Congruence-Anticongruence Closure

Ján Kľuka

kluka@fmph.uniba.sk

Department of Applied Informatics
Faculty of Mathematics, Physics And Informatics
Comenius University Bratislava
Slovakia

1st Doctoral Workshop on
Mathematical and Engineering Methods in Computer Science,
Znojmo, Czech Republic, 2005

Outline

1. Motivation
   - An Overview of Our Aims
   - Existing Solutions

2. A Small Step Towards a Suitable Proof Assistant
   - The Context and the Goal
   - Definitions
   - Clauses, Congruence, Anticongruence
   - CA-Closure
   - An Example
   - Proof System and Algorithm

3. Conclusion and Further Work

Outline

Outline

An Overview of Our Aims

- Formal verification of declarative programs
  - programs in a comfortable conservative extension of Peano arithmetic
  - i.e., untyped first-order functional programs
  - assisted, but not automated verification (theorem proving)
- Teaching tool (*CL*)
- Plan on being more practical
  - modularization
  - compilation with in-place updates
- Our old but still used proof assistant

An Overview of Our Aims

- Formal verification of declarative programs
  - programs in a comfortable conservative extension of Peano arithmetic
  - i.e., untyped first-order functional programs
  - assisted, but not automated verification (theorem proving)
- Teaching tool (*CL*)
- Plan on being more practical
  - modularization
  - compilation with in-place updates
- Our old but still used proof assistant
  - based on Shostak combination of decision procedures

An Overview of Our Aims

- Formal verification of declarative programs
  - programs in a comfortable conservative extension of Peano arithmetic
  - i.e., untyped first-order functional programs
  - assisted, but not automated verification (theorem proving)
- Teaching tool (*CL*)
- Plan on being more practical
  - modularization
  - compilation with in-place updates
- Our old but still used proof assistant
  - based on Shostak combination of decision procedures
  - no longer suits our needs
    - difficult to extend (especially to automatically use lemmas)
    - difficult to generate a proof from automated steps
    - too eager to rewrite $\Longrightarrow$ confusing (sometimes even for an experienced user)

An Overview of Our Aims

- Formal verification of declarative programs
  - programs in a comfortable conservative extension of Peano arithmetic
  - i.e., untyped first-order functional programs
  - assisted, but not automated verification (theorem proving)
- Teaching tool (*CL*)
- Plan on being more practical
  - modularization
  - compilation with in-place updates
- Our old but still used proof assistant
  - based on Shostak combination of decision procedures
  - no longer suits our needs
    - difficult to extend (especially to automatically use lemmas)
    - difficult to generate a proof from automated steps
    - too eager to rewrite $\Longrightarrow$ confusing (sometimes even for an experienced user)

An Overview of Our Aims

- Formal verification of declarative programs
  - programs in a comfortable conservative extension of Peano arithmetic
  - i.e., untyped first-order functional programs
  - assisted, but not automated verification (theorem proving)
- Teaching tool (*CL*)
- Plan on being more practical
  - modularization
  - compilation with in-place updates
- Our old but still used proof assistant
  - based on Shostak combination of decision procedures
  - no longer suits our needs
    - difficult to extend (especially to automatically use lemmas)
    - difficult to generate a proof from automated steps
    - too eager to rewrite $\Longrightarrow$ confusing (sometimes even for an experienced user)

An Overview of Our Aims

- Formal verification of declarative programs
  - programs in a comfortable conservative extension of Peano arithmetic
  - i.e., untyped first-order functional programs
  - assisted, but not automated verification (theorem proving)
- Teaching tool (*CL*)
- Plan on being more practical
  - modularization
  - compilation with in-place updates
- Our old but still used proof assistant
  - based on Shostak combination of decision procedures
  - no longer suits our needs
    - difficult to extend (especially to automatically use lemmas)
    - difficult to generate a proof from automated steps
    - too eager to rewrite $\Longrightarrow$ confusing (sometimes even for an experienced user)

Properties of a Suitable Proof Assistant

We would like the proof assistant to be

- simple
- able to automatically use already proved theorems
- able to generate an independently checkable step-by-step proof from automated inference
- non-confusing

Properties of a Suitable Proof Assistant

We would like the proof assistant to be

- simple
- able to automatically use already proved theorems
- able to generate an independently checkable step-by-step proof from automated inference
- non-confusing
    - finish the proof if possible

Properties of a Suitable Proof Assistant

We would like the proof assistant to be

- simple
- able to automatically use already proved theorems
- able to generate an independently checkable step-by-step proof from automated inference
- non-confusing
  - finish the proof if possible
  - do not interfere otherwise (no automated rewriting, source of confusion)

Properties of a Suitable Proof Assistant

We would like the proof assistant to be

- simple
- able to automatically use already proved theorems
- able to generate an independently checkable step-by-step proof from automated inference
- non-confusing
  - finish the proof if possible
  - do not interfere otherwise (no automated rewriting, source of confusion)

Properties of a Suitable Proof Assistant

We would like the proof assistant to be

- simple
- able to automatically use already proved theorems
- able to generate an independently checkable step-by-step proof from automated inference
- non-confusing
  - finish the proof if possible
  - do not interfere otherwise (no automated rewriting, source of confusion)

Properties of a Suitable Proof Assistant

We would like the proof assistant to be

- simple
- able to automatically use already proved theorems
- able to generate an independently checkable step-by-step proof from automated inference
- non-confusing
  - finish the proof if possible
  - do not interfere otherwise (no automated rewriting, source of confusion)

Existing Solutions

Two main schools of assisted theorem proving

"West-Coast"

- Nelson-Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* "small theories"
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

"East-Coast (and European?)"

Existing Solutions

Two main schools of assisted theorem proving

"West-Coast"

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* "small theories"
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

"East-Coast (and European?)"

Existing Solutions

Two main schools of assisted theorem proving

“West-Coast”

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* “small theories”
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

“East-Coast (and European?)”

Existing Solutions

Two main schools of assisted theorem proving

“West-Coast”

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* “small theories”
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

“East-Coast (and European?)”

Existing Solutions

Two main schools of assisted theorem proving

"West-Coast"

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* "small theories"
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

"East-Coast (and European?)"

Existing Solutions

Two main schools of assisted theorem proving

“West-Coast”

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* “small theories”
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

“East-Coast (and European?)”

Existing Solutions

Two main schools of assisted theorem proving

“West-Coast”

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* “small theories”
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

“East-Coast (and European?)”

