Speedup phenomena in subrecursive settings

Andrea Asperti

DISI, University of Bologna
Mura Anteo Zamboni 7, 40127, Bologna, ITALY
Email: asperti@cs.unibo.it

Scientific meeting in honor of Pierre-Louis Curien

September 9-11, 2013, Venice, Italy

Blum's abstract framework

Abstract complexity measure [Blum [3]]

A pair $\langle \varphi, \Phi \rangle$ is an *abstract complexity measure* if φ is a principal effective enumeration of partial recursive functions and Φ satisfies the following axioms:

(a) $\varphi_i(\vec{n}) \downarrow \leftrightarrow \Phi_i(\vec{n}) \downarrow$
(b) the predicate $\Phi_i(\vec{n}) = m$ is decidable

Not a real axiomatization.
Often used in conjunction with Church's Thesis.

Blum's speedup Theorem

Let $\langle \varphi, \Phi \rangle$ be a complexity measure.

Speedup theorem [Blum [3]]

For any speedup function r there exists a computable function f such that for any algorithm i computing f we can find a different algorithm j such that $r(\Phi_j(x)) \leq \Phi_i$ a.e.

Corollary: a computable function does not have, in general, a inherent complexity.

Contribution

All proofs of the speedup theorem make an essential use of Kleene's fixed point theorem to close a suitable diagonal argument.

As a consequence, very little is known about its validity in subrecursive settings, where there is no universal machine, and no fixpoints.

In this talk we discuss an alternative proof of the speedup theorem that allows us to spare the invocation of the fix point theorem and sheds more light on the actual complexity of the function f.

The work is part of a long term program of formal revisitation of the foundations of complexity theory, via a reverse methodological approach.

A complex proof

> *"The proof of this theorem is probably the most difficult in this book"*
>
> N.J.Cutland. Computability, p.219 [5]

The proof is traditionally splitted in two parts, proving first a pseudo-speedup theorem, where we only expect $\varphi_j = f$ a.e.

The speedup theorem is then a simple corollary.

Outline of the proof

Let φ_i be an enumeration of computable functions.

Let h be a binary computable function.
We define a family $g_i^h(x)$ of functions such that

- $g_i^h(x) = g_0^h(x)$ almost everywhere
- if $g_0^h(x) = \varphi_i$, then for no $x > i$, $\varphi_i(x) \downarrow h(i+1, x)$,

Given r, we prove that there exists h^r such that the complexity of computing $g_i^{r \circ h^r}(x)$ is less than $h^r(i, x)$.

If $f = g_0^{r \circ h^r} = \varphi_i$, the complexity of $\varphi_i(x)$ is definitely larger than $r(h^r(i+1, x))$, but $g_{i+1}^{r \circ h^r}(x)$ computes an almost equal function with complexity $h^r(i+1, x)$.

The definition of g_i^h

Try to define a function different from any function i that for some input terminates with complexity $h(i+1,y)$

$$w^h(i,x) = \mu_{\{y \in [i+1,x]\}}(\varphi_i(y) \downarrow h(i+1,y))$$

If $w^h(i,x) = x$ we say that i is cancelled at stage x:

$$C_u^h(x) = \{i \in [u,x[\mid w^h(i,x) = x\}$$

$$g_u^h(x) = 1 + max_{i \in C_u^h(x)} \varphi_i(x)$$

Properties of g_i^h

- if $x \leq u$, $g_u^h(x) = 1$
- for any u there exists n_u such that for any $x > n_u$, no $i < u$ belongs to $C_0^h(x)$.
  Hence, for $x > n_u$, $C_0^h(x) = C_u^h(x)$ and

