; \forall x.L[x]$$

Induction on Complex Formulas

Induction on literals is not always sufficient.

However, we can plug in arbitrary formula into induction axioms:

$$F[0] \wedge \forall y \in \mathbb{N}.(F[y] \rightarrow F[\mathrm{s}(y)]) \;\rightarrow\; \forall x \in \mathbb{N}.F[x]$$

$$\forall x.(\overline{F}[x] \rightarrow \exists y.(x \succ y \wedge \overline{F}[y])) \;\rightarrow\; \forall x.F[x]$$

Induction with an ∃-Quantified Variable

Consider the following conjecture: $\forall x \in \mathbb{N}. \exists y \in \mathbb{N}.\ \mathtt{half}(y) = x$

To prove it, we need to instantiate the standard structural induction axiom with an ∃-quantified formula – i.e., we use $F[z] := \exists x. L[z, x]$:

$$\exists v_0. L[0, v_0] \ \wedge\ \forall y. (\exists w. L[y, w] \rightarrow \exists v_{\mathrm{s}}. L[\mathrm{s}(y), v_{\mathrm{s}}]) \ \rightarrow\ \forall z. \exists x. L[z, x]$$

Induction with an $\exists$-Quantified Variable

Consider the following conjecture: $\forall x \in \mathbb{N}.\exists y \in \mathbb{N}.\ \mathtt{half}(y) = x$

To prove it, we need to instantiate the standard structural induction axiom with an $\exists$-quantified formula – i.e., we use $F[z] := \exists x. L[z, x]$:

$$\exists v_0. L[0, v_0] \ \wedge\ \forall y. \big(\exists w. L[y, w] \rightarrow \exists v_{\mathrm{s}}. L[\mathrm{s}(y), v_{\mathrm{s}}]\big) \ \rightarrow\ \forall z. \exists x. L[z, x]$$

We use it in the induction rule:

$$\frac{\overline{L}[t, x] \vee C}{\exists v_0. L[0, v_0] \ \wedge\ \forall y. \big(\exists w. L[y, w] \rightarrow \exists v_{\mathrm{s}}. L[\mathrm{s}(y), v_{\mathrm{s}}]\big) \ \rightarrow\ \forall z. \exists x. L[z, x]}$$

Induction with an $\exists$-Quantified Variable

Consider the following conjecture: $\forall x \in \mathbb{N}. \exists y \in \mathbb{N}.\ \mathtt{half}(y) = x$

To prove it, we need to instantiate the standard structural induction axiom with an $\exists$-quantified formula – i.e., we use $F[z] := \exists x. L[z, x]$:

$$\exists v_0. L[0, v_0] \ \wedge\ \forall y. \big(\exists w. L[y, w] \rightarrow \exists v_s. L[\mathrm{s}(y), v_s]\big) \ \rightarrow\ \forall z. \exists x. L[z, x]$$

We use it in the induction rule:

$$\frac{\overline{L}[t, x] \vee C}{\exists v_0. L[0, v_0] \ \wedge\ \forall y. \big(\exists w. L[y, w] \rightarrow \exists v_s. L[\mathrm{s}(y), v_s]\big) \ \rightarrow\ \forall z. \exists x. L[z, x]}$$

Time for a demo!

Solving the Example

Conjecture: $\forall x. \exists y.\ \mathtt{half}(y) = x$

Axioms: $\mathtt{half}(0) = 0, \quad \mathtt{half}(\mathtt{s}(0)) = 0, \quad \forall x.\ \mathtt{half}(\mathtt{s}(\mathtt{s}(x))) = \mathtt{s}(\mathtt{half}(x))$

Solving the Example

Conjecture: $\forall x. \exists y.\ \mathtt{half}(y) = x$

Axioms: $\mathtt{half}(0) = 0,\quad \mathtt{half}(\mathtt{s}(0)) = 0,\quad \forall x.\ \mathtt{half}(\mathtt{s}(\mathtt{s}(x))) = \mathtt{s}(\mathtt{half}(x))$

1.	$\mathtt{half}(y) \neq \sigma$	[preprocessed specification]
2.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y \vee \mathtt{half}(\sigma_x(z)) = z$	[Ind 1]
3.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\sigma_y) \vee \mathtt{half}(\sigma_x(z)) = z$	[Ind 1]
4.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y$	[BR 1, 2]
5.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\sigma_y)$	[BR 1, 3]
6.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\mathtt{half}(\sigma_w))$	[Sup 4, 5]
7.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{half}(\mathtt{s}(\mathtt{s}(\sigma_w)))$	[Sup A3, 6]
8.	$\mathtt{half}(v_0) \neq 0$	[ER 7]
9.	$\square$	[BR 8, A2]

Solving the Example

Conjecture: $\forall x. \exists y.\ \mathtt{half}(y) = x$

Axioms: $\mathtt{half}(0) = 0, \quad \mathtt{half}(\mathtt{s}(0)) = 0, \quad \forall x.\ \mathtt{half}(\mathtt{s}(\mathtt{s}(x))) = \mathtt{s}(\mathtt{half}(x))$

