LOGIC TECHNOLOGY FOR CS EDUCATION

RISCAL – The RISC Algorithm Language

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Wolfgang.Schreiner@risc.jku.at
http://www.risc.jku.at
Systematic Problem Solving

A key competence in modern professional life.

- “Computational” thinking
  - Express precisely how to carry out a problem solution.
  - Development of solution descriptions: algorithms/programs.
  - Ultimate goal is to let computers execute the solutions.
- But is a proposed solution really adequate?
  - Does it really solve the problem?
- “Specificational” thinking
  - Elaborate and express precisely what problem to solve.
  - Development of problem descriptions: specifications.
  - Ultimate goal is to let computers (help to) validate/verify the correctness of problem solutions.

Specifications come before programs.
A Sample Problem

“Given an array $a$ with $n$ elements, find the maximum $m$ of $a$."

- For instance, if $n = 3$ and $a$ is $[1, 2, -1]$, then $m = 2$.
  - Indices $0, 1, 2$ with $a[0] = 1, a[1] = 2, a[2] = -1$.

Does this algorithm (procedure) solve the problem?

```
proc maxProc(a:array, n:int): elem {
  var m:elem := 0;
  for var i:int := 0; i < n; i := i+1 do {
    if a[i] > m then m := a[i];
  }
  return m;
}
```

Before judging the algorithm, we have to specify the problem.
The Importance of Language

Forming, formulating and expressing ideas needs *language*.

- Computations are described in *programming languages*.
  - First: low-level instruction sets of computer processors.
  - Today: high-level languages understandable by humans.

- Specifications are described in the *language of logic*.
  - First: Aristotelian logic, sentences like “Aristoteles is a man”.
  - Today: first-order logic, sentences like “for every number $x$ there exists some number $y$ such that $y$ is greater than $x$”.
  - Precise form (syntax), meaning (semantics), and rules of reasoning (inference calculus).
  - Expressive enough to characterize computational problems.

Like any language, logic is learned by usage in practical context.
Problem Specification

“Given an array $a$ with $n$ elements, find the maximum $m$ of $a$.”

- **Given** arbitrary $a, n$ that satisfy the *precondition*.
  - “$a$ has $n$ elements.”
- **Find** some $m$ that satisfies the *postcondition*.
  - “$m$ is the maximum of array $a$ with $n$ elements.”

We are now going to formalize this specification.
Defining the Value Domains

We are considering arrays up to some maximum length $N$ with integer elements of some maximum absolute value $M$.

```plaintext
val N:N; val M:N;
type int = ℤ[-N,N];
type elem = ℤ[-M,M];
type array = Array[N,elem];
```

Values $a, n, m$ have types array, int, elem, respectively.
Formalizing Pre- and Postconditions

“a has n elements”

- n is an integer greater equal zero, and
- a holds beyond the valid indices 0, . . . , n − 1 only zeroes.
  - For every integer k, if k is greater equal n and less than N, then a holds at k value 0.

\[ n \geq 0 \land \forall k: \text{int. } n \leq k \land k < N \Rightarrow a[k] = 0; \]

We abbreviate this statement to a predicate hasLength(a,n):

\[ \text{pred hasLength(a:array, n:int) } \iff \\
    n \geq 0 \land \forall k: \text{int. } n \leq k \land k < N \Rightarrow a[k] = 0; \]
Formalizing Pre- and Postconditions

- “a has n elements”
  
  \[
  \text{pred pre}(a: \text{array}, n: \text{int}) \iff \text{hasLength}(a, n);
  \]

- “m is the maximum of array a with n elements.”
  
  - “m must be at least as big as every element of a.”
  - For every integer k, if k is greater equal 0 and less than n, then m is greater equal that element that a holds at k.

  \[
  \text{pred post}(a: \text{array}, n: \text{int}, m: \text{elem}) \iff \forall k: \text{int}. \ 0 \leq k \land k < n \Rightarrow m \geq a[k];
  \]

How to validate this specification attempt?
RISCAL: RISC Algorithm Language

A language/system for algorithm specification and verification.

In RISCAL all definitions are executable.
Validating the Specification

A specification implicitly defines a mathematical function.

fun maxFun(a:array, n:int):elem
   requires pre(a,n);
   = choose m:elem with post(a,n,m);

- For all $a, n$ that satisfy $pre(a, n)$, $maxFun(a, n)$ denotes some $m$ that satisfies $post(a, n, m)$.
  - If no such $m$ exists, the function result is undefined.
  - If multiple such $m$ exist, the result is not uniquely defined.

In RISCAL also this function is executable.
Checking the Implicitly Defined Function

Apply \texttt{maxFun} to all possible inputs.

Postcondition is too weak, allows \( m \) that does not occur in \( a \).
Strengthening the Postcondition

“m is the maximum of array a with n elements.”

