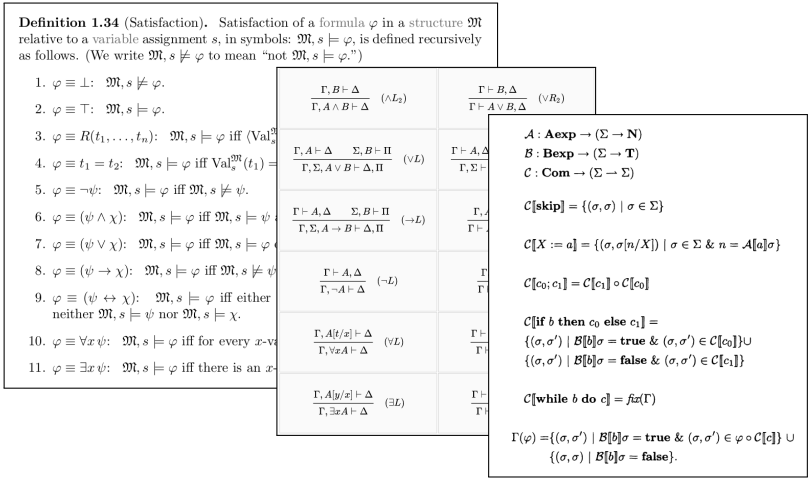
THE RISCAL MODEL CHECKER

Teaching Logic, Formalization, and Verification by Analyzing Theories and Algorithms in Finite Domains

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

JOHANNES KEPLER UNIVERSITY LINZ

Formal Modeling & Reasoning in Education



Definition 1.34 (Satisfaction). Satisfaction of a formula φ in a structure $\mathfrak{M}$ relative to a variable assignment s, in symbols: $\mathfrak{M}, s \models \varphi$, is defined recursively as follows. (We write $\mathfrak{M}, s \not\models \varphi$ to mean "not $\mathfrak{M}, s \models \varphi$.")

1. $\varphi \equiv \bot$: $\mathfrak{M}, s \not\models \varphi$.
2. $\varphi \equiv \top$: $\mathfrak{M}, s \models \varphi$.
3. $\varphi \equiv R(t_1, \ldots, t_n)$: $\mathfrak{M}, s \models \varphi$ iff $\langle \mathrm{Val}_s^{\mathfrak{M}}$
4. $\varphi \equiv t_1 = t_2$: $\mathfrak{M}, s \models \varphi$ iff $\mathrm{Val}_s^{\mathfrak{M}}(t_1) =$
5. $\varphi \equiv \neg\psi$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \not\models \psi$.
6. $\varphi \equiv (\psi \wedge \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \models \psi$
7. $\varphi \equiv (\psi \vee \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \models \varphi$
8. $\varphi \equiv (\psi \to \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \not\models \psi$
9. $\varphi \equiv (\psi \leftrightarrow \chi)$: $\mathfrak{M}, s \models \varphi$ iff either neither $\mathfrak{M}, s \models \psi$ nor $\mathfrak{M}, s \models \chi$.
10. $\varphi \equiv \forall x\, \psi$: $\mathfrak{M}, s \models \varphi$ iff for every x-va
11. $\varphi \equiv \exists x\, \psi$: $\mathfrak{M}, s \models \varphi$ iff there is an x-

$$\frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ (\wedge L_2) \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ (\vee R_2)$$

$$\frac{\Gamma, A \vdash \Delta \qquad \Sigma, B \vdash \Pi}{\Gamma, \Sigma, A \vee B \vdash \Delta, \Pi}\ (\vee L) \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma, \Sigma \vdash}$$

$$\frac{\Gamma \vdash A, \Delta \qquad \Sigma, B \vdash \Pi}{\Gamma, \Sigma, A \to B \vdash \Delta, \Pi}\ (\to L) \qquad \frac{\Gamma, A}{\Gamma \vdash}$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\ (\neg L) \qquad \frac{\Gamma}{\Gamma}$$

$$\frac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\ (\forall L) \qquad \frac{\Gamma \vdash}{\Gamma}$$

$$\frac{\Gamma, A[y/x] \vdash \Delta}{\Gamma, \exists x A \vdash \Delta}\ (\exists L) \qquad \frac{\Gamma \vdash}{\Gamma}$$

$\mathcal{A} : \mathbf{Aexp} \to (\Sigma \to \mathbf{N})$
$\mathcal{B} : \mathbf{Bexp} \to (\Sigma \to \mathbf{T})$
$\mathcal{C} : \mathbf{Com} \to (\Sigma \to \Sigma)$

$\mathcal{C}[\![\mathbf{skip}]\!] = \{(\sigma, \sigma) \mid \sigma \in \Sigma\}$

$\mathcal{C}[\![X := a]\!] = \{(\sigma, \sigma[n/X]) \mid \sigma \in \Sigma \ \&\ n = \mathcal{A}[\![a]\!]\sigma\}$