1.	$\mathtt{half}(y) \neq \sigma$	[preprocessed specification]
2.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y \vee \mathtt{half}(\sigma_x(z)) = z$	[Ind 1]
3.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_s) \neq \mathtt{s}(\sigma_y) \vee \mathtt{half}(\sigma_x(z)) = z$	[Ind 1]
4.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y$	[BR 1, 2]
5.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_s) \neq \mathtt{s}(\sigma_y)$	[BR 1, 3]
6.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_s) \neq \mathtt{s}(\mathtt{half}(\sigma_w))$	[Sup 4, 5]
7.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_s) \neq \mathtt{half}(\mathtt{s}(\mathtt{s}(\sigma_w)))$	[Sup A3, 6]
8.	$\mathtt{half}(v_0) \neq 0$	[ER 7]
9.	$\square$	[BR 8, A2]

Solving the Example

Conjecture: $\forall x.\exists y.\ \mathtt{half}(y) = x$

Axioms: $\mathtt{half}(0) = 0, \quad \mathtt{half}(\mathtt{s}(0)) = 0, \quad \forall x.\ \mathtt{half}(\mathtt{s}(\mathtt{s}(x))) = \mathtt{s}(\mathtt{half}(x))$

1.	$\underline{\mathtt{half}(y) \neq \sigma}$	[preprocessed specification]
2.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y \vee \mathtt{half}(\sigma_x(z)) = z$	[Ind 1]
3.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_s) \neq \mathtt{s}(\sigma_y) \vee \mathtt{half}(\sigma_x(z)) = z$	[Ind 1]
4.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y$	[BR 1, 2]
5.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_s) \neq \mathtt{s}(\sigma_y)$	[BR 1, 3]
6.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_s) \neq \mathtt{s}(\mathtt{half}(\sigma_w))$	[Sup 4, 5]
7.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_s) \neq \mathtt{half}(\mathtt{s}(\mathtt{s}(\sigma_w)))$	[Sup A3, 6]
8.	$\mathtt{half}(v_0) \neq 0$	[ER 7]
9.	$\square$	[BR 8, A2]

Solving the Example

Conjecture: $\forall x. \exists y.\ \mathtt{half}(y) = x$

Axioms: $\mathtt{half}(0) = 0, \quad \mathtt{half}(\mathtt{s}(0)) = 0, \quad \forall x.\ \mathtt{half}(\mathtt{s}(\mathtt{s}(x))) = \mathtt{s}(\mathtt{half}(x))$

1. $\underline{\mathtt{half}(y) \neq \sigma}$ [preprocessed specification]
2. $\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y \vee \underline{\mathtt{half}(\sigma_x(z)) = z}$ [Ind 1]
3. $\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\sigma_y) \vee \mathtt{half}(\sigma_x(z)) = z$ [Ind 1]
4. $\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y$ [BR 1, 2]
5. $\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\sigma_y)$ [BR 1, 3]
6. $\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\mathtt{half}(\sigma_w))$ [Sup 4, 5]
7. $\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{half}(\mathtt{s}(\mathtt{s}(\sigma_w)))$ [Sup A3, 6]
8. $\mathtt{half}(v_0) \neq 0$ [ER 7]
9. $\square$ [BR 8, A2]

Solving the Example

Conjecture: $\forall x. \exists y.\ \mathtt{half}(y) = x$

Axioms: $\mathtt{half}(0) = 0, \quad \mathtt{half}(\mathtt{s}(0)) = 0, \quad \forall x.\ \mathtt{half}(\mathtt{s}(\mathtt{s}(x))) = \mathtt{s}(\mathtt{half}(x))$

1.	$\underline{\mathtt{half}(y) \neq \sigma}$	[preprocessed specification]
2.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y \vee \mathtt{half}(\sigma_x(z)) = z$	[Ind 1]
3.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_\mathtt{s}) \neq \mathtt{s}(\sigma_y) \vee \underline{\mathtt{half}(\sigma_x(z)) = z}$	[Ind 1]
4.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y$	[BR 1, 2]
5.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_\mathtt{s}) \neq \mathtt{s}(\sigma_y)$	[BR 1, 3]
6.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_\mathtt{s}) \neq \mathtt{s}(\mathtt{half}(\sigma_w))$	[Sup 4, 5]
7.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_\mathtt{s}) \neq \mathtt{half}(\mathtt{s}(\mathtt{s}(\sigma_w)))$	[Sup A3, 6]
8.	$\mathtt{half}(v_0) \neq 0$	[ER 7]
9.	$\square$	[BR 8, A2]

Solving the Example

Conjecture: $\forall x. \exists y.\ \mathtt{half}(y) = x$

Axioms: $\mathtt{half}(0) = 0, \quad \mathtt{half}(\mathtt{s}(0)) = 0, \quad \forall x.\ \mathtt{half}(\mathtt{s}(\mathtt{s}(x))) = \mathtt{s}(\mathtt{half}(x))$