- written in ML (HOL, nuPRL, Isabelle)

Existing Solutions

Two main schools of assisted theorem proving

“West-Coast”

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* “small theories”
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

“East-Coast (and European?)”

- written in ML (HOL, nuPRL, Isabelle)
- proof procedures (strategies) always generate step-by-step proof
- correctness ensured by ML typing
- tied to higher-order logic (HO unification)
- rewriting

Existing Solutions

Two main schools of assisted theorem proving

"West-Coast"

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* "small theories"
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

"East-Coast (and European?)"

- written in ML (HOL, nuPRL, Isabelle)
- proof procedures (strategies) always generate step-by-step proof
- correctness ensured by ML-typing
- tied to higher-order logic (HO unification)
- rewriting

Existing Solutions

Two main schools of assisted theorem proving

“West-Coast”

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* “small theories”
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

“East-Coast (and European?)”

- written in ML (HOL, nuPRL, Isabelle)
- proof procedures (strategies) always generate step-by-step proof
- correctness ensured by ML-typing
- tied to higher-order logic (HO unification)
- rewriting

Existing Solutions

Two main schools of assisted theorem proving

“West-Coast”

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* “small theories”
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

“East-Coast (and European?)”

- written in ML (HOL, nuPRL, Isabelle)
- proof procedures (strategies) always generate step-by-step proof
- correctness ensured by ML-typing
- tied to higher-order logic (HO unification)
- rewriting

Existing Solutions

Two main schools of assisted theorem proving

"West-Coast"

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* "small theories"
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

"East-Coast (and European?)"

- written in ML (HOL, nuPRL, Isabelle)
- proof procedures (strategies) always generate step-by-step proof
- correctness ensured by ML-typing
- tied to higher-order logic (HO unification)
- rewriting

Existing Solutions

Two main schools of assisted theorem proving

“West-Coast”

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* “small theories”
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

“East-Coast (and European?)”

- written in ML (HOL, nuPRL, Isabelle)
- proof procedures (strategies) always generate step-by-step proof
- correctness ensured by ML-typing
- tied to higher-order logic (HO unification)
- rewriting

Existing Solutions

Two main schools of assisted theorem proving

"West-Coast"

- Nelson–Oppen
- Shostak (PVS)
- combination of decision procedures for *built-in* "small theories"
- powerful, but no automated use of already proved lemmas
- rarely generate checkable proofs
- Shostak: long history of problems with correctness
- Shostak: canonization (i.e., rewriting) is central

"East-Coast (and European?)"

- written in ML (HOL, nuPRL, Isabelle)
- proof procedures (strategies) always generate step-by-step proof
- correctness ensured by ML-typing
- tied to higher-order logic (HO unification)
- rewriting

Outline

The Context and the Goal

- Formal system: Gentzen sequent calculus or tableaux for first-order formulas with equalities
- State of the proof is given by a *sequent* $\Gamma \Rightarrow \Delta$
  - Γ, Δ are sets of first-order formulas
  - all *assumptions* Γ hold; among others axioms and lemmas
  - at least one of Δ to prove
- Sufficient to prove $\Gamma' \Rightarrow \Delta'$ for some $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$
- Minimal useful choice of Γ', Δ' for automation:
- Our choice: *closed, quantifier-free formulas*
- $\mathscr{NP}$-complete, but. . .

The Context and the Goal

- Formal system: Gentzen sequent calculus or tableaux for first-order formulas with equalities
- State of the proof is given by a *sequent* $\Gamma \Rightarrow \Delta$
  - Γ, Δ are sets of first-order formulas
  - all *assumptions* Γ hold; among others axioms and lemmas
  - at least one *goal* of Δ to prove
- Sufficient to prove $\Gamma' \Rightarrow \Delta'$ for some $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$
- Minimal useful choice of Γ', Δ' for automation:
- Our choice: *closed, quantifier-free formulas*
- $\mathscr{NP}$-complete, but. . .

The Context and the Goal

- Formal system: Gentzen sequent calculus or tableaux for first-order formulas with equalities
- State of the proof is given by a *sequent* $\Gamma \Rightarrow \Delta$
  - Γ, Δ are sets of first-order formulas
  - all *assumptions* Γ hold; among others axioms and lemmas
  - at least one *goal* of Δ to prove
- Sufficient to prove $\Gamma' \Rightarrow \Delta'$ for some $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$
- Minimal useful choice of Γ', Δ' for automation:
- Our choice: *closed, quantifier-free formulas*
- $\mathcal{NP}$-complete, but...

The Context and the Goal

- Formal system: Gentzen sequent calculus or tableaux for first-order formulas with equalities
- State of the proof is given by a *sequent* $\Gamma \Rightarrow \Delta$
  - Γ, Δ are sets of first-order formulas
  - all *assumptions* Γ hold; among others axioms and lemmas
  - at least one *goal* of Δ to prove
- Sufficient to prove $\Gamma' \Rightarrow \Delta'$ for some $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$
- Minimal useful choice of Γ', Δ' for automation:
- Our choice: *closed, quantifier-free formulas*
- $\mathcal{NP}$-complete, but. . .

The Context and the Goal

- Formal system: Gentzen sequent calculus or tableaux for first-order formulas with equalities
- State of the proof is given by a *sequent* $\Gamma \Rightarrow \Delta$
  - Γ, Δ are sets of first-order formulas
  - all *assumptions* Γ hold; among others axioms and lemmas
  - at least one *goal* of Δ to prove
- Sufficient to prove $\Gamma' \Rightarrow \Delta'$ for some $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$
- Minimal useful choice of Γ', Δ' for automation:

  [illegible] $s = t$ of closed (ground) terms

- Our choice: *closed, quantifier-free formulas*
- $\mathcal{NP}$-complete, but. . .

The Context and the Goal

- Formal system: Gentzen sequent calculus or tableaux for first-order formulas with equalities
- State of the proof is given by a *sequent* $\Gamma \Rightarrow \Delta$
  - Γ, Δ are sets of first-order formulas
  - all *assumptions* Γ hold; among others axioms and lemmas
  - at least one *goal* of Δ to prove
- Sufficient to prove $\Gamma' \Rightarrow \Delta'$ for some $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$
- Minimal useful choice of Γ', Δ' for automation:
  - equalities (identities) $s = t$ of closed (ground) terms
  - decidable by the congruence closure algorithm in $O(n \log n)$
- Our choice: *closed, quantifier-free formulas*
- $\mathcal{NP}$-complete, but. . .

