Handling Integer Arithmetic in KeY

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

14th June 2007
This is about methods for ground problems in integer arithmetic built into KeY:

- Simplification heuristics
- Linear arithmetic
- Nonlinear polynomial arithmetic

Short history:

- Development started in the end of 2005
  ⇒ Support for induction proofs
- Before that: Simplify and ICS to handle arithmetic
  (+ purely interactive reasoning)
- Everything is implemented and on main branch

“A Sequent Calculus for Integer Arithmetic with Counterexample Generation,” Verify 2007
Wish List for Integer Arithmetic Support

Integrate automated and interactive proving:
- Readable (history of) proof goals
- Terminating automated methods (that don’t cause splitting)

Construct counterexamples for invalid formulas:
- Important e.g. for induction/invariant proofs

Efficiently handle different integer semantics for Java:
- Idealised, mathematical integers
- Machine integers (modulo arithmetic)
- Mathematical integers + overflow checks

Nontrivial programs + specifications:
- (Nonlinear) arithmetic, bitwise operations, quantifiers

Support for metavariables:
- Quantifier handling, model construction, disproving
External theorem provers?
Computer algebra systems?
Built-in procedures?
Not Really Solvable . . .

- External theorem provers?
- Computer algebra systems?
- Built-in procedures?

- Different algebra algorithms as sequent calculi
- Implemented as taclets and KeY proof strategy
  \( \approx 110 \) taclets, part of JavaDL Strategy)
- All taclets are verified using the KeY lemma mechanism
Levels of Integer Theories in KeY

Actual machine operations: +, −, *, /, %, «, &, ==, <=, etc
(addJint,...)
- Many operations with (broken) modulo semantics
- No reasoning on this level

Elementary mathematical operations: +, *, /, %, =, <=, >=
- Polynomial arithmetic + division with remainder
- Normal mathematical semantics
- Simplification of expressions on this level

Pure polynomial arithmetic: +, *, =, <=, >=
- Real reasoning is done here
Simplification of Terms and Formulas
Polynomials are fully expanded, terms are sorted:

\[(a + b) \times (c - d + 1)\]
\[= a + b + c\times a + c\times b + d\times a\times -1 + d\times b\times -1\]

Used orderings:
- Lexicographic path ordering on terms
- Graded lexicographic ordering on monomials
Basically polynomial division:

\[(a \times 3) / 2 = a + a / 2\]

\[(a \% 10 + b + 8) \% 5 = (a + b - 2) \% 5\]

\[
\text{addJint}(\text{mulJint}(a, b), c) = \ldots = \text{addJint}(a \times b, c)
\]

⇒ Simple, but extremely useful to handle machine integers
Only the relations <=, =, >= are used
All inequalities are moved to antecedent

Greatest monomial in each formula is moved to left side:
\[(a + b) \times (c - d + 1) \geq 0\]
\[\iff\]
d*b \geq a + b + c*a + c*b + d*a*-1

Common factors are eliminated, rounding appropriately:
\[10*b = 15*a \iff 2*b = 3*a\]
\[2*a = 3 \iff \text{false}\]
\[7*a \geq 3 \iff a \geq 1\]
Linear Integer Arithmetic
<table>
<thead>
<tr>
<th>Linear Equations</th>
<th>Linear Inequalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian Elimination</td>
<td>Fourier-Motzkin Elimination</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Euclidian Algorithm</td>
<td>Case Analysis</td>
</tr>
</tbody>
</table>

- Complete for linear integer arithmetic
- Complete for producing counterexamples
Examples

Solves systems of equations:

\[-5x_1 - 2x_2 + x_3 - x_4 + x_5 = 0 \quad \&\]
\[9x_1 + 62x_2 - 5x_3 - 3x_4 + 101x_5 = 0 \quad \&\]
\[56x_1 - 34x_2 - 11x_3 + 67x_4 - 98x_5 = 0\]

