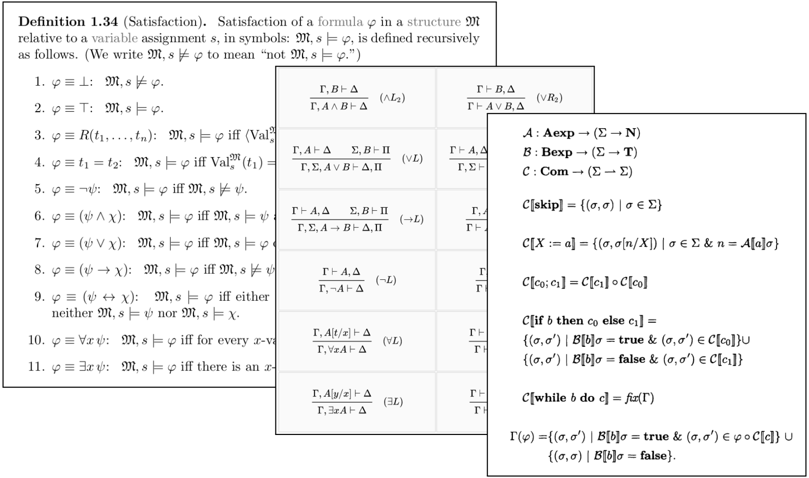
MATHEMATICAL MODEL CHECKING FOR COMPUTER SCIENCE EDUCATION

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. $\varphi \equiv R(t_1, \dots, t_n)$: $\mathfrak{M}, s \models \varphi$ 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. $\varphi \equiv (\psi \land \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \models \psi$ …
7. $\varphi \equiv (\psi \lor \chi)$: $\mathfrak{M}, s \models \varphi$ 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-…

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

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

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

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

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

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

$\mathcal{A} : \mathbf{Aexp} \to (\Sigma \to \mathbf{N})$

$\mathcal{B} : \mathbf{Bexp} \to (\Sigma \to \mathbf{T})$

$\mathcal{C} : \mathbf{Com} \to (\Sigma \rightharpoonup \Sigma)$

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

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

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

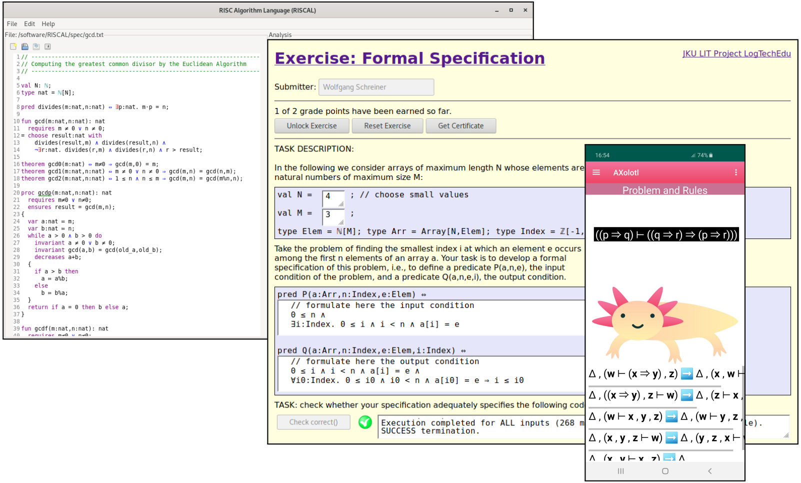
$\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!] =$
$\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{true} \;\&\; (\sigma, \sigma') \in \mathcal{C}[\![c_0]\!]\} \cup$
$\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{false} \;\&\; (\sigma, \sigma')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

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

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

Typically still presented as “paper and pencil” topics.

Formal Modeling&Reasoning in Education



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

Projects LOGTECHEDU and SemTech

- LOGTECHEDU: Logic Technologies for Computer Science Education.
  - JKU LIT (Linz Institute of Technology), 2018–2020.
  - Institutes FMV (Biere, Cerna, Seidl) and RISC (Schreiner, Windsteiger).
  - `http://fmv.jku.at/logtechedu`
- SemTech: Semantic Technologies for Computer Science Education.
  - Austrian OEAD WTZ and Slovak SRDA, 2018–2019.
  - JKU Linz (Schreiner) and TU Kosice (Novitzká, Steingartner).
  - `https://www.risc.jku.at/projects/SemTech`

Investigate the potential of formal modeling&reasoning software for education.

