The Integration of SMT Solvers into the RISCAL Model Checker

First Master Thesis Report

Franz Reichl

October 31, 2019

Overview

- Motivation and Aim for the Thesis
- Foundations
- Setting of the Problem
- Quantifier Elimination
- Theory Translations
- Conclusion & Current Work

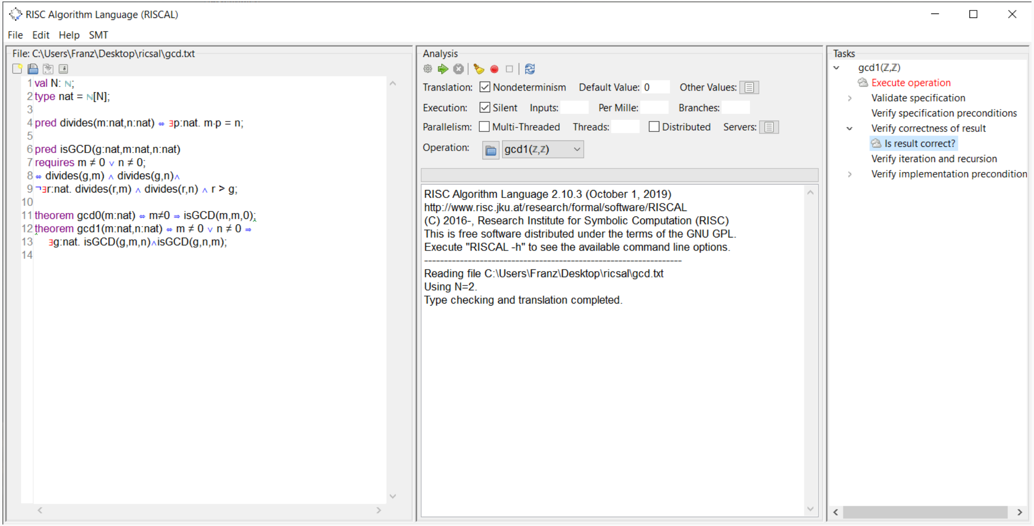
Motivation and Aim of the Thesis

- Elaboration of an alternative way of checking RISCAL formulae.
  - Translation of the RISCAL language to SMT-LIB
  - Application of an SMT solver
- Increase the efficiency of checking formulae.
- Other specification languages already use such translations.
  - JML - OpenJML
  - TLA+
  - Event-B

RISC Algorithm Language - RISCAL

- Specification language developed by Prof. Schreiner at RISC
- RISCAL is freely available at `https://www3.risc.jku.at/research/formal/software/RISCAL/`
- Supports the specification of both mathematical theories and algorithms
- Rich language supporting booleans, integers, tuples, arrays, $\cdots$
- Provides automatic checking (by explicit evaluation)
- Based on a finite type system

Foundations - RISCAL



Foundations

Reminder (multi-sorted) first order Logic.

$\forall x \in A : Q(x) \Rightarrow \exists y \in B : P(x, y)$

- Quantifiers, variables
- Logical connectives
- Sort symbols A, B
- Operation symbols Q, P

We associate a meaning to such formulae by means of interpretations.

- We assign sets to the sort symbols.
- We assign functions to the operation symbols.

Satisfiability - Validity

A formula is:

- *satisfiable* iff there is an interpretation that makes it true.
- *valid* iff any interpretation makes it true.

Satisfiability and Validity are dual properties:

- A formula is valid iff its negation is unsatisfiable.

Satisfiability Modulo Theories (SMT) - Motivation

Consider the following formula $f(1) < f(1)$

- In the usual context of mathematics we would consider this formula as incorrect.
- But this formula is satisfiable.

Conclusion: Restrict the considered interpretations

Satisfiability Modulo Theories

A *theory* denotes a class of interpretations.

Let T be a theory then a formula is:

- satisfiable modulo T iff there is an interpretation from T that makes it true.
- valid modulo T iff any interpretation from T makes it true.

