Algorithms and Programs

Erasmus Award Lecture

Kurt Mehlhorn

max planck institut
informatik

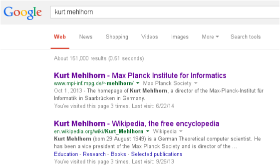
Goals of Talk

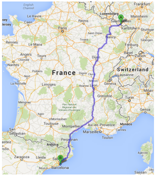
- Better understanding of Informatics (Computer Science)
- My fascination for the field and glimpses at my work
- A bit entertaining

slides available on my homepage

Outline of Talk

- Information Technology has changed the world





- Early Fascination
- Algorithms and Programs
  - What is an Algorithm? What is a Program?
  - Laws of Computation
  - Connections to Other Fields
  - Theory and Practice, Mathematics and Engineering
- Future

My First Encounter with Informatics

- Informatics was introduced as an area of study in Germany in 1968; I started studying in 1968 (Math, Physics, CS).
- I had no idea what informatics was about; I had never seen a computer except in pictures.
- The lectures
  - Math: polished, consistent in difficulty, basement of a tower of knowledge, instructor had material at his fingertips.
  - Informatics: rough, uneven in difficulty, bits and pieces, instructor (F. L. Bauer) struggled with content.

Ena and I, 73

F. L. Bauer

Early Fascination

Formal Languages and Programming Languages

- Syntax and semantics defined without any ambiguity
- Programming language = language for specifying computations
- Algol 68: definition finished in Dec. ’68; given to read in Jan. ’69

Early Fascination

Formal Languages and Programming Languages

- Syntax and semantics defined without any ambiguity
- Programming language = language for specifying computations
- Algol 68: definition finished in Dec. '68; given to read in Jan. '69

Algorithms and Programs

- An algorithm is a step-by-step procedure for solving a certain class of problems.
- Two examples

Al-Khwarizmi, 780 – 850

The Compendious Book on Calculation by Completion and Balancing

A First Algorithm: Calf's Liver

1-pound piece calf's liver, 3/4 pound onions, sliced paper-thin, 4 tablespoons unsalted butter, 1/3 cup water, 1 tablespoon chopped parsley

1. Rinse liver to remove any traces of blood and discard membrane and any tough veins. Cut cross-wise into 1/8-inch-thick slices.
2. Cook onions in 2 tablespoons butter in a 12-inch heavy skillet over moderate heat, stirring for 1 minute. Cover skillet and continue to cook, stirring occasionally, until softened and golden brown, about 10 minutes. Transfer to a bowl and keep warm, covered.
3. Pat half of liver slices dry and toss with salt and pepper to taste. . . .

A Second Algorithm: Solving a Quadratic Equation

Algorithm	Sample Execution
write the equation as $x^2 + bx + c = 0$	$x^2 + 8x - 9 = 0$
move the constant term to the other side	$x^2 + 8x = 9$
add $(b/2)^2$ on both sides	$x^2 + 8x + 4^2 = 9 + 4^2$
write LHS as $(x + b/2)^2$, simplify RHS	$(x + 4)^2 = 25$
if RHS is negative, STOP (no solution)	
remove ^{2} on LHS, replace RHS by $\pm\sqrt{\text{RHS}}$	$x + 4 = \pm\sqrt{25}$
move constant term from LHS to RHS	$x = -4 \pm \sqrt{25}$

Algorithm is in Al-Kwarizmi's book.

A Second Algorithm: Solving a Quadratic Equation

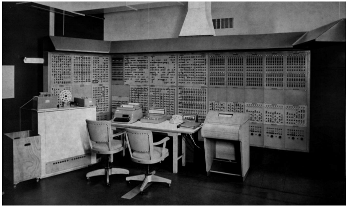
Algorithm	Sample Execution
write the equation as $x^2 + bx + c = 0$	$x^2 + 8x - 9 = 0$
move the constant term to the other side	$x^2 + 8x = 9$
add $(b/2)^2$ on both sides	$x^2 + 8x + 4^2 = 9 + 4^2$
write LHS as $(x + b/2)^2$, simplify RHS	$(x + 4)^2 = 25$
if RHS is negative, STOP (no solution)	
remove ^{2} on LHS, replace RHS by $\pm\sqrt{\text{RHS}}$	$x + 4 = \pm\sqrt{25}$
move constant term from LHS to RHS	$x = -4 \pm \sqrt{25}$

Algorithm is in Al-Kwarizmi's book.