$\mathcal{C}[\![c_0; c_1]\!] = \mathcal{C}[\![c_1]\!] \circ \mathcal{C}[\![c_0]\!]$

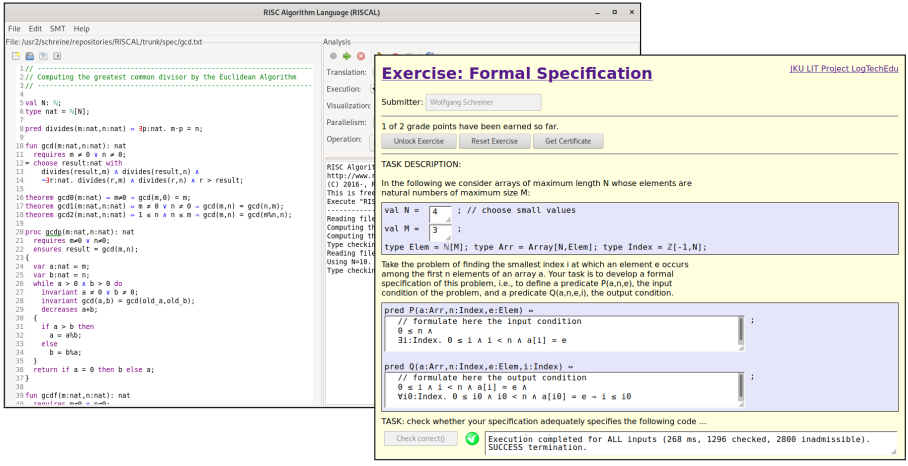
$\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!] =$
$\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{true}\ \&\ (\sigma, \sigma') \in \mathcal{C}[\![c_0]\!]\} \cup$
$\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{false}\ \&\ (\sigma, \sigma') \in \mathcal{C}[\![c_1]\!]\}$

$\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!] = \mathit{fix}(\Gamma)$

$\Gamma(\varphi) = \{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{true}\ \&\ (\sigma, \sigma') \in \varphi \circ \mathcal{C}[\![c]\!]\} \cup$
$\{(\sigma, \sigma) \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{false}\}.$

Typically presented as "paper and pencil" topics.

Formal Modeling & Reasoning in Education



But today the educational process can be substantially supported by *software*.

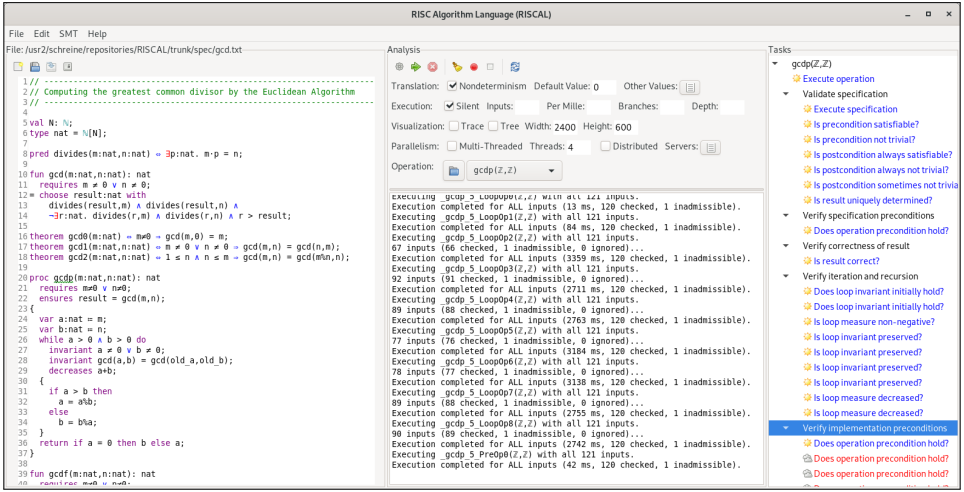
The RISCAL Software

A model checker for discrete mathematical theories and algorithms.

- Theory and algorithm language ("RISC Algorithm Language").
  - Strongly typed variant of first-order logic and set theory.
  - Integers, tuples/records, sets, arrays/maps, algebraic types.
  - Parameterized finite domains: decidable theories and algorithms.
- Purpose: quick validation/falsification of theories and algorithms.
  - General verification problem: $\forall N \in \mathbb{N}.\ F[N]$.
  - Validated by $F[0], F[1], \ldots$.
  - Falsified by any $N \in \mathbb{N}$ with $\neg F[N]$.
- Alternative decision mechanisms.
  - Internal: semantic evaluation of formulas and algorithms.
  - External: translation of formulas to a decidable SMT-LIB logic ($\mathrm{QF\_UFBV}$).