Satisfiability Modulo Theories - Example

Lets come back to the motivating example: $f(1) < f(1)$

The theory of integers I contains only interpretations that:

- Assign the expected values to the constants $0, 1, 2, \cdots$
- Assign the expected meaning to the operations $+, -, <, >, \cdots$

$f(1) < f(1)$ is unsatisfiable modulo I.

Satisfiability Modulo Theories - Quantifier free Theory of Bit Vectors

In this thesis we use the bit vector theory because:

- Bit vectors are well suited for modelling the RISCAL types.
- Similar to RISCAL all types in this theory are finite.

Satisfiability Modulo Theories - Quantifier free Theory of Bit Vectors

In this thesis we use the bit vector theory because:

- Bit vectors are well suited for modelling the RISCAL types.
- Similar to RISCAL all types in this theory are finite.

In this theory we have:

- Formulae contain no quantifiers.
- Sequences of arbitrary but fixed length, of zeroes and ones.
- Various operations including:
    - Bitwise arithmetic operations
    - Shift operations

SMT-LIB

SMT-LIB was founded in 2002 by Silvio Ranise and Cesare Tinelli.

SMT-LIB consists of three parts:

- A standardized description of theories.
- A collection of benchmarks for SMT solvers.
- An input and output language for SMT solvers.

For more information on SMT-LIB see `http://smtlib.cs.uiowa.edu/`

SMT-LIB Example

```
(declare-fun x() (_ BitVec 4))
(define-fun y() (_ BitVec 4)#b0001)
(assert (not (bvule x (bvadd x y))))
(check-sat)
```

Applying the SMT-solver Boolector to this script yields the result satisfiable.
⇝ If we assign 1111 to x the asserted property holds.

The problem setting

Given a list of correct RISCAL declarations $D_1, \cdots, D_n$ and a RISCAL theorem Th

Check if Th is valid.

The problem setting

Given a list of correct RISCAL declarations $D_1, \cdots, D_n$ and a RISCAL theorem Th

Check if Th is valid.

We can model RISCAL declarations by formulae $F_1, \cdots, F_m$ and F which are interpreted with respect to the RISCAL theory R

Thus we have to check whether all R interpretations that make $F_1, \cdots F_n$ true also make F true. (i.e. $F_1, \cdots, F_m \vDash_R F$.)

The problem setting

We know

$F_1, \cdots, F_m \models F$ iff $\bigwedge_{i=1}^{n} F_i \Rightarrow F$ is valid.

Similar we have for a theory T

$F_1, \cdots, F_m \models_T F$ iff $\bigwedge_{i=1}^{n} F_i \Rightarrow F$ is valid modulo T.

Therefore we have to show that $(\bigwedge_{i=1}^{n} F_i) \Rightarrow F$ is valid modulo R.
This is equivalent to showing that $(\bigwedge_{i=1}^{n} F_i) \wedge \neg F$ is unsatisfiable modulo R.

The problem setting

Our goal is to find a translation Ψ such that for any RISCAL-formula F the following properties of $\Psi(F)$ are fulfilled:

- Contains no quantifiers
- $\Psi(F)$ is satisfiable modulo the bit vector theory iff F is satisfiable modulo R.

Quantifier Elimination

Unquantification Algorithm

Input: A formula F_0

Require: $\mathit{free}(F_0) = \emptyset$

Output: A formula F_{out}

Ensure: F_0 and F_{out} are equisatisfiable with respect to the theory of RISCAL

Ensure: $\nexists p \in Pos(F_{out}), x^i \in \mathcal{V}, F' \in \mathcal{L} : F_{out}\langle p \rangle = \forall_{\mathcal{L}} x^i F'$

