

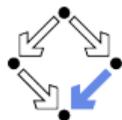
LOGIC & SEMANTIC TECHNOLOGIES FOR COMPUTER SCIENCE EDUCATION



Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC), JKU Linz, Austria

Joint work with V. Novitzká, W. Steingartner (TU Kosice)
and A. Biere, D. Cerna, M. Seidl, W. Windsteiger (JKU Linz)



Logic and Semantics 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. $\varphi \equiv \perp$: $\mathfrak{M}, s \not\models \varphi$.
2. $\varphi \equiv \top$: $\mathfrak{M}, s \models \varphi$.
3. $\varphi \equiv R(t_1, \dots, t_n)$: $\mathfrak{M}, s \models \varphi$ iff $\langle \text{Val}_s^{\mathfrak{M}}(t_1), \dots, \text{Val}_s^{\mathfrak{M}}(t_n) \rangle \in R$.
4. $\varphi \equiv t_1 = t_2$: $\mathfrak{M}, s \models \varphi$ iff $\text{Val}_s^{\mathfrak{M}}(t_1) = \text{Val}_s^{\mathfrak{M}}(t_2)$.
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 $\mathfrak{M}, s \models \psi$ and $\mathfrak{M}, s \models \chi$.
7. $\varphi \equiv (\psi \vee \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \models \psi$ or $\mathfrak{M}, s \models \chi$.
8. $\varphi \equiv (\psi \rightarrow \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \not\models \psi$ or $\mathfrak{M}, s \models \chi$.
9. $\varphi \equiv (\psi \leftrightarrow \chi)$: $\mathfrak{M}, s \models \varphi$ iff either neither $\mathfrak{M}, s \models \psi$ nor $\mathfrak{M}, s \models \chi$, or both $\mathfrak{M}, s \models \psi$ and $\mathfrak{M}, s \models \chi$.
10. $\varphi \equiv \forall x \psi$: $\mathfrak{M}, s \models \varphi$ iff for every x -valued assignment s' , $\mathfrak{M}, s' \models \psi$.
11. $\varphi \equiv \exists x \psi$: $\mathfrak{M}, s \models \varphi$ iff there is an x -valued assignment s' such that $\mathfrak{M}, s' \models \psi$.

$\frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} (\wedge L_2)$	$\frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \vee B, \Delta} (\vee R_2)$
$\frac{\Gamma, A \vdash \Delta \quad \Sigma, B \vdash \Pi}{\Gamma, \Sigma, A \vee B \vdash \Delta, \Pi} (\vee L)$	$\frac{\Gamma \vdash A, \Delta}{\Gamma, \Sigma \vdash \Delta} (\wedge R)$
$\frac{\Gamma \vdash A, \Delta \quad \Sigma, B \vdash \Pi}{\Gamma, \Sigma, A \rightarrow B \vdash \Delta, \Pi} (\rightarrow L)$	$\frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \Delta} (\rightarrow R)$
$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta} (\neg L)$	$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta} (\neg R)$
$\frac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta} (\forall L)$	$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta} (\forall R)$
$\frac{\Gamma, A[y/x] \vdash \Delta}{\Gamma, \exists x A \vdash \Delta} (\exists L)$	$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta} (\exists R)$

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

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

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

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

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

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

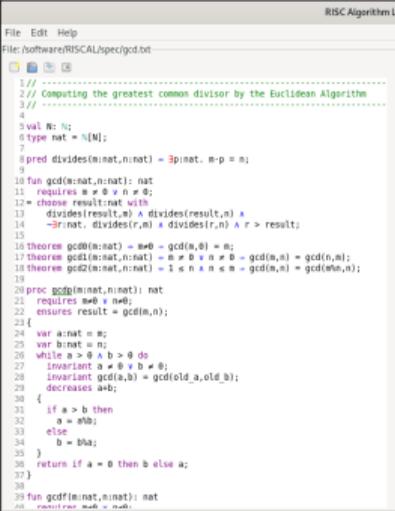
$\mathcal{C}[\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1] =$
 $\{(\sigma, \sigma') \mid \mathcal{B}[b]\sigma = \mathbf{true} \text{ \& } (\sigma, \sigma') \in \mathcal{C}[c_0]\} \cup$
 $\{(\sigma, \sigma') \mid \mathcal{B}[b]\sigma = \mathbf{false} \text{ \& } (\sigma, \sigma') \in \mathcal{C}[c_1]\}$

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

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

Typically still presented as “paper and pencil” topics.

