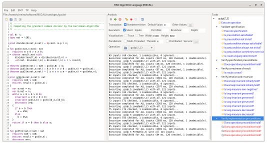
THE RISCTP SOFTWARE

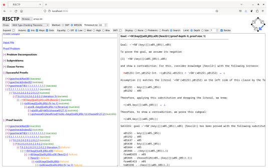
Combining Multiple Proving Strategies

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

JOHANNES KEPLER
UNIVERSITY LINZ

RISCAL & RISCTP





- RISCAL Language and Software
  - Variant of FOL over finite domains of some size N.
  - Rich variety of mathematical constructions and types.
  - Fixed size $N := c$: model checking.
  - Arbitrary size $N \in \mathbb{N}$: theorem proving.
- RISCTP Theorem Proving Interface
  - Language with abstraction level lower than RISCAL.
  - FOL with equality, integers, maps (arrays, sets), algebraic data types (tuples).
  - Interface to SMT-LIB based theorem provers (cvc5, Vampire, Z3).
  - MESON prover for FOL with support for above theories.
    - Construction and visualization of human-understandable proofs.

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

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 = choose i:Index with from ≤ i ∧ i ≤ to in
    key(a[i]) < x ⇒ ¬∃j:Index. from ≤ j ∧ j < i ∧ key(a[j]) = x;
```

Baseline Goal

Automatically generate "reasonably understandable" proofs for the verification conditions generated by RISCAL for programs that operate on arrays.

- Minimum/maximum element and position.
- Summation.
- Linear and binary search.
- Sorting.
- ...

Typical problems presented in my "Formal Methods" course; now handled in RISCAL by checking finite models via state space enumeration or SMT solving and in RISCTP by applying external SMT-based provers as "black boxes".

The Processing Pipeline

RISCAL

↓ Language Reduction

RISCTP

↓ Language Reduction

First-Order Logic

↓ Problem Decomposition/Simplification/Knowledge Generation

Sequent Problems

↓ Clause Generation

Clause Problems

↓ MESON ↔ SMT Solving, ↔ Equality Handling (Viktoria: Resolution)

Proofs

This presentation focuses on the relationship of MESON, SMT solving, equality handling, and problem decomposition/simplification/knowledge generation.

The Core: A MESON Prover

MESON: Model Elimination, Subgoal-Oriented.

- Judgment $Rs \vdash^{Ls}_{\sigma} G$: $(\bigwedge(Rs \cup Ls) \Rightarrow G)\sigma$ is valid.
  - Set Rs of "rules" $(\forall x \dots)(A_1 \wedge \dots \wedge A_a \Rightarrow B_1 \vee \dots \vee B_b)$ [$= (\forall x \dots)(L_1 \vee \dots \vee L_{a+b})$].
    - Atoms A_i, B_i, positive or negative atoms (literals) L_i.
  - "Goal" $G = (\exists x \dots)(G_1 \wedge \dots \wedge G_g)$ with literals G_i.
  - Set Ls of literals, variable substitution σ.

$$\frac{}{Rs \vdash^{Ls}_{\sigma} \top}\ \text{(AX)} \qquad \frac{L \in Ls \qquad G_1\sigma \text{ and } L\sigma \text{ have mgu } \sigma_1 \qquad Rs \vdash^{Ls}_{\sigma\sigma_1} (G_2 \wedge \dots \wedge G_g)}{Rs \vdash^{Ls}_{\sigma} (G_1 \wedge G_2 \wedge \dots \wedge G_{g\geq 1})}\ \text{(ASS)}$$

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

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):

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

- 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.
  - All clauses arising from the theorem to be proved have to be attempted (but not the clauses arising from theory axioms provided that they are satisfiable).

An intuitively understandable strategy.

A Note on Proofs by Cases

$$Rs := \{p \vee q, p \Rightarrow r, q \Rightarrow r\} \quad G := r$$

- Natural style reasoning: we have $p \vee q$.
  - In case of p, $(p \Rightarrow r)$ implies r.
  - In case of q, $(q \Rightarrow r)$ implies r.
- MESON pursues goal sequence $r \rightarrow p \rightarrow \neg q \rightarrow \neg r$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{Rs \vdash^{\{\neg r, \neg p, q\}} \neg r}\ (\text{ASS})}{Rs \vdash^{\{\neg r, \neg p\}} \neg q}\ (q \Rightarrow r)}{Rs \vdash^{\{\neg r\}} p}\ (p \vee q)}{Rs \vdash^{\emptyset} r}\ (p \Rightarrow r)$$

  - The case condition $(p \vee q)$ “inverts” the proof direction.

MESON cannot apply “case distinction” (the sequent calculus “cut rule”) to split proof situations (a “deficiency” mitigated a bit by some measures shown later).

Theories: SMT Solving

Especially consider theory symbols, i.e., symbols with "fixed" interpretation.

$$\frac{(\bigwedge(Rs) \wedge \bigwedge(Ls)\sigma \wedge \neg G_1\sigma) \text{ is unsatisfiable} \quad Rs \vdash_{\sigma}^{Ls} (G_2 \wedge \ldots \wedge G_g)}{Rs \vdash_{\sigma}^{Ls} G := (G_1 \wedge G_2 \wedge \ldots \wedge G_g)} \text{ (SMT)}$$

- $(\bigwedge(Rs) \wedge \bigwedge(Ls)\sigma \wedge \neg G_1\sigma)$ is unsatisfiable:
  - Consider only unquantified (variable-free) clauses from Rs.
  - Replace variables in $\bigwedge(Ls)\sigma \wedge \neg G_1\sigma$ by constants.
  - Result is a quantifier-free closed formula.
- RISCTP option "SMT":
  - Apply an external SMT solver (cvc5, Z3).
  - Unrestricted application slows down proof search substantially.
  - However, when applied up to depth 2 only, many proofs are sped up.

Still an explicit axiomatization of theories is needed to expose proof situations where a goal (G_1) follows from facts (Rs) and collected assumptions (Ls).

Axiomatization of Theories

- Maps/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
  - Axiomatization of constructor, selecter, tester operations. . .
- Integers
  - A (necessarily incomplete) axiomatization of the integer operations. . .

Axiomatization of Integers

