

Handling Integer Arithmetic and Quantifiers in a Sequent Calculus

Philipp Rümmer
Chalmers University of Technology, Gothenburg
philipp@chalmers.se

Deduction and Decision Procedures
4th October 2007

Part 1: Ground sequent calculus for integer arithmetic:

- Complete for linear arithmetic
- Incomplete handling of nonlinear polynomial arithmetic
- Complete for producing counterexamples

Verify '07: *“A Sequent Calculus for Integer Arithmetic with Counterexample Generation”*

Part 2: Adding quantifiers and other theories:

- (Work in progress)

Context: KeY Tool, full functional verification

- Based on dynamic logic for Java
- Symbolic program execution + first-order reasoning
- *Interactive + automated reasoning*

Scenario 1: Simplification during Prog. Verification

Permanent “shallow” reasoning:

- Simplify arithmetic expressions
- Handle linear equations and inequalities
- Afterwards: proof is continued interactively
- Should always terminate, not cause proof splitting
- Human-readable reasoning

Scenario 1: Simplification during Prog. Verification

Permanent “shallow” reasoning:

- Simplify arithmetic expressions
- Handle linear equations and inequalities
- Afterwards: proof is continued interactively
- Should always terminate, not cause proof splitting
- Human-readable reasoning

Example:

```
a != null, (i-1) >= 0, i < a.length  
==>  
\<{ a[i-1] = 5; }\> true
```

Scenario 1: Simplification during Prog. Verification

Permanent “shallow” reasoning:

- Simplify arithmetic expressions
- Handle linear equations and inequalities
- Afterwards: proof is continued interactively
- Should always terminate, not cause proof splitting
- Human-readable reasoning

Example: [Simplification of path predicate](#)

```
a != null, (i-1) >= 0, i < a.length  
==>  
\<{ a[i-1] = 5; }\> true
```

Scenario 1: Simplification during Prog. Verification

Permanent “shallow” reasoning:

- Simplify arithmetic expressions
- Handle linear equations and inequalities
- Afterwards: proof is continued interactively
- Should always terminate, not cause proof splitting
- Human-readable reasoning

Example: [Simplification of path predicate](#)

```
a != null, i >= 1, i <= a.length-1  
==>  
\<{ a[i-1] = 5; }\> true
```

Scenario 1: Simplification during Prog. Verification

Permanent “shallow” reasoning:

- Simplify arithmetic expressions
- Handle linear equations and inequalities
- Afterwards: proof is continued interactively
- Should always terminate, not cause proof splitting
- Human-readable reasoning

Example: [Derivation of side-conditions](#)

```
a != null, i >= 1, i <= a.length-1
==>
\<{ a[i-1] = 5; }\> true
```

Scenario 1: Simplification during Prog. Verification

Permanent “shallow” reasoning:

- Simplify arithmetic expressions
- Handle linear equations and inequalities
- Afterwards: proof is continued interactively
- Should always terminate, not cause proof splitting
- Human-readable reasoning

Example: [Derivation of side-conditions](#)

```
a != null, i >= 1, i <= a.length-1
==>
\<{ a[i-1] = 5; }\> true
```

No exceptions, because:

```
i >= 1, i <= a.length-1 ==> 0 <= i-1 < a.length
```

Scenario 2: Handling more complex problems

- Meta-reasoning, verification of rules
- Programs with nonlinear arithmetic
- Often: with user guidance

Scenario 2: Handling more complex problems

- Meta-reasoning, verification of rules
- Programs with nonlinear arithmetic
- Often: with user guidance

Example: **Derive laws of integer division**

$$c > 0 \ \& \ b \neq 0 \implies a / (b * c) = (a / c) / b$$

Scenario 2: Handling more complex problems

- Meta-reasoning, verification of rules
- Programs with nonlinear arithmetic
- Often: with user guidance

Example: **Verify gaussian sum** (invariant has to be provided)

$$c > 0 \ \& \ b \neq 0 \implies a/(b*c) = (a/c)/b$$

$$i = 0, \ s = 0, \ n \geq 0$$

\implies

$\{$

 while (i < n) {

 s = s + i;

 i = i + 1;

 }

$\} \ s = n * (n-1) / 2$

$$\sum_{i=0}^{n-1} i = \frac{n \cdot (n-1)}{2}$$

Reasoner?

