On Time and Space in Higher Order Boolean Circuits

Damiano Mazza
CNRS, UMR 7030, LIPN, Université Paris 13, Sorbonne Paris Cité
Damiano.Mazza@lipn.univ-paris13.fr

It is well known that the usual complexity measures of time and space (defined using Turing machines) are strongly related to *size* and *depth* of Boolean circuits. The size of a circuit is the number of its gates; the depth is the longest path in the circuit, seen as a directed acyclic graph (from inputs to outputs). Given a family $(C_n)_{n\in\mathbb{N}}$ of Boolean circuits (on the standard fan-in 2 basis $\{\neg, \wedge, \vee\}$), such that C_n has n inputs and 1 output, we say that it *decides* a language $L \subseteq \{0,1\}^*$ if, for all $x \in \{0,1\}^n$, $x \in L$ iff $C_{|x|}(x) = 1$. We then denote by $\mathsf{SIZE}(f)$ (resp. $\mathsf{DEPTH}(f)$) the class of languages decided by families of circuits $(C_n)_{n\in\mathbb{N}}$ such that the size (resp. depth) of C_n is bounded by $f(n)$.

With the above definitions, we have (see [14, 4]):

Theorem 1 (Fisher and Pippenger, Borodin)**.**
1. $\mathsf{TIME}(f) \subseteq \mathsf{SIZE}(O(f \log f))$;
2. $\mathsf{SIZE}(f) \subseteq \mathsf{TIME}(O(f))$;
3. $\mathsf{NSPACE}(f) \subseteq \mathsf{DEPTH}(O(f^2))$;
4. $\mathsf{DEPTH}(f) \subseteq \mathsf{SPACE}(O(f))$.

In (2) and (4), some notion of *uniformity* on circuit families must be assumed of course. The details are irrelevant for our purposes; it suffices to say that a family is uniform if its circuits may be generated by a program of small complexity (say, logspace).

Our purpose is to investigate what happens in the higher order world, with the ultimate goal of providing general, abstract cost models for higher-order programs, the preferred setting for the theory of programming languages. In particular:

(a) $\mathsf{TIME}(\cdot)$ and $\mathsf{SPACE}(\cdot)$ must be replaced by notions of time and space complexity for λ-terms;

(b) a suitable notion of higher-order Boolean circuit must be introduced, and the corresponding notions of size and depth defined for it.

For what concerns time, one may resort to the number of head-reduction steps, which was proved by Accattoli and Dal Lago [1] to be an invariant cost

measure, *i.e.*, if we denote by $\lambda\mathsf{TIME}(f)$ the class of languages decided (using *e.g.* Church binary strings and Church Booleans) by λ-terms in at most $f(n)$ steps of head reduction for inputs of size n, then:

Theorem 2 (Accattoli and Dal Lago)**.**
1. $\mathsf{TIME}(f) \subseteq \lambda\mathsf{TIME}(O(f))$*;*
2. $\lambda\mathsf{TIME}(f) \subseteq \mathsf{TIME}(O(f^k))$ *for some constant* k*.*

The above tells us we have a good notion of time complexity for λ-terms. When it comes to space complexity, however, much less is known. The ideal would be to find an abstract measure, something that is as much as possible machine-independent. Resorting to notions which are purely internal to the theory of the λ-calculus should guarantee a good level of abstraction (counting head-reductions is a perfect example). The work of Spoonhower et al. [16] is a very interesting proposal, although it still relies on a low-level description of λ-terms, which is not entirely satisfactory from our point of view.

Let us temporarily ignore the issue and let us address directly point (b): what is a higher-order Boolean circuit? We believe that this should be taken to coincide with *linear λ-terms*. There are several reasons for this choice: like Boolean circuits, linear λ-terms may only compute finite functions; like Boolean circuits, the (sequential) runtime of a linear λ-term coincides with its size; finally, Boolean circuits may be seen as morphisms of a free symmetric monoidal category (in fact, a PROP), while (normal) linear λ-terms are morphisms in a free *closed* symmetric monoidal category. In fact, for technical reasons it is better to consider *affine* λ-terms (erasing is permitted, not duplication), which are a minor variant still adhering to the above picture.

The relationship between affine λ-terms and general λ-terms may be refined by formalizing the intuition (already present in [7]) that the affine λ-calculus is "dense" in the full λ-calculus. This is the object of [8], which may be informally summarized as follows:

- there is a notion of *affine approximation* $t \sqsubset M$ of a λ-term by affine terms t, such that M may be seen as the limit of its affine approximations;
- reduction is continuous: if $M \to^* N$, then for all $u \sqsubset N$ there exists $t \sqsubset M$ such that $t \to^* u$.

Although this looks like a good start, we immediately run into trouble: linear-step computations in the λ-calculus may require exponentially large affine approximations. In other words, $\lambda\mathsf{TIME}(f)$ is *not* included in $\lambda\mathsf{SIZE}(O(f^k))$ for any constant k, where we consider the size of affine terms to be the usual one. A possible solution, which we started to explore, is to shift to the *parsimonious* λ-calculus, introduced by the author [9, 10, 13]. Another solution is to consider *linear explicit substitutions*, as in [1, 2]. In both cases, we obtain a result corresponding to points 1 and 2 of Theorem 1. For instance, if $\mathrm{p}\lambda\mathsf{TIME}(f)$ is the analogue of $\lambda\mathsf{TIME}(f)$ for the parsimonious λ-calculus, we have

Theorem 3 ([12])**.**
1. $\mathrm{p}\lambda\mathsf{TIME}(f) \subseteq \lambda\mathsf{SIZE}(O(f^k))$ *for a constant* k*;*
2. $\lambda\mathsf{SIZE}(f) \subseteq \mathrm{p}\lambda\mathsf{TIME}(O(f^2))$.