The Context and the Goal

- Formal system: Gentzen sequent calculus or tableaux for first-order formulas with equalities
- State of the proof is given by a *sequent* $\Gamma \Rightarrow \Delta$
  - Γ, Δ are sets of first-order formulas
  - all *assumptions* Γ hold; among others axioms and lemmas
  - at least one *goal* of Δ to prove
- Sufficient to prove $\Gamma' \Rightarrow \Delta'$ for some $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$
- Minimal useful choice of Γ', Δ' for automation:
  - *equalities* (*identities*) $s = t$ of closed (ground) terms
  - decidable by the *congruence closure algorithm* in $O(n \log n)$
- Our choice: *closed, quantifier-free formulas*
- $\mathcal{NP}$-complete, but...

The Context and the Goal

- Formal system: Gentzen sequent calculus or tableaux for first-order formulas with equalities
- State of the proof is given by a *sequent* $\Gamma \Rightarrow \Delta$
  - Γ, Δ are sets of first-order formulas
  - all *assumptions* Γ hold; among others axioms and lemmas
  - at least one *goal* of Δ to prove
- Sufficient to prove $\Gamma' \Rightarrow \Delta'$ for some $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$
- Minimal useful choice of Γ', Δ' for automation:
  - *equalities* (*identities*) $s = t$ of closed (ground) terms
  - decidable by the *congruence closure algorithm* in $O(n \log n)$
- Our choice: *closed, quantifier-free formulas*
- $\mathcal{NP}$-complete, but. . .

The Context and the Goal

- Formal system: Gentzen sequent calculus or tableaux for first-order formulas with equalities
- State of the proof is given by a *sequent* $\Gamma \Rightarrow \Delta$
  - Γ, Δ are sets of first-order formulas
  - all *assumptions* Γ hold; among others axioms and lemmas
  - at least one *goal* of Δ to prove
- Sufficient to prove $\Gamma' \Rightarrow \Delta'$ for some $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$
- Minimal useful choice of Γ', Δ' for automation:
  - *equalities* (*identities*) $s = t$ of closed (ground) terms
  - decidable by the *congruence closure algorithm* in $O(n \log n)$
- Our choice: *closed, quantifier-free formulas*
- $\mathcal{NP}$-complete, but. . .

The Context and the Goal

- Formal system: Gentzen sequent calculus or tableaux for first-order formulas with equalities
- State of the proof is given by a *sequent* $\Gamma \Rightarrow \Delta$
  - Γ, Δ are sets of first-order formulas
  - all *assumptions* Γ hold; among others axioms and lemmas
  - at least one *goal* of Δ to prove
- Sufficient to prove $\Gamma' \Rightarrow \Delta'$ for some $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$
- Minimal useful choice of Γ', Δ' for automation:
  - *equalities* (*identities*) $s = t$ of closed (ground) terms
  - decidable by the *congruence closure algorithm* in $O(n \log n)$
- Our choice: *closed, quantifier-free formulas*
- $\mathcal{NP}$-complete, but. . .

The Context and the Goal

- Formal system: Gentzen sequent calculus or tableaux for first-order formulas with equalities
- State of the proof is given by a *sequent* $\Gamma \Rightarrow \Delta$
  - Γ, Δ are sets of first-order formulas
  - all *assumptions* Γ hold; among others axioms and lemmas
  - at least one *goal* of Δ to prove
- Sufficient to prove $\Gamma' \Rightarrow \Delta'$ for some $\Gamma' \subseteq \Gamma$, $\Delta' \subseteq \Delta$
- Minimal useful choice of Γ', Δ' for automation:
  - *equalities* (*identities*) $s = t$ of closed (ground) terms
  - decidable by the *congruence closure algorithm* in $O(n \log n)$
- Our choice: *closed, quantifier-free formulas*
- $\mathscr{NP}$-complete, but. . .

Closed, Quantifier-Free Formulas with Equalities
Definition

Language $\mathscr{L}$:

- countable set of function and relation symbols, *arity* of each symbol
- no variables; at least one constant—function symbol 0 of arity 0

Terms $(s, t, \ldots)$ defined inductively: the set of all $f(t_1, \ldots, t_n)$

- f is a function symbol of arity $n \geq 0$,
- if $n > 0$, $t_1, \ldots, t_n$ are terms

Atomic formulas

- $s = t$ for terms s, t
- $R(t_1, \ldots, t_n)$ for terms t_i, $i = 1, \ldots, n$

Formulas $(A, B, \ldots)$ defined inductively

- atomic formulas
- $\top$, $\bot$, $\neg A$, $A \wedge B$, $A \vee B$, $A \rightarrow B$, if A, B are formulas
- $\Gamma \Rightarrow \Delta$ (*sequent*) if Γ, Δ are *finite* sets of formulas
- no variables $\Longrightarrow$ no quantifiers

Closed, Quantifier-Free Formulas with Equalities

Definition

Language $\mathscr{L}$:

- countable set of function and relation symbols, *arity* of each symbol
- no variables; at least one constant—function symbol 0 of arity 0

Terms $(s, t, \ldots)$ defined inductively: the set of all $f(t_1, \ldots, t_n)$

- f is a function symbol of arity $n \geq 0$,
- if $n > 0$, $t_1, \ldots, t_n$ are terms

Atomic formulas

- $s = t$ for terms s, t
- $R(t_1, \ldots, t_n)$ for terms t_i, $i = 1, \ldots, n$

Formulas $(A, B, \ldots)$ defined inductively

- atomic formulas
- $\top$, $\bot$, $\neg A$, $A \wedge B$, $A \vee B$, $A \rightarrow B$, if A, B are formulas
- $\Gamma \Rightarrow \Delta$ (*sequent*) if Γ, Δ are *finite* sets of formulas
- no variables $\Longrightarrow$ no quantifiers

Closed, Quantifier-Free Formulas with Equalities

Definition

Language $\mathscr{L}$:

- countable set of function and relation symbols, *arity* of each symbol
- no variables; at least one constant—function symbol 0 of arity 0

Terms $(s, t, \dots)$ defined inductively: the set of all $f(t_1, \dots, t_n)$

- f is a function symbol of arity $n \geq 0$,
- if $n > 0$, $t_1, \dots, t_n$ are terms

Atomic formulas

- $s = t$ for terms s, t
- $R(t_1, \dots, t_n)$ for terms t_i, $i = 1, \dots, n$

Formulas $(A, B, \dots)$ defined inductively

- atomic formulas
- $\top$, $\bot$, $\neg A$, $A \wedge B$, $A \vee B$, $A \to B$, if A, B are formulas
- $\Gamma \Rightarrow \Delta$ (*sequent*) if Γ, Δ are *finite* sets of formulas
- no variables $\Longrightarrow$ no quantifiers