Wolfgang Schreiner, Alexander Brunhuemer, Christoph Fürst: Teaching the Formalization of Mathematical Theories and Algorithms via the Automatic Checking of Finite Models. 6th International Workshop on Theorem proving components for Educational software (ThEdu'17), EPTCS 267, pp. 120-139., August 2018.

The RISCAL Software



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

Theories and Theorems

```
val N: ℕ;
type Literal = ℤ[-N,N] with value ≠ 0;
type Clause  = Set[Literal] with ∀l∈value. -l ∉ value;
type Formula = Set[Clause];
type Valuation  = Set[Literal] with ∀l∈value. -l ∉ value;

pred satisfies(v:Valuation, l:Literal) ⇔ l∈v;
pred satisfies(v:Valuation, c:Clause) ⇔ ∃l∈c. satisfies(v, l);
pred satisfies(v:Valuation, f:Formula) ⇔ ∀c∈f. satisfies(v,c);
pred satisfiable(f:Formula) ⇔ ∃v:Valuation. satisfies(v,f);
pred valid(f:Formula) ⇔ ∀v:Valuation. satisfies(v,f);
fun not(f: Formula):Formula = { c | c:Clause with ∀d∈f. ∃l∈d. -l∈c };

theorem notValid(f:Formula) ⇔ valid(f) ⇔ ¬satisfiable(not(f));
```

First-order logic, integers, tuples/records, arrays/maps, sets, algebraic types.

Declarative Algorithms

```
fun literals(f:Formula):Set[Literal] = {l | l:Literal with ∃c∈f. l∈c};

fun substitute(f:Formula,l:Literal):Formula = {c\{-l} | c∈f with ¬(l∈c)};

multiple pred DPLL(f:Formula)
  ensures result ⇔ satisfiable(f);
  decreases |literals(f)|;
⇔
  if f = ø[Clause] then
    ⊤
  else if ø[Literal] ∈ f then
    ⊥
  else
    choose l∈literals(f) in
    DPLL(substitute(f,l)) ∨ DPLL(substitute(f,-l));
```

Functions, predicates, implicitly defined constants and functions.

Imperative Algorithms

```
proc DPLL2(f:Formula): Bool
  ensures result ⇔ satisfiable(f);
{
  var satisfiable: Bool := ⊥;
  var stack: Array[N+1,Formula] := Array[N+1,Formula](ø[Clause]);
  var number: ℕ[N+1] := 0;
  stack[number] := f; number := number+1;
  while ¬satisfiable ∧ number>0 do
    invariant 0 ≤ number ∧ number ≤ N+1;
    invariant number > 0 ∧ stack[number-1] ≠ ø[Clause] ∧ ¬ø[Literal] ∈ stack[number-1] ⇒ number < N+1;
    invariant satisfiable(f) ⇔ satisfiable ∨ ∃i:ℕ[N+1] with i<number. satisfiable(stack[i]);
    decreases if satisfiable then 0 else ∑k:ℕ[N] with k<number. size(stack[k]);
  {
    number := number-1;
    var g:Formula := stack[number];
    if g = ø[Clause] then
      satisfiable := ⊤;
    else if ¬ø[Literal]∈g then
    {
      choose l∈literals(g);
      stack[number] := substitute(g,-l); number := number+1;
      stack[number] := substitute(g,l); number := number+1;
    }
  }
  return satisfiable;
}
```

Procedures, variables, loops.

RISCAL Checking

```
Using N=2.
Evaluating the domain of Clause...
Evaluating the domain of Valuation...
Computing the value of m...
Type checking and translation completed.

Executing notValid(Set[Clause]) with all 512 inputs.
Execution completed for ALL inputs (212 ms, 512 checked, 0 inadmissible).
Executing DPLL(Set[Clause]) with all 512 inputs.
Execution completed for ALL inputs (1036 ms, 512 checked, 0 inadmissible).
Executing DPLL2(Set[Clause]) with all 512 inputs.
483 inputs (483 checked, 0 inadmissible, 0 ignored)...
Execution completed for ALL inputs (2149 ms, 512 checked, 0 inadmissible).
Executing _DPLL_9_OutputCorrect0(Set[Clause]) with all 512 inputs.
Execution completed for ALL inputs (623 ms, 512 checked, 0 inadmissible).
```

Automatic checking of theorems, algorithms, generated verification conditions.

RISCAL Checking

```
Using N=2.
Type checking and translation completed.
The SMT solver Yices started execution.

Theorem _DPLL2_12_PostSat is valid.
Theorem _DPLL2_12_PostNotTrivialAll is valid.
Theorem _DPLL2_12_PostNotTrivialSome is valid.
Theorem _DPLL2_12_PostUnique is valid.
Theorem _DPLL2_12_CorrOp0 is valid.
Theorem _DPLL2_12_LoopOp0 is valid.
Theorem _DPLL2_12_LoopOp1 is valid.
Theorem _DPLL2_12_LoopOp2 is valid.
Theorem _DPLL2_12_LoopOp3 is valid.
Theorem _DPLL2_12_LoopOp4 is valid.
Theorem _DPLL2_12_LoopOp5 is valid.
...
Total time: 267 ms, translation: 231 ms, decision: 33 ms.
```

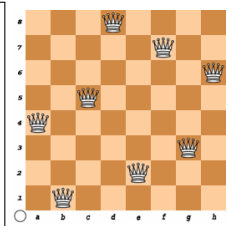
The checking of formulas can be considerably accelerated by SMT solving.

