A Saturation-Based Automated Theorem Prover for RISCAL

Viktoria Langenreither

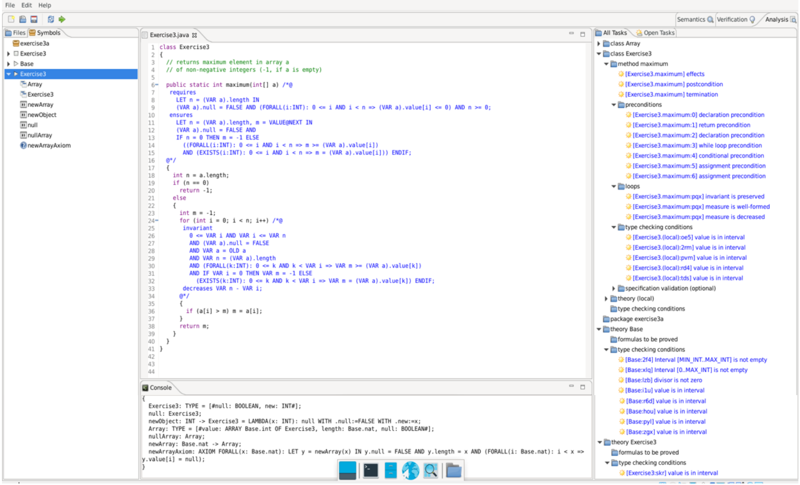
06.12.2022

Content

RISCAL

- RISC Algorithm Language
- consists of a specification language and an associated software system
- designed to simplify this process of specifying and verifying mathematical algorithms
- fully automatically check theorems and verification conditions
- all types are finite (upper bound specified by a model parameter)
- includes an interface to various SMT solvers
- linked to the RISCTP theorem proving interface to provide theorem proving capabilities

RISCAL


File Edit Help
Semantics
Verification
Analysis
Files
Symbols
exercise3a
Exercise3
Base
Exercise3
Array
Exercise3
newArray
newObject
null
nullArray
newArrayAxiom
Exercise3.java
class Exercise3
{
// returns maximum element in array a
// of non-negative integers (-1, if a is empty)
public static int maximum(int[] a) /*@
requires
LET n = (VAR a).length IN
(VAR a).null = FALSE AND (FORALL(i:INT): 0 <= i AND i < n => (VAR a).value[i] <= 0) AND n >= 0;
ensures
LET n = (VAR a).length, m = VALUE@NEXT IN
(VAR a).null = FALSE AND
IF n = 0 THEN m = -1 ELSE
((FORALL(i:INT): 0 <= i AND i < n => m >= (VAR a).value[i])
AND (EXISTS(i:INT): 0 <= i AND i < n => m = (VAR a).value[i])) ENDIF;
@*/
{
int n = a.length;
if (n == 0)
return -1;
else
{
int m = -1;
for (int i = 0; i < n; i++) /*@
invariant
0 <= VAR i AND VAR i <= VAR n
AND (VAR a).null = FALSE
AND VAR a = OLD a
AND VAR n = (VAR a).length
AND (FORALL(k:INT): 0 <= k AND k < VAR i => VAR m >= (VAR a).value[k])
AND IF VAR i = 0 THEN VAR m = -1 ELSE
(EXISTS(k:INT): 0 <= k AND k < VAR i => VAR m = (VAR a).value[k]) ENDIF;
decreases VAR n - VAR i;
@*/
{
if (a[i] > m) m = a[i];
}
return m;
}
}
}
Console
Exercise3: TYPE = [#null: BOOLEAN, new: INT#];
null: Exercise3;
newObject: INT -> Exercise3 = LAMBDA(x: INT): null WITH .null:=FALSE WITH .new:=x;
Array: TYPE = [#value: ARRAY Base.int OF Exercise3, length: Base.nat, null: BOOLEAN#];
nullArray: Array;
newArray: Base.nat -> Array;
newArrayAxiom: AXIOM FORALL(x: Base.nat): LET y = newArray(x) IN y.null = FALSE AND y.length = x AND (FORALL(i: Base.nat): i < x =>
y.value[i] = null);
All Tasks
Open Tasks
class Array
class Exercise3
method maximum
[Exercise3.maximum] effects
[Exercise3.maximum] postcondition
[Exercise3.maximum] termination
preconditions
[Exercise3.maximum:0] declaration precondition
[Exercise3.maximum:1] return precondition
[Exercise3.maximum:2] declaration precondition
[Exercise3.maximum:3] while loop precondition
[Exercise3.maximum:4] conditional precondition
[Exercise3.maximum:5] assignment precondition
[Exercise3.maximum:6] assignment precondition
loops
[Exercise3.maximum:pqx] invariant is preserved
[Exercise3.maximum:pqx] measure is well-formed
[Exercise3.maximum:pqx] measure is decreased
type checking conditions
[Exercise3.(local):oe5] value is in interval
[Exercise3.(local):2rm] value is in interval
[Exercise3.(local):pvm] value is in interval
[Exercise3.(local):rd4] value is in interval
[Exercise3.(local):tds] value is in interval
specification validation (optional)
theory (local)
type checking conditions
package exercise3a
theory Base
formulas to be proved
type checking conditions
[Base:2f4] Interval [MIN_INT..MAX_INT] is not empty
[Base:xlq] Interval [0..MAX_INT] is not empty
[Base:lzb] divisor is not zero
[Base:i1u] value is in interval
[Base:r6d] value is in interval
[Base:hou] value is in interval
[Base:pyl] value is in interval
[Base:zgx] value is in interval
theory Exercise3
formulas to be proved
type checking conditions
[Exercise3:skr] value is in interval