Logic and Semantics in Education



```

1 // .....
2 // Computing the greatest common divisor by the Euclidean Algorithm
3 // .....
4
5 val N: %
6 type nat = N[N];
7
8 pred divides(m:nat, n:nat) = ∃p.nat. m = p * n;
9
10 fun gcd(m:nat, n:nat): nat
11   requires m ≠ 0 ∧ n ≠ 0;
12   choose result:nat with
13     divides(result, m) ∧ divides(result, n) ∧
14     ¬∃r.nat. divides(r, m) ∧ divides(r, n) ∧ r > result;
15
16 theorem gcd(m:nat) = gcd(m, 0) = m;
17 theorem gcd(m:nat, n:nat) = m * B + n * D = gcd(m, n) = gcd(n, m);
18 theorem gcd(m:nat, n:nat) = I ≤ n ∧ a ≤ m = gcd(m, n) = gcd(m, n);
19
20 proc gcd(m:nat, n:nat): nat
21   requires m ≠ 0 ∧ n ≠ 0;
22   ensures result = gcd(m, n);
23 {
24   var a:nat = m;
25   var b:nat = n;
26   while a > 0 ∧ b > 0 do
27     invariant a = B + b ≠ 0;
28     invariant gcd(a, b) = gcd(oid_a, oid_b);
29     decreases a+b;
30   {
31     if a > b then
32       a = a-b;
33     else
34       b = b-a;
35   }
36   return if a = 0 then b else a;
37 }
38
39 fun gcdf(m:nat, n:nat): nat
40   requires m ≠ 0 ∧ n ≠ 0;

```

Exercise: Formal Specification

JKU LIT Project LogTechEdU

Submitter:

1 of 2 grade points have been earned so far.

TASK DESCRIPTION:

In the following we consider arrays of maximum length N whose elements are natural numbers of maximum size M:

```

val N = 4 ; // choose small values
val M = 3 ;
type Elem = N[M]; type Arr = Array[N, Elem]; type Index = Z[-1,

```

Take the problem of finding the smallest index i at which an element e occurs among the first n elements of an array a . Your task is to develop a formal specification of this problem, i.e., to define a predicate $P(a, n, e)$, the input condition of the problem, and a predicate $Q(a, n, e, i)$, the output condition.

```

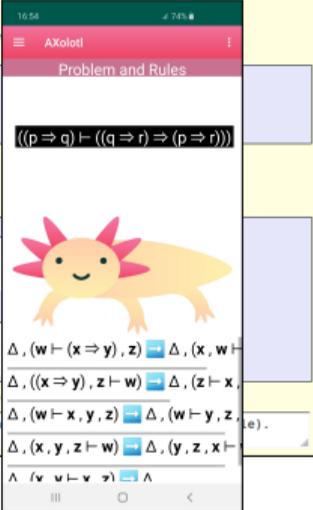
pred P(a:Arr, n:Index, e:Elem) =
  // formulate here the input condition
  0 ≤ n ∧
  ∃i:Index. 0 ≤ i ∧ i < n ∧ a[i] = e

pred Q(a:Arr, n:Index, e:Elem, i:Index) =
  // formulate here the output condition
  0 ≤ i ∧ i < n ∧ a[i] = e ∧
  ∀i0:Index. 0 ≤ i0 ∧ i0 < n ∧ a[i0] = e → i ≤ i0

```

TASK: check whether your specification adequately specifies the following code

✔
Execution completed for ALL inputs (268 n
SUCCESS termination.



$((p \Rightarrow q) \vdash ((q \Rightarrow r) \Rightarrow (p \Rightarrow r)))$



$\Delta, (w \vdash (x \Rightarrow y), z) \Rightarrow \Delta, (x, w \vdash y)$
 $\Delta, ((x \Rightarrow y), z \vdash w) \Rightarrow \Delta, (z \vdash x \wedge w)$
 $\Delta, (w \vdash x, y, z) \Rightarrow \Delta, (w \vdash y, z)$
 $\Delta, (x, y, z \vdash w) \Rightarrow \Delta, (y, z, x \vdash w)$
 $\wedge (\forall v \vdash v \Rightarrow \top) \Rightarrow \wedge$

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 logic&semantics-based software for education.

Educating with the Help of Logic&Semantics

- Today much of logic&semantics 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 logic&semantics.
 - May increase the motivation of students to model and to reason.
- The ultimate goal is logic&semantics-based **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 (*Jane*)
- Logic across the Subjects in Primary, Secondary and Higher Education

Various aspects of the general idea.

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 (by evaluating their semantics).
- Correctness of all algorithms is decidable (by evaluating their semantics).
- Automatic generation of (again decidable) verification conditions.

