THE RISCTP SOFTWARE

A Model Elimination Prover

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University Linz, Austria

JOHANNES KEPLER UNIVERSITY LINZ

The RISCTP Theorem Proving Interface

An extension of the RISCAL model checker by theorem proving capabilities.

- RISCTP: an intermediate language for stating proof problems.
  - Lower level of abstraction than RISCAL, higher level than SMT-LIB.
  - Like SMT-LIB: typed variant of first-order logic, algebraic data types, functional arrays with extensionality, integer arithmetic.
  - Beyond SMT-LIB: overloading, subtypes, tuple types, choose expressions.
  - Implementation by a Java library and as a standalone program.
- Solution of proof problems possible in various ways.
  - Existing: translation to SMT-LIB with quantification and SMT-LIB logics `ArraysEx` and `Ints`; connection to external SMT solvers/provers cvc5, Vampire, Z3.
  - Ongoing: internal provers based on resolution (V. Langenreither) and model elimination (W. Schreiner), respectively.

`https://www.risc.jku.at/research/formal/software/RISCTP`

The RISCTP Language

```
// problem file "arrays.txt"
const N:Nat; axiom posN ⇔ N > 0;
type Index = Nat with value < N;
type Value; type Elem = Tuple[Int,Value]; type Array = Map[Index,Elem];
fun key(e:Elem):Int = e.1;
pred sorted(a:Array,from:Index,to:Index) ⇔
  ∀i:Index,j:Index. from ≤ i ∧ i < j ∧ j ≤ to ⇒ key(a[i]) ≤ key(a[j]);
theorem T ⇔
  ∀a:Array,from:Index,to:Index,x:Int.
    from ≤ to ∧ sorted(a,from,to) ⇒
    // let i = (from+to)/2 in
    let i = choose i:Index with from ≤ i ∧ i ≤ to in
    key(a[i]) < x ⇒ ¬∃j:Index. from ≤ j ∧ j < i ∧ key(a[j]) = x;
```

Translation to the SMT-LIB language for external SMT solving and to classical first-order logic for internal theorem proving.

Processing Steps

1. Parse.
2. Type-check.
3. Remove subtypes.
4. Resolve overloading.
5. Remove choose expressions.
6. (FOL) Replace constants denoting variables by actual variables.
7. (FOL) Replace datatype declarations and match expressions.
8. (FOL) Replace let expressions.
9. (FOL) Replace function definitions by axioms.
10. (FOL) Separate terms from formulas.
11. Determine theorems and prune problem accordingly.
12. (SMT) Translation to SMT-LIB.
13. (FOL) Decompose problem into subproblems by sequent calculus.
14. (FOL) Transform problem into clausal form.
15. (FOL) Transform clausal form into a more efficient representation.

Starting point of first-order proofs by model elimination.

Model Elimination/MESON

- Model Elimination: a first-order proof calculus (Loveland, 1968).
  - MESON (model elimination subgoal-oriented): a reformulation of model elimination as a problem reduction strategy (Loveland, 1978).
  - PPTP: a "Prolog Technology Theorem Prover" implementation (Stickel, 1988).
  - See also (Letz and Stenz, 2001) and (Harrison, 2009) (OCaml implementation).

[1] John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, Cambridge, UK, 2009. http://www.cambridge.org/az/academic/subjects/computer-science/programming-languages-and-applied-logic/handbook-practical-logic-and-automated-reasoning.

[2] Reinhold Letz and Gernot Stenz. Model Elimination and Connection Tableau Procedures. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 28, pages 2015–2114. North-Holland, Amsterdam, The Netherlands, 2001. doi:10.1016/B978-044450813-3/50030-8.

[3] Donald W. Loveland. Mechanical Theorem-Proving by Model Elimination. *Journal of the ACM*, 15(2):236—251, April 1968. doi:10.1145/321450.321456.

[4] Donald W. Loveland. *Automated Theorem Proving: A Logical Basis*, volume 6 of *Fundamental Studies in Computer Science*. North-Holland, Amsterdam, The Netherlands, 1978. doi:10.1016/c2009-0-12705-8.

[5] Mark E. Stickel. A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler. *Journal of Automated Reasoning*, 4:353–380, 1988. doi:10.1007/BF00297245.