1. $\mathtt{half}(y) \neq \sigma$ [preprocessed specification]
2. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(\sigma_w) = \sigma_y \lor \mathtt{half}(\sigma_x(z)) = z$ [Ind 1]
3. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(v_s) \neq \mathtt{s}(\sigma_y) \lor \mathtt{half}(\sigma_x(z)) = z$ [Ind 1]
4. $\mathtt{half}(v_0) \neq 0 \lor \underline{\mathtt{half}(\sigma_w) = \sigma_y}$ [BR 1, 2]
5. $\mathtt{half}(v_0) \neq 0 \lor \underline{\mathtt{half}(v_s) \neq \mathtt{s}(\sigma_y)}$ [BR 1, 3]
6. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(v_s) \neq \mathtt{s}(\mathtt{half}(\sigma_w))$ [Sup 4, 5]
7. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(v_s) \neq \mathtt{half}(\mathtt{s}(\mathtt{s}(\sigma_w)))$ [Sup A3, 6]
8. $\mathtt{half}(v_0) \neq 0$ [ER 7]
9. $\square$ [BR 8, A2]

Solving the Example

Conjecture: $\forall x. \exists y.\ \mathtt{half}(y) = x$

Axioms: $\mathtt{half}(0) = 0, \quad \mathtt{half}(\mathtt{s}(0)) = 0, \quad \forall x.\ \mathtt{half}(\mathtt{s}(\mathtt{s}(x))) = \mathtt{s}(\mathtt{half}(x))$

1. $\mathtt{half}(y) \neq \sigma$ [preprocessed specification]
2. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(\sigma_w) = \sigma_y \lor \mathtt{half}(\sigma_x(z)) = z$ [Ind 1]
3. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(v_\mathtt{s}) \neq \mathtt{s}(\sigma_y) \lor \mathtt{half}(\sigma_x(z)) = z$ [Ind 1]
4. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(\sigma_w) = \sigma_y$ [BR 1, 2]
5. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(v_\mathtt{s}) \neq \mathtt{s}(\sigma_y)$ [BR 1, 3]
6. $\mathtt{half}(v_0) \neq 0 \lor \underline{\mathtt{half}(v_\mathtt{s}) \neq \mathtt{s}(\mathtt{half}(\sigma_w))}$ [Sup 4, 5]
7. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(v_\mathtt{s}) \neq \mathtt{half}(\mathtt{s}(\mathtt{s}(\sigma_w)))$ [Sup A3, 6]
8. $\mathtt{half}(v_0) \neq 0$ [ER 7]
9. $\square$ [BR 8, A2]

Solving the Example

Conjecture: $\forall x. \exists y.\ \mathtt{half}(y) = x$

Axioms: $\mathtt{half}(0) = 0, \quad \mathtt{half}(\mathtt{s}(0)) = 0, \quad \forall x.\ \mathtt{half}(\mathtt{s}(\mathtt{s}(x))) = \mathtt{s}(\mathtt{half}(x))$

1. $\mathtt{half}(y) \neq \sigma$ [preprocessed specification]
2. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(\sigma_w) = \sigma_y \lor \mathtt{half}(\sigma_x(z)) = z$ [Ind 1]
3. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\sigma_y) \lor \mathtt{half}(\sigma_x(z)) = z$ [Ind 1]
4. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(\sigma_w) = \sigma_y$ [BR 1, 2]
5. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\sigma_y)$ [BR 1, 3]
6. $\mathtt{half}(v_0) \neq 0 \lor \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\mathtt{half}(\sigma_w))$ [Sup 4, 5]
7. $\mathtt{half}(v_0) \neq 0 \lor \underline{\mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{half}(\mathtt{s}(\mathtt{s}(\sigma_w)))}$ [Sup A3, 6]
8. $\mathtt{half}(v_0) \neq 0$ [ER 7]
9. $\square$ [BR 8, A2]

Solving the Example

Conjecture: $\forall x. \exists y.\ \mathtt{half}(y) = x$

Axioms: $\mathtt{half}(0) = 0$, $\quad \mathtt{half}(\mathtt{s}(0)) = 0$, $\quad \forall x.\ \mathtt{half}(\mathtt{s}(\mathtt{s}(x))) = \mathtt{s}(\mathtt{half}(x))$

1.	$\mathtt{half}(y) \neq \sigma$	[preprocessed specification]
2.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y \vee \mathtt{half}(\sigma_x(z)) = z$	[Ind 1]
3.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\sigma_y) \vee \mathtt{half}(\sigma_x(z)) = z$	[Ind 1]
4.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(\sigma_w) = \sigma_y$	[BR 1, 2]
5.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\sigma_y)$	[BR 1, 3]
6.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{s}(\mathtt{half}(\sigma_w))$	[Sup 4, 5]
7.	$\mathtt{half}(v_0) \neq 0 \vee \mathtt{half}(v_{\mathtt{s}}) \neq \mathtt{half}(\mathtt{s}(\mathtt{s}(\sigma_w)))$	[Sup A3, 6]
8.	$\underline{\mathtt{half}(v_0) \neq 0}$	[ER 7]
9.	$\square$	[BR 8, A2]

How to...

1. *... construct the induction scheme?*
   Based on the formulas occurring in the search space. If they have a common induction term, we can use them in the same induction.

[Hajdu, Kovács, Rawson, Voronkov PAAR'22]
[Hozzová, Amrollahi, Hajdu, Kovács, Vorkonkov, Wagner IJCAR'24]

How to...

1. *…construct the induction scheme?*
   Based on the formulas occurring in the search space. If they have a common induction term, we can use them in the same induction.
2. *…use these schemes in saturation?*
   We need to "resolve" the CNF of the axiom with all the premises (using suitable substitutions, since the premises might not be ground anymore).