Closed, Quantifier-Free Formulas with Equalities
Definition

Language $\mathscr{L}$:

- countable set of function and relation symbols, *arity* of each symbol
- no variables; at least one constant—function symbol 0 of arity 0

Terms $(s, t, \dots)$ defined inductively: the set of all $f(t_1, \dots, t_n)$

- f is a function symbol of arity $n \geq 0$,
- if $n > 0$, $t_1, \dots, t_n$ are terms

Atomic formulas

- $s = t$ for terms s, t
- $R(t_1, \dots, t_n)$ for terms t_i, $i = 1, \dots, n$

Formulas $(A, B, \dots)$ defined inductively

- atomic formulas
- $\top$, $\bot$, $\neg A$, $A \wedge B$, $A \vee B$, $A \rightarrow B$, if A, B are formulas
- $\Gamma \Rightarrow \Delta$ (*sequent*) if Γ, Δ are *finite* sets of formulas
- no variables $\Longrightarrow$ no quantifiers

Quantifier-Free Formulas with Equality
Semantics

- Standard notion of *structure* $\mathscr{M}$:
  - domain D
  - $f^{\mathscr{M}} : D^n \longrightarrow D$
  - $R^{\mathscr{M}} \subseteq D^n$
- Recursively defined *valuation* of terms $t^{\mathscr{M}}$
- Satisfaction relation $\mathscr{M} \models A$
  - $\mathscr{M} \models s = t$ iff $s^{\mathscr{M}} = t^{\mathscr{M}}$
  - $\mathscr{M} \models R(s_1, \ldots, s_n)$ iff $(s_1^{\mathscr{M}}, \ldots, s_n^{\mathscr{M}}) \in R^{\mathscr{M}}$
  - $\mathscr{M} \models A \land B$ iff $\mathscr{M} \models A$ and $\mathscr{M} \models B$

    ...
  - $\mathscr{M} \models \Gamma \Rightarrow \Delta$ iff whenever $\mathscr{M} \models A$ for all $A \in \Gamma$, then there is a $B \in \Delta$ such that $\mathscr{M} \models B$
- Consequence $T \models A$
  - for all $\mathscr{M}$ such that $\mathscr{M} \models T$, $\mathscr{M} \models A$

Quantifier-Free Formulas with Equality
Semantics

- Standard notion of *structure* $\mathscr{M}$:
  - domain D
  - $f^{\mathscr{M}} : D^n \longrightarrow D$
  - $R^{\mathscr{M}} \subseteq D^n$
- Recursively defined *valuation* of terms $t^{\mathscr{M}}$
- Satisfaction relation $\mathscr{M} \models A$
  - $\mathscr{M} \models s = t$ iff $s^{\mathscr{M}} = t^{\mathscr{M}}$
  - $\mathscr{M} \models R(s_1, \ldots, s_n)$ iff $(s_1^{\mathscr{M}}, \ldots, s_n^{\mathscr{M}}) \in R^{\mathscr{M}}$
  - $\mathscr{M} \models A \wedge B$ iff $\mathscr{M} \models A$ and $\mathscr{M} \models B$
  - ...
  - $\mathscr{M} \models \Gamma \Rightarrow \Delta$ iff whenever $\mathscr{M} \models A$ for all $A \in \Gamma$, then there is a $B \in \Delta$ such that $\mathscr{M} \models B$
- Consequence $T \models A$
  - for all $\mathscr{M}$ such that $\mathscr{M} \models T$, $\mathscr{M} \models A$

Quantifier-Free Formulas with Equality

Semantics

- Standard notion of *structure* $\mathscr{M}$:
  - domain D
  - $f^{\mathscr{M}} : D^n \longrightarrow D$
  - $R^{\mathscr{M}} \subseteq D^n$
- Recursively defined *valuation* of terms $t^{\mathscr{M}}$
- Satisfaction relation $\mathscr{M} \models A$
  - $\mathscr{M} \models s = t$ iff $s^{\mathscr{M}} = t^{\mathscr{M}}$
  - $\mathscr{M} \models R(s_1, \ldots, s_n)$ iff $(s_1^{\mathscr{M}}, \ldots, s_n^{\mathscr{M}}) \in R^{\mathscr{M}}$
  - $\mathscr{M} \models A \wedge B$ iff $\mathscr{M} \models A$ and $\mathscr{M} \models B$

  ...
  - $\mathscr{M} \models \Gamma \Rightarrow \Delta$ iff whenever $\mathscr{M} \models A$ for all $A \in \Gamma$, then there is a $B \in \Delta$ such that $\mathscr{M} \models B$
- Consequence $T \models A$
  - for all $\mathscr{M}$ such that $\mathscr{M} \models T$, $\mathscr{M} \models A$

Quantifier-Free Formulas with Equality

Semantics

- Standard notion of *structure* $\mathscr{M}$:
  - domain D
  - $f^{\mathscr{M}} : D^n \longrightarrow D$
  - $R^{\mathscr{M}} \subseteq D^n$
- Recursively defined *valuation* of terms $t^{\mathscr{M}}$
- Satisfaction relation $\mathscr{M} \models A$
  - $\mathscr{M} \models s = t$ iff $s^{\mathscr{M}} = t^{\mathscr{M}}$
  - $\mathscr{M} \models R(s_1, \ldots, s_n)$ iff $(s_1^{\mathscr{M}}, \ldots, s_n^{\mathscr{M}}) \in R^{\mathscr{M}}$
  - $\mathscr{M} \models A \wedge B$ iff $\mathscr{M} \models A$ and $\mathscr{M} \models B$

  ...
  - $\mathscr{M} \models \Gamma \Rightarrow \Delta$ iff whenever $\mathscr{M} \models A$ for all $A \in \Gamma$, then there is a $B \in \Delta$ such that $\mathscr{M} \models B$
- Consequence $T \models A$
  - for all $\mathscr{M}$ such that $\mathscr{M} \models T$, $\mathscr{M} \models A$

Quantifier-Free Formulas with Equality

Semantics

- Standard notion of *structure* $\mathscr{M}$:
  - domain D
  - $f^{\mathscr{M}} : D^n \longrightarrow D$
  - $R^{\mathscr{M}} \subseteq D^n$
- Recursively defined *valuation* of terms $t^{\mathscr{M}}$
- Satisfaction relation $\mathscr{M} \models A$
  - $\mathscr{M} \models s = t$ iff $s^{\mathscr{M}} = t^{\mathscr{M}}$
  - $\mathscr{M} \models R(s_1, \ldots, s_n)$ iff $(s_1^{\mathscr{M}}, \ldots, s_n^{\mathscr{M}}) \in R^{\mathscr{M}}$
  - $\mathscr{M} \models A \wedge B$ iff $\mathscr{M} \models A$ and $\mathscr{M} \models B$