Both algorithms are for human computers,
programs for real computers are much more detailed.

Algorithms and Programs

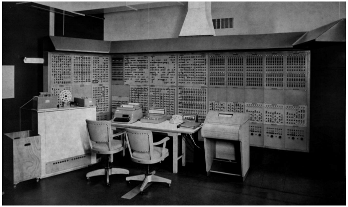
- An algorithm is a step-by-step procedure for solving a certain class of problems.
- Can be executed mechanically.
- Program = algorithm formulated in a programming language



The PERM

Algorithms and Programs

- An algorithm is a step-by-step procedure for solving a certain class of problems.
- Can be executed mechanically.
- Program = algorithm formulated in a programming language



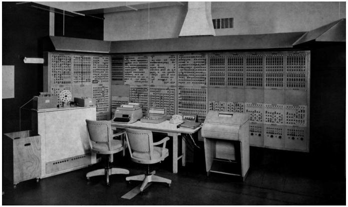
The PERM

Formative Insights

- Intellectually demanding tasks can be mechanized.

Algorithms and Programs

- An algorithm is a step-by-step procedure for solving a certain class of problems.
- Can be executed mechanically.
- Program = algorithm formulated in a programming language

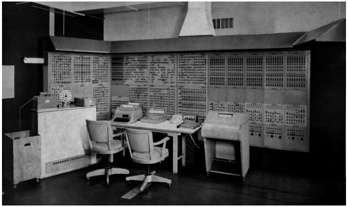


The PERM

Formative Insights

- Intellectually demanding tasks can be mechanized.
- Computers amplify brainpower.

- An algorithm is a step-by-step procedure for solving a certain class of problems.
- Can be executed mechanically.
- Program = algorithm formulated in a programming language



The PERM

Formative Insights

- Intellectually demanding tasks can be mechanized.
- Computers amplify brainpower.
- Algorithm design and programming are acts of creation.

Laws of Computation

Physicists discover laws of nature;

informaticians discover laws of computation,

Physicists discover laws of nature;
informaticians discover laws of computation,

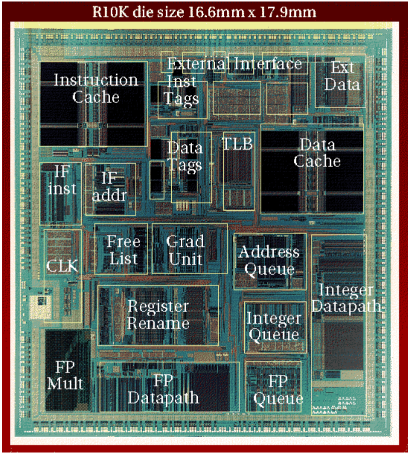
in particular, limits of efficiency for performing certains tasks.

We measure efficiency in terms of

- Time (number of elementary steps needed), and
- Space (number of symbols needed to store intermediate results).

A Computer Chip (Vintage 1995)

Der MIPS R10K Prozessorchip.



The block FP-Mult
multiplies numbers.

It occupies about 3 by 4 mm of space.

It multiplies numbers in 5 microseconds

Is this good?

How Hard is it to Multiply Numbers?