SLD-Resolution/Prolog

A calculus for reasoning about Horn clauses (Robert Kowalski, 1974)

- Rules: a set of Horn clauses $F = \{(\forall x)\ (A_1 \wedge \ldots \wedge A_{a\geq 0} \Rightarrow B), \ldots\}$.
- Goal: a negated Horn clause $G = (\exists y)\ (G_1 \wedge \ldots \wedge G_{g\geq 0})$.
  - Atoms (positive literals) $A_1, \ldots, A_a, B, G_1, \ldots, G_g$.
- Judgement $F \vdash G$: is $(F \Rightarrow G)$ valid?
  - Can be reduced to judgment $F \vdash_\emptyset G$.
  - $F \vdash_\sigma G$: is $(F \Rightarrow G)\sigma$ valid (with variable substitution σ)?

$$\frac{}{F \vdash_\sigma \top}\ \text{(AX)}$$

$F = \{C, \ldots\} \quad C = (A_1 \wedge \ldots \wedge A_a \Rightarrow B) \quad G = (G_1 \wedge G_2 \wedge \ldots \wedge G_g)$

σ_0 is a bijective renaming of the variables in $C\sigma$ such that $C\sigma\sigma_0$ and $G\sigma$ have no common variables

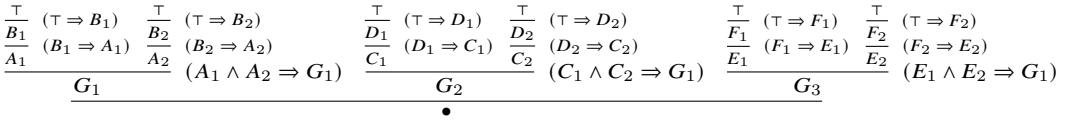
$B\sigma\sigma_0$ and $G_1\sigma$ have mgu σ_1

$$\frac{F \vdash_{\sigma\sigma_0\sigma_1} (A_1 \wedge \ldots \wedge A_a) \quad F \vdash_{\sigma\sigma_0\sigma_1} (G_2 \wedge \ldots \wedge G_g)}{F \vdash_\sigma G}\ \text{(SLD)}$$

The calculus applies subgoal-oriented backward reasoning (“backward chaining”).

Proof Search

An implementation of the calculus (implicitly) constructs a proof tree.



- Solving substitution σ: determined during the construction of the tree.
  - Starting with $\sigma = \emptyset$, rule (SLD) chooses for every node some rule and extends σ.
- Completeness of the proof search.
  - All possible rule choices have to be considered; this requires a suitable organization of the construction process.
  - Prolog applies a simple and efficient but incomplete strategy: top-down/left-to-right construction with backtracking on dead ends.

The proof search can be based on an intuitively understandable strategy.

Model Elimination/MESON

- Rules: a set of clauses $F = \{(\forall x)\ (A_1 \wedge \ldots \wedge A_{a\geq 0} \Rightarrow B_1 \vee \ldots \vee B_{b\geq 0}), \ldots\}$.
  - Atoms (positive literals) $A_1, \ldots, A_a, B_1, \ldots, B_b$.
- Goal: a negated clause $G = (\exists y)\ (G_1 \wedge \ldots \wedge G_{g\geq 0})$.
  - Positive/negative literals $G_1 \wedge \ldots \wedge G_g$.
- Judgement $F \vdash G$: is $(F \Rightarrow G)$ valid?
  - Can be reduced to judgement $F \vdash^{\emptyset}_{\emptyset} G$.
  - $F \vdash^{Ls}_{\sigma} G$: is $(F \wedge Ls \Rightarrow G)\sigma$ valid (with variable substitution σ and literal set Ls)?

$$\frac{}{F \vdash^{Ls}_{\sigma} \top}\ \text{(AX)} \qquad \frac{Ls = \{L, \ldots\} \qquad G_1\sigma \text{ and } L\sigma \text{ have mgu } \sigma_0 \qquad F \vdash^{Ls}_{\sigma\sigma_0} (G_2 \wedge \ldots \wedge G_g)}{F \vdash^{Ls}_{\sigma} (G_1 \wedge G_2 \wedge \ldots \wedge G_g)}\ \text{(ASS)}$$

$F = \{C, \ldots\} \quad C = (L_1 \vee \ldots \vee L_i \vee \ldots \vee L_{a+b}) \quad G = (G_1 \wedge G_2 \wedge \ldots \wedge G_g)$

σ_0 is a bijective renaming of the variables in $C\sigma$ such that $C\sigma\sigma_0$ and $G\sigma$ have no common variables

$$\frac{L_i\sigma\sigma_0 \text{ and } G_1\sigma \text{ have mgu } \sigma_1 \qquad F \vdash^{Ls\cup\{\overline{G_1}\}}_{\sigma\sigma_0\sigma_1} (\overline{L_1} \wedge \ldots \wedge \overline{L_{i-1}} \wedge \overline{L_{i+1}} \wedge \ldots \wedge \overline{L_{a+b}}) \qquad F \vdash^{Ls}_{\sigma\sigma_0\sigma_1} (G_2 \wedge \ldots \wedge G_g)}{F \vdash^{Ls}_{\sigma} G}\ \text{(MESON)}$$

A generalization of Prolog-like backward chaining to full first-order logic.