Transition Systems

```
type Num = ℕ[N];
type Queens = Array[N,Num];

pred admissible(q:Queens,n:Num,p:Num) ⇔
  n < N ∧ p < N ∧
  ∀i:Num with i < n. q[i] ≠ p ∧ q[i]-i ≠ p-n ∧ q[i]+i ≠ p+n;

shared system Queens
{
  var q:Queens = Array[N,Num](N);
  var n:Num = 0;
  invariant n = N ⇒ printtrace in printall;
  action place(p:Num) with admissible(q,n,p);
  {
    q[n] ≔ p;
    n ≔ n+1;
  }
}
```



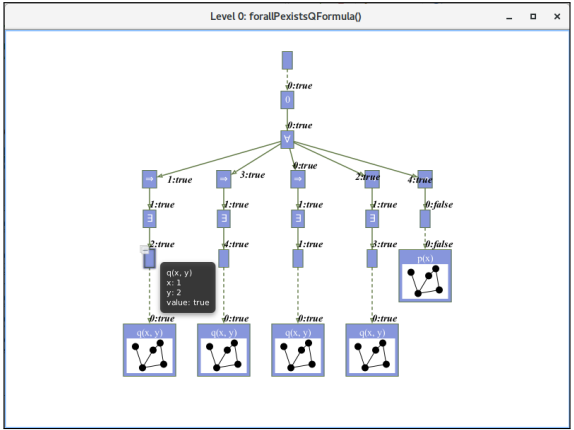
Nondeterministic systems defined by state transitions.

RISCAL Checking

```
Using N=4.
...
Type checking and translation completed.
Executing system Queens.
  0:[q:[4,4,4,4],n:0]->place(1)->
  1:[q:[1,4,4,4],n:1]->place(3)->
  2:[q:[1,3,4,4],n:2]->place(0)->
  3:[q:[1,3,0,4],n:3]->place(2)->
  4:[q:[1,3,0,2],n:4]
  0:[q:[4,4,4,4],n:0]->place(2)->
  1:[q:[2,4,4,4],n:1]->place(0)->
  2:[q:[2,0,4,4],n:2]->place(3)->
  3:[q:[2,0,3,4],n:3]->place(1)->
  4:[q:[2,0,3,1],n:4]
17 system states found with search depth 5.
Execution completed (69 ms).
```

The state space of the system is elaborated and investigated.

RISCAL Visualization



A pruned evaluation tree explains the truth value of a formula.

RISCAL Counterexample Generation

```
theorem _search_0_LoopOp6(a:array, x:elem) ⇔
  ∀i:int, r:int. ((((((0 ≤ i) ∧ (i ≤ N)) ∧ ...) ⇒
    (let i = i+1 in
      (∀j:int. (((0 ≤ j) ∧ (j < i)) ⇒ (a[j] ≠ x))))));

ERROR in execution of _search_0_LoopOp6([0,0],0): evaluation of
  _search_0_LoopOp6
at unknown position:
  theorem is not true
ERROR encountered in execution.

Executing __search_0_LoopOp6_refute().
This sequence of assignments leads to a counterexample
(note the underlined editor lines):
a=[0,0],x=0
i=0,r=-1
i=1
j=0
```

```
var i:int = 0;
var r:int = -1;
while i < N ∧ r = -1 do
  invariant 0 ≤ i ∧ i ≤ N;
  invariant ∀j:int. 0 ≤ j ∧ j < i ⇒ a[j] ≠ x;
  invariant r = -1 ∨ (r = i ∧ i < N ∧ a[r] = x);
  decreases if r = -1 then N-i else 0;
{
  if a[i] = x then r = i;
  i = i+1;
}
return r;
```

Core information to explain the invalidity of a formula.

Application: Analyzing Algorithms

```
proc gcdp(m:nat,n:nat): nat
  requires m≠0 ∨ n≠0;
  ensures result = gcd(m,n);
{
  var a:nat = m;
  var b:nat = n;
  while a > 0 ∧ b > 0 do
    invariant a ≠ 0 ∨ b ≠ 0;
    invariant gcd(a,b) = gcd(old_a,old_b);
    decreases a+b;
  {
    if a > b then
      a = a%b;
    else
      b = b%a;
  }
  return if a = 0 then b else a;
}
```


```
Executing gcdp(ℤ,ℤ) with all 121 inputs.
Execution completed for ALL inputs (172 ms, 120 checked, 1 inadmissible).
...
Executing _gcdp_5_PreOp3(ℤ,ℤ) with all 121 inputs.
87 inputs (86 checked, 1 inadmissible, 0 ignored)...
Execution completed for ALL inputs (2843 ms, 120 checked, 1 inadmissible).
```

Validating algorithms, their specification, annotations, verification conditions.