```
axiom '0+1' ⇔ 0+1 = 1;
axiom '1+0' ⇔ 1+0 = 1;
axiom '+-1' ⇔ ∀x:Int. (x+1)-1 = x;
axiom '-+1' ⇔ ∀x:Int. (x-1)+1 = x;
axiom 'comm+' ⇔ ∀x:Int,y:Int. x+y = y+x;
axiom 'assoc+' ⇔ ∀x:Int,y:Int,z:Int. x+(y+z) = (x+y)+z;
axiom 'neut+' ⇔ ∀x:Int. x+0 = x;
axiom 'inv+' ⇔ ∀x:Int. x-x = 0;
axiom 'def-' ⇔ ∀x:Int,y:Int. x-y = x+(-y);
axiom 'inv-' ⇔ ∀x:Int. -(-x) = x;
axiom 'distrib-' ⇔ ∀x:Int,y:Int. -(x+y) = (-x)+(-y);

axiom 'div2a' ⇔ ∀x:Int,y:Int. let z = (x+y)/2 in x ≤ y ⇒
  ('='(x,y) {*} ∧ z = x ∧ z = y) ∨ (x < y ∧ x ≤ z ∧ z < y);
axiom 'div2b' ⇔ ∀x:Int,y:Int. let z = (x+y)/2 in
  x ≤ y ⇒ x ≤ z ∧ z ≤ y;

axiom 'preserve<+1' ⇔ ∀x:Int,y:Int,z:Int. x < y ⇒ x+z < y+z;
axiom 'preserve<+2' ⇔ ∀x:Int,y:Int,z:Int. x < y ⇒ z+x < z+y;
axiom 'preserve≤+1' ⇔ ∀x:Int,y:Int,z:Int.  x ≤ y ⇒ x+z ≤ y+z;
axiom 'preserve≤+2' ⇔ ∀x:Int,y:Int,z:Int. x ≤ y ⇒ z+x ≤ z+y;
axiom 'preserve<-' ⇔ ∀x:Int,y:Int,z:Int.  x < y ⇒ z-y < z-x;
axiom 'preserve≤-' ⇔ ∀x:Int,y:Int,z:Int.  x ≤ y ⇒ z-y ≤ z-x;
axiom 'add<' ⇔ ∀x:Int,y:Int.  0 < y ⇒ x < x+y;
axiom 'add≤' ⇔ ∀x:Int,y:Int.  0 ≤ y ⇒ x ≤ x+y;
```

