Infinity in Informatics: Endless Loops

Symposium ‘Blik op ∞neindig’
18 November 2009 / De Leidsche Flesch / Leiden

Tom Verhoeoff
Technische Universiteit Eindhoven / Wiskunde & Informatica

© 2009, T. Verhoeoff @ TUE.NL

Motivations for Working on Scientific Problems

- Direct application in real life
- Foundation for working on other problems
- Aid to acquisition of knowledge and development of skills
- Fun and enjoyment

Informatics

What is Informatics?

Also known as Computing Science (UK) and Computer Science (US)

Branch of Mathematics?

Study of algorithms and related concepts

Actual infinity (finished) versus potential infinity (process)

Computation Tasks

Given an input, determine an output satisfying the task specification

A valid task input is also called problem instance

Not just computations on numbers:

- Find a word in a (sorted) list
- Sort a list of words
- Find shortest path in a road network
- Find an Euler circuit or a Hamiltonian circuit in a graph
- Schedule jobs on machines to minimize processing time
The D-L-F Counting Problems

Given is a sequence of $N$ characters in memory (an array $A$)

Efficiently count the number of occurrences of the character $L$ when

1. Only characters $D$ and $L$ occur, no $L$ left of $D$

   $\text{DDDDDLLLLL}$

2. Only characters $D$, $L$, and $F$ occur, no $L$ left of $D$, no $F$ left of $L$ or $D$

   $\text{DDDDDLLLLLLLLFF}$

3. Only characters $D$ and $L$ occur, no $D$ between $L$'s

   $\text{DDDDDLLLLLD} D D D D$

© 2009, T. Verhoeoff @ TUE.NL 5/30 Infinity in Informatics

Algorithms

An algorithm is an effective solution method for a computational task:

- Given any valid input (problem instance)
- the algorithm must exactly prescribe what steps to perform,
- this must terminate after a finite number of steps, and
- produce a correct output (satisfying the task specification)

Input and output can be abstract entities, possibly infinitely many

One does not need to understand why an algorithm works and how it was discovered when applying it to solve given problem instances

© 2009, T. Verhoeoff @ TUE.NL 6/30 Infinity in Informatics

(Computer) Programs

A mathematically precise algorithm can be executed by an automaton

A program is a sequence of instructions executable by a computer

Many instruction sets or programming languages are possible

All sufficiently rich programming languages are universal and, hence, equivalent in terms of computational expressibility

- Not every program is an algorithm: programs need not terminate
- Every algorithm can be expressed by a suitable program

*If the programming language is universal (Church–Turing Thesis)

© 2009, T. Verhoeoff @ TUE.NL 7/30 Infinity in Informatics

Computer as Universal Automaton

Programming = rewriting algorithms into programs

Universal programming language:

Integer variables with statements: $\text{inc}(v)$ $\text{dec}(v)$ while $v$ do ... od

© 2009, T. Verhoeoff @ TUE.NL 8/30 Infinity in Informatics
Central Questions of Informatics

- What tasks are (not) algorithmically solvable?
- What tasks are efficiently solvable?
- What bounds are there on the efficiency for solving certain tasks?
- How to design efficient algorithms?
- How to reason about the correctness and efficiency of algorithms?

Termination of Computations: A Recent Success

- Start with finite sequence over \( \{a, b, c\} \)
  
  \[
  \text{bbaa}
  \]

- Repeatedly replace subsequences:
  
  \[
  \begin{align*}
  aa & \rightarrow bc \\
  bb & \rightarrow ac \\
  cc & \rightarrow ab
  \end{align*}
  \]

Example:

\[
\begin{align*}
\text{bbaa} & \rightarrow \text{bbbc} \rightarrow \text{baac} \rightarrow \text{baab} \rightarrow \text{bbcb} \rightarrow \text{aceb} \rightarrow \text{aabb} \rightarrow \text{aaac} \rightarrow \text{abcc} \rightarrow \text{abab}
\end{align*}
\]

Does this terminate for every start sequence?

Termination of Computations: Famous Open Problems

- If \( N \) even \( \rightarrow N/2 \), if \( N \) odd \( \rightarrow 3N + 1 \) (Collatz)
  
  \[
  3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow \ldots
  \]

- While \( N \) is not a palindrome, add it to its reverse
  
  \[
  152 \rightarrow 152 + 251 = 403 \rightarrow 403 + 304 = 707 \quad 196 \rightarrow ?
  \]

Marble Game 1

Each time take a marble:

- Remove it

Does this terminate? After how many steps?
Marble Game 2

Repeatedly take a marble:

- If blue, then remove
- If white, then replace by one blue marble

Marble Game Analysis

$W$ white marbles  $B$ blue marbles

**Game 1.** Terminates after $f_1(W, B) = W + B$ steps

**Game 2.** Terminates after $f_2(W, B) = 2 \cdot W + B$ steps

**Game 3.** Terminates after ten hoogste $f_3(W, B) = \omega \cdot W + B$ steps

Lotto Ball Game

Repeatedly take an $N$-numbered lotto ball:

- Replace by arbitrary number of balls with smaller numbers
  i.e. $(n)$ replaced by $(<n) (<n) \cdots (<n)$

© 2009, T. Verhoeff @ TUE.NL  13/30  Infinity in Informatics

© 2009, T. Verhoeff @ TUE.NL  14/30  Infinity in Informatics

© 2009, T. Verhoeff @ TUE.NL  15/30  Infinity in Informatics

© 2009, T. Verhoeff @ TUE.NL  16/30  Infinity in Informatics
How Large Is the Set of All Programs?

