Quo Vadis Program Verification

Krzysztof R. Apt

CWI, Amsterdam, the Netherlands,

University of Amsterdam

We would like to use correct programs.

Programming Language Matters (1)

Correctness proofs of

- `quicksort` in Haskell,
- the type checking program for pure λ-calculus in Prolog,
- the program solving Sudoku puzzles in ECL^{i}PS^e,

are straightforward.

Programming Language Matters (2)

Compare ALMA-0 program

```
MODULE queens;
CONST  N = 8;
TYPE board = ARRAY [1..N] OF [1..N];

PROCEDURE queens(VAR x: board);
  VAR i, column, row: [1..N];
  BEGIN
    FOR column := 1 TO N DO
      SOME row := 1 TO N DO
        FOR i := 1 TO column-1 DO
          x[i] <> row;
          x[i] <> row+column-i;
          x[i] <> row+i-column
        END;
        x[column] = row
      END
    END
  END queens;
END queens.
```

```
public class Queens {

   /***********************************************/
    * Return true if queen placement q[n] does not conflict with
    * other queens q[0] through q[n-1]
   /***********************************************/
    public static boolean isConsistent(int[] q, int n) {
        for (int i = 0; i < n; i++) {
            if (q[i] == q[n])             return false;
            if ((q[i] - q[n]) == (n - i)) return false;
            if ((q[n] - q[i]) == (n - i)) return false;
        }
        return true;
    }

   /***********************************************/
    *  Try all permutations using backtracking
   /***********************************************/
    public static void enumerate(int N) {
        int[] a = new int[N];
        enumerate(a, 0);
    }
```

```
    public static void enumerate(int[] q, int n) {
        int N = q.length;
        if (n == N) printQueens(q);
        else {
            for (int i = 0; i < N; i++) {
                q[n] = i;
                if (isConsistent(q, n)) enumerate(q, n+1);
            }
        }
    }


    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        enumerate(N);
    }

}
```

Mathematics Matters

Examples

- Simplex algorithm with Bland anti-cycling rule,
- Gröbner's basis,
- Hungarian method,
- $\ldots$,

Program refinement matters

Small personal story: Constraint propagation algorithms.

- Several algorithms proposed in the literature (AC-3, PC-2, DAC, bounds consistency, relational consistency, . . .)
- They turned out to be special cases of two generic chaotic iteration algorithms.
  - K.R. Apt, *The essence of constraint propagation*, TCS 221(1-2), 179-210 (1999).
  - K.R. Apt, *The role of commutativity in constraint propagation algorithms*, ACM Toplas, 22(6), 1002-1036 (2000).

So far so good, but . . .

- Programs are mostly written in mainstream programming languages.
- Translation of theorems into programs is not a formal process.
- Translation of simplest statements to these programming languages is clumsy.

Example

Translate: ‘If a[1..m][1..n] has a zero entry’ to Java.

Translation to ALMA-0

```
IF
  SOME i := 1 TO m DO
  SOME j := 1 TO n DO
    a[i,j] = 0
  END
  END
THEN ...
```

K. R. Apt, J. Brunekreef, V. Partington, A. Schaerf,
ALMA-O*: An Imperative Language That Supports Declarative Programming*,
ACM Toplas 20(5): 1014-1066 (1998).

Program Verification

Assertional approach

- Basic Idea:
  Reason on the level of assertions instead of states.
- Axioms and proof rules to reason about **while** programs (Hoare '69),
- Example:

$$\frac{\{p \wedge B\}\ S\ \{p\}}{\{p\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg B\}}$$

(p is the loop invariant).

Some Theoretical Milestones

- Recursive procedures (Hoare '71),
- Arrays (Hoare and Wirth '73, Gries '78, De Bakker '80),
- Parallel programs (Owicki and Gries, '76, Lamport ('77)),
- Distributed programs (Apt, De Roever and Francez, '80),
- Notion of completeness (Cook '78),
- Impossibility of completeness for 'full ALGOL' (Clarke '79).

Drawbacks and Remedies

Deterministic programs

- Specifications in first-order logic can be clumsy or impossible.
- Remedy: use appropriate specification languages (Z of Abrial '74, ISO standard: 2002).
- Correctness proofs are tedious and error-prone.
- Remedy 1: develop the program together with its correctness proof (Dijkstra '76).
- Remedy 2: certify proofs.
- Another tack: Higher-level system development (Abrial '96, '09).