[Hajdu, Kovács, Rawson, Voronkov PAAR'22]
[Hozzová, Amrollahi, Hajdu, Kovács, Vorkonkov, Wagner IJCAR'24]

Outline

Example

We sometimes need induction schemes different from the structural induction axiom.

Consider the following conjecture:

$$\forall x, y.(\texttt{even}(x) \wedge \texttt{even}(y) \rightarrow \texttt{even}(x + y))$$

Example

We sometimes need induction schemes different from the structural induction axiom.

Consider the following conjecture:

$$\forall x, y.(\mathtt{even}(x) \land \mathtt{even}(y) \rightarrow \mathtt{even}(x + y))$$

Function $+$ and predicate $\mathtt{even}$ are defined as:

$$\forall y \in \mathbb{N}.\ 0 + y = y$$
$$\forall x, y \in \mathbb{N}.\ \mathtt{s}(x) + y = \mathtt{s}(x + y)$$
$$\mathtt{even}(0)$$
$$\neg\mathtt{even}(\mathtt{s}(0))$$
$$\forall x \in \mathbb{N}.(\mathtt{even}(\mathtt{s}(\mathtt{s}(x))) \leftrightarrow \mathtt{even}(x))$$

Example

We sometimes need induction schemes different from the structural induction axiom.

Consider the following conjecture:

$$\forall x, y.(\mathtt{even}(x) \land \mathtt{even}(y) \to \mathtt{even}(x + y))$$

Function $+$ and predicate $\mathtt{even}$ are defined as:

$$\forall y \in \mathbb{N}.\ 0 + y = y$$
$$\forall x, y \in \mathbb{N}.\ \mathtt{s}(x) + y = \mathtt{s}(x + y)$$
$$\mathtt{even}(0)$$
$$\neg\mathtt{even}(\mathtt{s}(0))$$
$$\forall x \in \mathbb{N}.(\mathtt{even}(\mathtt{s}(\mathtt{s}(x))) \leftrightarrow \mathtt{even}(x))$$

How to prove the conjecture by induction?

Solving the Example (1)

We use the following induction axiom:

$$F[0] \land F[\mathrm{s}(0)] \land \forall y \in \mathbb{N}.(F[y] \to F[\mathrm{s}(\mathrm{s}(y))]) \;\to\; \forall x \in \mathbb{N}.F[x]$$

Solving the Example (1)

We use the following induction axiom:

$$F[0] \wedge F[\mathtt{s}(0)] \wedge \forall y \in \mathbb{N}.(F[y] \rightarrow F[\mathtt{s}(\mathtt{s}(y))]) \ \rightarrow \ \forall x \in \mathbb{N}.F[x]$$

Negated and clausified conjecture:

$$\mathtt{even}(\sigma_x)$$
$$\mathtt{even}(\sigma_y)$$
$$\neg\mathtt{even}(\sigma_x + \sigma_y)$$

We apply induction with $F[x] := \mathtt{even}(x) \rightarrow \mathtt{even}(x + \sigma_y)$.

Solving the Example (1)

We use the following induction axiom:

$$F[0] \land F[\mathrm{s}(0)] \land \forall y \in \mathbb{N}.(F[y] \rightarrow F[\mathrm{s}(\mathrm{s}(y))]) \;\rightarrow\; \forall x \in \mathbb{N}.F[x]$$

Negated and clausified conjecture:

$$\mathtt{even}(\sigma_x)$$
$$\mathtt{even}(\sigma_y)$$
$$\neg\mathtt{even}(\sigma_x + \sigma_y)$$

We apply induction with $F[x] := \mathtt{even}(x) \rightarrow \mathtt{even}(x + \sigma_y)$.

When we instantiate the axiom, clausify it and resolve with $\mathtt{even}(\sigma_y)$ and $\neg\mathtt{even}(\sigma_x + \sigma_y)$, we are left with refuting the negated base and step cases.

Solving the Example (2)

Negated base cases we need to refute:

1. $\mathtt{even}(0) \wedge \neg\mathtt{even}(0 + \sigma_y)$

Solving the Example (2)

Negated base cases we need to refute:

1. $\mathtt{even}(0) \wedge \neg\mathtt{even}(0 + \sigma_y)$
   Using an axiom of $+$, the second literal is rewritten to $\neg\mathtt{even}(\sigma_y)$, which contradicts the input $\mathtt{even}(\sigma_y)$.

Solving the Example (2)

Negated base cases we need to refute:

1. $\mathtt{even}(0) \wedge \neg\mathtt{even}(0 + \sigma_y)$
   Using an axiom of $+$, the second literal is rewritten to $\neg\mathtt{even}(\sigma_y)$, which contradicts the input $\mathtt{even}(\sigma_y)$.
2. $\mathtt{even}(\mathtt{s}(0)) \wedge \neg\mathtt{even}(\mathtt{s}(\mathit{zero}) + \sigma_y)$

Solving the Example (2)

Negated base cases we need to refute:

1. $\texttt{even}(0) \wedge \neg\texttt{even}(0 + \sigma_y)$
   Using an axiom of $+$, the second literal is rewritten to $\neg\texttt{even}(\sigma_y)$, which contradicts the input $\texttt{even}(\sigma_y)$.
2. $\texttt{even}(\texttt{s}(0)) \wedge \neg\texttt{even}(\texttt{s}(\mathit{zero}) + \sigma_y)$
   The first literal contradicts an axiom of $\texttt{even}$.