Application: Analyzing Theories

```
val N:ℕ; // variablex x0,...,xN
val M:ℕ; // values 0,...,M

type Var  = ℕ[N];          // a variable
type Val  = ℕ[M];          // a value
type Ass  = Map[Var,Val]; // an assignment of variables to values
type Pred = Set[Ass];      // a predicate as a set of assignments

val Ass = { a | a:Ass };

pred independent(P:Pred, x:Var) ⇔
  ∀a:Ass, v1:Val, v2:Val.
    (a with [x] = v1) ∈ P ⇔ (a with [x] = v2) ∈ P;

fun EXISTS(x:Var, P:Pred):Pred =
  { a | a:Ass with ∃v:Val. (a with [x] = v) ∈ P } ;
theorem Exists1(x:Var, P:Pred) ⇔
  ∀Q:Pred with independent(Q,x). Q = EXISTS(x,P) ⇔
    P ⊆ Q ∧ ∀Q0:Pred with independent(Q0,x). P ⊆ Q0 ⇒ Q ⊆ Q0;
theorem Exists2(x:Var, P:Pred) ⇔
  EXISTS(x,P) = ⋂{ Q | Q:Pred with independent(Q,x) ∧ P ⊆ Q };
```

W. Schreiner, W. Steingartner, V. Novitzká:
A Novel Categorical Approach to the Semantics of Relational First-Order Logic.
Symmetry 12(10), 1584, MDPI. September 2020.

```
Executing Exists1(ℤ,Set[Array[ℤ]]) with all 768 inputs.
Execution completed for ALL inputs (4311 ms, 768 checked, 0 inadmissible).
Executing Exists2(ℤ,Set[Array[ℤ]]) with all 768 inputs.
Execution completed for ALL inputs (1674 ms, 768 checked, 0 inadmissible).
```

Validating conjectures (respectively the formalization of theorems).

Model Checking by Semantic Evaluation

$$ComSem := Single + Multiple$$
$$Single := Command \to (Context \to Context)$$
$$Multiple := Command \to (Context \to Seq(Context))$$
$$Seq(T) := Unit \to (Null + Next(T, Seq(T)))$$

$$[\![\,.\,]\!] : \texttt{Command} \to Single$$
$$[\![\texttt{if } E \texttt{ then } C]\!] := \lambda c.\ \text{if } [\![E]\!](c) \text{ then } [\![C]\!](c) \text{ else } c$$

```
interface ComSem  {
  public interface Single extends ComSem, Function<Context,Context> { }
  public interface Multiple extends ComSem, Function<Context,Seq<Context> > { }
}
interface Seq<T> extends Supplier<Seq.Next<T> > { ... }

ComSem.Single ifThenElse(BoolExpSem.Single E, ComSem.Single C)
{ return (Context c) -> E.apply(c) ? C.apply(c) : c; }
```

Translation of every syntactic phrase (command, formula, . . .) to (an executable version of) its (potentially nondeterministic) denotational semantics.

Formula Decision by SMT Solving

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

- Translation of RISCAL theory to SMT-LIB.
  - Franz-Xaver Reichl (now Algorithms and Complexity Group, TU Wien).
- SMT-LIB logic QF_UFBV.
  - Quantifier-free formulas over bitvectors with uninterpreted functions.
  - Well supported by various SMT solvers: CVC4, Yices, Yices, Z3, ...
- Translation of integers, tuples/records, arrays/maps, sets, ... to bit vectors.
  - Non-trivial because, e.g., RISCAL uses “true” mathematical integers.
- Elimination of quantifiers by Skolemization and expansion.
  - Skolemization requires uninterpreted functions.

Much faster in many cases, but not in all.

Alternative Formula Decision Mechanisms

- Semantic Evaluation:

$[\![\forall x{:}D.\ F[x]]\!] :=$

```
e := enumerate(D)
loop
  if empty(e) then return true
  x := next(e); e := rest(e)
  if ¬call([[ F ]], x) then return false
```

$[\![\exists x{:}D.\ F[x]]\!] :=$

```
e := enumerate(D)
loop
  if empty(e) then return false
  x := next(e); e := rest(e)
  if call([[ F ]], x) then return true
```

- SMT Solving: encoding of theories and quantifier elimination.