```
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 'trich' ⇔ ∀x:Int,y:Int. x < y ∨ y < x  ∨ '='(x,y) {*} ;
axiom 'part1' ⇔ ∀x:Int,y:Int. x ≤ y ∨ y < x;
axiom 'part2' ⇔ ∀x:Int,y:Int. ¬(x ≤ y ∧ y < x);
axiom 'def1≤'  ⇔ ∀x:Int,y:Int. x < y ∨ x = y ⇒ x ≤ y ;
axiom 'def2≤'  ⇔ ∀x:Int,y:Int. x ≤ y ⇒ x < y ∨ '='(x,y) {*} ;
axiom 'excl<' ⇔ ∀x:Int,y:Int. ¬(x < y ∧ x = y);
axiom 'excl2<' ⇔ ∀x:Int,y:Int. ¬(y < x ∧ x = y);
axiom '+-1<' ⇔ ∀x:Int,y:Int. '<'(x,y){*} ⇒ ¬(y < x+1) ∧ ¬(y-1 < x);
axiom '+1≤' ⇔ ∀x:Int,y:Int. x < y ⇔ x+1 ≤ y;
axiom '+1<' ⇔ ∀x:Int,y:Int. x ≤ y ⇔ x < y+1 ;
axiom '-1≤' ⇔ ∀x:Int,y:Int. x < y ⇔ x ≤ y-1;
axiom '-1<' ⇔ ∀x:Int,y:Int. x ≤ y ⇔ x-1 < y;
axiom 'x-1<x' ⇔ ∀x:Int. x-1 < x;
axiom 'x<x+1' ⇔ ∀x:Int. x < x+1;
axiom '≤0' ⇔ ∀x:Int. 0 ≤ x ⇒ -x ≤ 0;
axiom '<0' ⇔ ∀x:Int. 0 < x ⇒ -x < 0;
axiom 'x≤y' ⇔ ∀x:Int,y:Int. x ≤ y ⇒ 0 ≤ y-x;
axiom 'x<y' ⇔ ∀x:Int,y:Int. x < y ⇒ 0 < y-x;
axiom '0≤0' ⇔ 0 ≤ 0;
axiom '0<1' ⇔ 0 < 1;
axiom '-1<0' ⇔ -1 < 0;
axiom 'irrefl<' ⇔ ∀x:Int. ¬(x < x);
axiom 'refl≤' ⇔ ∀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 'trich' ⇔ ∀x:Int,y:Int. x < y ∨ y < x  ∨ '='(x,y) {*} ;
```

Without this, the proof search space may explode.

Equality: Paramodulation-Style Rewriting

A natural adaptation of rule (MESON).

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

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

Search space explodes; application of the rule has to be appropriately limited.

Rewriting Control

- Avoid rewrite cycles: if t_1 has been rewritten to t_2, do not rewrite t_2 to t_1 in same proof branch.
- Do not apply non-goals: ignore equalities marked as $\{*\}$.
- Restrict rewrite positions: only consider term positions in uninstantiated literal G_i (not in $G_i\sigma$).
- Prohibit variable rewrites: do not rewrite variable x to some term t.
- 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: "Off" (no rewriting), "Min" (rewriting with all restrictions, the default), "Med" (also consider non-goals, do not restrict rewrite positions), "High" (also allow variable rewrites), "Max" (also do not direct equations).

More Equality Rules

Actually RISCTP also implements the following rules.

$$\frac{t\sigma = s\sigma \quad Rs \vdash^{Ls}_{\sigma} G}{Rs \vdash^{Ls}_{\sigma} (t = s) \wedge G} \text{ (EQAX)} \qquad \frac{x \notin sup(\sigma) \quad x \neq t \quad Rs \vdash^{Ls}_{\sigma[x \mapsto t\sigma]} G}{Rs \vdash^{Ls}_{\sigma} (x = t) \wedge G} \text{ (EQSUBST)}$$

$$\frac{t\sigma \neq s\sigma \quad Rs \vdash^{Ls}_{\sigma} (t = s) \wedge G}{Rs \vdash^{Ls}_{\sigma} f(t_1, \ldots, t, \ldots, t_n) = f(t_1, \ldots, s, \ldots, t_n) \wedge G)} \text{ (FEQ)}$$