- External theorem provers/SMT solvers?
- Computer algebra systems?
- Built-in procedures?

Reasoner?

- External theorem provers/SMT solvers?
- Computer algebra systems?
- Built-in procedures?

- External theorem provers/SMT solvers?
- Computer algebra systems?
- Built-in procedures?

Programming KeY:

- Taclets: language for specifying new calculus rules
- Proof strategies control the rule applications (instead of tactics)
- “LCF approach”

- Calculus shown here really consists of ≈ 110 taclets
- All taclets are verified and sound

Simplification of Terms and Formulas

Expansion of Polynomials

Polynomials are fully expanded, terms are sorted:

$$\begin{aligned} & (a + b) * (c - d + 1) \\ & = a + b + c*a + c*b + d*a*-1 + d*b*-1 \end{aligned}$$

⇒ Realised as rewriting system

Used orderings:

- Lexicographic path ordering on terms
- Graded lexicographic ordering on monomials

Simplify Equations and Inequalities

- Only the relations \leq , $=$, \geq are used
- All inequalities are moved to antecedent
- Separate greatest monomial in each formula:

$$(a + b) * (c - d + 1) \geq 0$$

\Leftrightarrow

$$d*b \geq a + b + c*a + c*b + d*a*-1$$

- Common factors are eliminated, rounding:

$$10*b = 15*a \quad \Leftrightarrow \quad 2*b = 3*a$$

$$2*a = 3 \quad \Leftrightarrow \quad \text{false}$$

$$7*a \geq 3 \quad \Leftrightarrow \quad a \geq 1$$

Linear Integer Arithmetic

Sequent Calculus for Linear Arithmetic

Linear Equations	Linear Inequalities
Gaussian Elimination + Euclidian Algorithm	Fourier-Motzkin Elimination + Case Analysis

- Complete for linear integer arithmetic
- Complete for producing counterexamples
- Case analysis is disabled by default
⇒ Incompleteness is not an issue in practice

Calculus Rules

Linear equations:

$$\frac{\Gamma, s = t \vdash \phi[s' + u \cdot (s - t)], \Delta}{\Gamma, s = t \vdash \phi[s'], \Delta} \text{ RED}$$

$$\frac{\Gamma, \alpha \cdot (u + x') = s, x = u + x' \vdash \Delta}{\Gamma, \alpha x = s \vdash \Delta} \text{ COL-RED}$$

$$\frac{\Gamma \vdash s \leq t, \Delta \quad \Gamma \vdash s \geq t, \Delta}{\Gamma \vdash s = t, \Delta} \text{ SPLIT-EQ}$$

Linear inequalities:

$$\frac{\Gamma, \alpha s \geq t, \beta s \leq t', \beta t \leq \alpha t' \vdash \Delta}{\Gamma, \alpha s \geq t, \beta s \leq t' \vdash \Delta} \text{ FM-ELIM}$$

if $\alpha > 0, \beta > 0$

$$\frac{\Gamma, s < t \vdash \Delta \quad \Gamma, s = t \vdash \Delta}{\Gamma, s \leq t \vdash \Delta} \text{ STRENGTHEN}$$

Nonlinear Integer Arithmetic

Sequent Calculus for Nonlinear Arithmetic

Nonlinear Equations	Nonlinear Inequalities
Gröbner Bases (Buchberger's algorithm)	Cross-Multiplication + Case Analysis

- Incomplete method for proving validity
- Complete for producing counterexamples
- Cross-Multiplication, case analysis disabled by default

Calculus Rules (Equations)

Reduction with pseudo-division:

$$\frac{\Gamma, \alpha s = t \vdash \phi[u \cdot t = \alpha t'], \Delta}{\Gamma, \alpha s = t \vdash \phi[s' = t'], \Delta} \text{ PSEUDO-RED} \quad \text{if } s' = u \cdot s$$

Completion with S-polynomials:

$$\frac{\Gamma, s = t, s' = t', s'_r \cdot t = s_r \cdot t' \vdash \Delta}{\Gamma, s = t, s' = t' \vdash \Delta} \text{ S-POLY} \quad \begin{array}{l} s = \text{gcd}(s, s') \cdot s_r, \\ s' = \text{gcd}(s, s') \cdot s'_r \end{array}$$

