Limits of Computation

15 - Hierarchy Theorems

Bernhard Reus

The complexity story so far

- how to measure running time for different models
- runtime bounds
- complexity classes, e.g. **LIN** and **P**
- Cook's (Invariance) Thesis and Cook-Karp Thesis

THIS TIME

Hierarchy Theorems

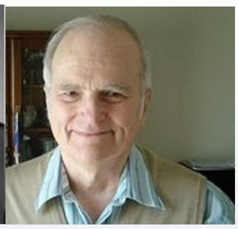
- Tackle the question: *"Can we decide more problems if we have 'more' time?"*
- *Exact formulation leads to* Hierarchy Theorems:
  - for WH¹LE constants in time bounds **do** matter.
  - in general where 'more' means (asymptotically) faster growing time bounds

"Runtime Complexity" Hierarchy – for Animals (on land)

image: www.eduplace.com (Leigh Haeger)

Can we decide more problems if we increase the time bound?

The "fathers" of computational complexity (this line of work started in 1965):


J. Hartmanis R.E. Stearns P. M. Lewis II

Turing Award 1993

...in other words, if time bound f is "smaller" than g, are there problems in $\mathrm{TIME}^{\mathtt{L}}(g)$ that are not in $\mathrm{TIME}^{\mathtt{L}}(f)$?

investigate this for different languages L and functions f,g

Constants do matter

Does $a < b$ imply $\mathbf{TIME}^{\mathtt{L}}(a \times n) \subsetneq \mathbf{TIME}^{\mathtt{L}}(b \times n)$?

proper inclusion

Does $a < b$ imply $\mathbf{TIME}^{\mathtt{L}}(b \times n) \setminus \mathbf{TIME}^{\mathtt{L}}(a \times n) \neq \emptyset$?

Can `L`-programs decide ***more*** problems with a ***larger*** constant in their ***linear*** "running time allowance"?

It can be shown that – in line with our intuition – this is true for the language $\mathtt{WH}^1\mathtt{LE}$.

Need some proof technique: ***timed universal program***

Time Hierarchy for $\texttt{WH}^1\texttt{LE}$

constants **do actually matter** for $\texttt{WH}^1\texttt{LE}$.

Theorem (Linear Time Hierarchy Theorem, [5, Thm. 19.3.1]). *There is a constant b such that for all $a \geq 1$ there is a problem A in $\textbf{TIME}^{\texttt{WH}^1\texttt{LE}}(a \times b \times n)$ that is not in $\textbf{TIME}^{\texttt{WH}^1\texttt{LE}}(a \times n)$.*

The proof uses a well known technique again that we used in the proof of the Undecidability of the Halting Problem (and the Barber's Paradox): self-reference (diagonalisation).

Open problem

- Does this theorem hold for `WHILE` and `GOTO` as well?
- Remark: one can show, however, an analogous theorem for `SRAM` (using logarithmic time measure).

Let us now consider super-linear time bounds and other languages (not just $\mathtt{WH^1LE}$)

What does "smaller" mean for general (super-linear) time bounds?

- f,g time bounds (functions mapping natural numbers to natural numbers)
- g "smaller than" f if
  f ***grows*** *(asymptotically)* ***much faster*** than g.
- f grows much faster than g if eventually $f(n) > g(n)$ for all n "large enough":
$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$$

asymptotic growth

Time constructible bounds

Definition A time bound $f : \mathbb{N} \to \mathbb{N}$ is called *time constructible* if there is a program p and a constant $c > 0$ such that for all $n \geq 0$ we have that

$$[\![p]\!](\ulcorner n \urcorner) = \ulcorner f(n) \urcorner \quad \text{and} \quad time_p(d) \leq c \times f(|d|) \quad \text{for all } d$$

The time bound is computable.
The time it takes to compute the time bound is itself bounded by the time bound up to a constant factor.

Big-O

should be familiar from module Program Analysis

Definition **(Big-O).** Let $f : \mathbb{N} \to \mathbb{N}$ be a function. The *order of* f, short $\mathscr{O}(f)$, is the set of all functions defined below:

$$\{\, g : \mathbb{N} \to \mathbb{N} \mid \forall n > n_0.\ g(n) \leq c \times f(n) \text{ for some } c \in \mathbb{N} \setminus \{0\} \text{ and } n_0 \in \mathbb{N} \}$$

In other words $\mathscr{O}(f)$ are those functions that up to constant factors grow at most as fast as f (or, in other words, not faster) and are thus at least "not worse" a runtime bound than f (maybe even "better"). For $g \in \mathscr{O}(f)$ we also say g is $\mathscr{O}(f)$.

Generalising $\mathbf{TIME}^{\mathtt{L}}(f)$

Definition For any timed programming language L we define another complexity class using "Big-O" as follows:

$$\mathbf{TIME}^{\mathtt{L}}(\mathcal{O}(f)) = \bigcup_{g\in\mathcal{O}(f)} \mathbf{TIME}^{\mathtt{L}}(g)$$

This is the class of problems L-decidable in $\mathcal{O}(f)$, that is with a runtime bound asymptotically growing, up to constants, not more than f.

This definition relaxes $\mathbf{TIME}^{\mathtt{L}}(f)$ with given worst case time bound f in the spirit of *asymptotic* complexity.

Little-o

should be familiar from module Program Analysis

Definition **(Little-o).** Let f and g be functions of type $\mathbb{N} \to \mathbb{N}$. Then $o(g)$ are those functions f that eventually grow much more slowly than g. Formally we can define this as follows:

$f \in o(g)$ iff for all $0 < \varepsilon \in \mathbb{R}$ there exists an $N \in \mathbb{N}$ s.t. $\varepsilon \times g(n) \geq f(n)$ for all $n \geq N$

The above definition is equivalent to

$$f \in o(g) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

Hierarchy Theorem

Arbitrary asymptotic growth in time

Theorem (Asymptotic Hierarchy Theorem $\mathtt{WH}^1\mathtt{LE}$). *If functions f and g are* time constructible, *$f(n) \geq n$, $g(n) \geq n$ and $g \in o(f)$ then it holds that:*

$$\boldsymbol{TIME}^{\mathtt{WH}^1\mathtt{LE}}(\mathcal{O}(f)) \setminus \boldsymbol{TIME}^{\mathtt{WH}^1\mathtt{LE}}(\mathcal{O}(g)) \neq \emptyset$$

If f grows asymptotically faster than g (under given assumptions on f and g) then we can decide more problems with $\mathtt{WH}^1\mathtt{LE}$ programs in time O(f) than O(g).

Proof similar to Linear Time Hierarchy Theorem

More Theorems

- Similar Hierarchy Theorems hold for `SRAM` and `TM`
- Other fascinating results (Gap-theorem, Blum's speedup theorem, Levin's optimality theorem). Not enough time here, but recommended to interested reader.

Hierarchy

Runtime Complexity Hierarchy – for classes of `WH`^{1}`LE`-programs

proper inclusion

…	…	
Time($d*2^n$)	exponentially slow	
Time($d*n^k$ logn)	…	
Time($d*n^3$)	cubic	
Time($c*n^2$)	quadratic	
Time(c*n*log n)		
Time(a*249*n)	larger constant	
Time(a*n)	linear: fast	
Time(a)	constant: super fast	

"Runtime Complexity" Hierarchy – for Animals

END

Next time: important problems and their complexity classes