- Complexity measures

$$\begin{aligned} A &= \text{area of the circuit} \\ T &= \text{execution time} \end{aligned}$$

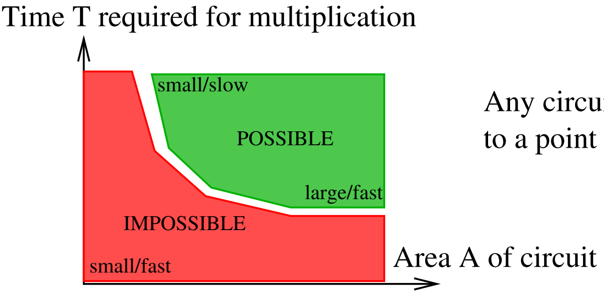
- There are many different circuits for multiplication:
  - Circuits that are fast, but also large,
  - Circuits that are small, but also slow.

Is there a circuit that is small and fast?

Is there a Circuit that is Small and Fast?

NO

Is there a Circuit that is Small and Fast?

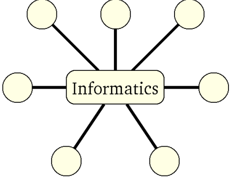


Any circuit corresponds to a point in this diagram.

Every combination of A and T in red region is impossible.

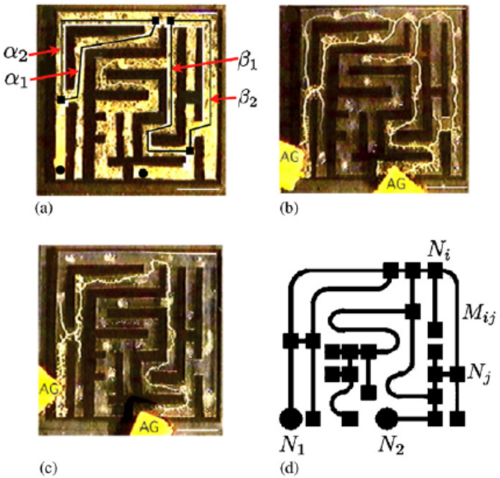
Every combination of A and T in green region can be realized.

Connections to Other Fields



Engineering	embedded systems, circuit design, autonomous systems, computer vision
Linguistics	machine translation, speech recognition, information retrieval
Biology and medicine	bioinformatics, computations in nature
Economics	business processes, game theory, algorithmic economics
Art	computer graphics
Natural sciences	high performance computing

The Physarum Computer, Nakagaki, Yamada, Tóth, Nature 2000



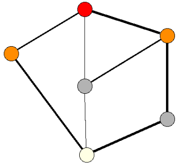
Physarum,
a slime mold

single cell,
several nuclei

show video

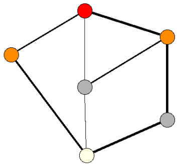
Mathematical Model (Tero et al., J. Theor. Biology 2007)

- Physarum is a network of pipes.
- Flow of liquids determined by concentration differences and lengths and diameters of pipes.
- Pipes adapt: if flow through a pipe is high (low) relative to diameter of the pipe, the diameter grows (shrinks).



Mathematical Model (Tero et al., J. Theor. Biology 2007)

- Physarum is a network of pipes.
- Flow of liquids determined by concentration differences and lengths and diameters of pipes.
- Pipes adapt: if flow through a pipe is high (low) relative to diameter of the pipe, the diameter grows (shrinks).



Theorem (Bonifaci, KM, Varma, 2012, J. Theor. Biology)

Network converges to shortest path, i.e.,

the diameter of tubes not on the shortest path converges to zero.

Mathematician in the Morning, Engineer in the Afternoon

Mathematician

works on problems of a fundamental nature that

- are intrinsic to the field, e.g., laws of computation or new algorithms, or
- come up in applications (and are too hard for engineers)

Engineer

- turns algorithms into programs
- designs and builds systems
- turns ideas into working systems
- identifies the need for further theory

Library of Efficient Data Structures and Algorithms

joint work with S. Näher and C. Uhrig

- When I asked former students, ...