  ...
  - $\mathscr{M} \models \Gamma \Rightarrow \Delta$ iff whenever $\mathscr{M} \models A$ for all $A \in \Gamma$, then there is a $B \in \Delta$ such that $\mathscr{M} \models B$
- Consequence $T \models A$
  - for all $\mathscr{M}$ such that $\mathscr{M} \models T$, $\mathscr{M} \models A$

Congruence and Congruence Closure

The congruence closure algorithm:

- a data structure representing an *equivalence* over terms

  reflexivity $s = s$

  symmetry $s = t \rightarrow t = s$

  transitivity $s = t \wedge t = u \rightarrow s = u$

- iterative closing under *congruence* (the substitution axiom for function symbols)

$$s_1 = t_1 \wedge \cdots \wedge s_n = t_n \rightarrow f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$$

- the sequent form of congruence:

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$$

Congruence and Congruence Closure

The congruence closure algorithm:

- a data structure representing an *equivalence* over terms

  reflexivity $s = s$

  symmetry $s = t \rightarrow t = s$

  transitivity $s = t \wedge t = u \rightarrow s = u$

- iterative closing under *congruence*
  (the substitution axiom for function symbols)

$$s_1 = t_1 \wedge \cdots \wedge s_n = t_n \rightarrow f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$$

- the sequent form of congruence:

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$$

Congruence and Congruence Closure

The congruence closure algorithm:

- a data structure representing an *equivalence* over terms

  reflexivity $s = s$

  symmetry $s = t \rightarrow t = s$

  transitivity $s = t \wedge t = u \rightarrow s = u$

- iterative closing under *congruence*
  (the substitution axiom for function symbols)

$$s_1 = t_1 \wedge \cdots \wedge s_n = t_n \rightarrow f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$$

- the sequent form of congruence:

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$$

Horn Clauses and Congruence

The Key Observation

A *Horn clause* with only equalities as atomic formulas

$$s_1 \neq t_1 \vee \cdots \vee s_n \neq t_n \vee u = v$$

sequent form

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow u = v$$

is equivalent to

$$g(s_1, \ldots, s_n) = u$$
$$g(t_1, \ldots, t_n) = v$$

plus congruence

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow g(s_1, \ldots, s_n) = g(t_1, \ldots, t_n)$$

for a *new* function symbol g

Horn Clauses and Congruence

The Key Observation

A *Horn clause* with only equalities as atomic formulas

$$s_1 \neq t_1 \vee \cdots \vee s_n \neq t_n \vee u = v$$

sequent form

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow u = v$$

is equivalent to

$$g(s_1, \ldots, s_n) = u$$
$$g(t_1, \ldots, t_n) = v$$

plus congruence

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow g(s_1, \ldots, s_n) = g(t_1, \ldots, t_n)$$

for a *new* function symbol g

Horn Clauses and Congruence

The Key Observation

A *Horn clause* with only equalities as atomic formulas

$$s_1 \neq t_1 \vee \cdots \vee s_n \neq t_n \vee u = v$$

sequent form

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow u = v$$

is equivalent to

$$g(s_1, \ldots, s_n) = u$$
$$g(t_1, \ldots, t_n) = v$$

plus congruence

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow g(s_1, \ldots, s_n) = g(t_1, \ldots, t_n)$$

for a *new* function symbol g

Horn Clauses and Congruence

The Key Observation

A *Horn clause* with only equalities as atomic formulas

$$s_1 \neq t_1 \vee \cdots \vee s_n \neq t_n \vee u = v$$

sequent form

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow u = v$$

is equivalent to

$$g(s_1, \ldots, s_n) = u$$
$$g(t_1, \ldots, t_n) = v$$

plus congruence

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow g(s_1, \ldots, s_n) = g(t_1, \ldots, t_n)$$

for a *new* function symbol g

Clauses and Anticongruence

A general *clause* with only equalities as atomic formulas (sequent form)

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow u_1 = v_1, \ldots, u_m = v_m$$

is equivalent to

$$g(s_1, \ldots, s_n) = h(u_1, \ldots, u_m)$$
$$g(t_1, \ldots, t_n) = h(v_1, \ldots, v_m)$$

plus congruence

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow g(s_1, \ldots, s_n) = g(t_1, \ldots, t_n)$$

for a *new* function symbol g
plus *anticongruence*

$$h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m) \Rightarrow u_1 = v_1, \ldots, u_m = v_m$$

for a *new anticongruence function symbol* h

Clauses and Anticongruence

A general *clause* with only equalities as atomic formulas (sequent form)

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow u_1 = v_1, \ldots, u_m = v_m$$

is equivalent to

$$g(s_1, \ldots, s_n) = h(u_1, \ldots, u_m)$$
$$g(t_1, \ldots, t_n) = h(v_1, \ldots, v_m)$$

plus congruence

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow g(s_1, \ldots, s_n) = g(t_1, \ldots, t_n)$$

for a *new* function symbol g
plus *anticongruence*

$$h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m) \Rightarrow u_1 = v_1, \ldots, u_m = v_m$$

for a *new anticongruence function symbol* h

Clauses and Anticongruence

A general *clause* with only equalities as atomic formulas (sequent form)

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow u_1 = v_1, \ldots, u_m = v_m$$

is equivalent to

$$g(s_1, \ldots, s_n) = h(u_1, \ldots, u_m)$$
$$g(t_1, \ldots, t_n) = h(v_1, \ldots, v_m)$$

plus congruence

$$s_1 = t_1, \ldots, s_n = t_n \Rightarrow g(s_1, \ldots, s_n) = g(t_1, \ldots, t_n)$$

for a *new* function symbol g

plus *anticongruence*

$$h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m) \Rightarrow u_1 = v_1, \ldots, u_m = v_m$$

for a *new anticongruence function symbol* h

Clauses and Anticongruence

A general *clause* with only equalities as atomic formulas (sequent form)

$$s_1 = t_1, \dots, s_n = t_n \Rightarrow u_1 = v_1, \dots, u_m = v_m$$

is equivalent to

$$g(s_1, \dots, s_n) = h(u_1, \dots, u_m)$$
$$g(t_1, \dots, t_n) = h(v_1, \dots, v_m)$$

plus congruence

$$s_1 = t_1, \dots, s_n = t_n \Rightarrow g(s_1, \dots, s_n) = g(t_1, \dots, t_n)$$

for a *new* function symbol g
plus *anticongruence*

$$h(u_1, \dots, u_m) = h(v_1, \dots, v_m) \Rightarrow u_1 = v_1, \dots, u_m = v_m$$

for a *new anticongruence function symbol* h

CA-Closure for Assisted Theorem Proving

- Equality is central, must always be handled automatically (by congruence closure)
- Propositional part can be reduced to equality and handled by *anticongruence closure*
- Several steps are needed:
  - Elimination of relation symbols (makes equality the only relation symbol)