Ensure: $\nexists p \in Pos(F_{out}), x^i \in \mathcal{V}, F' \in \mathcal{L} : F_{out}\langle p \rangle = \exists_{\mathcal{L}} x^i F'$

function $\textsc{Unquantify}(F_0)$
- $F_1 \leftarrow \textsc{RemImpEqu}(F_0)$
- $F_2 \leftarrow \textsc{SinkNeg}(F_1)$
- $F_3 \leftarrow \textsc{UniQVars}(F_2, \emptyset, \mathit{bound}(F))$
- $(F_4, FS) \leftarrow \textsc{Skol}(F_3, \mathit{getOpSyms}(F_3) \cup Op_{RISCAL} \cup Op_{BV})$
- $F_5 \leftarrow \textsc{ExpAll}(F_4)$
- **return** F_5

end function

Quantifier Elimination

Quantifier Expansion

Reminder: In RISCAL we have finite types.

Let T be a type and $T = \{t_1, \cdots, t_n\}$ then

- $\forall x \in T : P(x) \equiv \bigwedge_{i=1}^{n} P(t_i)$.
- $\exists x \in T : P(x) \equiv \bigvee_{i=1}^{n} P(t_i)$.

Example:

$\forall x \in \mathbb{Z}[0, 2] : x < 3 \rightsquigarrow (0 < 3) \wedge (1 < 3) \wedge (2 < 3)$

Quantifier Elimination

Skolemisation

Preparatory steps:

- Removal of implications and equivalences
- Push negations inwards
- Rename variables

Quantifier Elimination

Skolemisation

Preparatory steps:

- Removal of implications and equivalences
- Push negations inwards
- Rename variables

After this preparatory steps we have to

- Determine the free variables in the respective expression.
- Find an unused operation symbol.
- Replace each occurrence of the quantified variable by the operation symbol applied to the free variables.

Skolemisation

Preservation of satisfiability

- The preparatory steps preserve equality.
- The skolemised formula is satisfiable iff the original formula is satisfiable.

Quantifier Elimination

Skolemisation

Preservation of satisfiability

- The preparatory steps preserve equality.
- The skolemised formula is satisfiable iff the original formula is satisfiable.

Example: $\forall x \in A : (\forall y \in B : P(x, y)) \Rightarrow \forall x \in C : Q(x)$
First we rewrite this formula to:
$\forall x \in A : (\exists y \in B : \neg P(x, y)) \vee \forall z \in C : Q(z)$
The only free variable in $\exists y \in B : \neg P(x, y)$ is y therefore we get
$\forall x \in A : \neg P(x, f(x)) \vee \forall z \in C : Q(z)$

Theory Translations

Booleans

- Logical connectives are available in both theories.
- We will deal with special predicates (like $<, >$) when we discuss the translation of the types of the predicate's arguments.

Theory Translations

Integers

There is a close relation between bit vectors and integers.

Let b be a bit vector of length n then we assign integers to bit vectors in the following ways:

- Signed Representation: $-b[n] \cdot 2^n + \sum_{i=1}^{n-1} b[i] \cdot 2^i$
- Unsigned Representation: $\sum_{i=1}^{n} b[i] \cdot 2^i$

Theory Translations

Integers

For several integer operations in the RISCAL theory there are related operations in the bit vector theory. (e.g. $+, -, <, >, \cdots$)

Especially two problems arise:

- The operations from the bit vector theory require arguments of the same length.
- The arithmetic operations from the bit vector theory correspond to modular integer arithmetic.

Example: Consider the expression $4 + 1$ from the RISCAL theory. We can associate 4 to the bit vector 100 and 1 to the bit vector 1.
$\rightsquigarrow$ There is no addition function for arguments of different lengths.

Integers

Solution to the problems: Find an appropriate vector length and associate the integers to bitvectors of this length.

For an RISCAL integer expression we determine

- The vector length necessary to represent the expression itself.
- The vector lengths necessary to represent the integer sub expressions.