$$\begin{aligned} valid[\![\forall x{:}D.\ F[x]]\!] &\equiv \neg sat[\![\neg \forall x{:}D.\ F[x]]\!] \\ &\equiv \neg sat[\![\exists x{:}D.\ \neg F[x]]\!] \equiv \neg sat[\![\neg F[f(x_1, \ldots, x_m)]]\!] \end{aligned}$$

$$\begin{aligned} valid[\![\exists x{:}D.\ F[x]]\!] &\equiv \neg sat[\![\neg \exists x{:}D.\ F[x]]\!] \\ &\equiv \neg sat[\![\forall x{:}D.\ \neg F[x]]\!] \equiv \neg sat[\![\neg F[e_1] \wedge \ldots \wedge \neg F[e_n]]\!] \end{aligned}$$

Potential Problems with SMT Solving

- Expansion of existential formulas:

$$valid[\![\, \exists x{:}D.\ F[x] \,]\!] \equiv \neg sat[\![\, \neg F[e_1] \wedge \ldots \wedge \neg F[e_n] \,]\!]$$

  - Expansion factor $n = |D|$.
- Domain axioms for Skolem functions:

$$valid[\![\, \forall x{:}D.\ F[x] \,]\!] \equiv \neg sat[\![\, \neg F[f(x_1, \ldots, x_m)] \,]\!]$$

  - In the SMT-LIB translation, the range of Skolem function f has to be constrained by an axiom to those bitvectors that indeed represent elements from D:

$$\bigwedge_{(d_1,\ldots,d_n)\in D^m} p_D(f(t(d_1), \ldots, t(d_m)))$$

- Translation of choice expressions to axiomatized functions:

$$(\text{choose } y{:}D.\ F[x, y]) \quad \rightsquigarrow \quad f_F(x)$$

  - Free variable $x{:}D$, fresh function $f_F{:}D \to D$ with axiom $\forall x{:}D.\ F[x, f_F(x)]$.
  - The expanded version of the axiom has to be added to the SMT-LIB encoding.
- Bitvector encodings of RISCAL types and operations.

Artificial Benchmarks: Quantification Patterns

We analyze the effect of various quantification patterns.

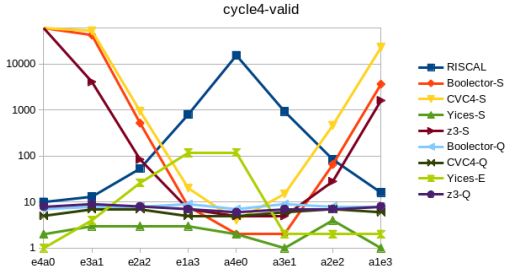
$$Q_{x_1,x_2,x_3,x_4}.\ F_{x_1,x_2,x_3,x_4}$$

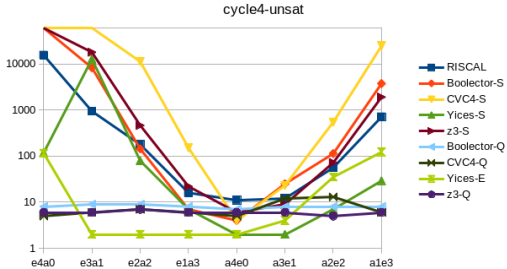
- Variables $x_1, x_2, x_3, x_4 \in D$ with $D := \mathbb{N}_{2^N}$ and $N := 6$.
- Quantifier prefix Q: $\exists^4\forall^0$, $\exists^3\forall^1$,$\exists^2\forall^2$, $\exists^1\forall^3$, $\forall^4\exists^0$, $\forall^3\exists^1$, $\forall^2\exists^2$, $\forall^1\exists^3$.
- Formula F: *cycle4-valid* or *cycle4-sat1*.

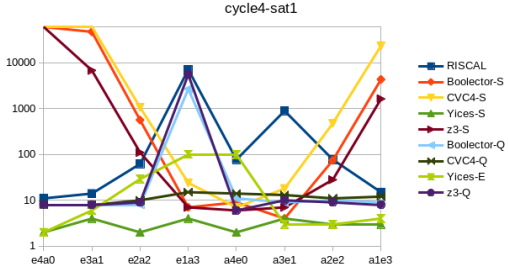
$$\textit{cycle4-valid} \equiv \neg(x_1 < x_2 \wedge x_2 < x_3 \wedge x_3 < x_4 \wedge x_4 < x_1)$$
$$\textit{cycle4-sat1} \equiv \neg(x_1 < x_2 \wedge x_2 < x_3 \wedge x_3 < x_4 \wedge x_4 < x_1 + 4)$$

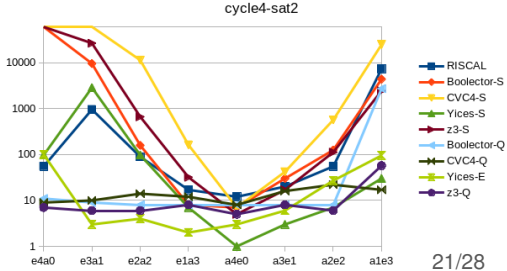
- Formula *cycle4-valid*: valid, its negation unsatisfiable.
- Formula *cycle4-sat1*: “mostly valid”, its negation “mostly unsatisfiable”.