Solutions are:

\[x_1 = l_4 \times -74 + l_3 \times 72,\]
\[x_2 = l_4 \times -133 + l_3 \times 94,\]
\[x_3 = l_4 \times -740 + l_3 \times 623,\]
\[x_4 = l_4 \times -54 + l_3 \times 43,\]
\[x_5 = l_4 \times 50 + l_3 \times -32\]
Examples

Solves systems of equations:

\[-5x_1 - 2x_2 + x_3 - x_4 + x_5 = 0 \quad \& \quad 9x_1 + 62x_2 - 5x_3 - 3x_4 + 101x_5 = 0 \quad \& \quad 56x_1 - 34x_2 - 11x_3 + 67x_4 - 98x_5 = 0\]

\[\rightarrow \text{false}\]

Counterexamples are:

\[x_1 = l_4 \times -74 + l_3 \times 72,\]
\[x_2 = l_4 \times -133 + l_3 \times 94,\]
\[x_3 = l_4 \times -740 + l_3 \times 623,\]
\[x_4 = l_4 \times -54 + l_3 \times 43,\]
\[x_5 = l_4 \times 50 + l_3 \times -32.\]
Examples (2)

Proves that inequalities are contradictory:

\[
a + b \leq 5 \land a \geq 0 \land a - 2 \cdot b \leq -20 \\
\rightarrow \\
false
\]
Examples (3)

Proves the following formula:
(with machine integers)

\[
\text{inInt}(\text{start}) \& \text{inInt}(\text{end}) \\
\rightarrow
\]

\[
\langle \text{middle} = (\text{start} + \text{end}) / 2; \rangle \\
(\text{start} \leq \text{middle} \& \text{middle} \leq \text{end} \\
| \text{end} \leq \text{middle} \& \text{middle} \leq \text{start})
\]
Examples (3)

Produces counterexamples for the following formula:
(with machine integers)

\[
in\text{Int}(\text{start}) \ & \ in\text{Int}(\text{end})
\rightarrow
\]

\[
\langle \ \text{middle} = ( \ \text{start} + \text{end} \ ) / 2; \ \rangle
\]

( \text{start} \leq \text{middle} \ & \ \text{middle} \leq \text{end}
| \ \text{end} \leq \text{middle} \ & \ \text{middle} \leq \text{start} )
Examples (3)

Produces counterexamples for the following formula:
(with machine integers)

\[
\text{inInt}(\text{start}) \& \text{inInt}(\text{end}) \\
\rightarrow \\
\langle \text{middle} = (\text{start} + \text{end}) / 2; \rangle \\
\quad (\text{start} \leq \text{middle} \& \text{middle} \leq \text{end} \\
\quad \quad | \text{end} \leq \text{middle} \& \text{middle} \leq \text{start} )
\]

\begin{align*}
\text{start} &= 2147483647, \text{end} = 1 \\
\text{start} &= 2147483647, \text{end} = 2147483646 \\
\text{start} &= -2147483648, \text{end} = -3 \\
\text{...}
\end{align*}
public void add(short e, short f) {
    intPart += e;

    if (intPart > 0 && decPart < 0) {
        intPart--;
        decPart = (short)(decPart + PRECISION);
    } else if (intPart < 0 && decPart > 0) {
        intPart++;
        decPart = (short)(decPart - PRECISION);
    }

    decPart += f;
    if (intPart > 0 && decPart < 0) {
        intPart--;
        decPart = (short)(decPart + PRECISION);
    } else if (intPart < 0 && decPart > 0) {
        intPart++;
        decPart = (short)(decPart - PRECISION);
    } else {
        short retenue = 0;
        short signe = 1;
        if (decPart < 0) {
            signe = -1;
            decPart = (short)(-decPart);
        }
        retenue = (short)(decPart / PRECISION);
        decPart = (short)(decPart % PRECISION);
        retenue *= signe;
        decPart *= signe;
        intPart += retenue;
    }
}
Case Splits are Disabled by Default

\[
\Gamma, \ s < t \vdash \Delta \quad \Gamma, \ s = t \vdash \Delta \\
\Gamma, \ s \leq t \vdash \Delta \quad \text{STRENGTHEN}
\]