Mechanical Verification

- Use a theorem prover /proof assistant.
- Underlying assumption:
  the theorem prover is a correct program.
- Verify mechanically soundness of the used proof systems.
- Establish correctness of a given program by verifying mechanically its correctness proof in a sound proof system.

Gap between Theory and Practice

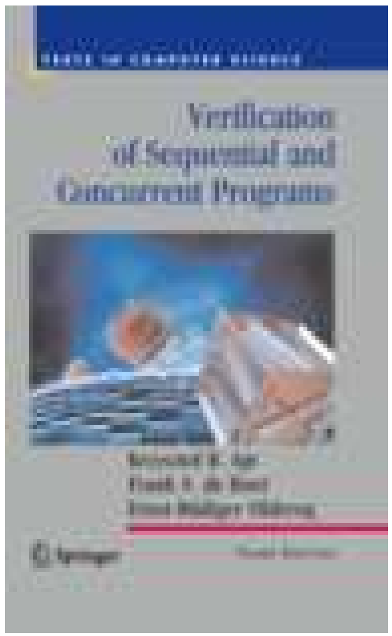
Grand Challenge in Program Verification

- Build a library of provably correct OO programs dealing with data structures.
- Example: Verify the programs in
  LEDA: A Platform for Combinatorial and Geometric Computing,
  Mehlhorn and Näher, '99.
  Cambridge University Press, 1034 pages.
- Main difficulty: these are C++ programs; extensively use classes.

Verification of OO Programs

- Initial idea: De Boer, '91,
- Presented using program transformation in

  Verification of Sequential and Concurrent Programs,
  Apt, De Boer and Olderog,
  Springer, 2009, 502 pages.



Main difficulties

How to deal with

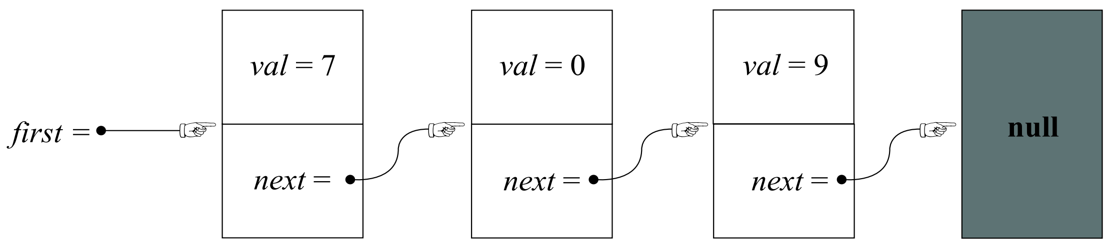
- instance variables,
- transfer of control between caller and callee,
- void references (calls on **null** object).

Approach

- Carefully choose a kernel language.
- Provide a syntax-directed transformation of object-oriented programs to the kernel language.
- Enrich the assertion language to reason about objects.
- Use this translation to derive the proof rules.
- Detailed omitted (here).

Example

```
find :: if val = 0  then return := this
        else if next ≠ null
                then next.find
                else return := null
                fi
        fi
```



- val is an instance integer variable,
- $next$ is an instance object variable,
- $first$ and $return$ are normal object variables.
- Intuition: $first.find$ returns the first object that stores 0. The search starts at the object stored in $first$.

Back to the Grand Challenge (1)

Missing Features

- object creation (handled in ABO '09),
- access to instance variables of arbitrary objects (handled in ABO '10),
- inheritance, subtyping (Pierik and De Boer, '05),
- exception handling, . . .

Back to the Grand Challenge (2)

Are Mechanical Proofs Needed?

- Rules can be unsound.
  Example: SUBSTITUTION RULE (ABO '09)

$$\frac{\{p\}\ S\ \{q\}}{\{p[\bar{z} := \bar{t}]\}\ S\ \{q[\bar{z} := \bar{t}]\}}$$

  where $(\{\bar{z}\} \cup var(\bar{t})) \cap change(S) = \emptyset$.

  Correct version (ABO '10):

  where $(\{\bar{z}\} \cap var(S)) \cup (var(\bar{t}) \cap change(S)) = \emptyset$.

- $find$ program may not terminate for cyclic lists.

Back to the Grand Challenge (3)

We need to

- rely on mathematical theorems

and combine them with

- stepwise refinement,
- program refinement and transformations,
- assertional verification,

in one framework.

Summary

- Focus on libraries of existing OO programs.
- Create a catalogue of mechanically certified programs.
- Small comment: one needs first to choose the assertion language and the programming language . . .,
- . . . and ideally prove mechanically the underlying mathematical theorems.

Is this realistic?