Solving the Example (3)

Next we refute the negation of the step case:

$$\neg\forall x.((\mathtt{even}(x) \to \mathtt{even}(x + \sigma_y)) \to (\mathtt{even}(\mathtt{s}(\mathtt{s}(x))) \to \mathtt{even}(\mathtt{s}(\mathtt{s}(x)) + \sigma_y))$$

The clausal form:

$$\neg\mathtt{even}(\sigma) \vee \mathtt{even}(\sigma + \sigma_y) \tag{1}$$

$$\mathtt{even}(\mathtt{s}(\mathtt{s}(\sigma))) \tag{2}$$

$$\neg\mathtt{even}(\mathtt{s}(\mathtt{s}(\sigma)) + \sigma_y)) \tag{3}$$

Solving the Example (3)

Next we refute the negation of the step case:

$$\neg\forall x.((\mathtt{even}(x) \rightarrow \mathtt{even}(x + \sigma_y)) \rightarrow (\mathtt{even}(\mathtt{s}(\mathtt{s}(x))) \rightarrow \mathtt{even}(\mathtt{s}(\mathtt{s}(x)) + \sigma_y))$$

The clausal form:

$$\neg\mathtt{even}(\sigma) \vee \mathtt{even}(\sigma + \sigma_y) \tag{1}$$

$$\mathtt{even}(\mathtt{s}(\mathtt{s}(\sigma))) \tag{2}$$

$$\neg\mathtt{even}(\mathtt{s}(\mathtt{s}(\sigma)) + \sigma_y)) \tag{3}$$

From the definition of $\mathtt{even}$ and (2) we obtain $\mathtt{even}(\sigma)$.

Solving the Example (3)

Next we refute the negation of the step case:

$$\neg\forall x.((\mathtt{even}(x) \rightarrow \mathtt{even}(x + \sigma_y)) \rightarrow (\mathtt{even}(\mathtt{s}(\mathtt{s}(x))) \rightarrow \mathtt{even}(\mathtt{s}(\mathtt{s}(x)) + \sigma_y))$$

The clausal form:

$$\neg\mathtt{even}(\sigma) \vee \mathtt{even}(\sigma + \sigma_y) \tag{1}$$

$$\mathtt{even}(\mathtt{s}(\mathtt{s}(\sigma))) \tag{2}$$

$$\neg\mathtt{even}(\mathtt{s}(\mathtt{s}(\sigma)) + \sigma_y)) \tag{3}$$

From the definition of $\mathtt{even}$ and (2) we obtain $\mathtt{even}(\sigma)$.
We resolve that with (1) to obtain $\mathtt{even}(\sigma + \sigma_y)$.

Solving the Example (3)

Next we refute the negation of the step case:

$$\neg\forall x.((\mathtt{even}(x) \to \mathtt{even}(x + \sigma_y)) \to (\mathtt{even}(\mathtt{s}(\mathtt{s}(x))) \to \mathtt{even}(\mathtt{s}(\mathtt{s}(x)) + \sigma_y)))$$

The clausal form:

$$\neg\mathtt{even}(\sigma) \vee \mathtt{even}(\sigma + \sigma_y) \tag{1}$$

$$\mathtt{even}(\mathtt{s}(\mathtt{s}(\sigma))) \tag{2}$$

$$\neg\mathtt{even}(\mathtt{s}(\mathtt{s}(\sigma)) + \sigma_y)) \tag{3}$$

From the definition of $\mathtt{even}$ and (2) we obtain $\mathtt{even}(\sigma)$.
We resolve that with (1) to obtain $\mathtt{even}(\sigma + \sigma_y)$.
That together with the definition of $+$ and $\mathtt{even}$ suffices to contradict (3).

Solving the Example (3)

Next we refute the negation of the step case:

$$\neg\forall x.((\mathtt{even}(x) \to \mathtt{even}(x + \sigma_y)) \to (\mathtt{even}(\mathtt{s}(\mathtt{s}(x))) \to \mathtt{even}(\mathtt{s}(\mathtt{s}(x)) + \sigma_y))$$

The clausal form:

$$\neg\mathtt{even}(\sigma) \vee \mathtt{even}(\sigma + \sigma_y) \tag{1}$$
$$\mathtt{even}(\mathtt{s}(\mathtt{s}(\sigma))) \tag{2}$$
$$\neg\mathtt{even}(\mathtt{s}(\mathtt{s}(\sigma)) + \sigma_y)) \tag{3}$$

From the definition of $\mathtt{even}$ and (2) we obtain $\mathtt{even}(\sigma)$.
We resolve that with (1) to obtain $\mathtt{even}(\sigma + \sigma_y)$.
That together with the definition of $+$ and $\mathtt{even}$ suffices to contradict (3).

Time for a demo!