Artificial Benchmarks: Quantification Patterns

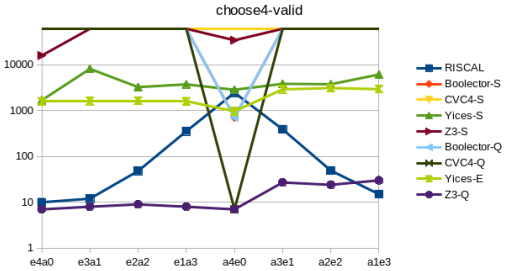


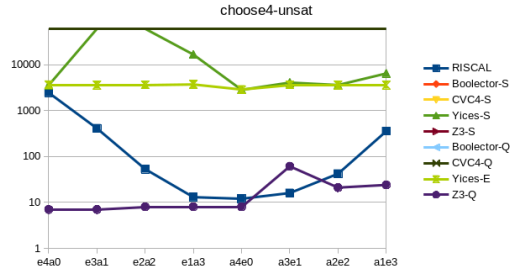


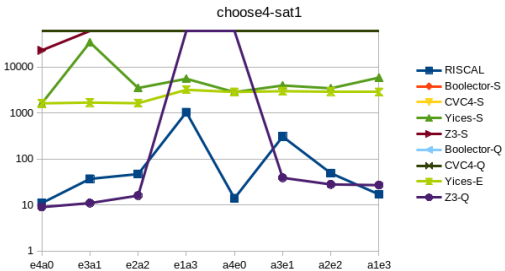


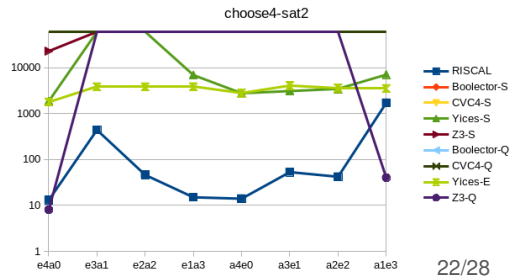


Artificial Benchmarks: Axiomatized Functions

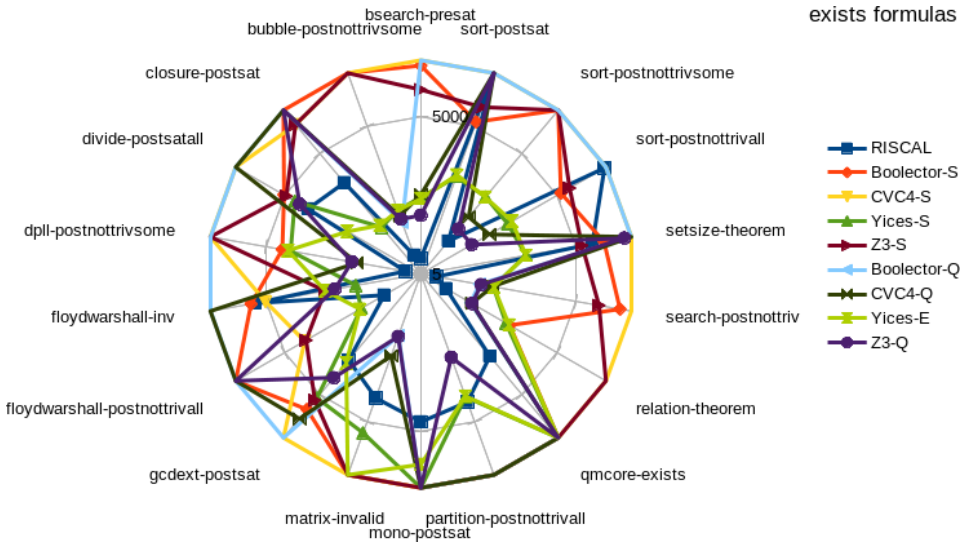




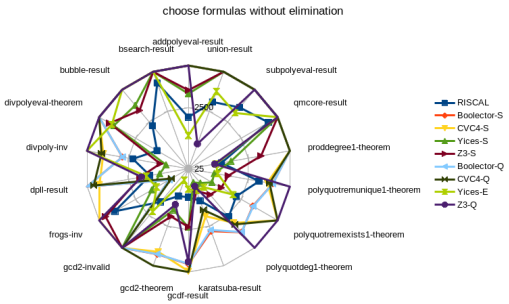


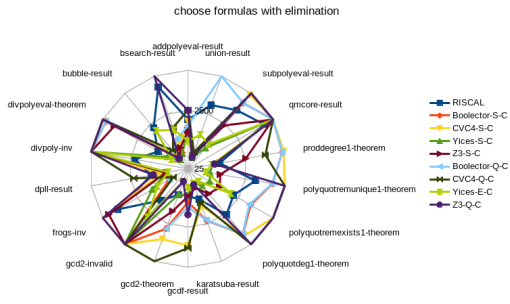


Real-Life Benchmarks: Existential Formulas



Real-Life Benchmarks: Axiomatized Functions





Optimization implemented to remove certain applications of axiomatized functions.

