

A formal proof of Borodin-Trakhtenbrot's Gap Theorem

Andrea Asperti

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

Certified Programs and Proofs

December 11-13, 2013, Melbourne, Australia

A long term program

A formalization of the well known Gap Theorem of Complexity Theory, asserting the existence of arbitrarily large gaps between complexity classes.

A case study in a much wider research program, aiming to a

synthetic theory of complexity

through a

reverse investigation

of its main results.

An analogy with HoTT:

The HoTT Book, pag.59

An important difference between homotopy type theory and classical homotopy theory is that homotopy type theory provides a synthetic description of spaces, in the following sense. Synthetic geometry is geometry in the style of Euclid: one starts from some basic notions (points and lines), constructions (a line connecting any two points), and axioms (all right angles are equal), and deduces consequences logically. This is in contrast with analytic geometry, where notions such as points and lines are represented concretely using cartesian coordinates in R^n - lines are sets of points - and the basic constructions and axioms are derived from this representation. While classical homotopy theory is analytic (spaces and paths are made of points), homotopy type theory is synthetic: points, paths, and paths between paths are basic, indivisible, primitive notions.

Synthetic Complexity

Classical Complexity Theory is **analytical**.

One starts with a specific computational model (typically, Turing Machines) and concrete resources (tapes, registers, discrete transitions) and derives the basic notions and results from the given representation.

This is immediately followed by the claim that (up to some polynomial factor), the specific model *does not matter*.

But, **WHAT MATTERS**, then? Answering the question would bring us to a synthetic approach to Complexity Theory.

Not a matter of expressiveness

Passing from an analytical approach to a synthetic one, one does not expect to gain expressiveness (on the contrary, one could loose some).

- ▶ having a more symbolic (logical) approach (more fun!)
- ▶ open a new perspective on the field
- ▶ possibly suggest unconventional models

A reverse approach

Try to avoid any prejudice and conceptual bias.

Adopt a **reverse methodological** approach, reconstructing from proofs the basic notions and assumptions underlying the major results of this field.

Methodology is similar to **reverse mathematics** (Friedman [10], Simpson [12]), but aims are different.

The final goal of a synthetic approach to complexity naturally entails the use of formal **proof assistants**.

A minimal linguistic framework

Where do we start?

We need a minimal linguistic framework to talk about the complexity of a not better specified computational agent.

Abstract complexity measure [Blum [7]]

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 relation

$$\begin{cases} \varphi_i(n) = a \\ \Phi_i(n) = m \end{cases}$$

is equivalently expressed by Kleene's T predicate

$$T(i, n, a, m)$$

expressing that the computational agent i on input n returns a with computational resources m .

Basic framework

We shall work with a functional version of T , namely a ternary function U returning an optional result.

The intuition is that

$$U(i, n, m) = \begin{cases} \text{Some } (a) & \text{if } T(i, n, a, m) \\ \text{None} & \text{otherwise} \end{cases}$$

But this is an intuition. The only assumption on U is monotonicity.

axiom $U: \text{nat} \rightarrow \text{nat} \rightarrow \text{nat} \rightarrow \text{option nat}.$

axiom $\text{monotonic_}U: \forall i, n, m, y. n \leq m \rightarrow$

$U \ i \times n = \text{Some } y \rightarrow U \ i \times m = \text{Some } y.$

In other words, we work with a “bounded” applicative algebra.

More axiom “on demand”

For instance, the basic framework suffices for the proof of the gap theorem.

Notation:

$$\varphi_i(n) \downarrow t \Leftrightarrow \exists a. U(i, n, t) = \text{Some } a$$

$$\varphi_i(n) \not\downarrow t \Leftrightarrow U(i, n, t) = \text{None}$$

The gap theorem

Gap theorem [Borodin [9]]

Let g be a nondecreasing recursive function such that $\forall x. x \leq g(x)$. Then there exists a nondecreasing recursive function t such that, for any i and for any sufficiently large n ,

$$\varphi_i(n) \downarrow t(n) \quad \text{or} \quad \varphi_i(n) \not\downarrow g \circ t(n)$$

“no matter how much better one computer may seem compared to the other, there will be a t such that the set of functions computable in time t is the same for both computers” [9].

Define t as follows:

- ▶ $t(0) = 1$,
- ▶ $t(n) = \mu k \geq t(n-1) \{ \forall i < n. [\varphi_i(n) \downarrow k \text{ or } \varphi_i(n) \not\downarrow g(k)] \}$

Then:

1. for any n , k exists, since for all $i < n$ if $\varphi_i(n) \uparrow$ then $\forall k. \varphi_i(n) \not\downarrow g(k)$, and if $\varphi_i(n) \downarrow$ then $\exists k. \varphi_i(n) \downarrow k$.
2. k can be found recursively, since $\varphi_i(n) \downarrow k$ and $\varphi_i(n) \not\downarrow g(k)$ are decidable predicates.
3. t satisfies the theorem, since $n > i$ implies that either $\varphi_i(n) \downarrow t(n)$ or $\varphi_i(n) > g \circ t(n)$.

QED.

The fact that φ_i is a principal enumeration of all partial recursive functions is not used.