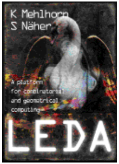
Library of Efficient Data Structures and Algorithms

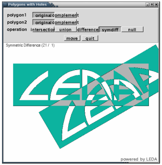
joint work with S. Näher and C. Uhrig

- When I asked former students, ...
- Insight (1990): Writing books and articles does not suffice.

  Must turn knowledge of the field into software that is

  - easy-to-use,
  - correct, and
  - efficient.





- Started LEDA project in 1990, later CGAL, STXXL, SCIL
- In use at thousands of academic and industrial sites

  Algorithmic Solutions GmbH

The Project Almost Ruined My Reputation

Some of our programs were incorrect.

Some of our programs were incorrect.

What had gone wrong and what did we do about it?

- We had followed the state of the art, however,

The Project Almost Ruined My Reputation

Some of our programs were incorrect.

What had gone wrong and what did we do about it?

- We had followed the state of the art, however,
- there was no scientific basis available for geometric computations.

We and others created a basis over the next 20 years.

Some of our programs were incorrect.

What had gone wrong and what did we do about it?

- We had followed the state of the art, however,
- there was no scientific basis available for geometric computations.

  We and others created a basis over the next 20 years.

- We made mistakes and wrote incorrect programs.

  Adopted new design principle: certifying algorithms.

Some of our programs were incorrect.

What had gone wrong and what did we do about it?

- We had followed the state of the art, however,
- there was no scientific basis available for geometric computations.

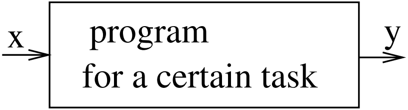
We and others created a basis over the next 20 years.

- We made mistakes and wrote incorrect programs.

Adopted new design principle: certifying algorithms.

Today, LEDA is a main source of my reputation.

The Problem



- A user feeds x to the program, the program returns y.
- How can the user be sure that

  y is indeed the correct output for input x?

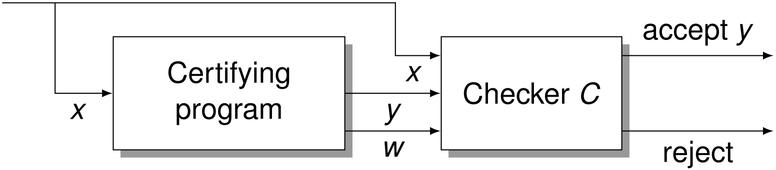
  The user has no way to know.

analogy: construction company

The Proposal

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

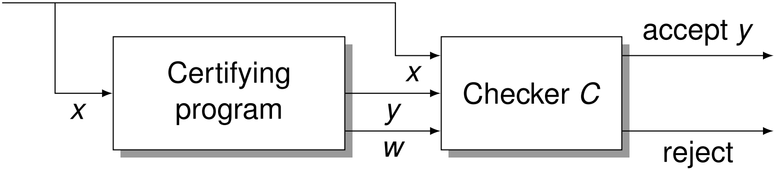
Certifying Algorithms



On input x, a certifying algorithm computes

- the function value y and
- a witness w. (convincing evidence that y is the correct output for x)

Certifying Algorithms

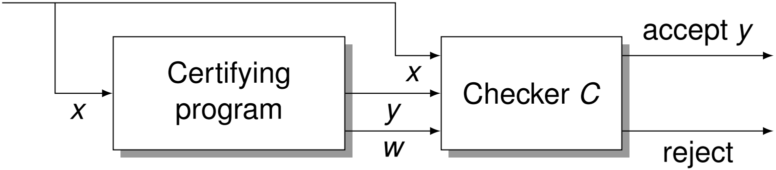


On input x, a certifying algorithm computes

- the function value y and
- a witness w. (convincing evidence that y is the correct output for x)

w is inspected by either the human user of the certifying program,

Certifying Algorithms



On input x, a certifying algorithm computes

- the function value y and
- a witness w. (convincing evidence that y is the correct output for x)

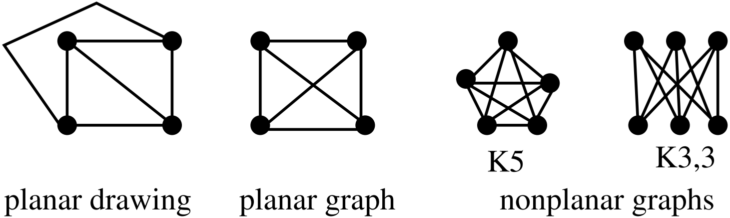
w is inspected by either the human user of the certifying program,

or more elegantly, by a checker program C.