$$g_0^h(x) = g_u^h(x)$$

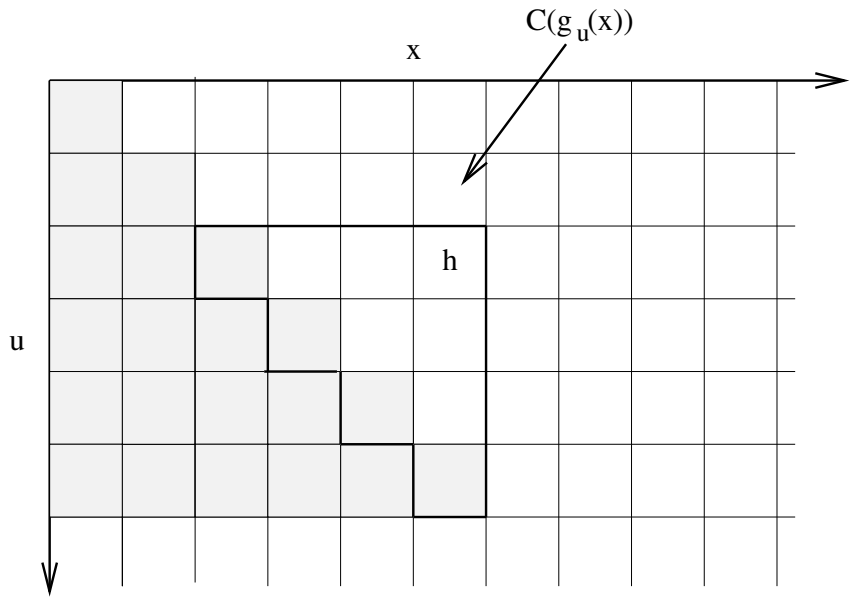
- if $\varphi_i = g_0^h$, then for all $x > i$,

$$\varphi_i(x) \not\downarrow h(i+1, x)$$

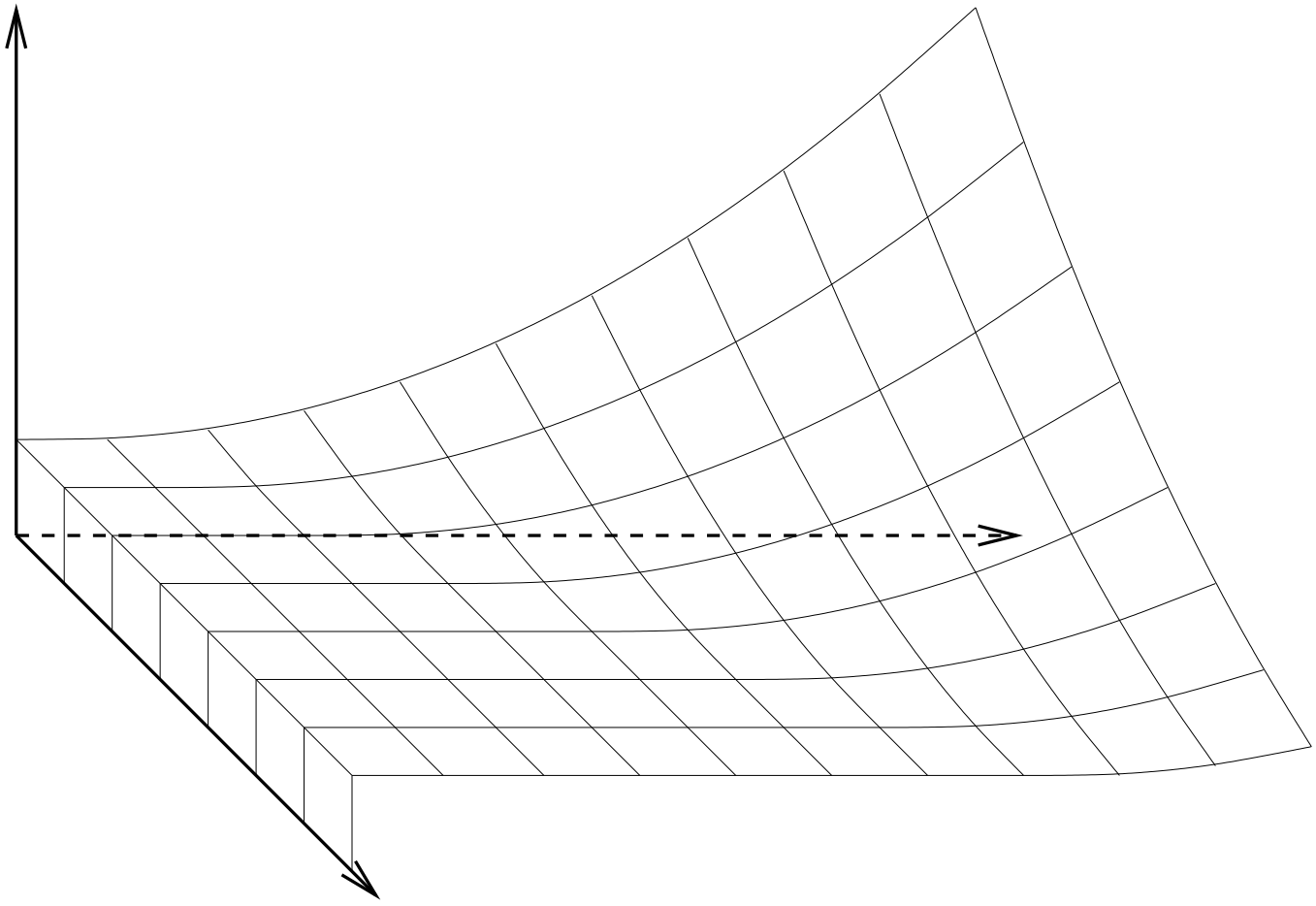
since otherwise i would be cancelled for the first such $x > i$,and by definition $g_0^h(x) > \varphi_i(x)$