We use the maximum of these lengths as the length of our bit vectors

Theory Translations

Integers

Example: $8 - (4 + 4)$

- We need a vector of length 1 to represent $0 = 8 - (4 + 4)$
- We need a vector of length 4 to represent 8
- We need a vector of length 3 to represent 4

Thus we use bit vectors of length 4 i.e. (*bvsub* 1000 (*bvadd* 0100 0100))

Theory Translations

Integers

Example: $8 - (4 + 4)$

- We need a vector of length 1 to represent $0 = 8 - (4 + 4)$
- We need a vector of length 4 to represent 8
- We need a vector of length 3 to represent 4

Thus we use bit vectors of length 4 i.e. ($bvsub$ 1000 ($bvadd$ 0100 0100))

This procedure ensures that

- The arguments of functions have the same length.
- No overflows occur.

Translation - Example

```
val N: ℕ;
type nat = ℕ[N];

pred divides(m:nat,n:nat) ⇔ ∃p:nat. m·p = n;

pred isGCD(g:nat,m:nat,n:nat)
requires m ≠ 0 ∨ n ≠ 0;
⇔ divides(g,m) ∧ divides(g,n)∧¬∃r:nat. divides(r,m) ∧ divides(r,n) ∧ r > g;

theorem gcd0(m:nat) ⇔ m≠0 ⇒ isGCD(m,m,0);
theorem gcd1(m:nat,n:nat) ⇔ m ≠ 0 ∨ n ≠ 0 ⇒ ∃g:nat. isGCD(g,m,n)∧isGCD(g,n,m);
```

We want to show that the theorem *gcd1* holds.

Translation - Example

```
(set-logic QF_UFBV)
(define-fun N () (_ BitVec 2) #b10)
(define-sort nat ()(_ BitVec 2))
(define-fun divides ((m (_ BitVec 2)) (n (_ BitVec 2)))Bool (or (let ((p #b00)) (= (bvmul ((_
     zero_extend 1) m) ((_ zero_extend 1) p)) ((_ zero_extend 1) n))) (let ((p #b01)) (= (bvmul ((_
     zero_extend 1) m) ((_ zero_extend 1) p)) ((_ zero_extend 1) n))) (let ((p #b10)) (= (bvmul ((_
     zero_extend 1) m) ((_ zero_extend 1) p)) ((_ zero_extend 1) n)))))
(define-fun isGCD ((g (_ BitVec 2)) (m (_ BitVec 2)) (n (_ BitVec 2)))Bool (and (and (divides g m) (
     divides g n)) (and (let ((r #b00)) (or (or (not (divides r m)) (not (divides r n))) (bvule r g)))
     (let ((r #b01)) (or (or (not (divides r m)) (not (divides r n))) (bvule r g))) (let ((r #b10)) (or
      (or (not (divides r m)) (not (divides r n))) (bvule r g))))))
(declare-fun f ()(_ BitVec 2))
(assert (and (bvule #b00 f) (bvuge #b10 f)))
(declare-fun f_1 ()(_ BitVec 2))
(assert (and (bvule #b00 f_1) (bvuge #b10 f_1)))
(assert (let ((m f))(let ((n f_1))(let (( result (or (and (= m #b00) (= n #b00)) (or (let ((g #b00)) (
     and (isGCD g m n) (isGCD g n m))) (let ((g #b01)) (and (isGCD g m n) (isGCD g n m))) (let ((g #b10
     )) (and (isGCD g m n) (isGCD g n m))))))) (not result)))))
(check-sat)(exit)
```

Applying the SMT-solver Boolector to this script yields the result unsatisfiable.

Future Work - Conclusion

Remaining work:

- Essentially the program is finished. The search for potential bugs remains.
- The remaining theories need to be formalized.
- Tests with the program have to be conducted in order to compare its performance with the performance of the existing checking mechanism.

Preliminary conclusion:

- Tests of the program already indicate that in certain cases the SMT solvers are much faster than the current evaluation mechanism.
- But the tests also indicate that for certain cases the SMT solver approach is much slower.