

THE RISCTP SOFTWARE

Some Recent Developments...



Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC)

Johannes Kepler University Linz, Austria



The RISCTP Theorem Proving Interface

An abstraction layer for equipping RISCAL with theoring 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.**
 - Currently: translation to SMT-LIB using full quantification and SMT-LIB logics `ArraysEx` and `Ints`; connection to SMT solvers/provers `cvc5`, `Vampire`, `Z3`.
 - Ongoing: internal provers (with interfaces to SMT solvers) 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  $\Leftrightarrow$  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)  $\Leftrightarrow$ 
   $\forall i,j:Index. \text{from} \leq i \wedge i < j \wedge j \leq \text{to} \Rightarrow \text{key}(a[i]) \leq \text{key}(a[j]);$ 
theorem T  $\Leftrightarrow$ 
   $\forall a:Array,from:Index,to:Index,x:Int.$ 
     $\text{from} \leq \text{to} \wedge \text{sorted}(a,from,to) \Rightarrow$ 
      // let i = (from+to)/2 in
      let i = choose i:Index with  $\text{from} \leq i \wedge i \leq \text{to}$  in
       $\text{key}(a[i]) < x \Rightarrow \neg \exists j:Index. \text{from} \leq j \wedge j < i \wedge \text{key}(a[j]) = x;$ 
```

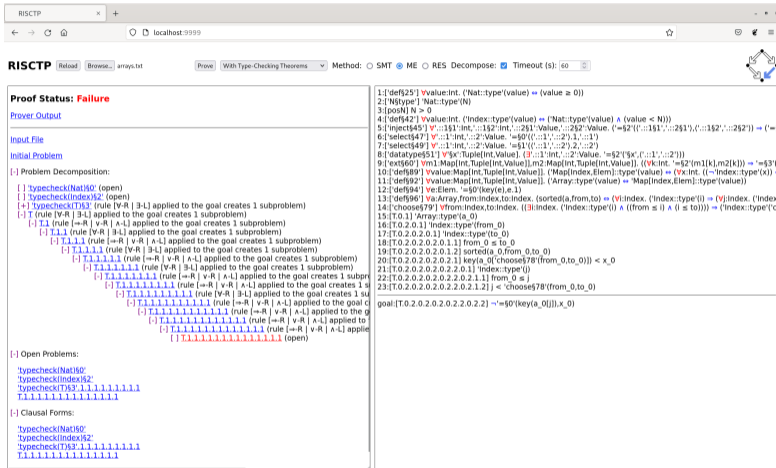
Translation to the SMT-LIB language for SMT solving and to classical first-order logic for theorem proving.

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.

Starting point of first-order proofs.

RISCTP GUI



The screenshot displays the RISCTP web-based interface. At the top, there's a browser window with the URL 'localhost:9999'. Below the browser, the RISCTP application header includes a 'Reload' button, a 'Browse...' button, and a file path 'arrays.tst'. The 'Method' dropdown is set to 'SMT', and the 'Timeout (s)' is set to '60'. A 'Prove' button is visible.

The main content area is divided into two panels. The left panel, titled 'Proof Status: Failure', contains the following information:

- Prover Output:** A link to view the prover's output.
- Input File:** A link to view the input file.
- Initial Problem:** A link to view the initial problem.
- Problem Decomposition:** A list of goals and their decomposition into subproblems, represented by a tree structure of '+' and '-' signs.
- Open Problems:** A list of open problems, including 'typecheck(Nat150)', 'typecheck(Index152)', and 'typecheck(T153)'. The latter two are followed by a long sequence of '1' characters.
- Clausal Forms:** A list of clausal forms, including 'typecheck(Nat150)', 'typecheck(Index152)', and 'typecheck(T153)'. The latter two are followed by a long sequence of '1' characters.

The right panel displays a complex proof script in a monospace font, consisting of several lines of code defining variables and goals. The script includes:

```
1:[def$25] vvalue:INT. ('Nat::type'(value) ==> (value >= 0))
2:['N8type'] 'Nat::type'(N)
3:[posN] N > 0
4:[def$42] vvalue:INT. ('Index::type'(value) ==> ('Nat::type'(value) & (value < N)))
5:[inject$45] v:::1$1'int,:::152'int,:::251'Value,:::252'Value. ('==52(':::151',:::251'),(':::152',:::252)) -> ('==52(':::151',:::251)')
6:[select$47] v:::1'int,:::2'Value. ==50(':::1',:::2),1,:::1)
7:[select$49] v:::1'int,:::2'Value. ==51(':::1',:::2),2,:::2)
8:[datatypes51] v$X:Tuple(INT,Value). (3,:::1'int,:::2'Value. ==52('5x',(':::1',:::2)))
9:[datatypes51] vm1:Map(INT,Tuple(INT,Value)),m2:Map(INT,Tuple(INT,Value)). (('vk:int. ==52('m1[k],m2[k])) -> '=53('m1[k],m2[k]))
10:[def$89] vvalue:Map(INT,Value). ('Map(Index,Elem)::type'(value) ==> (vk:int. (vk -> Index::type'(x)) -> v))
11:[def$92] vvalue:Map(INT,Value). ('Array::type'(value) ==> 'Map(Index,Elem)::type'(value))
12:[def$94] ve:Elem. ==50('key(e),e.1)
13:[def$96] va:Array:from:Index:to:Index. (sorted(a,from,to) ==> (vi:Index. ('Index::type'(i) ==> (vj:Index. ('Index::type'(j) ==> (i ==> v[j])))))
14:[choose$79] v:from:Index:to:Index. ((3i:Index. ('Index::type'(i) & ((from <= i) & (i <= to))) -> 'Index::type'(i))
15:[T.0.1] 'Array::type'(a_0)
16:[T.0.2.0.1] 'Index::type'(from_0)
17:[T.0.2.0.2.0.1] 'Index::type'(to_0)
18:[T.0.2.0.2.0.2.0.1.1] from_0 <= to_0
19:[T.0.2.0.2.0.2.0.1.2] sorted(a_0,from_0,to_0)
20:[T.0.2.0.2.0.2.0.2.1] key(a_0['choose$78'(from_0,to_0)]) < x_0
21:[T.0.2.0.2.0.2.0.2.0.1] 'Index::type'(i)
22:[T.0.2.0.2.0.2.0.2.0.2.1.1] from_0 <= j
23:[T.0.2.0.2.0.2.0.2.0.2.1.2] j < 'choose$78'(from_0,to_0)
goal:[T.0.2.0.2.0.2.0.2.0.2.2] '-==50('key(a_0)[],x_0)
```

A web-based frontend to monitor prover and inspect proofs.