Educating with the Help of Formal Models

- Today much of modeling&reasoning can be automated by computer software.
  - Substantial advances in computational logic (automated reasoning, model checking, satisfiability solving).
- By the application of such software education may be supported.
  - May demonstrate the practical usefulness of theory.
  - May increase the motivation of students to model and to reason.
- The ultimate goal is self-directed learning.
  - Teachers become “enablers” by providing basic knowledge and skills.
  - Students “educate themselves” by solving problems.
    - (Voluntary) quizzes, (mandatory) assignments, possibly (graded) exams.

Core idea: let students actively engage with lecturing material by solving concrete problems and by receiving immediate feedback from the software.

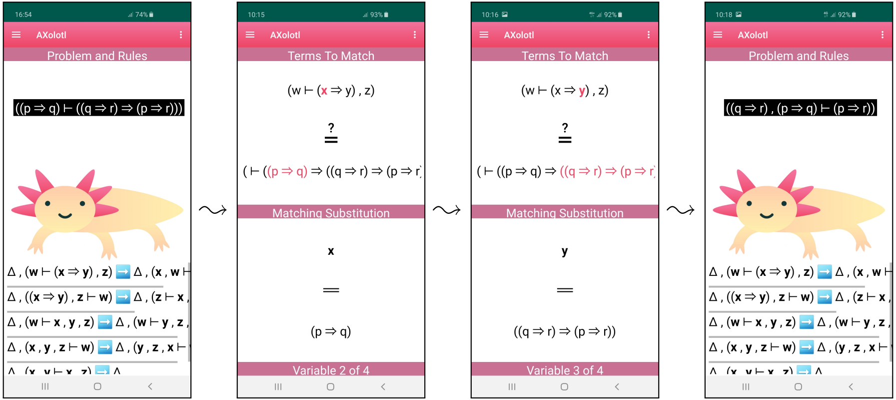
Research Strands

- Solver Guided Exercises (Limboole, Boolector)
- Teaching Solver Technology (Limboole, Boolector)
- Proof Assistants for Education (Theorema, AXolotl)
- Specification and Verification Systems for Education (RISCAL)
- Formal Semantics of Programming Languages ($\mathscr{J}ane$)
- Logic across the Subjects in Primary, Secondary and Higher Education

Various aspects of the general idea.

Example: AXolotl

Author: David Cerna; Google Play Store (search for “AXolotl Logic Software”)



Proving on a smartphone by a purely touch-based interface (no keyboard input).

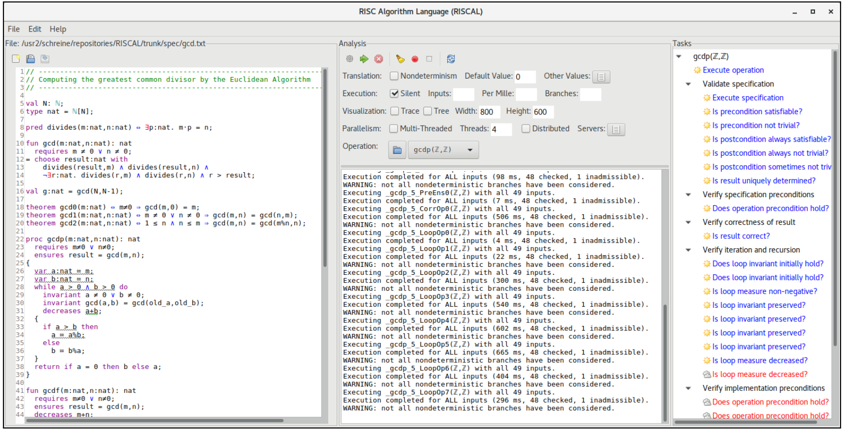
The RISC Algorithm Language (RISCAL)

A language and software system for investigating finite mathematical models (i.e., a “mathematical model checker”).