Calculus Rules (Inequalities)

Cross-multiplication:

$$\frac{\Gamma, s \leq t, s' \leq t', 0 \leq (t - s) \cdot (t' - s') \vdash \Delta}{\Gamma, s \leq t, s' \leq t' \vdash \Delta} \text{ CROSS-MULT}$$

Case splits:

$$\frac{\Gamma, x < 0 \vdash \Delta \quad \Gamma, x = 0 \vdash \Delta \quad \Gamma, x > 0 \vdash \Delta}{\Gamma \vdash \Delta} \text{ SIGN-CASES}$$

$$\frac{\Gamma, s < t \vdash \Delta \quad \Gamma, s = t \vdash \Delta}{\Gamma, s \leq t \vdash \Delta} \text{ STRENGTHEN}$$

⇒ With many further lemmas and heuristics

Example: Nonlinear Inequalities

$$x \geq 0, x \leq 10 \implies x^2 \leq 101$$

Example: Nonlinear Inequalities

Proof by contradiction

$$x \geq 0, x \leq 10 \implies x^2 \leq 101$$

$$x \geq 0, x \leq 10, x^2 \geq 102 \implies$$

Example: Nonlinear Inequalities

Cross-multiplication (CROSS-MULT)

$$x \geq 0, x \leq 10 \implies x^2 \leq 101$$

$$x \geq 0, x \leq 10, x^2 \geq 102 \implies$$

$$(x-0)(10-x) \geq 0, x \geq 0, x \leq 10, x^2 \geq 102 \implies$$

Example: Nonlinear Inequalities

Cross-multiplication (CROSS-MULT)

$$x \geq 0, x \leq 10 \implies x^2 \leq 101$$

$$x \geq 0, x \leq 10, x^2 \geq 102 \implies$$

$$x^2 \leq 10x, x \geq 0, x \leq 10, x^2 \geq 102 \implies$$

Example: Nonlinear Inequalities

Fourier-Motzkin variable elimination (FM-ELIM)

$$x \geq 0, x \leq 10 \implies x^2 \leq 101$$

$$x \geq 0, x \leq 10, x^2 \geq 102 \implies$$

$$x^2 \leq 10x, x \geq 0, x \leq 10, x^2 \geq 102 \implies$$

$$102 \leq 10x, x^2 \leq 10x, x \geq 0, x \leq 10, \\ x^2 \geq 102 \implies$$

Example: Nonlinear Inequalities

Fourier-Motzkin variable elimination (FM-ELIM)

$$x \geq 0, x \leq 10 \implies x^2 \leq 101$$

$$x \geq 0, x \leq 10, x^2 \geq 102 \implies$$

$$x^2 \leq 10x, x \geq 0, x \leq 10, x^2 \geq 102 \implies$$

$$x \geq 11, x^2 \leq 10x, x \geq 0, x \leq 10, \\ x^2 \geq 102 \implies$$

Example: Nonlinear Inequalities

$x \geq 0, x \leq 10 \implies x^2 \leq 101$

$x \geq 0, x \leq 10, x^2 \geq 102 \implies$

$x^2 \leq 10x, x \geq 0, x \leq 10, x^2 \geq 102 \implies$

$x \geq 11, x^2 \leq 10x, x \geq 0, x \leq 10,$
 $x^2 \geq 102 \implies$

closed

Counterexamples as “Obviously” Unprovable Goals

Example:

$$\implies x * x \geq 3$$

eventually leads to the goals (search terminates):

$$x = -1 \implies$$

$$x = 0 \implies$$

$$x = 1 \implies$$

Side-effect of splitting rules SIGN-CASES, STRENGTHEN:

- Possible models are enumerated
- In practice: also other rules derive bounds for variables

Free Variables, Quantifiers and Further Theories

(Work in Progress)

Beyond Ground Arithmetic (Wish List)

First-order arithmetic:

- Explicit quantifiers
- Full Presburger-Skolem arithmetic

(Rigid) free variables over integers:

- Program dis-verification:
Search for program inputs by constraint solving
- Non-termination-checking

