MAX-PLANCK-GESELLSCHAFT

Certifying Algorithms

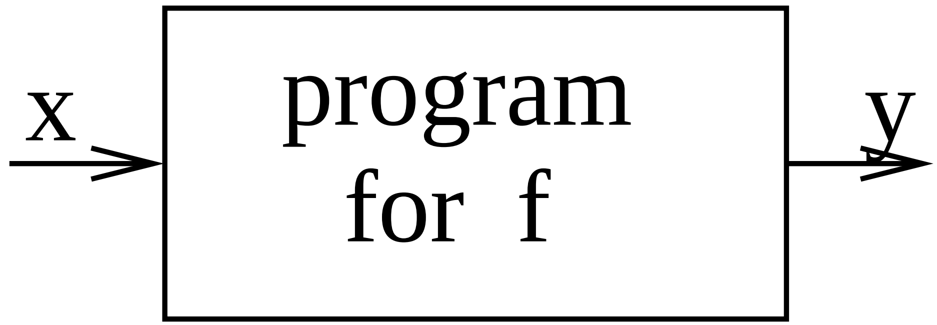
Kurt Mehlhorn

MPI für Informatik

Saarbrücken

Germany

The Problem Statement



- a user knows x and y.
- how can he/she be sure that, indeed, $y = f(x)$.
- he/she is at complete mercy of the program
- I do not like to depend on software in this way, not even for programs written by myself

Warning Examples

- Rhino3d (a CAD systems) fails to compute correct intersection of two cyclinders and two spheres

Warning Examples

- Rhino3d (a CAD systems) fails to compute correct intersection of two cyclinders and two spheres
- CPLEX (a linear programming solver) fails on benchmark problem *etamacro*.

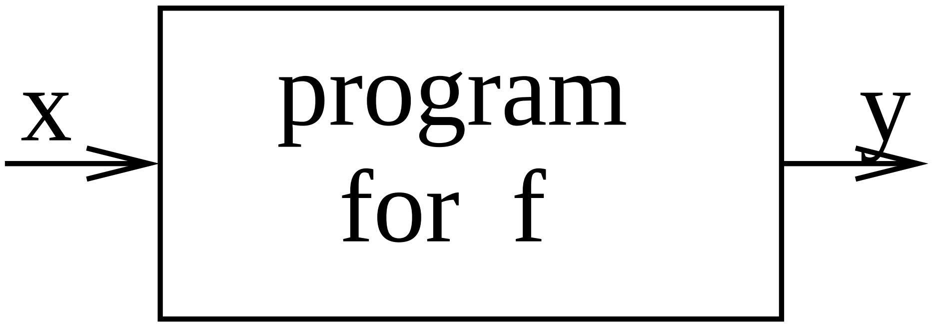
Warning Examples

- Rhino3d (a CAD systems) fails to compute correct intersection of two cyclinders and two spheres
- CPLEX (a linear programming solver) fails on benchmark problem *etamacro*.
- Mathematica 4.2 (a mathematics systems) fails to solve a small integer linear program

```
In[1] := ConstrainedMin[ x , {x==1,x==2} , {x} ]
Out[1] = {2, {x->2}}
```

```
In[1] := ConstrainedMax[ x , {x==1,x==2} , {x} ]
ConstrainedMax::lpsub": The problem is
unbounded."
Out[2] = {Infinity, {x -> Indeterminate}}
```

The Problem Statement



programs should justify (prove) their answers in a way that is easily checked by their users.

Certifying Algorithms

- a certifying program returns
  - the function value y and
  - a certificate (witness) w.
- w proves the equality $y = f(x)$.
- if $y \neq f(x)$, there should be no w such that (x, y, w) passes checking.
- formalization in second half of talk
- name introduced in Kratsch/McConnell/Mehlhorn/Spinrad: SODA 2003
- related work: Blum et al.: Programs that check their work

Outline of Talk

- problem definition and certifying algorithms
- examples of certifying algorithms
  - linear system solving
  - testing bipartiteness
  - matchings in graphs
  - planarity testing
  - convex hulls
  - dictionaries and priority queues
  - linear programming
- advantages of certifying algorithms
- do certifying algorithms always exist?
- verification of checkers
- collaboration of checking and verification

Linear System Solving

- does the linear system $A \cdot x = b$ have a solution?
- answer yes/no

- a solution x_0 witnesses solvability (= the answer yes)
- a vector c with $c^T A = 0$ and $c^T \cdot b \neq 0$ witnesses non-solvability (= the answer no)
  - assume x_0 is a solution, i.e., $Ax_0 = b$.
  - multiply with c^T from the left and obtain $c^T A x_0 = c^T b$
  - thus $0 \neq 0$.