CA-Closure for Assisted Theorem Proving

- Equality is central, must always be handled automatically (by congruence closure)
- Propositional part can be reduced to equality and handled by *anticongruence closure*
- Several steps are needed:
  - Elimination of relation symbols (makes equality the only relation symbol)
  - Transformation to conjunctive normal form (set of sequents with only atomic members—clauses)
  - Replacement of sequents by new special function symbols and equalities
  - Proof system for equalities with special function symbols
  - Algorithm for the proof system

CA-Closure for Assisted Theorem Proving

- Equality is central, must always be handled automatically (by congruence closure)
- Propositional part can be reduced to equality and handled by *anticongruence closure*
- Several steps are needed:
  - Elimination of relation symbols (makes equality the only relation symbol)
  - Transformation to conjunctive normal form (set of sequents with only atomic members—clauses)
  - Replacement of sequents by new special function symbols and equalities
  - Proof system for equalities with special function symbols
  - Algorithm for the proof system

CA-Closure for Assisted Theorem Proving

- Equality is central, must always be handled automatically (by congruence closure)
- Propositional part can be reduced to equality and handled by *anticongruence closure*
- Several steps are needed:
  - Elimination of relation symbols (makes equality the only relation symbol)
  - Transformation to conjunctive normal form (set of sequents with only atomic members—clauses)
  - Replacement of sequents by new special function symbols and equalities
  - Proof system for equalities with special function symbols
  - Algorithm for the proof system

CA-Closure for Assisted Theorem Proving

- Equality is central, must always be handled automatically (by congruence closure)
- Propositional part can be reduced to equality and handled by *anticongruence closure*
- Several steps are needed:
  - Elimination of relation symbols (makes equality the only relation symbol)
  - Transformation to conjunctive normal form (set of sequents with only atomic members—clauses)
  - Replacement of sequents by new special function symbols and equalities
  - Proof system for equalities with special function symbols
  - Algorithm for the proof system

CA-Closure for Assisted Theorem Proving

- Equality is central, must always be handled automatically (by congruence closure)
- Propositional part can be reduced to equality and handled by *anticongruence closure*
- Several steps are needed:
  - Elimination of relation symbols (makes equality the only relation symbol)
  - Transformation to conjunctive normal form (set of sequents with only atomic members—clauses)
  - Replacement of sequents by new special function symbols and equalities
  - Proof system for equalities with special function symbols
  - Algorithm for the proof system

CA-Closure for Assisted Theorem Proving

- Equality is central, must always be handled automatically (by congruence closure)
- Propositional part can be reduced to equality and handled by *anticongruence closure*
- Several steps are needed:
  - Elimination of relation symbols (makes equality the only relation symbol)
  - Transformation to conjunctive normal form (set of sequents with only atomic members—clauses)
  - Replacement of sequents by new special function symbols and equalities
  - Proof system for equalities with special function symbols
  - Algorithm for the proof system

CA-Closure for Assisted Theorem Proving

- Equality is central, must always be handled automatically (by congruence closure)
- Propositional part can be reduced to equality and handled by *anticongruence closure*
- Several steps are needed:
  - Elimination of relation symbols (makes equality the only relation symbol)
  - Transformation to conjunctive normal form (set of sequents with only atomic members—clauses)
  - Replacement of sequents by new special function symbols and equalities
  - Proof system for equalities with special function symbols
  - Algorithm for the proof system

An Example
Part 1 of 2

- Initial closed, quantifier-free part of the sequent to be proved

$$\left.\begin{array}{l} a<b \wedge b<c \rightarrow a<c, \\ a<b \vee b<a, \\ b<c, b=d \end{array}\right\} \Rightarrow a<c, d<a$$

- Elimination of relation symbols

$$\left.\begin{array}{l} <_*(a,b)=0 \wedge <_*(b,c)=0 \rightarrow <_*(a,c)=0, \\ <_*(a,b)=0 \vee <_*(b,a)=0, \\ <_*(d,c)=0, b=d \end{array}\right\} \Rightarrow <_*(a,c)=0, <_*(d,a)=0$$

- Transformation to CNF

$$<_*(a,b)=0, <_*(b,c)=0 \Rightarrow <_*(a,c)=0, \qquad \Rightarrow <_*(a,b)=0, <_*(b,a)=0,$$

$$\Rightarrow <_*(b,c)=0, \quad \Rightarrow b=d, \quad <_*(a,c)=0 \Rightarrow \quad , \quad <_*(d,a)=0 \Rightarrow$$

Avoid exponential blow-up

An Example
Part 1 of 2

- Initial closed, quantifier-free part of the sequent to be proved

$$\left.\begin{array}{l} a<b \wedge b<c \rightarrow a<c, \\ a<b \vee b<a, \\ b<c, b=d \end{array}\right\} \Rightarrow a<c, d<a$$

- Elimination of relation symbols

$$\left.\begin{array}{l} <_*(a,b)=0 \wedge <_*(b,c)=0 \rightarrow <_*(a,c)=0, \\ <_*(a,b)=0 \vee <_*(b,a)=0, \\ <_*(d,c)=0, b=d \end{array}\right\} \Rightarrow <_*(a,c)=0, <_*(d,a)=0$$

- Transformation to CNF

$$<_*(a,b)=0, <_*(b,c)=0 \Rightarrow <_*(a,c)=0, \qquad \Rightarrow <_*(a,b)=0, <_*(b,a)=0,$$

$$\Rightarrow <_*(b,c)=0, \quad \Rightarrow b=d, \quad <_*(a,c)=0 \Rightarrow \quad , \quad <_*(d,a)=0 \Rightarrow$$