Example: Planarity Test

Planar Graph

A graph is planar if it can be drawn in the plane without edge crossings.



Fact: Every non-planar graph contains a Kuratowski graph.

Story and Demo

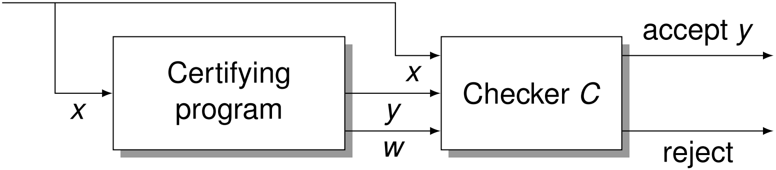
History

- I do not claim that I invented the concept; it is rather an old concept
  - al-Kwarizmi (780 – 850): multiplication
  - extended Euclid ($\approx$ 1700): gcd
  - primal-dual algorithms in combinatorial optimization
  - Blum et al.: Programs that check their work
- I do claim that Stefan Näher and I were the first (1995) to adopt the concept as the design principle for a software project:

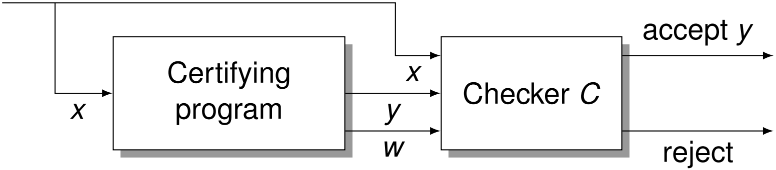
  LEDA (Library of Efficient Data Types and Algorithms)

- McConnell/M/Näher/Schweitzer (2010): 80 page survey

Who Checks the Checker?

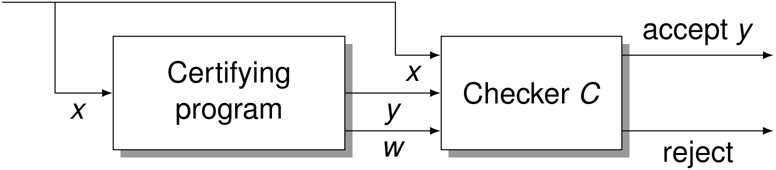


Who Checks the Checker?



Answer as of 2011: Checkers are simple programs, and hence, we get them correct.

Who Checks the Checker?



Answer as of 2011: Checkers are simple programs, and hence, we get them correct.

Today's answer: Checkers are simple programs, and hence, we can prove their correctness using formal mathematics.

Formal Mathematics

- Mathematics is carried in a formal language without any ambiguities.
- Proofs are machine-checked.
- Isabelle/HOL (L. Paulson/T. Nipkow)

definition *disjoint-edges* :: (α, β) *pre-graph* $\Rightarrow \beta \Rightarrow \beta \Rightarrow$ *bool* **where**
disjoint-edges $G\ e_1\ e_2 = ($
$start\ G\ e_1 \neq start\ G\ e_2 \wedge start\ G\ e_1 \neq target\ G\ e_2 \wedge$
$target\ G\ e_1 \neq start\ G\ e_2 \wedge target\ G\ e_1 \neq target\ G\ e_2)$

definition *matching* :: (α, β) *pre-graph* $\Rightarrow \beta$ *set* $\Rightarrow$ *bool* **where**
matching $G\ M = ($
$M \subseteq edges\ G \wedge$
$(\forall e_1 \in M.\ \forall e_2 \in M.\ e_1 \neq e_2 \longrightarrow disjoint\text{-}edges\ G\ e_1\ e_2))$