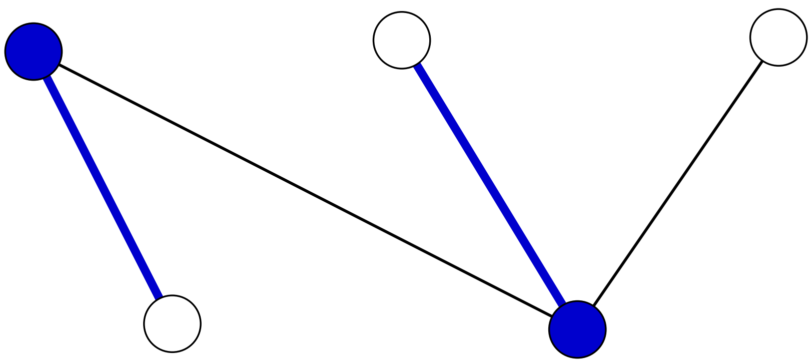
- Gaussian elimination computes solution x_0 or vector c
- checking is trivial

Bipartite Graphs

- is a given graph G bipartite?
- two-coloring witnesses bipartiteness
- odd cycle witnesses non-bipartiteness
- an algorithm
  - construct a spanning tree of G
  - use it to color the vertices with colors red and blue
  - check for all non-tree edges e whether the endpoints have different colors
  - if yes, the graph is bipartite and the coloring proves it
  - if no, let $e = \{u, v\}$ be a non-tree edge whose endpoints have the same color;
    - e together with the tree path from u to v is an odd cycle
    - tree path from u to v has even length since u and v have the same color

Bipartite Matching

- given a bipartite graph, compute a maximum matching
- a matching M is a set of edges no two of which share an endpoint
- a node cover C is a set of nodes such that every edge of G is incident to some node in C.
- $|M| \leq |C|$ for any matching M and any node cover C.
    - map $(u,v) \in M$ to an endpoint in C, this is possible and injective

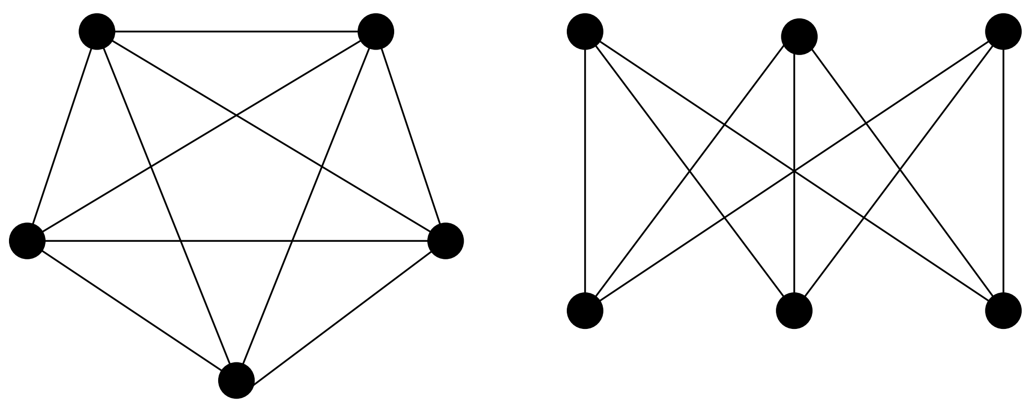


- a certifying alg returns M and C with $|M| = |C|$
- no need to understand that such a C exists **(!!!)**
- it suffices to understand the inequality $|M| \leq |C|$
- demo for general graphs