- Avoid exponential blow-up

An Example
Part 1 of 2

- Initial closed, quantifier-free part of the sequent to be proved

$$\left.\begin{aligned} &a < b \wedge b < c \rightarrow a < c,\\ &a < b \vee b < a,\\ &b < c, b = d \end{aligned}\right\} \Rightarrow a < c, d < a$$

- Elimination of relation symbols

$$\left.\begin{aligned} &<_*(a,b) = 0 \wedge <_*(b,c) = 0 \rightarrow <_*(a,c) = 0,\\ &<_*(a,b) = 0 \vee <_*(b,a) = 0,\\ &<_*(d,c) = 0, b = d \end{aligned}\right\} \Rightarrow <_*(a,c) = 0, <_*(d,a) = 0$$

- Transformation to CNF

$$<_*(a,b) = 0, <_*(b,c) = 0 \Rightarrow <_*(a,c) = 0, \qquad \Rightarrow <_*(a,b) = 0, <_*(b,a) = 0,$$
$$\Rightarrow <_*(b,c) = 0, \quad \Rightarrow b = d, \quad <_*(a,c) = 0 \Rightarrow \quad , \quad <_*(d,a) = 0 \Rightarrow$$

Avoid exponential blow-up

An Example
Part 1 of 2

- Initial closed, quantifier-free part of the sequent to be proved

$$\left.\begin{array}{l} a<b\wedge b<c\rightarrow a<c, \\ a<b\vee b<a, \\ b<c, b=d \end{array}\right\}\Rightarrow a<c, d<a$$

- Elimination of relation symbols

$$\left.\begin{array}{l} <_*(a,b)=0\wedge <_*(b,c)=0\rightarrow <_*(a,c)=0, \\ <_*(a,b)=0\vee <_*(b,a)=0, \\ <_*(d,c)=0, b=d \end{array}\right\}\Rightarrow <_*(a,c)=0, <_*(d,a)=0$$

- Transformation to CNF[1]

$$<_*(a,b)=0, <_*(b,c)=0\Rightarrow <_*(a,c)=0, \qquad \Rightarrow <_*(a,b)=0, <_*(b,a)=0,$$

$$\Rightarrow <_*(b,c)=0, \quad \Rightarrow b=d, \quad <_*(a,c)=0\Rightarrow 0=1, \quad <_*(d,a)=0\Rightarrow 0=1$$

Avoid exponential blow-up

An Example

Part 2 of 2

- After transformation to CNF[1]

$$
\begin{aligned}
&<_*(a,b)=0, <_*(b,c)=0 \Rightarrow <_*(a,c)=0\\
&\Rightarrow <_*(a,b)=0, <_*(b,a)=0,\\
&\Rightarrow <_*(b,c)=0, \quad \Rightarrow b=d,\\
&<_*(a,c)=0 \Rightarrow 0=1,\\
&<_*(d,a)=0 \Rightarrow 0=1
\end{aligned}
$$

- Reduction of sequents to equalities

$$
\begin{aligned}
&g_1(<_*(a,b), <_*(b,c)) = <_*(a,c), \quad g_1(0,0)=0\\
&h_2(<_*(a,b), <_*(b,a)) = h_2(0,0),\\
&<_*(b,c)=0, \quad b=d,\\
&g_5(<_*(a,c))=0, \quad g_5(0)=1,\\
&g_6(<_*(d,a))=0, \quad g_6(0)=1
\end{aligned}
$$

An Example
Part 2 of 2

- After transformation to CNF[1]

$$
\begin{aligned}
&<_*(a,b)=0, <_*(b,c)=0 \Rightarrow <_*(a,c)=0\\
&\Rightarrow <_*(a,b)=0, <_*(b,a)=0,\\
&\Rightarrow <_*(b,c)=0, \quad \Rightarrow b=d,\\
&<_*(a,c)=0 \Rightarrow 0=1,\\
&<_*(d,a)=0 \Rightarrow 0=1
\end{aligned}
$$

- Reduction of sequents to equalities

$$
\begin{aligned}
&g_1(<_*(a,b), <_*(b,c)) = <_*(a,c), \quad g_1(0,0)=0\\
&h_2(<_*(a,b), <_*(b,a)) = h_2(0,0),\\
&<_*(b,c)=0, \quad b=d,\\
&g_5(<_*(a,c))=0, \quad g_5(0)=1,\\
&g_6(<_*(d,a))=0, \quad g_6(0)=1
\end{aligned}
$$

Proof System and Algorithm

- The CA-proof system
  - C-rule

$$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \vDash_e \vec{s} = \vec{t}$$

  - A-rule

$$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$

    if there is an anticongruence symbol h such that $\Gamma \vDash_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$

- The CA-algorithm

- Branching causes exponential running time

Proof System and Algorithm

- The CA-proof system
  - C-rule

$$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \vDash_e \vec{s} = \vec{t}$$

  - A-rule

$$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$

    if there is an anticongruence symbol h such that $\Gamma \vDash_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$

- The CA-algorithm
  1. apply C-rule while possible (standard congruence closure)

- Branching causes exponential running time

Proof System and Algorithm

- The CA-proof system
  - *C*-rule

$$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \vDash_e \vec{s} = \vec{t}$$

  - *A*-rule

$$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$

  if there is an anticongruence symbol h such that $\Gamma \vDash_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$

- The CA-algorithm
  1. apply C-rule while possible (standard congruence closure)
  2. if $\Gamma \vDash_e 0 = 1$, return proved
  3. apply A-rule if possible

  4. otherwise return a counterexample

- Branching causes exponential running time

Proof System and Algorithm

- The CA-proof system
  - *C*-rule

$$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \models_e \vec{s} = \vec{t}$$

  - *A*-rule

$$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$

    if there is an anticongruence symbol h such that $\Gamma \models_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$
- The CA-algorithm
  1. apply C-rule while possible (standard congruence closure)
  2. if $\Gamma \models_e 0 = 1$, return proved
  3. apply A-rule if possible
  4. otherwise return a counterexample
- Branching causes exponential running time

Proof System and Algorithm

- The CA-proof system
  - *C*-rule

$$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \models_e \vec{s} = \vec{t}$$

  - *A*-rule

$$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$

    if there is an anticongruence symbol h such that
    $\Gamma \models_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$

- The CA-algorithm
  1. apply C-rule while possible (standard congruence closure)
  2. if $\Gamma \models_e 0 = 1$, return proved
  3. apply A-rule if possible
     - call the CA-algorithm recursively for each branch
  4. otherwise return a counterexample
- Branching causes exponential running time

