THE RISCTP SOFTWARE

Equality and Theory Support for the MESON Prover

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

JOHANNES KEPLER
UNIVERSITY LINZ

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;
```

Typed variant of first-order logic with equality and the theories of integers, maps (functional arrays with extensionality), algebraic data types (including tuples).

MESON: Model Elimination, Subgoal-Oriented

- 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 $G_1 \wedge \ldots \wedge G_g$.
- Judgment $F \vdash G$: is $(F \Rightarrow G)$ valid?
  - Can be reduced to judgment $F \vdash^{\emptyset}_{\emptyset} G$.
  - $F \vdash^{Ls}_{\sigma} G$: $(F \wedge Ls \Rightarrow G)\sigma$ is valid (with variable substitution σ and literal set Ls).

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

$$\frac{\begin{array}{c} F = \{C, \ldots\} \quad C = (L_1 \vee \ldots \vee L_i \vee \ldots \vee L_{a+b}) \quad G = (G_1 \wedge G_2 \wedge \ldots \wedge G_g) \\ \sigma_0 \text{ is a bijective renaming of the variables in } C\sigma \text{ such that } C\sigma\sigma_0 \text{ and } G\sigma \text{ have no common variables} \\ L_i\sigma\sigma_0 \text{ and } G_1\sigma \text{ have mgu } \sigma_1 \\ F \vdash^{Ls \cup \{\overline{G_1}\}}_{\sigma\sigma_0\sigma_1} (\overline{L_1} \wedge \ldots \wedge \overline{L_{i-1}} \wedge \overline{L_{i+1}} \wedge \ldots \wedge \overline{L_{a+b}}) \quad F \vdash^{Ls}_{\sigma\sigma_0\sigma_1} (G_2 \wedge \ldots \wedge G_g) \end{array}}{F \vdash^{Ls}_{\sigma} G}\ \text{(MESON)}$$

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

Proof Search

An implementation of the calculus (implicitly) constructs a proof tree (below the special case of Prolog-like Horn clauses is depicted):

$$\dfrac{\dfrac{\dfrac{\dfrac{\top}{B_1}\,(\top \Rightarrow B_1)}{A_1}\,(B_1 \Rightarrow A_1) \quad \dfrac{\dfrac{\top}{B_2}\,(\top \Rightarrow B_2)}{A_2}\,(B_2 \Rightarrow A_2)}{G_1}\,(A_1 \wedge A_2 \Rightarrow G_1) \quad \dfrac{\dfrac{\dfrac{\top}{D_1}\,(\top \Rightarrow D_1)}{C_1}\,(D_1 \Rightarrow C_1) \quad \dfrac{\dfrac{\top}{D_2}\,(\top \Rightarrow D_2)}{C_2}\,(D_2 \Rightarrow C_2)}{G_2}\,(C_1 \wedge C_2 \Rightarrow G_1) \quad \dfrac{\dfrac{\dfrac{\top}{F_1}\,(\top \Rightarrow F_1)}{E_1}\,(F_1 \Rightarrow E_1) \quad \dfrac{\dfrac{\top}{F_2}\,(\top \Rightarrow F_2)}{E_2}\,(F_2 \Rightarrow E_2)}{G_3}\,(E_1 \wedge E_2 \Rightarrow G_1)}{\bullet}$$

- Solving substitution σ: determined during the construction of the tree.
  - Starting with $\sigma = \emptyset$, rule (MESON) 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.
- Strategy applied in RISCTP:
  - Clauses are ordered according to their introduction in the proof problem file; later clauses are likely to represent higher-level "lemmas" and are tried first.

An intuitively understandable strategy.

MESON Theory Support

- Integration of SMT Solver
  - Decide $F \vdash_{\sigma}^{Ls} G_i$ by showing the unsatisfiability of $(Ls \wedge \neg G_i)\sigma$.
  - Slow and actually only effective if the proof decomposition is guided by appropriate theory axioms (not discussed further).
- Equality Reasoning
  - Add axiom $F \vdash_{\sigma}^{Ls} (t = t)$.
  - Apply paramodulation-style rewriting to goal literal.
- Axiomatization of Theories
  - Add axioms to (completely or incompletely) characterize the underlying theories.