Planarity Testing

- given a graph G, decide whether it is planar
- Tarjan (76): planarity can be tested in linear time
- a story and a demo
- combinatorial planar embedding is a witness for planarity
- Chiba et al (85): planar embedding of a planar G in linear time
- Kuratowski subgraph is a witness for non-planarity
- Hundack/M/Näher (97): Kuratowski subgraph of non-planar G in linear time

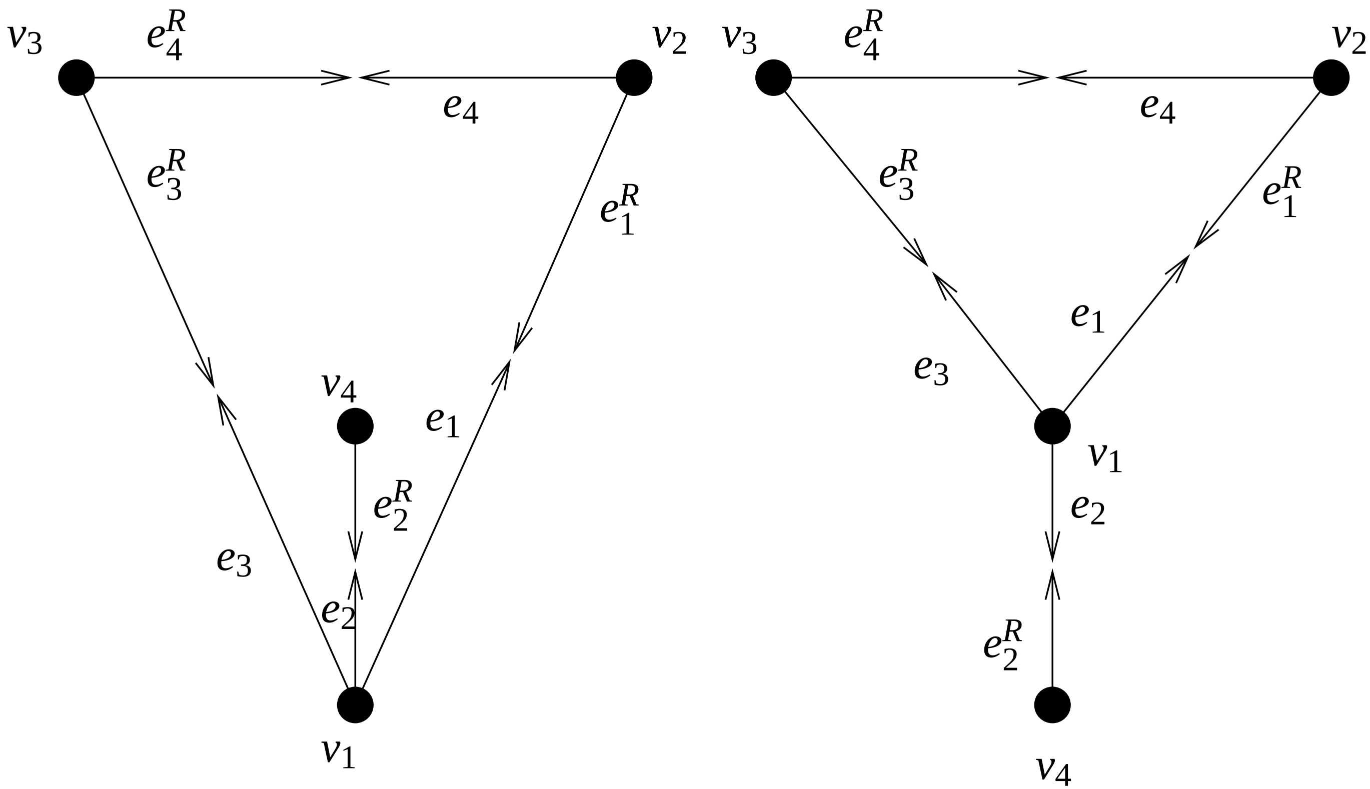
LEDAbook, Chapter 9



K_5 $K_{3,3}$

Planarity Testing: Checking the Witness I

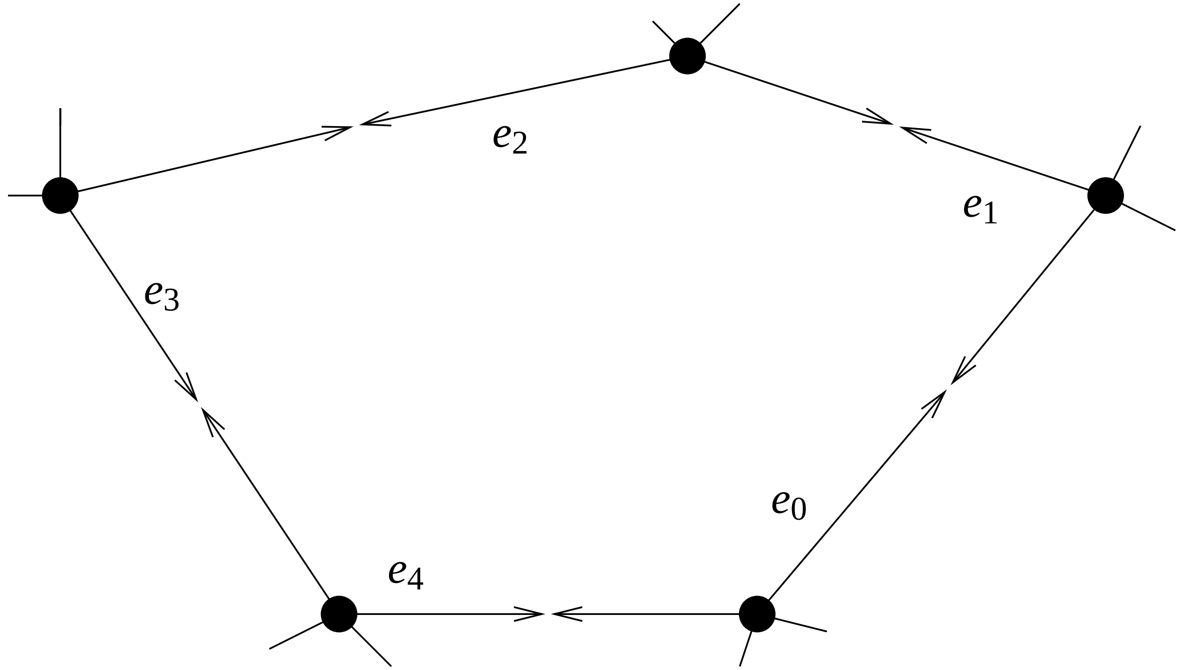
- combinatorial embedding: graph + cyclic order on the edges incident to any vertex



