Handling Integer Arithmetic in KeY

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

14th June 2007

Context, Outline

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
  $\Rightarrow$ 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

Not Really Solvable . . .

- 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

Expansion of Polynomials

Polynomials are fully expanded, terms are sorted:

```
(a + b) * (c - d + 1)
= a + b + c*a + c*b + d*a*-1 + d*b*-1
```

Used orderings:

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

Simplify Fractions and Modulo-Expressions

Basically polynomial division:

```
(a*3)/2 = a + a/2

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

addJint(mulJint(a, b), c)
   = ... = addJint(a*b, c)
```

$\Rightarrow$ Simple, but extremely useful to handle machine integers

Simplify Equations and Inequalities

- Only the relations <=, =, >= are used
- All inequalities are moved to antecedent

- Greatest monomial in each formula is moved to left side:

```
(a + b) * (c - d + 1) >= 0
<=>
d*b >= a + b + c*a + c*b + d*a*-1
```

- Common factors are eliminated, rounding appropriately:

```
10*b = 15*a   <=>  2*b = 3*a
 2*a = 3      <=>  false
 7*a >= 3     <=>  a >= 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

Examples

Solves systems of equations:

```
-5*x1 -  2*x2 +    x3 -    x4 +     x5 = 0 &
 9*x1 + 62*x2 -  5*x3 -  3*x4 + 101*x5 = 0 &
56*x1 - 34*x2 - 11*x3 + 67*x4 -  98*x5 = 0
```

Solutions are:

```
x1 = l_4 * -74  + l_3 * 72,
x2 = l_4 * -133 + l_3 * 94,
x3 = l_4 * -740 + l_3 * 623,
x4 = l_4 * -54  + l_3 * 43,
x5 = l_4 * 50   + l_3 * -32
```

Examples

Solves systems of equations:

```
  -5*x1 -  2*x2 +    x3 -    x4 +     x5 = 0 &
   9*x1 + 62*x2 -  5*x3 -  3*x4 + 101*x5 = 0 &
  56*x1 - 34*x2 - 11*x3 + 67*x4 -  98*x5 = 0
-> false
```

Counterexamples are:

```
  x1 = l_4 * -74  + l_3 * 72,
  x2 = l_4 * -133 + l_3 * 94,
  x3 = l_4 * -740 + l_3 * 623,
  x4 = l_4 * -54  + l_3 * 43,
  x5 = l_4 * 50   + l_3 * -32
```

Examples (2)

Proves that inequalities are contradictory:

```
a + b <= 5 & a >= 0 & a - 2 * b <= -20
->
false
```

Examples (3)

Proves the following formula:
(with machine integers)

```
  inInt(start) & inInt(end)
->

  \< middle = ( start + end ) / 2; \>
        (  start <= middle & middle <= end
         | end <= middle & middle <= start )
```

Examples (3)

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

```
  inInt(start) & inInt(end)
->

  \< middle = ( start + end ) / 2; \>
        (  start <= middle & middle <= end
         | end <= middle & middle <= start )
```

Examples (3)

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

```
  inInt(start) & inInt(end)
->

  \< middle = ( start + end ) / 2; \>
        (  start <= middle & middle <= end
         | end <= middle & middle <= start )
```

```
  start = 2147483647, end = 1
  start = 2147483647, end = 2147483646
  start = -2147483648, end = -3
  ...
```

Gemplus Example

```
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;
  } }
```

- Addition in fixed-point arithmetic
- Automatically verified with machine integers
- Originally: verified with Loop by Cees-Bart Breunesse

Case Splits are Disabled by Default

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

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

$\Rightarrow$ Can be switched on with option “Model search”

Nonlinear Integer Arithmetic

Sequent Calculus for Nonlinear Arithmetic

Nonlinear Equations	Nonlinear Inequalities
Gröbner Bases	Cross-Multiplication + Case Analysis

- 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*b = 1  <->  (a = b & (a = 1 | a = -1))

a^11 >= 1000  <->  a > 1

c > 0 & b != 0  ->  a/(b*c) = (a/c)/b
```

Examples (2)

Also proves the following formula:
(with machine integers)

```
  a != null & a.length >= 100 &
  x >= 0 & x <= 9
->
  \< y = a[x*x]; \> true
```

Examples (3)

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

```
  a != null & a.length >= 100 &
  x >= 0 & x <= 10
->
  \< y = a[x*x]; \> true
```

The (only) counterexample is:

```
  a.length = 100,
  x = 10
```

Calculus for Nonlinear Inequalities

Cross-multiplication (linear approximations of nonlinear terms) :

$$\frac{\Gamma,\ s \le t,\ s' \le t',\ 0 \le (t - s) \cdot (t' - s')\ \vdash\ \Delta}{\Gamma,\ s \le t,\ s' \le t'\ \vdash\ \Delta}\ \textsc{cross-mult}$$

Case splits:

$$\frac{\Gamma,\ x < 0\ \vdash\ \Delta \quad \Gamma,\ x = 0\ \vdash\ \Delta \quad \Gamma,\ x > 0\ \vdash\ \Delta}{\Gamma\ \vdash\ \Delta}\ \textsc{sign-cases}$$

$$\frac{\Gamma,\ s < t\ \vdash\ \Delta \quad \Gamma,\ s = t\ \vdash\ \Delta}{\Gamma,\ s \le t\ \vdash\ \Delta}\ \textsc{strengthen}$$

$\Rightarrow$ With many further lemmas and heuristics

$\Rightarrow$ 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

Back to the Wish List

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
  $\Rightarrow$ External search with proof generation
  (based on DPLL(T) framework?)
- Bitwise operations