- Proof splitting is unpopular
- Rule destroys termination
- Incompleteness is not an issue in practice
- But: case splits allow to construct counterexamples

⇒ Can be switched on with option “Model search”
Nonlinear Integer Arithmetic
Sequent Calculus for Nonlinear Arithmetic

<table>
<thead>
<tr>
<th>Nonlinear Equations</th>
<th>Nonlinear Inequalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gröbner Bases</td>
<td>Cross-Multiplication</td>
</tr>
<tr>
<td></td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Case Analysis</td>
</tr>
</tbody>
</table>

- Incomplete method for proving validity
- Complete for producing counterexamples
- Cross-Multiplication, case analysis disabled by default
- Procedures have so far mainly been useful to verify rules
Examples

Proves formulas like:

\[ a \times b = 1 \iff (a = b \land (a = 1 \lor a = -1)) \]

\[ a^{11} \geq 1000 \iff a > 1 \]

\[ c > 0 \land b \neq 0 \implies a / (b \times c) = (a / c) / b \]
Also proves the following formula:
(with machine integers)

\[
a \neq \text{null} \land a\.length \geq 100 \land x \geq 0 \land x \leq 9
\rightarrow
\langle y = a[x^2]; \rangle \text{ true}
\]
Examples (3)

Produces counterexample for the following formula: (with machine integers)

\[
\begin{align*}
  a &\ne \text{null} \& a.\text{length} \geq 100 \& \\
  x &\geq 0 \& x \leq 10 \\
  \rightarrow \\
  \langle y = a[x*x]; \rangle \text{ true}
\end{align*}
\]

The (only) counterexample is:

\[
\begin{align*}
  a.\text{length} &= 100, \\
  x &= 10
\end{align*}
\]
Calculus for Nonlinear Inequalities

Cross-multiplication (linear approximations of nonlinear terms):

\[
\Gamma, \ s \leq t, \ s' \leq t', \ 0 \leq (t - s) \cdot (t' - s') \vdash \Delta
\]

\[
\Gamma, \ s \leq t, \ s' \leq t' \vdash \Delta \quad \text{CROSS-MULT}
\]

Case splits:

\[
\Gamma, \ x < 0 \vdash \Delta \quad \Gamma, \ x = 0 \vdash \Delta \quad \Gamma, \ x > 0 \vdash \Delta
\]

\[
\Gamma \vdash \Delta \quad \text{SIGN-CASES}
\]

\[
\Gamma, \ s < t \vdash \Delta \quad \Gamma, \ s = t \vdash \Delta
\]

\[
\Gamma, \ s \leq t \vdash \Delta \quad \text{STRENGTHEN}
\]

⇒ With many further lemmas and heuristics

⇒ Similar method is implemented in ACL2
Evaluation ...
Usefulness of the Methods

Gaussian Elimination + Euclidian Algorithm:
- Essential, quite useful to handle machine integers
- Performance is more than sufficient

Fourier-Motzkin Elimination:
- Essential
- Performance is mostly sufficient

Gröbner Bases:
- Still searching for an application

Cross-Multiplication + Case Analysis:
- Important for meta-reasoning, “mathematical” programs
- But: does not scale very well
Automated+interactive proving, readable proof goals, etc:
- Much better than 1 year ago
- Proofs are too long, too many irrelevant steps are shown

Construct counterexamples for invalid formulas:
- Works for many interesting cases, but could scale better
- Often requires user guidance

Efficiently handle different integer semantics:
- Mostly solved; remaining show-stoppers are elsewhere

Verify nontrivial programs+specifications:
- Quite good handling of many arithmetic operations
- Bitwise operations are basically not supported
- Quantifier handling got better, but not good enough

Support for metavariables:
- Happens to work quite well, but to be investigated in detail
Future Work

- Add quantifier handling with metavariables and constraints (Goal: complete calculus for Presburger arithmetic)
- Standalone implementation of the calculus → External search with proof generation (based on DPLL(T) framework?)
- Bitwise operations