- also: “m must occur as an element of a”.
- There exists some integer k that is greater equal 0 and less than n such that m equals that element that a holds at k.

\[
\text{pred post(a:array, n:int, m:elem) } \iff \\
(\forall k:\text{int. } 0 \leq k \land k < n \Rightarrow m \geq a[k]) \land \\
(\exists k:\text{int. } 0 \leq k \land k < n \land m = a[k]);
\]
Strengthening the Postcondition

Now the postcondition apparently allows only one result.
Checking Specification Properties

For all $a, n$ that satisfy the precondition, there is not more than one $m$ that satisfies the postcondition:

\[
\text{theorem maxUnique}(a: \text{array}, n: \text{int})
\]

\[
\text{requires } \text{pre}(a, n);
\]

\[
\iff \forall m1: \text{elem}, m2: \text{elem}.
\]

\[
\text{post}(a, n, m1) \land \text{post}(a, n, m2) \Rightarrow m1 = m2;
\]

For all $a, n$ that satisfy the precondition, there is some $m$ that satisfies the postcondition:

\[
\text{theorem maxExists}(a: \text{array}, n: \text{int})
\]

\[
\text{requires } \text{pre}(a, n);
\]

\[
\iff \exists m: \text{elem}. \text{post}(a, n, m);
\]
Checking Specification Properties

Precondition is too weak, allows empty array ($n = 0$).
Strengthening the Precondition

\[
\text{pred pre}(a:\text{array}, \ n:\text{int}) \iff \text{hasLength}(a, \ n) \land n > 0;
\]

Now the specification seems adequate.
Checking the Algorithm

The algorithm violates the specification!
An Improved Algorithm

proc maxProc(a:array, n:int): elem
  requires pre(a,n);
  ensures  post(a,n,result);
{
  var m:elem := a[0];
  for var i:int := 1; i < n; i := i+1 do
    { 
      if a[i] > m then m := a[i];
    }
  return m;
}
Checking the Algorithm

This algorithm satisfies the specification for some $N, M$.
Verifying the Algorithm

Core question: is the algorithm correct for all $N, M$?

- Have now checked this for arrays of length at most $N = 3$.
  - Verification for arbitrary $N$ requires logi proof.
- Proof based on loop invariant that holds for arbitrary $N$.
  1. Proof that invariant holds before loop is started.
  2. Proof that invariant is preserved by every loop iteration.
  3. Proof that on termination invariant implies postcondition.
- Key: in iteration $i$, $m$ is the maximum of the first $i - 1$ values.

$$\text{pred inv}(a: \text{array}, n: \text{int}, m: \text{elem}, i: \text{int}) \iff$$

$$\begin{align*}
1 & \leq i \land i \leq n \land \\
(\forall k: \text{int}. \ 0 \leq k \land k < i \Rightarrow m \geq a[k]) \land \\
(\exists k: \text{int}. \ 0 \leq k \land k < i \land m = a[k]);
\end{align*}$$

- Use of automated provers or interactive proof assistants.
  - Inadequate invariant lets some of the proofs fail.

RISCAL can also validate the suitability of a proposed invariant.
Checking a Loop Invariant

The invariant is not too strong.
Checking Verification Conditions

The invariant may be still too weak.

```
theorem VC1(a: array, n: int, m: elem, i: int)
  requires pre(a,n);
  \iff m = a[0] \land i=1 \Rightarrow inv(a,n,m,i);
theorem VC2a(a: array, n: int, m: elem, i: int)
  requires pre(a,n);
  \iff inv(a,n,m,i) \land i < n \land a[i] > m \Rightarrow inv(a,n,a[i],i+1);
theorem VC2b(a: array, n: int, m: elem, i: int)
  requires pre(a,n);
  \iff inv(a,n,m,i) \land i < n \land \neg(a[i] > m) \Rightarrow inv(a,n,m,i+1);
theorem VC3(a: array, n: int, m: elem, i: int)
  requires pre(a,n);
  \iff inv(a,n,m,i) \land \neg(i < n) \Rightarrow post(a,n,m);
```

Need to check the verification conditions.
Checking Verification Conditions

Proof-based verification for general $N, M$ is likely to succeed.
Goal of RISCAL

Logic education in computer science and mathematics.

- **Self-directed and self-paced learning.**
  - Students may quickly validate *self-defined* theories, specifications, algorithms, meta-knowledge.
  - Exercises, assignments, online courses.

- **Current and future work:**
  - Automatic generation of formulas that support validation.
  - Application of SMT solvers to deciding validity of formulas.
  - Visualization of formula interpretation/evaluation.
  - Export of formulas to external proof assistants.
  - Development of educational materials.

- Supported by JKU LIT project LOGTECHEDU.

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