Proof System and Algorithm

- The CA-proof system
  - *C*-rule
    $$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \vDash_e \vec{s} = \vec{t}$$
  - *A*-rule
    $$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$
    if there is an anticongruence symbol h such that
    $\Gamma \vDash_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$
- The CA-algorithm
  1. apply C-rule while possible (standard congruence closure)
  2. if $\Gamma \vDash_e 0 = 1$, return proved
  3. apply A-rule if possible
     1. call the CA-algorithm recursively for each branch
     2. return proved if all recursive calls returned proved
  4. otherwise return a counterexample
- Branching causes exponential running time

Proof System and Algorithm

- The CA-proof system
  - *C*-rule
    $$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \models_e \vec{s} = \vec{t}$$
  - *A*-rule
    $$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$
    if there is an anticongruence symbol h such that $\Gamma \models_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$
- The CA-algorithm
  1. apply C-rule while possible (standard congruence closure)
  2. if $\Gamma \models_e 0 = 1$, return proved
  3. apply A-rule if possible
     1. call the CA-algorithm recursively for each branch
     2. return proved if all recursive calls returned proved
  4. otherwise return a counterexample
- Branching causes exponential running time

Proof System and Algorithm

- The CA-proof system
  - *C*-rule
  $$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \vDash_e \vec{s} = \vec{t}$$
  - *A*-rule
  $$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$
    if there is an anticongruence symbol h such that $\Gamma \vDash_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$
- The CA-algorithm
  1. apply C-rule while possible (standard congruence closure)
  2. if $\Gamma \vDash_e 0 = 1$, return proved
  3. apply A-rule if possible
     1. call the CA-algorithm recursively for each branch
     2. return proved if all recursive calls returned proved
  4. otherwise return a counterexample
- Branching causes exponential running time

Proof System and Algorithm

- The CA-proof system
  - *C*-rule
    $$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \models_e \vec{s} = \vec{t}$$
  - *A*-rule
    $$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$
    if there is an anticongruence symbol h such that $\Gamma \models_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$
- The CA-algorithm
  1. apply C-rule while possible (standard congruence closure)
  2. if $\Gamma \models_e 0 = 1$, return proved
  3. apply A-rule if possible
     1. call the CA-algorithm recursively for each branch
     2. return proved if all recursive calls returned proved
  4. otherwise return a counterexample
- Branching causes exponential running time

Proof System and Algorithm

- The CA-proof system
  - C-rule

$$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \models_e \vec{s} = \vec{t}$$

  - A-rule

$$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$

    if there is an anticongruence symbol h such that $\Gamma \models_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$

- The CA-algorithm
  1. apply C-rule while possible (standard congruence closure)
  2. if $\Gamma \models_e 0 = 1$, return proved
  3. apply A-rule if possible
     1. call the CA-algorithm recursively for each branch
     2. return proved if all recursive calls returned proved
  4. otherwise return a counterexample

- Branching causes exponential running time

Proof System and Algorithm

- The CA-proof system
  - *C*-rule
    $$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \vDash_e \vec{s} = \vec{t}$$
  - *A*-rule
    $$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$
    if there is an anticongruence symbol h such that
    $\Gamma \vDash_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$
- The CA-algorithm
  1. apply C-rule while possible (standard congruence closure)
  2. if $\Gamma \vDash_e 0 = 1$, return proved
  3. apply A-rule if possible
     1. call the CA-algorithm recursively for each branch
     2. return proved if all recursive calls returned proved
  4. otherwise return a counterexample
- Branching causes exponential running time

Proof System and Algorithm

- The CA-proof system
  - *C*-rule
    $$\frac{f(\vec{s}) = f(\vec{t}), \Gamma}{\Gamma} \quad \text{if } \Gamma \models_e \vec{s} = \vec{t}$$
  - *A*-rule
    $$\frac{u_1 = v_1, \Gamma \mid \cdots \mid u_m = v_m, \Gamma}{\Gamma}$$
    if there is an anticongruence symbol h such that
    $\Gamma \models_e h(u_1, \ldots, u_m) = h(v_1, \ldots, v_m)$
- The CA-algorithm
  1. apply C-rule while possible (standard congruence closure)
  2. if $\Gamma \models_e 0 = 1$, return proved
  3. apply A-rule if possible
     1. call the CA-algorithm recursively for each branch
     2. return proved if all recursive calls returned proved
  4. otherwise return a counterexample
- Branching causes exponential running time, but. . .

Conclusion and Further Work

- Congruence closure can be more useful than it seems
- Can derive from Horn clauses
- Can be extended to deal with any propositional content of formulas by “symmetric” anticongruences
- Pilot implementation only
- Work in progress on adding formulas

$$\forall x(s_1 = t_1, \ldots, s_n = t_n \Rightarrow u_1 = v_1, \ldots, u_m = v_m)$$

Will enable automated use of most common axioms and lemmas

Conclusion and Further Work

- Congruence closure can be more useful than it seems
- Can derive from Horn clauses
- Can be extended to deal with any propositional content of formulas by "symmetric" anticongruences
- Pilot implementation only
- Work in progress on adding formulas

$$\forall x(s_1 = t_1, \ldots, s_n = t_n \Rightarrow u_1 = v_1, \ldots, u_m = v_m)$$

Will enable automated use of most common axioms and lemmas

Conclusion and Further Work

- Congruence closure can be more useful than it seems
- Can derive from Horn clauses
- Can be extended to deal with any propositional content of formulas by "symmetric" anticongruences
- Pilot implementation only
- Work in progress on adding formulas

$$\forall x(s_1 = t_1, \ldots, s_n = t_n \Rightarrow u_1 = v_1, \ldots, u_m = v_m)$$

Will enable automated use of most common axioms and lemmas

Conclusion and Further Work

- Congruence closure can be more useful than it seems
- Can derive from Horn clauses
- Can be extended to deal with any propositional content of formulas by "symmetric" anticongruences
- Pilot implementation only
- Work in progress on adding formulas

$$\forall x(s_1 = t_1, \dots, s_n = t_n \Rightarrow u_1 = v_1, \dots, u_m = v_m)$$

Will enable automated use of most common axioms and lemmas

Conclusion and Further Work

- Congruence closure can be more useful than it seems
- Can derive from Horn clauses
- Can be extended to deal with any propositional content of formulas by "symmetric" anticongruences
- Pilot implementation only
- Work in progress on adding formulas

$$\forall x(s_1 = t_1, \ldots, s_n = t_n \Rightarrow u_1 = v_1, \ldots, u_m = v_m)$$

Will enable automated use of most common axioms and lemmas

Thank you!