definition edge_as_*set* :: $\beta \Rightarrow \alpha$ *set* **where**
edge_as_*set* e $\equiv$ {tail G e, head G e}

lemma *matching*_disjointness:
assumes *matching* G M
assumes $e_1 \in$ M **assumes** $e_2 \in$ M **assumes** $e_1 \neq e_2$
shows edge_as_*set* $e_1 \cap$ edge_as_*set* e_2 = {}
using assms
by (auto simp add: edge_as_*set*_def disjoint_arcs_def *matching*_def)

Formal Mathematics

- Mathematics is carried in a formal language without any ambiguities.
- Proofs are machine-checked.
- Isabelle/HOL (L. Paulson/T. Nipkow)

definition *disjoint-edges* :: (α, β) *pre-graph* $\Rightarrow \beta \Rightarrow \beta \Rightarrow$ *bool* **where**
  disjoint-edges $G\ e_1\ e_2 = ($
    start $G\ e_1 \neq$ *start* $G\ e_2 \wedge$ *start* $G\ e_1 \neq$ *target* $G\ e_2 \wedge$
    target $G\ e_1 \neq$ *start* $G\ e_2 \wedge$ *target* $G\ e_1 \neq$ *target* $G\ e_2)$

definition *matching* :: (α, β) *pre-graph* $\Rightarrow \beta$ *set* $\Rightarrow$ *bool* **where**
  matching $G\ M = ($
    $M \subseteq$ *edges* $G \wedge$
    $(\forall e_1 \in M.\ \forall e_2 \in M.\ e_1 \neq e_2 \longrightarrow$ *disjoint-edges* $G\ e_1\ e_2))$

definition edge_as_*set* :: $\beta \Rightarrow \alpha$ *set* **where**
  edge_as_*set* e $\equiv$ {tail G e, head G e}

lemma *matching*_disjointness:
  assumes *matching* G M
  assumes $e_1 \in$ M **assumes** $e_2 \in$ M **assumes** $e_1 \neq e_2$
  shows edge_as_*set* $e_1 \cap$ edge_as_*set* e_2 = {}
  using assms
  by (auto simp add: edge_as_*set*_def disjoint_arcs_def *matching*_def)

We formally verify

- the witness property (e.g., Kuratowski $\rightarrow$ non-planarity),
- termination of the checker program, and
- correctness of the checker program.

Formal Verification: Summary

Formal Instance Correctness

If a formally verified checker accepts a triple (x, y, w),

we have a formal proof that y is the correct output for input x.

- Highest achievable trust (only Isabelle kernel to be trusted)
- A way to build large libraries of trusted algorithms

Alkassar/Böhme/M/Rizkallah: Verification of Certifying Computations, JAR 2014

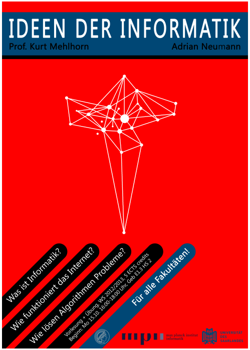
Noshinski/Rizkallah/M: Verification of Certifying Computations through AutoCorres and Simpl, NASA Formal Methods Symposium 2014

Trends

- From information to knowledge
- bBg data and machine learning
- Autonomous systems
- Security, privacy, and accountability

Ideas of Informatics (http://resources.mpi-inf.mpg.de/departments/d1/teaching/ws14/Ideen-der-Informatik/)

- An introduction to informatics for non-specialists (Studium Generale)
- Goal: informatics litteracy
- Presents concepts and their applications, e.g.,
  - cryptography and electronic banking
  - shortest path algs and navigation systems
  - machine learning and automatic translation
- In WS 14/15: (internal) online-course
- In WS 15/16: public on-line course (Iversity) with credit points.



Thank You