Integration with first-order reasoning:

- Uninterpreted functions, free variables, unification

Further theories:

- In particular lists

First-Order Arithmetic through Quantifier Elimination

Ground calculus works by eliminating big symbols:

- Linear part resembles Omega test
- Similar for nonlinear part (but of course incomplete)

⇒ Almost quantifier elimination

First-Order Arithmetic through Quantifier Elimination

Ground calculus works by eliminating big symbols:

- Linear part resembles Omega test
- Similar for nonlinear part (but of course incomplete)

⇒ Almost quantifier elimination

$$\frac{}{\frac{}{\frac{}{\vdash \forall x \dots \exists y \dots \forall z \dots \phi}}}}$$

First-Order Arithmetic through Quantifier Elimination

Ground calculus works by eliminating big symbols:

- Linear part resembles Omega test
- Similar for nonlinear part (but of course incomplete)

⇒ Almost quantifier elimination: **Skolemisation, free variables**

$$\frac{\vdash [x/c] \dots [y/Y] \dots [z/f(Y)] \dots \phi}{\vdash \forall x \dots \exists y \dots \forall z \dots \phi}$$

Term ordering: $c \prec Y \prec f(Y)$

First-Order Arithmetic through Quantifier Elimination

Ground calculus works by eliminating big symbols:

- Linear part resembles Omega test
- Similar for nonlinear part (but of course incomplete)

⇒ Almost quantifier elimination: [Arithmetic reasoning](#)

$$\frac{\vdots}{\frac{\vdash [x/c] \dots [y/Y] \dots [z/f(Y)] \dots \phi}{\vdash \forall x \dots \exists y \dots \forall z \dots \phi}}$$

Term ordering: $c \prec Y \prec f(Y)$

First-Order Arithmetic through Quantifier Elimination

Ground calculus works by eliminating big symbols:

- Linear part resembles Omega test
- Similar for nonlinear part (but of course incomplete)

⇒ Almost quantifier elimination: **Arithmetic reasoning ...**

$$\frac{\begin{array}{c} \vdash \psi_1, \dots \quad \vdash \psi_2, \dots \quad \dots \quad \vdash \psi_n, \dots \\ \hline \vdots \\ \hline \vdash [x/c] \dots [y/Y] \dots [z/f(Y)] \dots \phi \\ \hline \vdash \forall x \dots \exists y \dots \forall z \dots \phi \end{array}}{\vdash \forall x \dots \exists y \dots \forall z \dots \phi}$$

Term ordering: $c \prec Y \prec f(Y)$

First-Order Arithmetic through Quantifier Elimination

Ground calculus works by eliminating big symbols:

- Linear part resembles Omega test
- Similar for nonlinear part (but of course incomplete)

⇒ Almost quantifier elimination: **Collect non- $f(Y)$ -formulas**

$$\frac{\begin{array}{c} \vdash \psi_1, \dots \quad \vdash \psi_2, \dots \quad \dots \quad \vdash \psi_n, \dots \\ \vdots \\ \vdash [x/c] \dots [y/Y] \dots [z/f(Y)] \dots \phi \end{array}}{\vdash \forall x \dots \exists y \dots \forall z \dots \phi}$$

Term ordering: $c \prec Y \prec f(Y)$

First-Order Arithmetic through Quantifier Elimination

Ground calculus works by eliminating big symbols:

- Linear part resembles Omega test
- Similar for nonlinear part (but of course incomplete)

⇒ Almost quantifier elimination

$$\frac{\begin{array}{c} \vdash \psi_1, \dots \quad \vdash \psi_2, \dots \quad \dots \quad \vdash \psi_n, \dots \\ \vdots \\ \vdash [x/c] \dots [y/Y] \dots [z/f(Y)] \dots \phi \end{array}}{\vdash \forall x \dots \exists y \dots \forall z \dots \phi}$$

Term ordering: $c \prec Y \prec f(Y)$

- $\forall x \dots \exists y \dots (\psi_1 \wedge \dots \wedge \psi_n)$ is result of eliminating $\forall z$
- Iterate process to eliminate all quantifiers

More Formally: Constrained Sequents

$$\Gamma \vdash \Delta \Downarrow C$$

Γ, Δ ... sets of formulas

C ... formula (constraint)