Complexity of g_i^h

$$w^h(i,x) = \mu_{\{y \in [i+1,x]\}}(\varphi_i(y) \downarrow h(i+1,y))$$
$$C_u^h(x) = \{i \in [u,x[\mid w^h(i,x) = x\}$$
$$g_u^h(x) = 1 + max_{i \in C_u^h(x)} \varphi_i(x)$$



Complexity of g_i^h



An upper bound

$$h^r(i,x) = \begin{cases} 1 & \text{if } x \leq i \\ (x-i)^2 \cdot r(h^r(i+1,x)) & \text{otherwise} \end{cases}$$

Clearly, the complexity of $g_i^{r \circ h^r}(x)$ is less than $h^r(i,x)$.

This completes the proof.

Discussion

Our proof is a revisitation of Young's version of the proof [6].
In Young's proof, the function g is directly defined in terms of its own complexity:

Young [6]

We will also assume that it is legitimate to define a function recursively, not just from its earlier values, but also from its earlier run times. Intuitively, this amounts to assuming that if we used a program to calculate the value of a function at an early argument, we can know the resources used in the computation even if we do not explicitly know the entire program used for computing the function. Formally of course, one must use the recursion theorem or some other means to validate such an argument.

A proof based on the recursion theorem is given in Cutland [5].

Main remarks

- Using an upper bound h to the complexity of g is enough
- We can abstract the definition of g w.r.t. this function h, and instantiate later h according to the complexity of g.

Advantages:

- No need for the fixed point theorem
- Termination of g is not an issue

Drawbacks (?):

- Need to analize the complexity of g

Open issue

For any r, the speedup fucntion f has a complexity that is hyper-exponential. What can we say about speedup phenomena in

"feasable" complexity classes? (e.g. P)

Reverse Complexity

This work is part of a huge program of *formal* revisitation of the foundations of complexity theory via a reverse investigation of its proofs (reverse complexity [2, 1]).

Key ingredients that seems to emerge:

- Complexity of bounded arithmetics
- Complexity of bounded interpretation

Bibliography

Andrea Asperti.

The intensonal content of Rice's Theorem.
Proc. of of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM SIGPLAN Notices - POPL '08, V.43, n.1, pp. 113-119. 2008.

Andrea Asperti.

Reverse Complexity.
Submitted for publication.

Andrea Asperti.

A formal proof of Borodin-Trakhtenbrot's Gap Theorem.
Proc. of CPP'13, Melbourne, Australia, 11-13 December 2013, to appear.

Manuel Blum.

A machine-independent theory of the complexity of recursive functions.
J. ACM, 14(2):322–336, 1967.

Manuel Blum.

On Effective Procedures for Speeding Up Algorithms.
J. ACM, 18(2):290–305, 1971.

Nigel J. Cutland.

Computability: An Introduction to Recursive Function Theory.
Cambridge University Press, 1980.

Paul R. Young.

Easy constructions in complexity theory: gap and speed-up theorems.
Proceedings of A.M.S., 37(2):555–563, 1973.