Soundness

We show the soundness of rule (MESON) for a simple case (ignoring substitutions).

$$\frac{F = \{(G_1 \vee L_2), \ldots\} \quad F \vdash^{Ls \cup \{\overline{G_1}\}} \overline{L_2} \quad F \vdash^{Ls} G_2}{F \vdash^{Ls}_{\sigma} (G_1 \wedge G_2)} \text{ (MESON)}$$

- We assume (1) $(F \Rightarrow (G_1 \vee L_2))$, (2) $(F \wedge Ls \wedge \overline{G_1} \Rightarrow \overline{L_2})$, (3) $(F \wedge Ls \Rightarrow G_2)$, and (4) $(F \wedge Ls)$.
- We show $G_1 \wedge G_2$.
- From (3) and (4) we have G_2, thus it suffices to show G_1.
- We assume (5) $\overline{G_1}$ and show a contradiction.
- From (2), (4), and (5), we have (6) $\overline{L_2}$. But (4), (5), and (6) contradict (1). □

The intuition for the additional assumption $\overline{G_1}$ will become clearer in the software demo.

Completeness

MESON is complete for a goal G in clausal form. But how to apply MESON to prove a general first-order formula $G = (A \Rightarrow T)$?

1. Convert $\neg G = (A \wedge \neg T)$ to a clause set Cs. For every clause $C \in Cs$ that contains only negative literals, attempt a proof of $Cs \backslash \{C\} \vdash \neg C$.
   - If Cs does not contain any such clause, Cs is satisfiable and G is not valid (also if Cs fully consists of clauses with only positive literals).
2. Assume that A is satisfiable (which is the case if A represents some properties of an existing model). Convert A into a clause set As and $\neg T$ into a clause set Ts. For every clause $C \in Ts$, attempt a proof of $As \cup Ts \backslash \{C\} \vdash \neg C$.
   - This is the strategy applied in RISCTP to prove a `theorem T`: construct As from all formulas marked as `axiom` and Ts from $\neg T$.

Another level of nondeterminism (apart from rule selection).

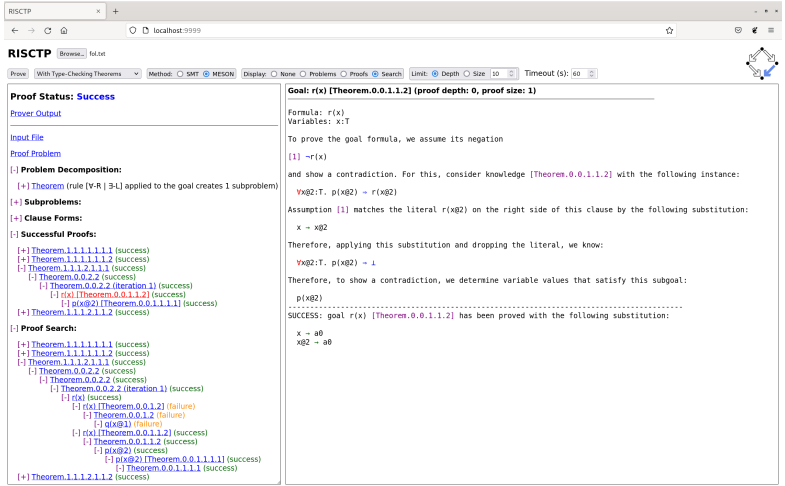
Proof Search

How to implement a proof search that preserves completeness?

- Iterative deepening: attempt Prolog-like proof search for proof bounds $1, 2, \ldots$ and terminate the attempt if the bound is exceeded.
  - Proof depth: depth of the proof tree.
  - Proof size: number of nodes in the proof tree.
- (Harrison, 2009) recommends proof size.
  - MESON may generate some long proof branches.
  - Harrison describes an optimization to spread the proof size "budget" more evenly over different proof branches (but this requires multiple attempts to order the subproofs appropriately).

RISCTP supports both kinds of bounds (but proof size without the Harrison optimization, which makes them rather ineffective). Proof depth seems more transparent and is actually effective for various examples; it is therefore the default.

MESON in the RISCTP GUI (Demo)



More options are available from the command line.

Next Steps

- Equality Reasoning
  - Paramodulation-style rewriting applied to every literal added to the literal set Ls.
- Theory Reasoning
  - Add an (incomplete) axiomatization of the integers (a complete axiomatization already exists for arrays and algebraic data types) to be handled by MESON.
  - Apply an external SMT solver to determine whether a goal literal is is a logical consequence of Ls (considering the theories of equality, integers, and arrays).
- Integration in RISCAL
  - Application to real verification conditions.
- Comparison with Resolution
  - Viktoria Langenreither's prover.

https://www.risc.jku.at/research/formal/software/RISCTP