RISCTP

- consists of a language for specifying prove problems and an associated software for solving them
- extend the RISCAL model checker with theorem proving capabilities
- interacts with external SMT solvers and theorem provers (Z3, cvc5 and Vampire)
- RISCTP language is based on a typed variant of first order-logic, which provides algebraic data types, functional arrays and integer arithmetic

RISCTP

Terminal

```
alan!17> RISCTP -method smt arrays.txt
RISC Theorem Proving Interface 1.0 (June 8, 2022)
https://www.risc.jku.at/research/formal/software/RISCTP
(C) 2022-, Research Institute for Symbolic Computation (RISC)
This is free software distributed under the terms of the GNU GPL.
Execute "RISCTP -h" to see the available command line options.
-----------------------------------------------------------------
Reading file /usr2/schreine/papers/RISCTP2022/problems/arrays.txt...
=== SMT solving
SMT solver: Z3 version 4.8.17 - 64 bit
Proving theorem 'typecheck(Nat)§0'...
SUCCESS: theorem was proved (27 ms).
Proving theorem 'typecheck(Index)§2'...
SUCCESS: theorem was proved (2 ms).
Proving theorem 'typecheck(T)§7'...
SUCCESS: theorem was proved (4 ms).
Proving theorem T...
SUCCESS: theorem was proved (3 ms).
===
SUCCESS termination.
alan!18>
```

Goals of this Thesis

- extension of RISCTP/RISCAL by a saturation-based automated theorem prover for first-order logic with equality
- the theoretical basis for such a prover and the support for special theories (integer and arrays)
- implementation of the prover
- experiments and tests with the prover

Saturation

- the prover generates from a set of first-order clauses in a systematic way all logical consequences until the unsatisfiability of the clause set can be shown
- invented by John Alan Robinson in 1965 (for the resolution calculus)
- searching for a contradiction proceeds by saturating the given set of clauses (the inference rules get applied systematically and exhaustively)
- a process with two levels of data structure manipulation — the level of deduction and the level of theorem proving derivations
- e.g. limited resource strategy algorithm, Discount algorithm and Otter saturation algorithm