$$\frac{\begin{array}{c}\neg(t{:}\mathsf{Int}) \quad R := (L_1 \vee \ldots \vee G_1[s] \vee \ldots \vee L_{a+b}) \in F \quad t\sigma \neq s\sigma \\ Rs \vdash^{Ls}_{\sigma} (\overline{L_1} \wedge \ldots \wedge \overline{L_{i-1}} \wedge \overline{L_{i+1}} \wedge \ldots \wedge \overline{L_{a+b}} \wedge (s = t)) \quad Rs \vdash^{Ls}_{\sigma} (G_2 \wedge \ldots \wedge G_g)\end{array}}{Rs \vdash^{Ls}_{\sigma} (G_1[t] \wedge G_2 \wedge \ldots \wedge G_{g \geq 1})} \text{ (EQ)}$$

$$\frac{\begin{array}{c}t{:}\mathsf{Int} \quad R := (L_1 \vee \ldots \vee G_1[s] \vee \ldots \vee L_{a+b}) \in F \quad t\sigma \neq s\sigma \\ Rs \vdash^{Ls}_{\sigma} (\overline{L_1} \wedge \ldots \wedge \overline{L_{i-1}} \wedge \overline{L_{i+1}} \wedge \ldots \wedge \overline{L_{a+b}} \wedge (s \leq t) \wedge \neg(s < t)) \quad Rs \vdash^{Ls}_{\sigma} (G_2 \wedge \ldots \wedge G_g)\end{array}}{Rs \vdash^{Ls}_{\sigma} (G_1[t] \wedge G_2 \wedge \ldots \wedge G_{g \geq 1})} \text{ (LEQ)}$$

The application of rule (LEQ) leads in the subsequent proof to a “goal split” based on the relative order of the values of integer terms t and s.

Completeness of Equality Reasoning

Does all of this make the equality reasoning complete?

- Resolution: paramodulation is complete.
  - Provided that we add the reflexivity axiom $x = x$ and one function reflexivity axiom $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$ for every function symbol f.
- MESON: paramodulation-style rewriting is incomplete.
  - $Rs := \{p(x) \Rightarrow f(x) = c, p(x) \Rightarrow g(x) = c, \neg p(x) \Rightarrow f(x) = d, \neg p(x) \Rightarrow g(x) = d\}$, $G := f(x) = g(x)$
  - Resolution: can derive from Rs the knowledge $p(x) \Rightarrow f(x) = g(x)$ and $\neg p(x) \Rightarrow f(x) = g(x)$ and from this the goal G.
  - MESON: no proof can be found (from any clause as a starting point).

Unclear (to me) whether/how extension to a complete calculus is possible that preserves the goal-directed flavor of MESON.

Problem Decomposition

Before applying MESON, a decomposition of the proof problem according to the rules of the sequent calculus is performed.

$$\frac{\Gamma, \Delta \vdash A, \Lambda}{\Gamma, \neg A, \Delta \vdash \Lambda}\ (\neg\text{-L}) \qquad \frac{A, \Gamma \vdash \Delta, \Lambda}{\Gamma \vdash \Delta, \neg A, \Lambda}\ (\neg\text{-R})$$

$$\frac{\Gamma, A, B, \Delta \vdash \Lambda}{\Gamma, A \wedge B, \Delta \vdash \Lambda}\ (\wedge\text{-L}) \qquad \frac{\Gamma \vdash \Delta, A, \Lambda \quad \Gamma \vdash \Delta, B, \Lambda}{\Gamma \vdash \Delta, A \wedge B, \Lambda}\ (\wedge\text{-R})$$

$$\frac{\Gamma, A, \Delta \vdash \Lambda \quad \Gamma, B, \Delta \vdash \Lambda}{\Gamma, A \vee B, \Delta \vdash \Lambda}\ (\vee\text{-L}) \qquad \frac{\Gamma \vdash \Delta, A, B, \Lambda}{\Gamma \vdash \Delta, A \vee B, \Lambda}\ (\vee\text{-R})$$

$$\frac{\Gamma, \Delta \vdash A, \Lambda \quad \Gamma, B, \Delta \vdash \Lambda}{\Gamma, A \Rightarrow B, \Delta \vdash \Lambda}\ (\Rightarrow\text{-L}) \qquad \frac{A, \Gamma \vdash \Delta, B, \Lambda}{\Gamma \vdash \Delta, A \Rightarrow B, \Lambda}\ (\Rightarrow\text{-R})$$