All three extensions have been implemented in RISCTP.

Paramodulation-Style Rewriting

A natural adaptation of rule (MESON).

$$\frac{\begin{array}{c} F = \{C, \ldots\} \quad C = (L_1 \vee \ldots \vee (l = r) \vee \ldots \vee L_{a+b}) \quad G = (G_1[t] \wedge G_2 \wedge \ldots \wedge G_g) \\ \sigma_0 \text{ is a bijective renaming of the variables in } C\sigma \text{ such that } C\sigma\sigma_0 \text{ and } G\sigma \text{ have no common variables} \\ t\sigma\sigma_0 \text{ and } l\sigma \text{ have mgu } \sigma_1 \\ F \vdash^{Ls}_{\sigma\sigma_0\sigma_1} (\overline{L_1} \wedge \ldots \wedge \overline{L_{i-1}} \wedge \overline{L_{i+1}} \wedge \ldots \wedge \overline{L_{a+b}}) \quad F \vdash^{Ls}_{\sigma\sigma_0\sigma_1} (G_1[r] \wedge G_2 \wedge \ldots \wedge G_g) \end{array}}{F \vdash^{Ls}_{\sigma} G} \text{(PARA)}$$

$L[t]$: literal L with subterm t.

Also applicable for $C = (L_1 \vee \ldots \vee (r = l) \vee \ldots \vee L_{a+b})$.

Rewriting Control

Uncontrolled rewriting lets space of proof search quickly explode.

- Avoid rewrite cycles: If t_1 has been rewritten to t_2, do not rewrite t_2 to t_1 in same proof branch.
- Prohibit variable rewrites: do not apply rule to rewrite variable x to some term t.
- Restrict rewrite positions: only apply rules to term positions in G_i (not in $G_i\sigma$).
- Direct equations: do not apply $l = r$ if $r \succ l$ for a variant of lexicographic path order:
  - $l \in var(r)$ and $l \neq r$.
  - $r = f(r_1, \ldots, r_m)$ and $l = g(l_1, \ldots, l_n)$ and
    - $r_i \succeq l$ for some i, or
    - $f \succ g$ and $r \succ l_j$ for all j, or
    - $f = g$ and $r \succ l_j$ for all j and $(r_1, \ldots, r_m) \succ_{\text{lex}} (l_1, \ldots, l_n)$.
  - We consider $f \succ g$ iff f was declared in the theory later than g.
  - Variant: $t \succ f(t)$ if t is of an algebraic data type and f is a selector of that type.

Various settings: “None” (no rewriting), “Min” (rewriting with all restrictions), “Med” (do not restrict rewrite positions), “Max” (also do not direct equations and do not prohibit rewriting into variables).

Axiomatization of Theories of Structured Types

- Arrays:

$$\forall a_1, a_2.\ (\forall i.\ a_1[i] = a_2[i]) \Rightarrow a_1 = a_2$$
$$\forall a, i.e.\ a[i \mapsto e][i] = e$$
$$\forall a, i, j, e.\ i \neq j \Rightarrow a[i \mapsto e][j] = a[j]$$

- Tuples:

$$\forall x_1, x_2, y_1, y_2.\ \langle x_1, x_2 \rangle = \langle y_1, y_2 \rangle \rightarrow x_1 = x_2 \wedge y_1 = y_2$$
$$\forall x_1, x_2.\ \langle x_1, x_2 \rangle.1 = x_1$$
$$\forall x_1, x_2.\ \langle x_1, x_2 \rangle.2 = x_2$$
$$\forall t, x_1.\ (t\ \texttt{with}\ .1 = x_1).1 = x_1$$
$$\forall t, x_2.\ (t\ \texttt{with}\ .2 = x_2).2 = x_2$$

- Algebraic Data Types:
  - Tuple types are just a special case.
  - Axiomatization of constructor, selecter, tester operations. . .

Axiom forms are tweaked and supplementary axioms are added to simplify proofs.

Axiomatization of Integers

- Necessarily incomplete.
- Inspired by axiomatization used in Vampire (cf. thesis of V. Langenreither).
- Literals $n \geq 2$ are inductively axiomatized as $n = n' + 1$ with literal $n' = n - 1$.
- Preprocessing applied to remove $>$ and $\geq$.
- Axiomatization of $0, 1, +, -, \cdot, =, <, \leq$.

RISCTP tries later axioms first, so order is important.

Axiomatization of Integers