Resolution

- the resolution rule derives form two clauses $p \vee C_1$ and $\neg p \vee C_2$ the resolvent $C_1 \vee C_2$
- the resolution calculus can in essence be described by the two inference rules (binary) resolution and (positive) factoring
- the first-order resolution principle of Robinson employs unification in order to resolve the most general forms of the clauses directly
- refutationally complete
- the bare resolution method could easily produce many thousands of clauses without reaching a poof

Resolution

Example

We show the unsatisfiability of this set of clauses:

$$
\begin{array}{lll}
(1): & C(u) \quad \vee \quad C(f(u)) & \\
(2): & \neg C(v) \quad \vee \quad C(f(w)) & \\
(3): & \neg C(x) \quad \vee \quad \neg C(f(x)) & \\
(4): & C(u) \vee C(f(w)) & \textit{by resolving (1) and (2) with } v = f(u) \\
(5): & C(f(w)) & \textit{by factoring (4) with } u = f(w) \\
(6): & \neg C(f(f(w'))) & \textit{by resolving (5) and (3) with } w = w', \\
 & & x = f(w') \\
(7): & \{\} & \textit{by resolving (5) and (6) with } w = f(w')
\end{array}
$$

We use three resolution steps and one factoring step.

Reasoning about Equality

The equality is first-order logic can be handled in many different ways:

- equality axioms
- equality elimination
- paramodulation
- superposition

Equality Axioms

One rather easy way of handling equality is to modify the input formulas by adding special equality axioms:

- $\forall x.\ x = x$ (reflexive)
- $\forall x, y.\ x = y \Rightarrow y = x$ (symmetric)
- $\forall x, y, z.\ x = y \wedge y = z \Rightarrow x = z$ (transitive)
- $\forall x_1, \ldots, x_n, y_1, \ldots, y_n, .\ x_1 = y_1 \wedge \ldots \wedge x_n = y_n \Rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots y_n)$ (function congruence)
- $\forall x_1, \ldots, x_n, y_1, \ldots, y_n, .\ x_1 = y_1 \wedge \ldots \wedge x_n = y_n \Rightarrow p(x_1, \ldots, x_n) = p(y_1, \ldots y_n)$ (predicate congruence)

Equality Elimination

Conside a binary relation R that is an equivalence relation:

- $\forall x.\ R(x,x) \wedge$
- $\forall x,y.\ R(x,y) \Rightarrow R(y,x) \wedge$
- $\forall x,y,z.\ R(x,y) \wedge R(y,z) \Rightarrow R(x,z)$

This is equivalent to
$\forall x,y.\ R(x,y) \Leftrightarrow (\forall z.\ R(x,z) \Rightarrow R(y,z))$.

A general form of this is $\forall x,y.\ R(x,y) \Leftrightarrow R^*[x,y]$ so we can think of rules for replacing each instance of $R(s,t)$ in a formula by $R^*[x,y]$.

Equality Elimination

- earlier equality elimination method (Brand 1975) similarly eliminates symmetry and transitivity, but keeps the reflexivity axiom
- advantage: expansive transformation may only be performed on positive occurrences of $R(s,t)$, while negative occurrences $\neg R(u,v)$ can be left alone.
- with R being the equality relation, these ways of eliminating symmetry and transitivity give Brand's S-modification and T-modification
- proceeding with the congruence axioms the basic idea is to repeatedly pull out non-variable immediate subterms t of function and predicate symbols (other than equality) — Brand's E-modification

Paramodulation

- first equality-based inference rule called demodulation (introduced in 1967 by Wos, Robinson, Carson and Shalla)
- core idea: use unit equality clauses (e. g. x + 0 = x) as rewrite rules to simplify other clauses
- this is not complete, therefore a more general rule called paramodulation was invented in 1969 by G. Robinson and Wos
- Simplified paramodulation: For two ground clauses $C_1 = L[s] \vee C_1'$ ($L[s]$ a literal containing a term s) and $C_2 = s \approx t \vee C_2'$ produce the so-called paramodulant $L[t] \vee C_1' \vee C_2'$