$$\frac{\Gamma, A[y/x], \Delta \vdash \Lambda}{\Gamma, (\exists x.\ A), \Delta \vdash \Lambda}\ (\exists\text{-L}) \qquad \frac{\Gamma \vdash \Delta, A[y/x], \Lambda}{\Gamma \vdash \Delta, (\forall x.\ A), \Lambda}\ (\forall\text{-R})$$

Resulting formulas are either atomic or quantified.

Problem Simplification and Knowledge Generation

In the presence of integer axioms, MESON proof search is only realistic up to depth 4 or so; thus proof problems have to be considerably simplified before/in the decomposition stage.

- Reduce operations: $>$, $\geq$, $\neq$ are reduced to $<$, $\leq$, $=$.
- Inline explicitly defined constants/functions: application $f(t)$ is replaced by $s[t]$.
- Insert axioms for implicitly defined functions: application $f(t)$ yields knowledge $F[t]$.
- Close the proof: apply axioms $(\Gamma, A, \Delta \vdash \Lambda, A, \Phi)$, $(\Gamma, \bot, \Delta \vdash \Lambda)$, $(\Gamma \vdash \Delta, \top, \Lambda)$.
- Cleanup the proof: apply rules $(\Gamma, \top, \Delta \vdash \Lambda) \rightarrow (\Gamma, \Delta \vdash \Lambda)$ and $(\Gamma \vdash \Delta, \bot, \Lambda) \rightarrow (\Gamma \vdash \Delta, \Lambda)$.
- Simplify formulas: apply (theory) knowledge to reduce (sub)formula to $\top/\bot$ and simplify result.
- Split arithmetic cases: replace $(t < s + 1)$ by $(t < s \vee t = s)$ and $(t \leq s + 1)$ by $(t \leq s \vee t = s + 1)$.
- Reduce arithmetic cases: replace knowledge $(t \leq s)$ and $\neg(t < s)$ by $t = s$.
- Normalize arithmetic equalities/inequalities: e.g., $a - b < a - c$ is transformed to $c < b$.
- Simplify arithmetic inequalities: replace $t \leq u + 1$ by $t < u$.
- Generalize arithmetic non-equalities: extend knowledge $t < u$ by $\neg(t = u)$ and $\neg(u = t)$.
- Apply arithmetic transitivity: extend, e.g., knowledge $t \leq s$ and $s < u$ by $t < u$.

Generate smaller problems with more knowledge; close simple problems.

Conclusions

What I (believe to) have learned so far...

- Pure first-order proving is *comparatively* simple (with the RISCTP implementation of MESON all proofs from Harrison Chapter 3 can be quickly found).
- However, in the presence of integer arithmetic, the "backward" proof search of MESON has to be complemented with "forward" proof decomposition, simplification, knowledge generation to be effective.
- SMT solving can be indeed helpful to enable/speed up some proofs; however with forward knowledge generation the direct use of integer rules is often competitive (at least for simple problems).
- Equality reasoning is the hardest part; it depends on a tricky trade-off between efficiency (reduce the space of applicability of rewriting rules) and reasoning strength (preserve the important rewrites).

Many of the stated goal problems can now be solved, I hope to soon provide a suitable release of RISCAL/RISCTP for my next semester's course.

Demo