```
type Int;
pred '=§0'(x:Int,y:Int);
pred '≠§0'(x:Int,y:Int);
pred <(x1:Int,x2:Int);
pred ≤(x1:Int,x2:Int);
pred >(x1:Int,x2:Int);
pred ≥(x1:Int,x2:Int);
type Nat = Int;
const 0:Int;
pred 'Nat::type'(value:Int);
axiom def§25 ⇔
  ∀value:Int. ('Nat::type'(value) ⇔ (¬(value < 0)));
theorem typecheck(Nat)§0 ⇔ ∃value:Int. 'Nat::type'(value);
fun +(x1:Int,x2:Int):Int;
fun -(x1:Int,x2:Int):Int;
fun '-§0'(x:Int):Int;
fun ·(x1:Int,x2:Int):Int;
axiom §comm+ ⇔ ∀x:Int,y:Int. '=§0'(x+y,y+x);
axiom §assoc+ ⇔ ∀x:Int,y:Int,z:Int. '=§0'(x+(y+z),(x+y)+z);
axiom §neut+ ⇔ ∀x:Int. '=§0'(x+0,x);
axiom §inv+ ⇔ ∀x:Int. '=§0'(x+'-§0'(x),0);
axiom §def- ⇔ ∀x:Int,y:Int. '=§0'(x-y,x+'-§0'(y));
axiom §comm* ⇔ ∀x:Int,y:Int. '=§0'(x·y,y·x);
axiom §assoc* ⇔ ∀x:Int,y:Int,z:Int. '=§0'(x·(y·z),(x·y)·z);
const 1:Int;
axiom §neut* ⇔ ∀x:Int. '=§0'(x·1,x);
axiom §absorb* ⇔ ∀x:Int. '=§0'(x·0,0);
axiom §inv- ⇔ ∀x:Int. '=§0'('-§0'('-§0'(x)),x);
axiom §distrib- ⇔ ∀x:Int,y:Int. '=§0'('-§0'(x+y),'-§0'(x)+'-§0'(y));
axiom §distrib* ⇔ ∀x:Int,y:Int,z:Int. '=§0'(x·(y+z),(x·y)+(x·z));
axiom §preserve<* ⇔ ∀x:Int,y:Int,z:Int.
  (((x < y) ∧ (0 < z)) ⇒ ((x·z) < (y·z)));
axiom §preserve<+ ⇔ ∀x:Int,y:Int,z:Int. ((x < y) ⇒ ((x+z) < (y+z)));
axiom §trans< ⇔ ∀x:Int,y:Int,z:Int. (((x < y) ∧ (y < z)) ⇒ (x < z));
axiom §trans<= ⇔ ∀x:Int,y:Int,z:Int. (((x ≤ y) ∧ (y ≤ z)) ⇒ (x ≤ z));
axiom §trans1<= ⇔ ∀x:Int,y:Int,z:Int. (((x ≤ y) ∧ (y < z)) ⇒ (x < z));
axiom §trans2<= ⇔ ∀x:Int,y:Int,z:Int. (((x < y) ∧ (y ≤ z)) ⇒ (x < z));
axiom §trichotomy ⇔ ∀x:Int,y:Int. (((x < y) ∨ (y < x)) ∨ '=§0'(x,y){*});
axiom §notequal< ⇔ ∀x:Int,y:Int. (((x < y) ∨ (y < x)) ⇒ (¬'=§0'(x,y)));
axiom §neqdef ⇔ ∀x:Int,y:Int. ('≠§0'(x,y) ⇔ (¬'=§0'(y,x)));
axiom §irrefl2< ⇔ ∀x:Int,y:Int. ('=§0'(x,y) ⇒ (¬(x < y)));
axiom §refl<= ⇔ ∀x:Int,y:Int. ('=§0'(x,y) ⇒ (x ≤ y));
axiom §def<= ⇔ ∀x:Int,y:Int. ((x ≤ y) ⇔ (¬(y < x)));
axiom §equiv< ⇔ ∀x:Int,y:Int. ((x < y) ⇔ (¬(y < (x+1))));
axiom §plus1<= ⇔ ∀x:Int,y:Int. ((x < y) ⇔ ((x+1) ≤ y));
axiom §minus1<= ⇔ ∀x:Int,y:Int. ((x < y) ⇔ (x ≤ (y-1)));
axiom §minus1< ⇔ ∀x:Int. ((x-1) < x);
axiom §plus1< ⇔ ∀x:Int. (x < (x+1));
axiom §0<1 ⇔ 0 < 1;
axiom §irrefl< ⇔ ∀x:Int. (¬(x < x));
```

Preventing Literals as Proof Targets

Clause $A_1 \wedge A_2 \Rightarrow B_1 \vee B_2$.

- Syntactic sugar for an "undirected" disjunction:

  $$\neg A_1 \vee \neg A_2 \vee B_1 \vee B_2$$

- Each atom becomes target of a proof rule:

  $$\begin{array}{lcl} A_2 \wedge \neg B_1 \wedge \neg B_2 & \Rightarrow & \neg A_1 \\ A_1 \wedge \neg B_1 \wedge \neg B_2 & \Rightarrow & \neg A_2 \\ A_1 \wedge A_2 \wedge \neg B_2 & \Rightarrow & B_1 \\ A_1 \wedge A_2 \wedge \neg B_1 & \Rightarrow & B_2 \end{array}$$

  - May lead to proof attempts that are unlikely to succeed.

- Clause $A_1\{*\} \wedge A_2\{*\} \Rightarrow B_1 \vee B_2\{*\}$ with atoms marked as "non-goals" $\{*\}$.
  - Only proof rule: $A_1 \wedge A_2 \wedge \neg B_2 \Rightarrow B_1$

```
axiom §trichotomy ⇔ ∀x:Int,y:Int. (((x < y) ∨ (y < x)) ∨ '=§0'(x,y){*});
```

Sacrifice completeness for efficiency.

Conclusions

- Effective solutions of various proof problems:
  - All equality problems in [Harrison].
  - The array and list examples from the RISCTP manual.
  - Some more examples on rewriting and basic arithmetic.
  - Sometimes competitive with SMT (often slower, also due to iterative deepening).
- Next steps:
  - Integration with RISCAL.
  - Application to RISCAL verification problems.
  - Comparison with Viktoria Langenreither's work.

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