Certifying Algorithms
An Attempt of a Theory

Kurt Mehlhorn

MPI für Informatik

Saarbrücken

Germany

Does every Program have a Certifying Counterpart?

- a formalization of certifying programs for programs computing functions
- Monte Carlo algs have no certifying counterpart
- every deterministic program has a certifying counterpart
- then formalization for programs with non-trivial preconditions
- there are programs which have certifying counterpart

Witness Predicates

$W : X \times Y \times W \mapsto \{0,1\}$ is a *witness predicate* for $f : X \mapsto Y$ if

1. W deserves is name:

$$\forall x, y \quad (\exists w \, W(x,y,w)) \quad \text{iff} \quad (y = f(x))$$

2. witness property is easy to understand, i.e., the implication

$$W(x,y,w) \rightarrow (y = f(x))$$

   has an elementary proof.

3. given x, y, and w, it is trivial to decide whether $W(x,y,w)$ holds.
   - a program for W is called a checker
   - checker has linear running time and simple structure
   - correctness of checker is obvious or can be established by an elementary proof

no assumption about difficulty of proving $\quad (y = f(x)) \rightarrow \exists w \, W(x,y,w)$

Does every Function have a Certifying Alg?

- let P be a program and let f be the function computed by P

- does there exist a program Q and a predicate W such that
  1. W is a witness predicate for f.
  2. On input x, Q computes a triple (x, y, w) with $W(x, y, w)$.
  3. the resource consumption (time, space) of Q on x is at most a constant factor larger than the resource consumption of P.

Thesis:

- Every deterministic algorithm can be made certifying
- Monte Carlo algorithms resist certification

Intuition:

- correctness proofs yield certifying algorithms
- a certifying Monte Carlo alg yields Las Vegas alg

Monte Carlo Algorithms resist Certification

MAX-PLANCK-GESELLSCHAFT

- assume we have a Monte Carlo algorithm for a function f, i.e.,
  - on input x it outputs $f(x)$ with probability at least $3/4$
  - the running time is bounded by $T(|x|)$.
- assume Q is a certifying alg with the same complexity
  - on input x, Q outputs a witness triple (x, y, w) with probability at least $3/4$.
  - it has running time $O(T(|x|))$.
- this gives rise to a Las Vegas alg for f with the same complexity
  - run Q and apply W to the triple (x, y, w) returned by Q
  - if W holds, we return y. Otherwise, we rerun Q.
  - this outputs $f(x)$ in expected time $O(T(|x|))$.

Every Deterministic Program has a Certifying Counterpart

- let P be a program computing f.
- certifying Q outputs $f(x)$ and a witness $w = (w_1, w_2, w_3)$
  - w_1 is the program text P, w_2 is a proof (in some formal system) that P computes f, and w_3 is the computation of P on input x
  - $W(x,y,w)$ holds if $w = (w_1, w_2, w_3)$, where w_1 is the program text of some program P, w_2 is a proof (in some formal system) that P computes f, w_3 is the computation of P on input x, and y is the output of w_3.
- we have
  1. W is clearly a witness predicate
  2. W is trivial to decide
  3. the proof of $W(x,y,w) \rightarrow (y = f(x))$ is elementary
  4. Q has same space/time complexity as P.
- construction is artificial, but assuring: certifying algs exist
- the challenge is to find natural certifying algs

And with Non-Trivial Preconditions

$$\{\varphi(x)\} \quad P \quad \{\psi(x,y)\}$$

- standard interpretation of total correctness: on an input x satisfying φ, the program P returns a y with $\psi(x,y)$. If x does not satisfy φ, the program may do anything.
- certifying program: on an input x, it either returns a proof for $\neg\varphi(x)$ or a y and a proof for $\psi(x,y)$.
- Example 1:
  - Precondition: x is the description of a Turing Machine halting on empty input
  - Output: the result of running x on empty input
  - this behavior is easily realized: a universal Turing Machine
  - formal correctness proof is feasible
  - but behavior cannot be realized by a certifying algorithm

Verification of Checkers

- the checker should be so simple that its correctness is “obvious”.
- we may hope to formally verify the correctness of the implementation of the checker

  this is a much simpler task than verifying the solution algorithm

  - the mathematics required for the checker is usually much simpler that the one underlying the algorithm for finding solutions and witnesses
  - checkers are simple programs
  - algorithmicists may be willing to code the checkers in languages which ease verification
  - logicians may be willing to verify the checkers

- **Remark:** for a correct program, verification of the checker is as good as verification of the program itself

Cooperation of Verification and Checking

- a sorting routine working on a set S

  (a) must not change S and

  (b) must produce a sorted output.
- I learned the example from Gerhard Goos
- the first property is hard to check (provably as hard as sorting)
- but usually trivial to prove, e.g.,
  if the sorting algorithm uses a *swap*-subroutine to exchange items.
- the second property is easy to check by a linear scan over the output,
  but hard to prove (if the sorting algorithm is complex).
- second example in handout

Design of Certifying Algorithms

- general approaches
  - linear programming duality: primal and dual solution certify each other, e.g.,
    matchings and covers, flows and cuts, shortest paths and potential functions
  - characterization theorem, e.g.,
    non-planarity and Kuratowski subgraphs, convex bodies and certifying rays

- however, there is no “Königsweg”