Schemes from Recursive Function Definitions

[Hajdu, Hozzová, Kovács, Voronkov FMCAD'21]

1. *How to construct the induction scheme?*
   When $L[t]$ contains a recursive function, we construct the scheme based on the function definition. Each branch in the function definition corresponds to one case in the scheme.

Schemes from Recursive Function Definitions

[Hajdu, Hozzová, Kovács, Voronkov FMCAD'21]

1. *How to construct the induction scheme?*
   When $L[t]$ contains a recursive function, we construct the scheme based on the function definition. Each branch in the function definition corresponds to one case in the scheme.

   If the function definition branch does not have recursive calls, we convert it into a base case:

$$\begin{aligned} \mathtt{even}(\mathtt{0}) \quad &\rightsquigarrow \quad F[\mathtt{0}] \\ \mathtt{even}(\mathtt{s}(\mathtt{0})) \quad &\rightsquigarrow \quad F[\mathtt{s}(\mathtt{0})] \end{aligned}$$

Schemes from Recursive Function Definitions

[Hajdu, Hozzová, Kovács, Voronkov FMCAD'21]

1. *How to construct the induction scheme?*

   When $L[t]$ contains a recursive function, we construct the scheme based on the function definition. Each branch in the function definition corresponds to one case in the scheme.

   If the function definition branch does not have recursive calls, we convert it into a base case:

$$\begin{aligned} \mathtt{even}(\mathtt{0}) \quad &\rightsquigarrow \quad F[\mathtt{0}] \\ \mathtt{even}(\mathtt{s}(\mathtt{0})) \quad &\rightsquigarrow \quad F[\mathtt{s}(\mathtt{0})] \end{aligned}$$

   If the function definition branch contains recursive calls, we use those in the induction hypothesis:

$$\mathtt{even}(\mathtt{s}(\mathtt{s}(x))) \leftrightarrow \mathtt{even}(x) \quad \rightsquigarrow \quad \forall x.(F[x] \rightarrow F[\mathtt{s}(\mathtt{s}(x))])$$

Schemes from Recursive Function Definitions

[Hajdu, Hozzová, Kovács, Voronkov FMCAD'21]

1. *How to construct the induction scheme?*
   When $L[t]$ contains a recursive function, we construct the scheme based on the function definition. Each branch in the function definition corresponds to one case in the scheme.

   If the function definition branch does not have recursive calls, we convert it into a base case:

$$\begin{array}{rcl} \mathtt{even}(0) & \rightsquigarrow & F[0] \\ \mathtt{even}(\mathtt{s}(0)) & \rightsquigarrow & F[\mathtt{s}(0)] \end{array}$$

   If the function definition branch contains recursive calls, we use those in the induction hypothesis:

$$\mathtt{even}(\mathtt{s}(\mathtt{s}(x))) \leftrightarrow \mathtt{even}(x) \quad \rightsquigarrow \quad \forall x.(F[x] \rightarrow F[\mathtt{s}(\mathtt{s}(x))])$$

2. *How to use these schemes in saturation?*
   Just as before! Resolve with the premise(s).

Outline

Application: Synthesis Based on Induction

We can use induction for synthesis of recursive functions!

Recall the conjecture $\forall x. \exists y.\ \mathtt{half}(y) = x$.
This can be understood as a program specification:

"for any input x, there is an output y such that half of y is x"

...how can we construct the program computing y given x?

Application: Synthesis Based on Induction

We can use induction for synthesis of recursive functions!

Recall the conjecture $\forall x.\exists y.\ \mathtt{half}(y) = x$.
This can be understood as a program specification:

"for any input x, there is an output y such that half of y is x"

...how can we construct the program computing y given x?

$$\underbrace{\exists v_0.L[0, v_0]}_{\text{program } v_0} \wedge \quad \underbrace{\forall z.(\exists w.L[z, w] \rightarrow \exists v_{\mathrm{s}}.L[\mathrm{s}(z), v_{\mathrm{s}}])}_{\text{program } v_{\mathrm{s}}} \quad \rightarrow \quad \underbrace{\forall x.\exists y.L[x, y]}_{\text{program } x?}$$

Application: Synthesis Based on Induction

We can use induction for synthesis of recursive functions!

Recall the conjecture $\forall x. \exists y. \ \mathtt{half}(y) = x$.
This can be understood as a program specification:

"for any input x, there is an output y such that half of y is x"

...how can we construct the program computing y given x?

$$\underbrace{\exists v_0 . L[0, v_0]}_{\text{program } v_0} \wedge \underbrace{\forall z . (\exists w . L[z, w] \rightarrow \exists v_\mathtt{s} . L[\mathtt{s}(z), v_\mathtt{s}])}_{\text{program } v_\mathtt{s}} \rightarrow \underbrace{\forall x . \exists y . L[x, y]}_{\text{program } x?}$$

For more, come to the IJCAR talk on July 3rd at 16:30.

Induction Everywhere

Suppose our aim is to automate math.

What is “proof by induction”?

Induction Everywhere

Suppose our aim is to automate math.

What is "proof by induction"?

Analogues of Induction:

- Axiom of choice;
- König's lemma
- ...

Induction Everywhere

Suppose our aim is to automate math.

What is “proof by induction”?

Analogues of Induction:

- Axiom of choice;
- König’s lemma
- ...