- combinatorial planar embedding: combinatorial embedding such that there is a plane drawing conforming to the ordering

Planarity Testing: Checking the Witness II

- face cycles



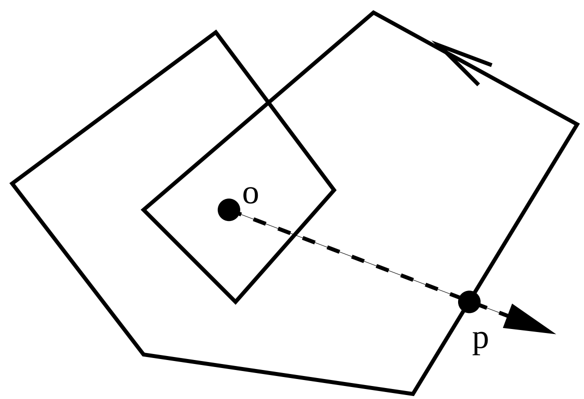
- face cycles are defined for combinatorial embeddings.
- **Theorem 0 (Euler, Poincaré)** *A combinatorial embedding of a connected graph is a combinatorial planar embedding iff*

$$f - e + n = 2$$

- theorem = easy check whether a combinatorial embedding is planar.

Convex Hulls

Given a simplicial, piecewise linear closed hyper-surface F in d-space decide whether F is the surface of a convex polytope.