Definition

$\Gamma \vdash \Delta \Downarrow C$ is valid iff

$$val(C) = tt \Rightarrow val(\Gamma \vdash \Delta) = tt$$

- Practically: constraints are filled in afterwards
- Generalisation of Giese's incremental closure method
 \Rightarrow First-order reasoning

Some Structural Calculus Rules

$$\frac{\Gamma \vdash \Delta \Downarrow C \quad \vdash C}{\Gamma \vdash \Delta} \text{ C-CUT}$$

$$\frac{*}{\Gamma, \phi_1, \dots \vdash \psi_1, \dots, \Delta \Downarrow (\neg\phi_1 \vee \dots \vee \psi_1 \vee \dots)} \text{ C-CLOSE}$$

$$\frac{\Gamma \vdash \phi, \Delta \Downarrow C \quad \Gamma \vdash \psi, \Delta \Downarrow D}{\Gamma \vdash \phi \wedge \psi, \Delta \Downarrow C \wedge D} \text{ AND-RIGHT}$$

To emulate FO reasoning with incremental closure:

$$\frac{*}{\Gamma, p(\bar{s}) \vdash p(\bar{t}), \Delta \Downarrow \bar{s} = \bar{t}} \text{ UNIFY}$$

Strict Iterated Application

$$\overline{\Gamma \vdash \Delta \Downarrow ?}$$

Strict Iterated Application

Handle programs, innermost quantifiers, uninterpreted funs.

$$\frac{\vdots}{\Gamma \vdash \Delta \Downarrow C_1}$$

Strict Iterated Application

Handle programs, innermost quantifiers, uninterpreted funs.

$$\frac{\vdots}{\Gamma \vdash \Delta \Downarrow C_1} \rightarrow$$

Strict Iterated Application

Handle arithmetic quantifiers

$$\frac{\Gamma \vdash \Delta \downarrow C_1}{\vdots} \rightarrow \overline{C_1 \vdash \downarrow}$$

Strict Iterated Application

Handle arithmetic quantifiers

$$\frac{\vdots}{\Gamma \vdash \Delta \Downarrow C_1} \rightarrow \frac{\vdots}{C_1 \vdash \Downarrow C_2}$$

Strict Iterated Application

Handle arithmetic quantifiers

$$\frac{\vdots}{\Gamma \vdash \Delta \Downarrow C_1} \rightarrow \frac{\vdots}{C_1 \vdash \Downarrow C_2} \rightarrow \dots$$

Strict Iterated Application

Handle arithmetic quantifiers

$$\frac{\vdots}{\Gamma \vdash \Delta \Downarrow C_1} \rightarrow \frac{\vdots}{C_1 \vdash \Downarrow C_2} \rightarrow \dots \rightarrow C_n \text{ ground}$$

Strict Iterated Application

Handle arithmetic quantifiers

$$\frac{\vdots}{\Gamma \vdash \Delta \Downarrow C_1} \rightarrow \frac{\vdots}{C_1 \vdash \Downarrow C_2} \rightarrow \dots \rightarrow C_n \text{ ground}$$

- Possible: external reasoning about constraints
- In practice: reason about partial constraints
⇒ Refinements of Giese's method apply directly
- Two different calculi: one is sound, one is complete

Related Approaches

Ground arithmetic handling in ACL2

- Also uses Fourier-Motzkin, Cross-multiplication,
- No case splits (?)

Cyrluk, Kapur, 1989: *“Reasoning about Nonlinear Inequality Constraints: A Multi-level Approach”*

Platzer, 2007: *“Combining deduction and algebraic constraints for hybrid system analysis”*

- Side-derivations instead of constraints
- Quantifier elimination done in Mathematica

Korovin, Voronkov, 2007: *“Integrating Linear Arithmetic into Superposition Calculus”*

- First-order logic + linear rational arithmetic
- No arithmetic quantifier elimination

SMT solvers, higher-order proof assistants . . .

- Finish [Part 2](#)
- Fragment of FOL + arithmetic that can be handled?
- Support further theories + operations relevant to Java:
Lists, exponentiation, bitwise operations
- Standalone implementation of the calculus
DPLL(T)?