Consistency of big chunks of modern math can be formalized using well-founded induction up to a certain ordinal

(A Simple) Example from a Textbook on Computational Logic

Lemma 1.12 (Independence lemma): *For any* LD*–refutation of $P \cup \{G\}$ of length n with answer substitution θ and any selection rule R, there is an* SLD*–refutation of $P \cup \{G\}$ via R of length n with an answer substitution ψ such that $G\psi$ is a renaming of $G\theta$.*

Proof: We proceed by induction on the length of the given LD–refutation. As usual, there is nothing to prove if it has length 1. Suppose we have an LD–refutation of G of length $k+1$ with answer substitution φ. Let A_j be the literal selected from G by R and suppose it is resolved on at step s of the given LD–refutation. Apply Lemma 1.11 $s-1$ times to get an LD–refutation $\langle G_0, C_0 \rangle, \ldots, \langle G_k, C_k \rangle$ of $P \cup \{G\} = P \cup \{G_0\}$ with answer substitution $\varphi = \varphi_0\varphi_1 \ldots \varphi_k$ in which A_j is the literal resolved on at step 1 and such that φ is a renaming of θ via λ. Now apply the induction hypothesis to the LD–refutation $\langle G_1, C_1 \rangle, \ldots, \langle G_k, C_k \rangle$ to

Proofs in Model Theory

Full of ordinal constructions and implicit induction (ultrafilters)

We begin with the idea that for each $i < \lambda$, a_i is the equivalence class in N of the sequence which is constantly equal to b_i. We then essentially doctor this sequence by winnowing $\mathcal{P}$, i.e. erasing some of the edges. Formally, of course, at each index t we choose a sequence $\langle b'_i[t] : i < \lambda \rangle$ of distinct elements of M (using the fact that M is universal for models of T of size $\leq \lambda$) such that for all $w \subseteq \lambda$, if $M \models R(\bar{b}'_w)$ then $M \models R(\bar{b}_w)$, but not necessarily the inverse. We will then set $a_i = \langle b'_i[t] : t \in I \rangle / \mathcal{D}$ for each $i < \lambda$. How to winnow edges? Following the notation of the proof of 4.1, fix an enumeration of $[\lambda]^k$ as $\langle v_\beta : \beta < \lambda \rangle$ without repetition, so the eventual type will be enumerated by $\{R(x, \bar{a}_{v_\beta}) : \beta < \lambda\}$. Let $\Omega = [\lambda]^{<\aleph_0}$. For each $s \in \Omega$, let the 'critical set' $\mathrm{cs}(s)$ be the set of $w \in \mathcal{P}$ such that each $v \in [w]^k$ is v_β for some $\beta \in w$. (Note that this is generally weaker than saying that $w \subseteq \mathrm{vert}(s)$.) The rule is that for each $t \in I$, and each $w \in \mathcal{P}$, we leave an edge on $\{b'_i : i \in w\}$ if and only if $t \in \mathbf{x}_{g_w}$. By the choice of g_w, no edge will persist in the ultrapower, so $\langle a_i : i < \lambda \rangle$

Proofs in Our Community

Example from my own slides on completeness of resolution (induction over a well-founded ordering).

Now we prove that I_R satisfies all clauses in S. Suppose it does not. Then there is a clause $F \in S$ such that $I_R \not\models F$. Note that F is non-empty, since S does not contain the empty clause.

Since $\succ$ is well-founded on clauses, the set of all clauses in S, which are false in I_R contains the least element. Denote this clause by F.

We will now show, by contradiction, that S contains a clause smaller than F and false in I_R. To prove this, we consider several cases, depending on which literal(s) are selected in F.

Proofs in Our Community

Suppose that F has a negative selected literal. Then F has the form $\underline{s \neq t} \vee D$. Consider the case when s coincides with t, then F has the form $\underline{s \neq s} \vee D$. Consider the equality resolution inference

$$\frac{\underline{s \neq s} \vee D}{D}$$

Since F is persistent and the process is fair, this inference was applied at some step, so D belongs to some search space S_i. Note that $F \succ D$ and $D \vdash F$. The last property implies that $I_R \not\models F$.

If $D \in S$, that is, D is persistent, than we are done, since we have found a clause in S that is false in I_R and smaller than D.

If $D \notin S$, then it was removed and so has a trace $D_1, \ldots, D_n$. Since $D_1, \ldots, D_n \vdash D$ and D is false, then there exists some D_i such that D_i is false too. Note that we have $D_i \prec D \prec F$, so again we have found a clause smaller than F and false in I_R.

Proofs in Math

1. FO reasoning
2. Theory-specific reasoning
3. Lemmas
4. Induction

2–4 are closely related.
There is hardly anything else . . .

What We (and Others) Did

We need to prove a formula of a certain shape (normally, universal) by induction.

We have a collection of induction schemas in this area of math.

Sometimes we can immediately apply some off-the-shelf induction schema (induction on length), sometime not.

Sometimes we have to generalize the formula we prove before applying induction.

Sometimes we prove one formula G (goal) but we need to apply induction to a different formula I.

Sometimes this formula I appears in the proof search, sometimes it does not.

Sometimes we can assemble I from formulas which appear in the proof search and then apply induction to I.

Is it About Induction?