In ongoing work, we are seeking to complete the picture by finding the analogue of points 3 and 4 of Theorem 1, taking the parsimonious λ-calculus as the underlying language. Our approach is based on an interesting correspondence between intersection types and affine approximations, described recently by the author [11]: a λ-term M has an intersection type iff there exists a simply-typable t such that $t \sqsubset M$. More precisely, one may build a non-idempotent, non-commutative intersection type system (equivalent to the standard one for what concerns typability) such that its types are isomorphic to the simple types for affine terms and its derivations are isomorphic to simply-typed affine terms and, furthermore, an affine term corresponding to a type derivation for M is actually an approximation of M. So, we may write $\Gamma \vdash t \sqsubset M : A$ to say that M is typable (in intersection types) of type A with a derivation isomorphic to t (whose simple type is also A). Thanks to this correspondence, which holds also for the parsimonious λ-calculus, and the idea, originally due to Schöpp [15], of using the geometry of interaction to perform space-efficient computation on λ-terms, we obtain

Theorem 4. *Let M be a parsimonious λ-term s.t. there exists a directed (w.r.t. the approximation order) sequence $(t_n)_{n\in\mathbb{N}}$ s.t., for all $n \in \mathbb{N}$,*

$$\vdash t_n \sqsubset M : \mathsf{Str}_n \multimap \mathsf{Bool},$$

where Str_n are (bigger and bigger) instances of the usual type of binary strings, and let d_n be the depth of Str_n (as a syntactic tree). Then, M decides a language in $\mathsf{SPACE}(O(d_n \log |t_n|))$.

The above result suggests that, if we take higher-order Boolean circuits to be affine terms of type $\mathsf{Str}[] \multimap \mathsf{Bool}$ (with $\mathsf{Str}[]$ some instance of the type of binary strings), then we should take as "depth" the depth of the formula $\mathsf{Str}[]$. This is in accord with what Terui did for Boolean proof nets [17] and agrees, up to a multiplicative instead of additive factor, with what Borodin [4] found for Turing machines and (first-order) Boolean circuits: a uniform family of circuits of depth d_n and size s_n decides a language in $\mathsf{SPACE}(d_n + \log s_n)$ (this is how point 4 of Theorem 1 is proved).

Observe that Theorem 4 may allow to bypass the definition of a space measure for parsimonious λ-terms: the existence of a family of suitable typings would constitute *per se* an abstract, machine-independent measure. The relationship between intersection types and time complexity of λ-terms was well known [6, 3, 18, 5]; it is interesting to see that it may work for space, too. In the talk, we will discuss and explain more in detail the above results and the perspectives they offer.

Acknowledgments. Partially supported by Coquas (ANR-12-JS02-006-01) and Elica (ANR-14-CE25-0005).

References

[1] Beniamino Accattoli and Ugo Dal Lago. On the invariance of the unitary cost model for head reduction. In *Proceedings of RTA*, pages 22–37, 2012.

[2] Beniamino Accattoli and Ugo Dal Lago. (Leftmost-outermost) Beta reduction is invariant, indeed. *Logical Methods in Computer Science*, 12(1), 2016.

[3] Alexis Bernadet and Stéphane Lengrand. Complexity of strongly normalising lambda-terms via non-idempotent intersection types. In *Proceedings of FOSSACS*, pages 88–107, 2011.

[4] Allan Borodin. On relating time and space to size and depth. *SIAM J. Comput.*, 6(4):733–744, 1977.

[5] Erika De Benedetti and Simona Ronchi Della Rocca. Bounding normalization time through intersection types. In *Proceedings of ITRS*, pages 48–57, 2013.

[6] Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *CoRR*, abs/0905.4251, 2009.

[7] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50(1):1–102, 1987.

[8] Damiano Mazza. An infinitary affine lambda-calculus isomorphic to the full lambda-calculus. In *Proceedings of LICS*, pages 471–480, 2012.

[9] Damiano Mazza. Non-uniform polytime computation in the infinitary affine lambda-calculus. In *Proceedings of ICALP, Part II*, pages 305–317, 2014.

[10] Damiano Mazza. Simple parsimonious types and logarithmic space. In *Proceedings of CSL*, pages 24–40, 2015.

[11] Damiano Mazza. Affine approximations and intersection types. Accepted for presentation at ITRS 2016. Available on the author's web page, 2016.

[12] Damiano Mazza. Church meets cook and levin. To appear in *Proceedings of LICS*. Available on the author's web page, 2016.

[13] Damiano Mazza and Kazushige Terui. Parsimonious types and non-uniform computation. In *Proceedings of ICALP, Part II*, pages 350–361, 2015.

[14] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[15] Ulrich Schöpp. Space-efficient computation by interaction. In *Proceedings of CSL*, pages 606–621, 2006.

[16] Daniel Spoonhower, Guy E. Blelloch, Robert Harper, and Phillip B. Gibbons. Space profiling for parallel functional programs. *J. Funct. Program.*, 20(5-6):417–461, 2008.

[17] Kazushige Terui. Proof nets and boolean circuits. In *Proceedings of LICS*, pages 182–191, 2004.

[18] Kazushige Terui. Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In *Proceedings of RTA*, pages 323–338, 2012.