- Formulation of mathematical theories and theorems.
- Formulation and specification of (also non-deterministic) algorithms.
- Rooted in strongly typed first order logic and set theory.
- All types are finite (with sizes determined by model parameters).
- All formulas are automatically decidable.
- Correctness of all algorithms is decidable.
- Automatic generation of (again decidable) verification conditions.

Checking in some model of fixed size *before* proving in models of arbitrary size.

The RISCAL Software



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

Theories and Theorems

```
val N: ℕ;
type Literal = ℤ[-N,N];
type Clause  = Set[Literal] with ¬∃l∈value. -l∈value;
type Formula = Set[Clause];
type Valuation  = Set[Literal];

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 = ∅[Clause] then
    ⊤
  else if ∅[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](∅[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] ≠ ∅[Clause] ∧ ¬∅[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 = ∅[Clause] then
      satisfiable ≔ ⊤;
    else if ¬∅[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.

Transition Systems

```
proc system(x0: Positions, y0: Positions): ()
  requires init(x0, y0);
{
  var x: Positions = x0; var y: Positions = y0;

  var rs: Array[N+1,Robot] = Array[N+1,Robot](0);
  var ds: Array[N+1,Direction] ≔ Array[N+1,Direction](Direction!Stop);

  for var i:ℕ[N] ≔ 0; i < N; i ≔ i+1 do
  {
    choose r: Robot, d: Direction with nextDir(x, y, r, d);
    rs[i] := r; ds[i] ≔ d;

    x ≔ moveX(x, r, d); y ≔ moveY(y, r, d);
    assert noCollision(x, y) ∨ print rs, ds in ⊥;
  }
}
```

Nondeterministic systems defined by initial state condition and next state relation.

RISCAL Checking

```
Using N=2.
Type checking and translation completed.
...

Executing notValid(Set[Set[ℤ]]) with selected 512 inputs.
Execution completed for SELECTED inputs (111 ms, 512 checked, 0 inadmissible).

Executing DPLL(Set[Set[ℤ]]) with selected 512 inputs.
Execution completed for SELECTED inputs (1219 ms, 512 checked, 0 inadmissible).

Executing DPLL2(Set[Set[ℤ]]) with selected 512 inputs.
435 inputs (435 checked, 0 inadmissible, 0 ignored)...
Execution completed for SELECTED inputs (2436 ms, 512 checked, 0 inadmissible).

Executing DPLL_OutputCorrect(Set[Set[ℤ]]) with selected 512 inputs.
Execution completed for SELECTED inputs (609 ms, 512 checked, 0 inadmissible).
```

Automatic checking of theorems, algorithms, generated verification conditions.

Application: Mathematical Modeling

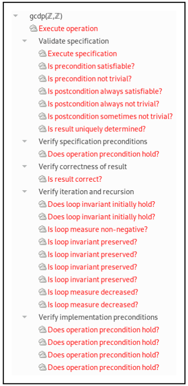
```
val N:ℕ; // variablex x0,...,xN
val M:ℕ; // values 0,...,M

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 };
```

```
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).
```

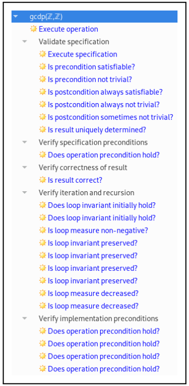
Validating conjectures (respectively the formalization of theorems).

Application: Specifying and Verifying 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.

RISCAL Approach to Model Checking/Formula Decision

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

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

```
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 RISCAL phrase to its (potentially nondeterministic) semantics and the execution of this semantics.