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

The RISCAL Software

The screenshot displays the RISCAL software interface, which is used for formal verification of algorithms. The main window is titled "RISC Algorithm Language (RISCAL)".

File: /usr2/schreine/repositories/RISCAL/trunk/spec/gcd.txt

Code (Left Panel):

```
1 //
2 // Computing the greatest common divisor by the Euclidean Algorithm
3 //
4
5 val N: N;
6 type nat = NIN;
7
8 pred divides(m:nat,n:nat) = ∃p:nat. m = p * n;
9
10 fun gcd(m:nat,n:nat): nat
11   requires m ≠ 0 ∨ n ≠ 0;
12   choose result:nat with
13     divides(result,m) ∧ divides(result,n) ∧
14     ¬∃r:nat. divides(r,m) ∧ divides(r,n) ∧ r > result;
15
16 val g:nat = gcd(N,N-1);
17
18 theorem gcd0(m:nat) = m = 0 → gcd(m,0) = m;
19 theorem gcd1(m:nat,n:nat) = m ≠ 0 ∨ n = 0 → gcd(m,n) = gcd(n,m);
20 theorem gcd2(m:nat,n:nat) = 1 ≤ n ∧ n ≤ m → gcd(m,n) = gcd(m/n,n);
21
22 proc gcdp(m:nat,n:nat): nat
23   requires m ≠ 0 ∧ n ≠ 0;
24   ensures result = gcd(m,n);
25 {
26   var a:nat = m;
27   var b:nat = n;
28   while a ≥ 0 ∧ b ≥ 0 do
29     invariant a ≠ 0 ∨ b ≠ 0;
30     invariant gcd(a,b) = gcd(old_a,old_b);
31     decreases a+b;
32   {
33     if a ≥ b then
34       a = a-b;
35     else
36       b = b-a;
37   }
38   return if a = 0 then b else a;
39 }
40
41 fun gcdf(m:nat,n:nat): nat
42   requires m ≠ 0 ∨ n ≠ 0;
43   ensures result = gcd(m,n);
44   decreases m+n;
```

Analysis (Middle Panel):

Translation: Nondeterminism Default Value: 0 Other Values: []

Execution: Silent Inputs: Per Mille: Branches:

Visualization: Trace Tree Width: 800 Height: 600

Parallelism: Multi-Threaded Threads: 4 Distributed Servers: []

Operation: gcdp(Z,Z)

Execution completed for ALL inputs (98 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing gcdp_5_PrefEn0(Z,Z) with all 49 inputs.
Execution completed for ALL inputs (7 ms, 48 checked, 1 inadmissible).
Executing gcdp_5_CorrOp0(Z,Z) with all 49 inputs.
Execution completed for ALL inputs (506 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing gcdp_5_LoopOp0(Z,Z) with all 49 inputs.
Execution completed for ALL inputs (4 ms, 48 checked, 1 inadmissible).
Executing gcdp_5_LoopOp1(Z,Z) with all 49 inputs.
Execution completed for ALL inputs (22 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing gcdp_5_LoopOp2(Z,Z) with all 49 inputs.
Execution completed for ALL inputs (308 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing gcdp_5_LoopOp3(Z,Z) with all 49 inputs.
Execution completed for ALL inputs (548 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing gcdp_5_LoopOp4(Z,Z) with all 49 inputs.
Execution completed for ALL inputs (602 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing gcdp_5_LoopOp5(Z,Z) with all 49 inputs.
Execution completed for ALL inputs (484 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing gcdp_5_LoopOp6(Z,Z) with all 49 inputs.
Execution completed for ALL inputs (665 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.
Executing gcdp_5_LoopOp7(Z,Z) with all 49 inputs.
Execution completed for ALL inputs (296 ms, 48 checked, 1 inadmissible).
WARNING: not all nondeterministic branches have been considered.

Tasks (Right Panel):

- gcdp(Z,Z)
 - Execute operation
 - Validate specification
 - Execute specification
 - Is precondition satisfiable?
 - Is precondition not trivial?
 - Is postcondition always satisfiable?
 - Is postcondition always not trivial?
 - Is postcondition sometimes not true?
 - Is result uniquely determined?
 - Verify specification preconditions
 - Does operation precondition hold?
 - Verify correctness of result
 - Is result correct?
 - Verify iteration and recursion
 - Does loop invariant initially hold?
 - Does loop invariant initially hold?
 - Is loop measure non-negative?
 - Is loop invariant preserved?
 - Is loop invariant preserved?
 - Is loop invariant preserved?
 - Is loop measure decreased?
 - Is loop measure decreased?
 - Verify implementation preconditions
 - Does operation precondition hold?
 - Does operation precondition hold?