Every program is a text over some suitable enumerable alphabet $\mathcal{A}$. Not every text over $\mathcal{A}$ is program: it must be syntactically correct according to the rules of the programming language.

A compiler checks the syntactical correctness of a text as a program (but not semantical correctness: whether the text is an algorithm).

Construct an enumeration of all programs from an enumeration of all texts over $\mathcal{A}$ by deleting all texts that are not syntactically correct:

$$P_0, P_1, P_2, \ldots, P_n, \ldots$$

where $P_i$ denotes the $i$-th program. $\text{|Programs|} = |\mathbb{N}|$

Every algorithm appears in the sequence, not every $P_i$ is an algorithm.* Each programming language gives rise to its own enumeration.

Provided the programming language is sufficiently expressive.

Intermezzo: The Power Set Consisting of All Subsets

For a set $S$, its power set $\mathcal{P}(S)$ consists of all subsets of $S$:

$$\mathcal{P}(S) = \{ T \mid T \subseteq S \}$$

E.g. $\mathcal{P}([0,1]) = \{ \emptyset, \{0\}, \{1\}, \{0,1\} \}$

Attempt to match $S$ and $\mathcal{P}(S)$:

\[
\begin{array}{cccccccc}
 x \in S & \rightarrow & \text{subset } T_x \subseteq S & : & y \in T_x \\
 a & & b & & c & & d & \ldots \\
 x \rightarrow & o & o & o & o & \ldots \\
 b \rightarrow & o & o & o & \ldots \\
 c \rightarrow & o & o & o & \ldots \\
 d \rightarrow & o & o & o & \ldots \\
 ? \rightarrow & o & o & o & \ldots \\
 & D = \text{complement of diagonal} \\
\end{array}
\]

How Large Is the Set of All Computational Tasks?

For $M \subseteq \mathbb{N}$, a decision problem $(\mathbb{N}, M)$ is defined by

- **Input:** a natural number $n \in \mathbb{N}$
- **Output:** YES if $n \in M$, NO if $n \notin M$

N.B.: $M$ is not an input of the problem, but a ‘built-in’ constant.

Example: for primality testing take $M := \{ 2, 3, 5, 7, 11, 13, 17, 19, \ldots \}$

$$|\text{Tasks}| \geq |\mathcal{P}(\mathbb{N})| = |\mathbb{N} \rightarrow \{ \text{NO, YES} \}| > |\mathbb{N}|$$

Intermezzo: A Set is Smaller Than Its Power Set (Cantor)

**Diagonalization method:** assume matching $\{ x \leftrightarrow T_x \mid x \in S, T_x \subseteq S \}$

\[
D = \{ x \in S \mid x \notin T_x \}
\]

$x \in D$ if and only if $x \in S$ and $x \notin T_x$ for all $x$.

Hence, $D \in \mathcal{P}(S)$ is not matched with any $x \in S$.

Consequently, $S$ is smaller than $\mathcal{P}(S)$: $|S| < |\mathcal{P}(S)|$, in particular

$|\mathbb{N}| < |\mathcal{P}(\mathbb{N})|$

The power set of $S$ is isomorphic to the set of mappings $S \rightarrow \{0,1\}$

$f : S \rightarrow \{0,1\}$ corresponds to $\{ x \in S \mid f(x) = 1 \}$

$\mathcal{P}(\mathbb{N})$ corresponds to the set of all N-infinite 0,1-sequences.
Not All Problems \((N,M)\) Are Decidable

Algorithm \(A\) solves decision problem \((N,M)\) when

for any given \(n \in \mathbb{N}\), \(A\) outputs \(\text{YES}\) if \(n \in M\) and \(\text{NO}\) if \(n \notin M\)

\((N,M)\) is called **decidable** when there exists an algorithm to solve it, and **undecidable** if no such algorithm exists.

Because \(|P(N)| > |\mathbb{N}|\), there are more problems \((N,M)\) than algorithms.

There exist \(M \subseteq \mathbb{N}\) such that \((N,M)\) is undecidable.

Decision Problem \((N,\text{DIAG})\)

Define

\[
\text{DIAG} = \{i \in \mathbb{N} \mid \text{program } P_i \text{ does not output YES on input } i\}
\]

N.B. Each way of enumerating all programs, gives rise to its own set \(\text{DIAG}\).