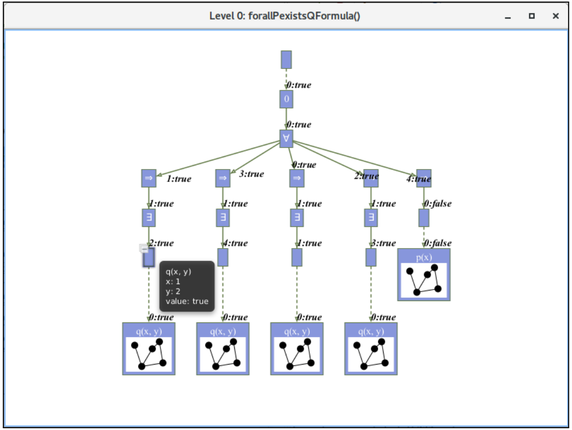
RISCAL Formula Decision (Experimental Alternative)

```
(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.
  - Author: Franz-Xaver Reichl (master thesis).
  - QF_UFBV: quantifier-free formulas over bitvectors with uninterpreted functions.
  - Well supported by various SMT solvers: Boolector, Z3, Yices, CVC4, ...
  - Elimination of quantifiers by skolemization and expansion.
  - Translation of integers, tuples/records, arrays/maps, sets, ... to bit vectors.
    - Non-trivial because, e.g., RISCAL uses “true” mathematical integers.

Much faster in many (not all) cases, systematic benchmarks under way.

RISCAL Visualization



Pruned evaluation trees to explain the truth value of a formula.

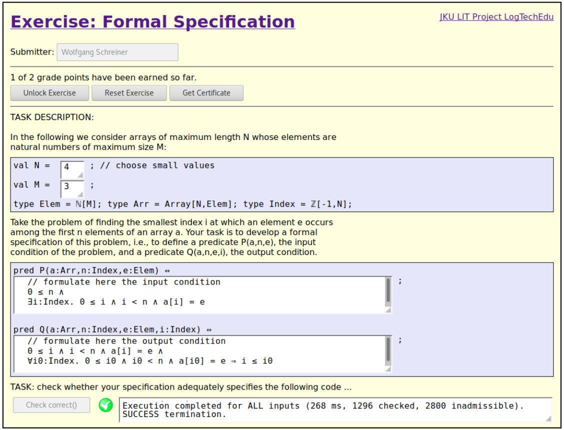
RISCAL Counterexample Generation

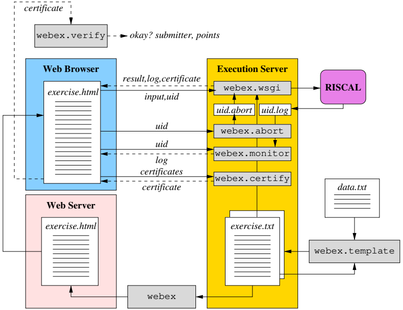
```
theorem _search_0_LoopOp6(a:array, x:elem) ⇔
  ∀i:int, r:int. ((((((0 ≤ i) ∧ (i ≤ N)) ∧ ...) ⇒
    (let i = i+1 in
      (∀j:int. (((0 ≤ j) ∧ (j < i)) ⇒ (a[j] ≠ x))))));

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.

RISCAL Web Exercises





Framework for web-based exercises checked by a RISCAL server.

Educational Usage

- “Formal Methods in Software Development” (JKU, master programs “Computer Science” and “Computer Mathematics”)
  - RISCAL: formal problem specifications; specification and verification of imperative programs.
- “Formal Methods and Specification” (TU Prague, Stefan Ratschan, master program “Informatics”)
  - RISCAL: formal specification and verification of imperative programs.
- “Formal Modeling” (JKU, bachelor program “Technical Mathematics”)
  - RISCAL: formal modeling of computational problems, search and scheduling problems (“puzzles”), dynamic systems.
- “Logic” (JKU, bachelor prog. “Computer Science” and “Artificial Intelligence”)
  - RISCAL, AXolotol, Theorema, Limboole, Boolector, Z3.
  - Bonus (RISCAL Web) and laboratory exercises (RISCAL desktop, AXolotol).
- Various Bachelor and Master Theses

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):
  - More used than other tools on FO and SMT (but less than SAT solvers).
  - Those who performed the exercises scored better in tests.
  - Students that scored poorly in tests did not use the software.
  - “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 Further Work

- Formal modeling&reasoning software can indeed be a factor to increase interest in “formal” topics and foster “self-directed” learning.
- However, students mainly profit if they already have certain abilities respectively some background.
- Care has to be taken to not “loose” the weaker beginners; these are easily overwhelmed by information overload or (trivial) syntactic/technical difficulties.
- We are currently running a beginner’s course with an easier to use web-based interface and will evaluate the difference it makes.
- Future work will concentrate on development of software-based course materials and on technical extensions (integration with interactive provers, modeling and reasoning about concurrency).

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