Define t as follows:

- ▶ $t(0) = 1$,
- ▶ $t(n) = \mu k \geq t(n-1) \{ \forall i < n. [\varphi_i(n) \downarrow k \text{ or } \varphi_i(n) \not\downarrow g(k)] \}$

Then:

1. for any n , k exists, since for all $i < n$ if $\varphi_i(n) \uparrow$ then $\forall k. \varphi_i(n) \not\downarrow g(k)$, and if $\varphi_i(n) \downarrow$ then $\exists k. \varphi_i(n) \downarrow k$.
2. k can be found recursively, since $\varphi_i(n) \downarrow k$ and $\varphi_i(n) \not\downarrow g(k)$ are decidable predicates.
3. t satisfies the theorem, since $n > i$ implies that either $\varphi_i(n) \downarrow t(n)$ or $\varphi_i(n) > g \circ t(n)$.

QED.

The fact that φ_i is a principal enumeration of all partial recursive functions is not used.

The formal proof in Matita is not sensibly more complex.

The main trouble comes from minimization, for which we need to provide an upper bound.

To this aim, let us consider the intervals

$$[g^i(b), g^{i+1}(b)[\text{ for } 0 \leq i \leq n$$

and all functions such that

$$\varphi_j(n) \leq g^{n+1}(b) \text{ for } j < n$$

We have at most n functions to distribute over $n + 1$ intervals, so at least one interval must remain empty.

So, $g^n(t(n))$ is an upper bound for the minimization.

the gap function in matita

```
let rec gap g n on n :=  
  match n with  
    [ 0  $\Rightarrow$  1  
    | S m  $\Rightarrow$  let b := gap g m in  $\mu_{-\{k \in [b, g^n b]\}}$  (gapb n n g k)  
    ].
```

where

```
definition gapb :=  $\lambda n, x, g, r.$   
   $\backslash \text{big}[andb, true]_{-\{i < n\}}$  ((term b i x r)  $\vee \neg$  (term b i x (g r))).
```

In general, in order to prove other theorems of Complexity Theory, you need to make additional (closure) assumption.

Studying such a minimal logical framework is precisely the purpose of Reverse Complexity.

Some definitions

code for a function:

definition $\text{code_for} := \lambda f, i.$

$$\forall x. \exists n. \forall m. n \leq m \rightarrow \bigcup i \times m = f \ x.$$

i is running in $O(s)$

definition $C := \lambda s, i.$

$$\exists c. \exists a. \forall x. a \leq |x| \rightarrow \exists y. \bigcup i \times (c * (s(|x|))) = \text{Some } y.$$

$f \in O(s)$

definition $CF := \lambda s, f. \exists i. \text{code_for } f \ i \wedge C \ s \ i.$

A bound interpreter

Internalization of U : a bound interpreter!

axiom sU: nat \rightarrow nat \rightarrow nat \rightarrow nat.

axiom CFU3: CF3 sU U.

This axiom and little more is enough to prove the hierarchy theorem, in a completely parametric way ([1]).

Conclusion

We formally revisited the gap theorem in the framework of the **Reverse Complexity** program, aiming to a **synthetic description** of Complexity Theory.

In different works, we applied the reverse methodology to the Hierarchy Theorems [1] and the Speedup Theorem [2].

The important point that seems to emerge is the fact that complexity theory can be perfectly investigated in a **subrecursive setting**: no need for universal machines, and general recursion.

The role played by the universal machine in computability is replaced by **bound interpretation** in complexity theory.

All Complexity Theory a-head.

Next milestone: **Savitch Theorem** [11].

Synthetic Complexity is a far away dream: avoid to address the problem directly (too many failures).

Bibliography



Andrea Asperti.

Reverse Complexity.

Submitted for publication.



Andrea Asperti.

Speedup phenomena in subrecursive settings.

Talk at *Curien's Festschrift*, Venice 2013.



Andrea Asperti.

The intensional content of Rice's theorem.

In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, January 7-12, 2008, San Francisco, California, USA, pages 113–119. ACM, 2008.



Andrea Asperti and Agata Ciabattoni.

Effective applicative structures.

In *Category Theory and Computer Science, 6th International Conference, CTCS '95, Cambridge, UK, Proceedings*, volume 953 of *Lecture Notes in Computer Science*, pages 81–95. Springer, 1995.



Andrea Asperti and Wilmer Ricciotti.

Formalizing turing machines.

In *Logic, Language, Information and Computation - 19th International Workshop, WoLLIC 2012, Buenos Aires, Argentina*, volume 7456 of *Lecture Notes in Computer Science*, pages 1–25, 2012.



Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi.

The Matita interactive theorem prover.

In *Proceedings of the 23rd International Conference on Automated Deduction (CADE-2011)*, Wroclaw, Poland, volume 6803 of *LNCS*, 2011.

Bibliography



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.



Allan Borodin.

Computational complexity and the existence of complexity gaps.
J. ACM, 19(1):158–174, 1972.



Harvey Friedman and Stephen G. Simpson.

Issues and problems in reverse mathematics.
Contemporary Mathematics, 257:127–143, 2000.



Walter J. Savitch.

Relationships between nondeterministic and deterministic tape complexities.
J. Comput. Syst. Sci., 4(2):177–192, 1970.



Stephen G. Simpson.

Subsystems of second order arithmetic.
Cambridge University Press, 2009.



Boris Trakhtenbrot.

Turing computations with logarithmic delay.
Algebra and Logic, 3(4):33–48, 1964.