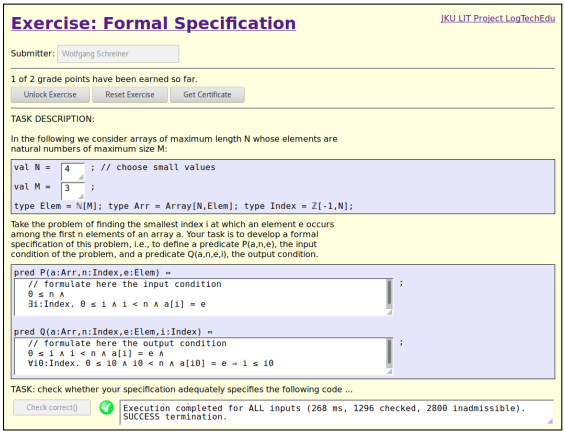
Wolfgang Schreiner, Franz-Xaver Reichl. First-Order Logic in Finite Domains: Where Semantic Evaluation Competes with SMT Solving. 9th International Symposium on Symbolic Computation in Software Science (SCSS 2021), EPTCS 342, pp. 99–113. September 2021.

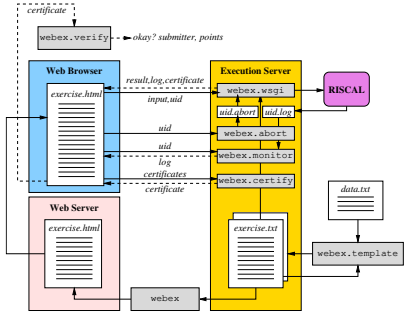
Educational Usage of RISCAL

- Course “Formal Methods in Software Development”
  - JKU master programs “Computer Science” and “Computer Mathematics”.
  - Formal problem specifications; specification and verification of imperative programs.
- Course “Formal Methods and Specification”
  - TU Prague (Stefan Ratschan) master program “Informatics”.
  - Formal specification and verification of imperative programs.
- Course “Formal Modeling”
  - JKU bachelor program “Technical Mathematics”
  - Modeling computational problems, search/scheduling problems (“puzzles”), transition systems.
- Course “Logic”.
  - JKU bachelor programs “Computer Science” and “Artificial Intelligence”.
  - Bonus (RISCAL web) and laboratory exercises (RISCAL desktop).
- Various Bachelor and Master Theses

Wolfgang Schreiner: Theorem and Algorithm Checking for Courses on Logic and Formal Methods. 7th International Workshop on Theorem proving components for Educational software (ThEdu’19), EPTCS 290, pp. 56-75, April 2019.

RISCAL Web Exercises





Framework for web-based exercises checked by a RISCAL server (another framework for RISCAL-enhanced educational material is under development).

RISCAL Experience

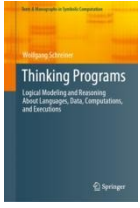
Observations, results of questionnaires, test/exam results.

- Students with some technical/formal background (2nd year and higher):
  - High satisfaction with ease of use.
  - Much more liked than “proof-based” logic software.
  - Many students were indeed enabled to independently develop adequate formal specifications, models, program annotations.
- Absolute beginners (1st semester):
  - Web interface is essential: self-installation of software is problematic.
  - Approx. 2/3 of the active students elaborate the RISCAL-based exercises.
  - Students that scored poorly in tests did mostly not do the exercises.
  - “Extrinsic motivation”: mainly used to get additional grade points.

From a certain background/level on, substantial increase in motivation and interest (but not a statistically significant effect on grades).

Conclusions and Future Work

- Model checking first-order logic over (small) finite domains may be beneficial:
  - Retains a formal framework that is familiar to most users.
  - Removes the uncertainty caused by undecidability over infinite domains.
    - Allows quick validation/falsification of theories and algorithms.
    - Yields further insight by the inspection of models.
  - However: susceptible to well-known “state explosion” problem.
    - Partially avertable by bitvector encoding and SMT solving.
- Future work:
  - RISCAL-enhanced course materials (lecture notes, exercises, . . .).
  - LTL model checking of RISCAL transition systems.
  - Addition of automated/interactive proving capabilities to RISCAL.
    - Generalization of verification to domains of arbitrary sizes.



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

Wolfgang Schreiner. Thinking Programs — Logical Modeling and Reasoning About Languages, Data, Computations, and Executions. Texts & Monographs in Symbolic Computation, Springer, 2021.