Paramodulation

Example

We show the unsatisfiability of this set of clauses:

$$
\begin{array}{lllll}
(1): & Q(c) & & & \\
(2): & \neg Q(c) & \vee & f(x) = x & \\
(3): & P(x) & \vee & P(f(c)) & \\
(4): & \neg P(x) & \vee & \neg P(f(x)) & \\
(5): & f(x) = x & & & \textit{by resolving (1) and (2)} \\
(6): & P(f(c)) & & & \textit{by factoring (3) with } x = f(c) \\
(7): & \neg P(f(f(c))) & & & \textit{by resolving (6) and (4) with } x = f(c) \\
(8): & \neg P(f(c)) & & & \textit{paramodulant of (5) and (7)} \\
(9): & \{\} & & & \textit{by resolving (8) and (6)}
\end{array}
$$

We use three resolution steps, one paramodulation step and one factorization step.

Superposition

- first introduced by Bachmair and Ganzinger in 1991
- specialization of the resolution method and paramodulation
- the superposition calculus consists of an inference system (respectively a family of systems) for first-order logic with equality, parametrised by a simplification ordering and a selection function
- the rules can in general be divided in generating and simplifying ones
- big advantage: leads to a smaller search space
- compatible with a wide variety of redundancy elimination criteria

Special Theories

The RISCTP language is modeled after SMT-LIB in that it supports among other things the following concepts/theories:

- linear integer arithmetic
- functional arrays with extensionality

An internally developed prover has to support these theories as well. Two ways of integrating them are:

- Axiomatization
- SMT Solvers

Axiomatisation

- adding first-order (incomplete) axiomatisation of the theory
- add theory axioms to the input problem
- special theories available in Vampire are the integers, reals and arrays

SMT solvers

- integrating SMT (Satisfiability Modulo Theory)
- constraining first-order logic (syntactically and/or semantically)
- Ex: Quantifier-free Linear Integer Arithmetic
- SMT-LIB theory `Ints` (Integer arithmetic)
- SMT-LIB theory `ArrayEx` (Functional arrays with extensionality)

Expected Results

- detailed formalization of reasoning in first-order logic with equality as well as proving based on the saturation principle
- justification for ultimately chosen saturation approach based on informal/semiformal arguments
- investigation about the integration of reasoning capabilities of external SMT solvers
- Java-based implementation of a saturation-based automated theorem prover (designed to be transparent)
- benchmarks and comparisons with already existing (external) checking mechanisms

References

- Wolfgang Schreiner. The RISC Algorithm Language (RISCAL). Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria. 2019. url: https://www.risc.jku.at/research/formal/software/RISCAL
- Wolfgang Schreiner. The RISCTP Theorem Proving Interface. Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria. 2022. url: https://www3.risc.jku.at/research/formal/software/RISCTP/.
- John Harrison. Handbook of Practical Logic and Automated Reasoning. Cambridge, UK: Cambridge University Press, 2009. doi: 10.1017/CBO9780511576430
- Laura Kovacs and Andrei Voronkov. "First-Order Theorem Proving and VAMPIRE". In: Computer Aided Verification. Springer, Berlin, Heidelberg, 2013, pp. 1–35. doi: 10.1007/978-3-642-39799-8_1.

References

- Alan Robinson and Andrei Voronkov. Handbook of Automated Reasoning. Vol. 1. printed in The Netherlands: Elsevier, Amsterdam and Co-publisher The MIT Press, Cambridge, Massachusetts, 2001. doi: 10.5555/581809
- Nikolaj De Moura Leonardo; Bjørner. "Satisfiability modulo theories : introduction and applications". In: Communications of the ACM 54 (9 2011), p. 69. issn: 0001-0782. doi: 10.1145/1995376.1995394