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

RISCAL Theories and Theorems

```
val N: ℕ;
type Literal = ℤ[-N,N];
type Clause  = Set[Literal];
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)
  requires formula(f);
⇔ valid(f) ⇔ ¬satisfiable(not(f));
```

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

RISCAL Algorithms

```
multiple pred DPLL(f:Formula)
  requires formula(f);
  ensures result  $\Leftrightarrow$  satisfiable(f);
  decreases |literals(f)|;
 $\Leftrightarrow$  if f =  $\emptyset$ [Clause] then
   $\top$ 
  else if  $\emptyset$ [Literal]  $\in$  f then
   $\perp$ 
  else choose l $\in$ literals(f) in
    DPLL(substitute(f,l))  $\vee$ 
    DPLL(substitute(f,-l));
```

```
proc DPLL2(f:Formula): Bool
  requires ...; ensures ...;
{
  var stack: Array[n+1,Formula] :=
    Array[n+1,Formula]( $\emptyset$ [Clause]);
  ...
  while  $\neg$ satisfiable  $\wedge$  number>0 do
    ...
    invariant satisfiable(f)  $\Leftrightarrow$  ...;
    decreases if satisfiable then 0 else
       $\sum_{k:\mathbb{N}[n]} \text{with } k < \text{number. size}(\text{stack}[k]);$ 
    {
      ...
      choose l $\in$ literals(g);
      ...
    }
    return satisfiable;
}
```

Functions, procedures, recursion, nondeterminism.

RISCAL Systems

```
proc system(s0:State): Tuple[N[N],Array[N,State]]
  requires init(s0);
{
  var s:State = s0;
  var i:N[N] = 0;
  var t:Array[N,State] = Array[N,Action](s);
  while  $\neg$ goal(s)  $\wedge$  i < N do
  {
    choose s1:State with next(s,s1);
    s = s1; i = i+1; t[i] = s;
  }
  return hi,ti;
}
```

Modeling and analysis of nondeterministic transition systems described by their 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)...

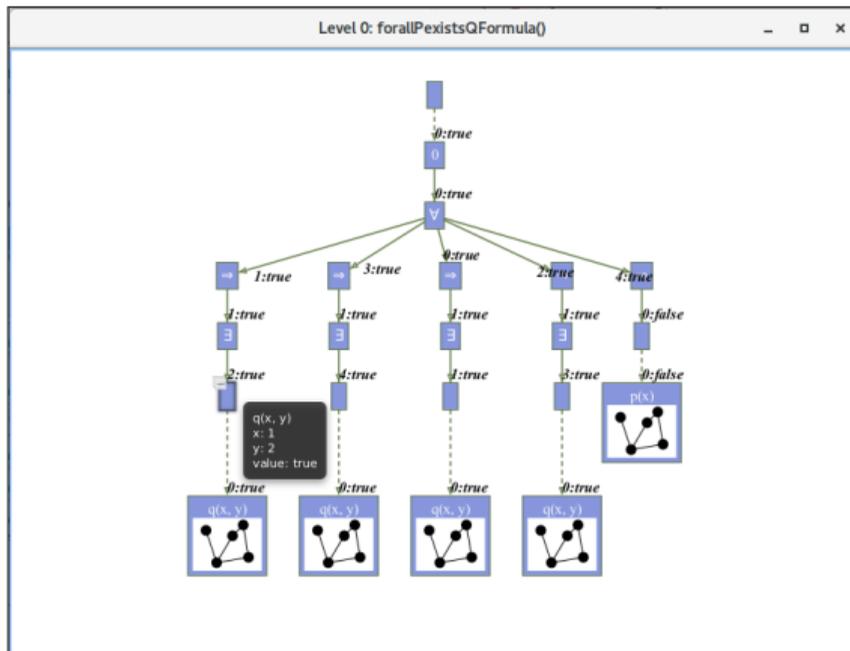
Execution completed for SELECTED inputs (2436 ms, 512 checked, 0 inadmissible).

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.

RISCAL Visualization



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

RISCAL Web Exercises

Exercise: Formal Specification JKU LIT Project LogTechEdu

Submitter:

1 of 2 grade points have been earned so far.