We need to prove a formula of a certain shape (normally, universal).

We have a collection of techniques used in the area (induction schemas, diagonalization, tricks for working with sequences, ultrafilters etc.).

Sometimes we can immediately apply some off-the-shelf technique (induction on length), sometime not.

Sometimes we have to generalize the formula we prove before applying some technique.

Sometimes we prove one formula G (goal) but we need to apply a technique to a different formula I.

Sometimes this formula I appears in the proof search, sometimes it does not.

Sometimes we can assemble I from formulas which appear in the proof search and then apply a technique to I.

Underlying Idea: Theory Lemma Generation

First-order theorem provers are good in proving theorems in pure first order logic, aka logic of uninterpreted functions.

Underlying Idea: Theory Lemma Generation

First-order theorem provers are good in proving theorems in pure first order logic, aka logic of uninterpreted functions.

Applications of reasoning (math, program analysis, verification), require reasoning with various domains, such in integers, reals, sequences etc. Adding domain axioms to FOL is hopeless, moreover, such axiomatizations are commonly infinite.

Underlying Idea: Theory Lemma Generation

First-order theorem provers are good in proving theorems in pure first order logic, aka logic of uninterpreted functions.

Applications of reasoning (math, program analysis, verification), require reasoning with various domains, such in integers, reals, sequences etc. Adding domain axioms to FOL is hopeless, moreover, such axiomatizations are commonly infinite.

The idea is to capture specific ways of reasoning in such domains, of which most important ones are various induction rules.

Underlying Idea: Theory Lemma Generation

First-order theorem provers are good in proving theorems in pure first order logic, aka logic of uninterpreted functions.

Applications of reasoning (math, program analysis, verification), require reasoning with various domains, such in integers, reals, sequences etc. Adding domain axioms to FOL is hopeless, moreover, such axiomatizations are commonly infinite.

The idea is to capture specific ways of reasoning in such domains, of which most important ones are various induction rules.

In saturation provers we implement this by Theory Lemma Generation:

1. Generate a theory lemma: a formula that is valid in the theory but not valid in FOL. The formula is selected based on the occurrence of formulas of certain kind in the search space.
2. Add this theory lemma (clausified) to the search space.

Conclusion

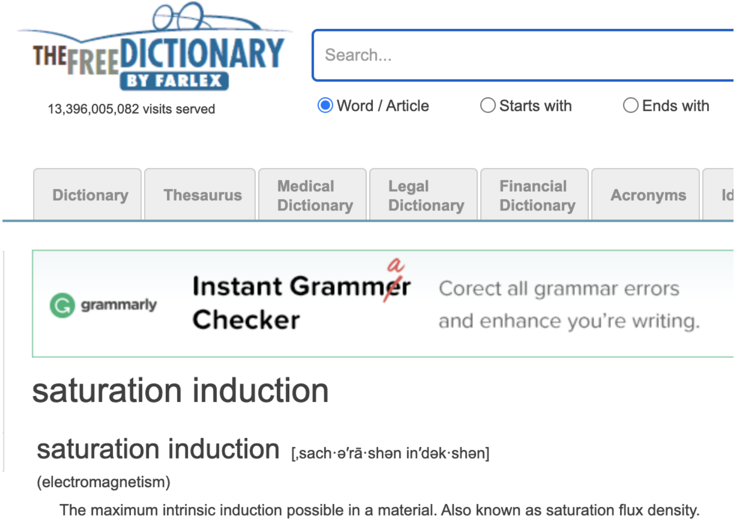
We have found a technique which allows one to apply saturation-based provers in various areas of math.

Human mathematicians are good in discovering new reasoning techniques in various areas of math.

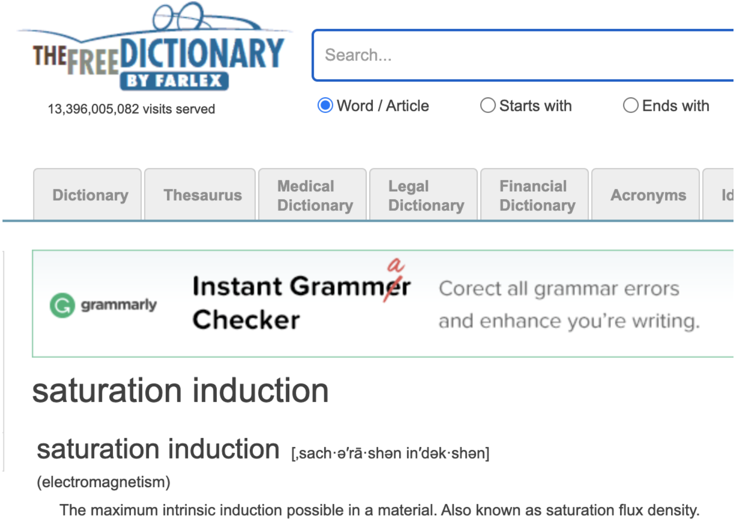
Modern saturation provers are good in combinatorics (FOL, use of decision procedures, case analysis).

Our new techniques allow theorem provers to prove more complex theorems in various parts of math.

Some Humor: Definition of Saturation Induction



Some Humor: Definition of Saturation Induction



The maximum intrinsic induction possible in a theorem prover