Problem \((N,\text{DIAG})\) is undecidable, because

no program \(P_i\) implements an algorithm \(A\) that solves \((N,\text{DIAG})\):

\[
\begin{align*}
\text{A outputs YES on input } i &\iff i \in \text{DIAG} \iff P_i \text{ does not output YES on input } i
\end{align*}
\]

Problems \text{UNIV} and \text{HALT}

More interesting problems:

**UNIV (the universal problem)**

Input: a program \(P\) and a natural number \(i \in \mathbb{N}\)

Output: \(\text{YES}\) if \(P\) outputs \(\text{YES}\) on input \(i\)

\(\text{NO}\) if \(P\) outputs \(\text{NO}\) or does not halt on input \(i\)

**HALT (the halting problem)**

Input: a program \(P\) and a natural number \(i \in \mathbb{N}\)

Output: \(\text{YES}\) if \(P\) halts on input \(i\)

\(\text{NO}\) if \(P\) does not halt on input \(i\)

N.B. Simulation of \(P\) will not work, because it need not terminate.

UNIV \(\leq_{\text{Alg}}\) HALT by Reduction

\[
\begin{array}{c}
P_i \quad i \\
\text{algorithm that decides the halting problem } \ A_{\text{HALT}} \\
\text{simulates the finite computation of } P \text{ on } i \\
P \quad S \quad B \\
\text{algorithm } B \text{ decides UNIV} \\
\end{array}
\]

\[
\begin{array}{c}
P \quad i \\
\text{YES} \\
\text{NO} \\
\text{YES} \\
\text{NO} \\
\end{array}
\]

\[
\begin{array}{c}
P \quad i \\
\text{YES} \\
\text{NO} \\
\text{YES} \\
\text{NO} \\
\end{array}
\]

\[
\begin{array}{c}
P \quad i \\
\text{YES} \\
\text{NO} \\
\text{YES} \\
\text{NO} \\
\end{array}
\]

\[
\begin{array}{c}
P \quad i \\
\text{YES} \\
\text{NO} \\
\text{YES} \\
\text{NO} \\
\end{array}
\]

\[
\begin{array}{c}
P \quad i \\
\text{YES} \\
\text{NO} \\
\text{YES} \\
\text{NO} \\
\end{array}
\]
HALT $\leq_{\text{Alg}}$ UNIV by Reduction

Modify \( P \) into \( P' \) in such a way, that \( P \) never answers NO by exchanging all occurrences of NO for YES

\[ P \]

\[ P' \]

\[ A_{\text{UNIV}} \]

\( A_{\text{UNIV}} \) decides whether \( i \) is in \( M(P) \) or not

\( C \)

\( D \)

algorithm that decides, whether \( P \) halts on \( i \)

\( \text{YES} \)

\( \text{NO} \)

\( \text{YES} \)

\( \text{NO} \)

\( \text{YES} \)

\( \text{NO} \)

\( \text{YES} \)

\( \text{NO} \)

(\( N, \text{DIAG} \) $\leq_{\text{Alg}}$ UNIV by Reduction)

\( A_{\text{gen}} \)

\( A_{\text{UNIV}} \)

\( A_{\text{UNIV}} \) accepts \( i \) if and only if the \( i^{\text{th}} \) program \( P_i \) does not accept \( i \)

\( A_{\text{UNIV}} \) decides whether \( i \) belongs to \( M(P_i) \) or not

\( \text{YES} \)

\( \text{NO} \)

\( \text{YES} \)

\( \text{NO} \)

\( \text{YES} \)

\( \text{NO} \)

Conclusion: UNIV and HALT are also not solvable by an algorithm

Undecidability Is Not Rare

- Decide* whether a Game of Life configuration stabilizes
- Decide whether a set of Wang tiles can tile the plane
- Decide whether a Diophantine equation (multivariable polynomial equation, like \( a^3 + b^3 = c^3 \)) has a solution in integers
- Decide whether a program has a specific non-trivial property, like whether it always halts, always outputs 0, ... [cf. Rice’s Theorem]

*In each case, the algorithm needs to work for all possible inputs (shown in yellow). All these decision problems turn out to involve a universal mechanism.

(Other) Infinities in Informatics

- Infinitely many computational tasks, with infinitely many inputs
- Infinitely many algorithms and programs
- Infinitely many execution traces, finite or infinite
- Infinite (nonterminating, reactive) computations: lists, trees, ...
- Inductively defined data types, having infinitely many values
- Computations on infinite sequences (generated by other programs)
- Infinite automata
Concluding Challenge to Grapple with Infinity

Write a program

- that is **not empty**
- that processes no input
- that produces a listing of its own source code

After some attempts, you might get into an infinite regress
But it is doable (and instructive to discover your own solution)

Try this Challenge on-line with Tom’s JavaScript Machine at

www.win.tue.nl/~wstomv/edu/javascript

---

**Literature**