FACT: F is convex iff it passes the following three tests MNSSSS

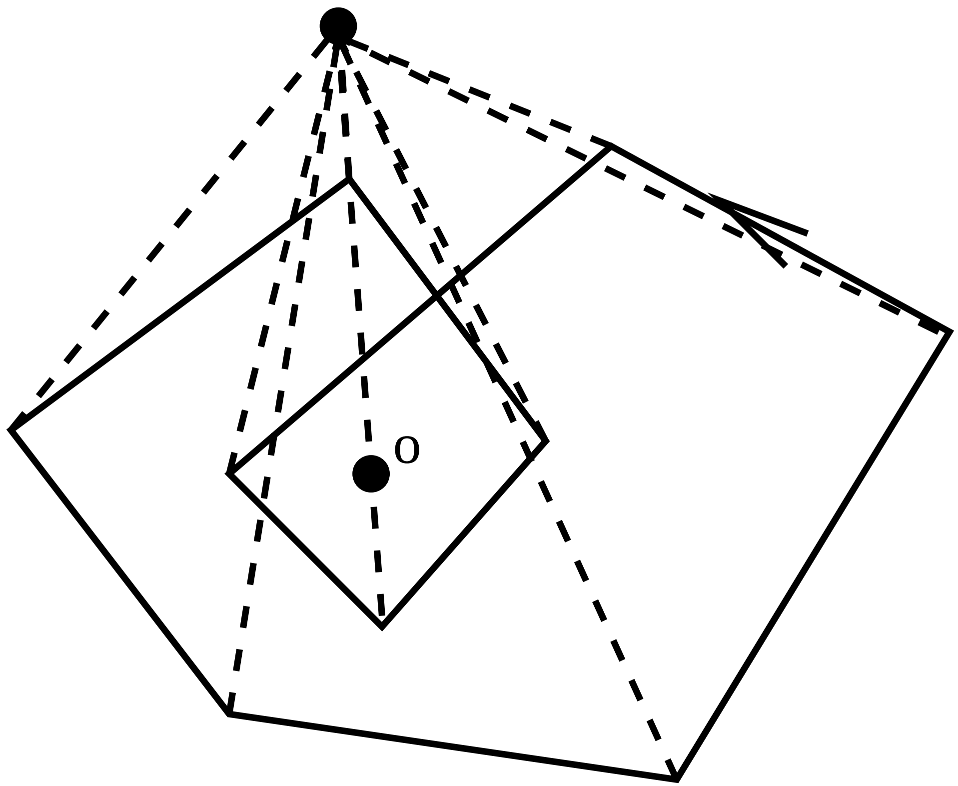
1. check local convexity at every ridge
2. $0 =$ center of gravity of all vertices

   check whether 0 is on the negative side of all facets
3. $p =$ center of gravity of vertices of some facet f

   check whether ray $\vec{0p}$ intersects closure of facet different from f

Sufficiency of Test is a Non-Trivial Claim

- ray for third test cannot be chosen arbitrarily, since in R^d, $d \geq 3$, ray may "escape" through lower-dimensional feature.



Monitoring Priority Queues I

a PQ maintains a set S (of real numbers) under the operations insert and delete_min

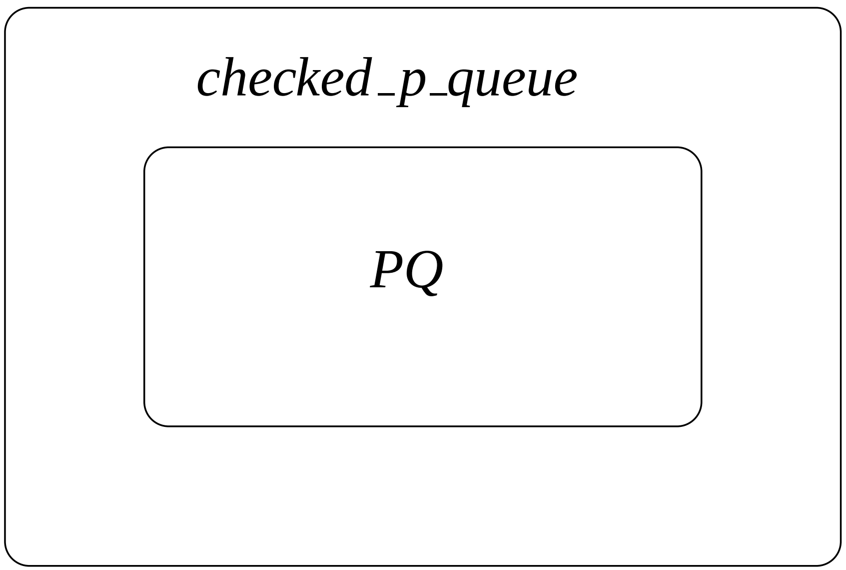
$insert(5)$,	$insert(2)$,	$insert(4)$,	$delete\_min$,	$insert(7)$,	$delete\_min$
			must return 2		must return 4
			returns 2		return 5