TASK DESCRIPTION:

In the following we consider arrays of maximum length N whose elements are natural numbers of maximum size M :

```
val N = 4 ; // choose small values
val M = 3 ;
type Elem = N[M]; type Arr = Array[N,Elem]; type Index = Z[-1,N];
```

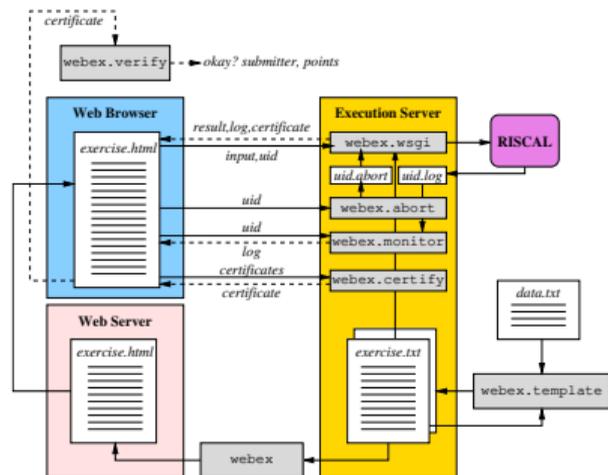
Take the problem of finding the smallest index i at which an element e occurs among the first n elements of an array a . Your task is to develop a formal specification of this problem, i.e., to define a predicate $P(a,n,e)$, the input condition of the problem, and a predicate $Q(a,n,e,i)$, the output condition.

```
pred P(a:Arr,n:Index,e:Elem) =
  // formulate here the input condition
  0 ≤ n ∧
  ∃i:Index. 0 ≤ i ∧ i < n ∧ a[i] = e

pred Q(a:Arr,n:Index,e:Elem,i:Index) =
  // formulate here the output condition
  0 ≤ i ∧ i < n ∧ a[i] = e ∧
  ∀i0:Index. 0 ≤ i0 ∧ i0 < n ∧ a[i0] = e - 1 ≤ i0
```

TASK: check whether your specification adequately specifies the following code ...

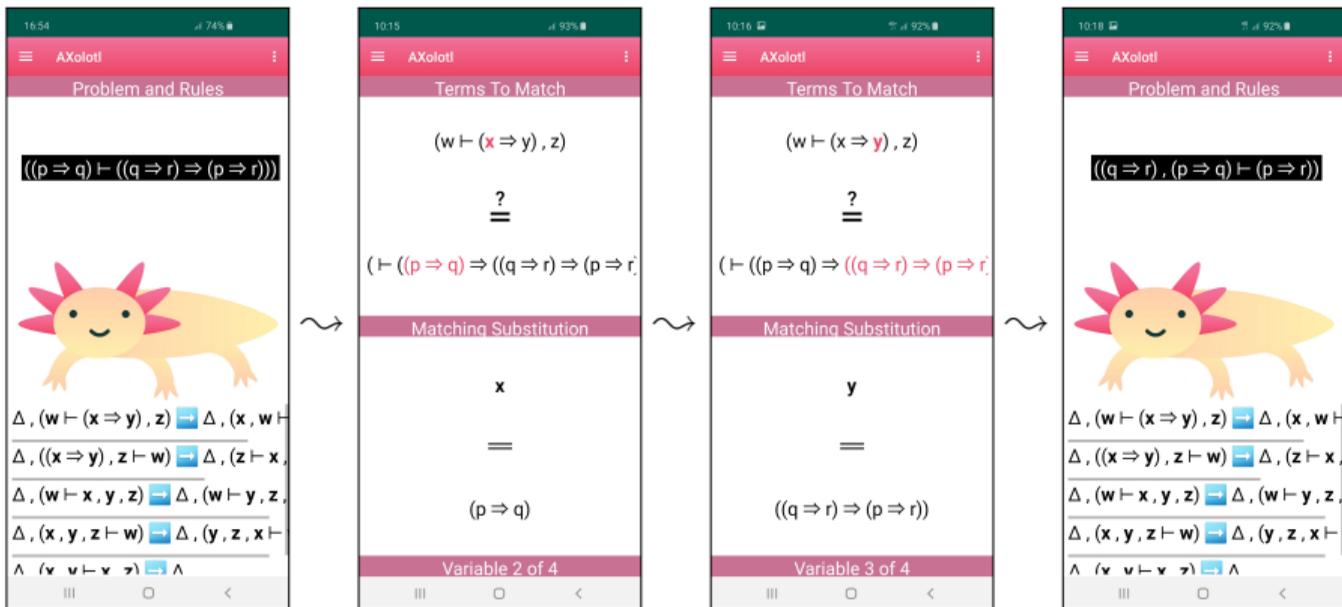
Execution completed for ALL inputs (268 ms, 1296 checked, 2800 inadmissible). SUCCESS termination.



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

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).

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).
- **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 (1. sem. 2018, RISCAL web interface not yet available):
 - 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

- Logic&semantics 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 SMT solvers and interactive provers, modeling and reasoning about concurrency).

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