Monitoring Priority Queues I

a PQ maintains a set S (of real numbers) under the operations insert and delete_min

$insert(5)$, $insert(2)$, $insert(4)$, $delete\_min$, $insert(7)$, $delete\_min$

$delete\_min$	$delete\_min$
must return 2	must return 4
returns 2	return 5



A checker wraps around any priority queue PQ and monitors its behavior.

- It offers the functionality of a priority queue
- It complains if PQ does not behave like a priority queue.
  - immediately
  - ultimately

Monitoring Priority Queues II

Fact: Priority queue implementations with logarithmic running time per operation exist.

Fact:

- There is a checker with additional constant amortized running time per operation.
  It catches errors ultimately, namely with linear delay
- Immediate error catching requires $\Omega(\log n)$ additional time per operation.

Finkler/Mehlhorn, SODA 99

Linear Programming

maximize $c^T x$ subject to $Ax \leq b \quad x \geq 0$

- linear programming is a most powerful algorithmic paradigm
- there is no linear programming solver that is guaranteed to solve large-scale linear programs to optimality. Every existing solver may return suboptimal or infeasible solutions.

Problem				**CPLEX**				**Exact Verification**
Name	C	R	NZ	T	V	Res	RelObjErr	T
degen3	1504	1818	26230	8.08	0	opt	6.91e-16	8.79
etamacro	401	688	2489	0.13	10	dfeas	1.50e-16	1.11
fffff800	525	854	6235	0.09	0	opt	0.00e+00	4.41
pilot.we	737	2789	9218	3.8	0	opt	2.93e-11	1654.64
scsd6	148	1350	5666	0.1	13	dfeas	0.00e+00	0.52

Dhiflaoui/Funke/Kwappik/M/Seel/Schömer/Schulte/Weber: SODA 03

The Advantages of Certifying Algorithms

MAX-PLANCK-GESELLSCHAFT

- certifying algs can be tested on
  - **every** input
  - and not just on inputs for which the result is known.
- certifying programs are reliable
  - either give the correct answer
  - or notice that they have erred
- there is no need to understand the program, understanding the witness property and the checking program suffices.
- formal verification of checkers is feasible
- one may even keep the program secret and only publish the checker
- most programs in LEDA are certifying

Does every Function have a Certifying Alg?

$W : X \times Y \times W \mapsto \{0,1\}$ 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. 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
3. witness property is easily verified, i.e., the implication

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

has an elementary proofs.

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

Does every Function have a Certifying Alg?

MAX-PLANCK-GESELLSCHAFT

- 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

- 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 Algorithm has a Certifying Counterpart

MAX-PLANCK-GESELLSCHAFT

- 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

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
- Harald Ganzinger and I are exploring the idea

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).
- give other examples where a combination of verification and checking does the job

Summary

MAX-PLANCK-GESELLSCHAFT

- certifying algs have many advantages over standard algs
  - can be tested on every input
  - can assumed to be reliable
  - can be relied on without knowing code
  - . . .
- they exist: every deterministic alg has a certifying counterpart
- they are non-trivial to find
- most programs in the LEDA system are certifying
- Monte Carlo algs resist certification

Summary

- certifying algs have many advantages over standard algs
  - can be tested on every input
  - can assumed to be reliable
  - can be relied on without knowing code
  - . . .
- they exist: every deterministic alg has a certifying counterpart
- they are non-trivial to find
- most programs in the LEDA system are certifying
- Monte Carlo algs resist certification

When you design your next algorithm,

make it certifying