Computability over locally finite structures from algebra

Siddharth Bhaskar*

April 7, 2018

Abstract

Working with a particular notion of computability over general, first-order structures, we argue that computability classes over certain locally finite structures should capture Turing machine complexity classes. We exhibit this phenomenon for some locally finite structures from algebra.

The central thesis of implicit complexity theory is that Turing machine complexity classes can be profitably identified with computability by programs that satisfy some syntactic constraint. In this talk, we show that we can also identify Turing machine complexity classes with computability by programs that are syntactically general, but which operate over "atypical" structures, i.e., structures unlike the natural numbers or binary strings. In other words, we connect complexity theory to *recursion over abstract structures*.

Any structure $\mathbf{A}$ has an associated family $\mathrm{rec}(\mathbf{A})$ of partial functions and relations recursive over $\mathbf{A}$. As our model of computation we use *(McCarthy) recursive programs* [6, 7]. These are minimal, functional programs that allow calls to the primitive operations (i.e., calls to Φ), branching, and recursive calls. We also restrict them to be *first-order*, forbidding higher-type objects like lists or sets over our base data type. (For a precise definition, see the Appendix on page 3.)

An important subclass of recursive programs are those which are *tail recursive*. These exactly capture iterative algorithms; i.e., those expressible with recursion no more complicated than while loops. If $\mathrm{tail}(\mathbf{A})$ is the family of tail recursive partial functions and relations, then $\mathrm{tail}(\mathbf{A}) \subseteq \mathrm{rec}(\mathbf{A})$ in general, but the converse is not necessary [5, 8].

In the case of $\mathbf{N} = (\mathbb{N}, 0, 1, S, Pd, =)$ where S is the successor and Pd the predecessor function, $\mathrm{rec}(\mathbf{N})$ and $\mathrm{tail}(\mathbf{N})$ coincide with the classically recursive functions. For general structures $\mathbf{A}$, there are several conditions, equivalent under weak hypotheses [3], that assert that $\mathrm{rec}(\mathbf{A})$ is "like $\mathrm{rec}(\mathbf{N})$:"

- $\mathbf{A}$ interprets $\mathbf{N}$ (in the appropriate sense),

*Department of Computer Science, Haverford College, PA, USA

- $\mathrm{rec}(\mathbf{A})$ has pairing and unpairing functions, and
- $\mathrm{rec}(\mathbf{A})$ admits a Gödel numbering with universal and $s-m-n$ functions.

As a fourth condition we may add $\mathrm{tail}(\mathbf{A}) = \mathrm{rec}(\mathbf{A})$, which is implied by these conditions, but cannot imply them without resolving some hard questions in computational complexity.

The family of recursive functions of such structures clearly more closely resembles classical computability rather than complexity classes. However, over other structures with "less computational power" it turns out that the family of recursive functions captures complexity classes. We focus our attention on *locally finite structures*, structures all of whose finitely generated substructures are finite. The fact that recursive programs, which lack quantifiers, are "local" in the sense that their computation on particular input takes place entirely in the substructure generated by that input, means that $\mathrm{rec}(\mathbf{A})$ for locally finite $\mathbf{A}$ cannot be "classical" in the first three senses above.

Among locally finite structures, there are some whose family of recursive functions is so weak that they are not obviously comparable to complexity classes. We ignore these, though it's a compelling open question to identify a clean model-theoretic property guaranteeing or forbidding such comparability. Among positive examples, our starting point is *cons-free computation* of Neil Jones [4], who looked at the structure

$$(2^{<\omega}; \epsilon, head, tail, =)$$

obtained by taking the usual structure of binary strings[1] and deleting the constructors "add 0" and "add 1." Here the unary recursive and tail recursive relations *are* exactly polynomial time and logarithmic space decidable languages respectively.

An almost identical analysis holds for the structure $(n^{<\omega}; \epsilon, head, tail, =)$ for $n < \omega$; when $n = 1$ we (basically) get the structure of "predecessor arithmetic" $\mathbf{N}_{Pd} = (\mathbb{N}; 0, 1, Pd, =)$ whose recursive and tail recursive relations correspond to exponential time and linear space under the standard bijection $\mathbb{N} \simeq 2^{<\omega}$.

Our own contribution is connecting computability over some locally finite algebraic structures to cons-free computation, and hence to complexity classes.[2] We consider certain infinite locally finite abelian groups and fields, for example, the field

$$\mathbf{F}_p = (\bar{\mathbb{F}}_p; 0, 1, +, -, \times, \div, =),$$

where $\bar{\mathbb{F}}_p$ is the algebraic closure of the finite field of size p. Here we can prove that:

Theorem. *$X \subseteq \bar{\mathbb{F}}_p$ is recursive over $\mathbf{F}$ iff $\rho[X] \subseteq 2^{<\omega}$ is decidable in exponential time. Similarly, X is tail recursive iff $\rho[X]$ is decidable in linear space.*

[1] Here *head* and *tail* refer to the operations that return the first character of a list and all but the first character respectively.

[2] This refers to work published in our thesis [1] and a subsequent paper [2].

(For an element $x \in \bar{\mathbb{F}}_p$, $\rho(x)$ is obtained by taking the minimal polynomial of x over $\mathbb{F}_p[t]$ and "converting it to binary.")

Using this, we can prove several recursion-theoretic consequences, such as:

Theorem. *rec*($\mathbf{F}$) *does not admit a Gödel numbering with a recursive universal function.*

Theorem. *rec*($\mathbf{F}$)=*tail*($\mathbf{F}$) *iff linear space is exponential time.*

Uniform families of interpretations The standard notion of interpretation from model theory is too "rigid" to identify, e.g., rec($\mathbf{F}$) with rec($\mathbf{N}_{Pd}$). Part of our technical contribution is in identifying an appropriate notion of interpretation, for the time being a *loose interpretation*, that allows us to connect computability over different structures. This is roughly a family of interpretations over substructures of the two structures that "cohere" in the right way.

Goals and future work One immediate goal is to expand our "library" of locally finite structures which loosely interpret cons-free computation, and hence complexity classes. More conjecturally, we wonder whether there is a set of recursion-theoretic or model-theoretic conditions that assert that rec($\mathbf{A}$) is "similar" to rec($\mathbf{N}_{Pd}$) or a similar cons-free structure, thus showing that such properties are robust. We have some ideas for these which we will review at the end of the talk.

Appendix: Recursive and tail recursive programs

Given a signature Φ, a Φ-*recursive program* is a set of simultaneous recursive equations of the form

$$\{\mathtt{p}_i(\vec{\mathtt{x}}_i) = E_i(\vec{\mathtt{x}}_i, \vec{\mathtt{p}})\}_{i<k}$$

where $\vec{\mathtt{p}} = (\mathtt{p}_0, \ldots, \mathtt{p}_{k-1})$, and the functionals E_i are given by the grammar

$$M := \mathtt{x}_n \mid \phi(M_1, \ldots, M_n) \mid \mathtt{p}_n(M_1, \ldots, M_n) \mid \text{if } M_0 \text{ then } M_1 \text{ else } M_2$$

where $\mathtt{x}_n$ and $\mathtt{p}_n$ come from some countable stock of variables and function-valued variables respectively. Semantics may be defined operationally or denotationally, with the function denoted by the "head" $\mathtt{p}_0$ being the function computed by the whole program. For example,

$$\mathtt{p}_0(\mathtt{x}) = \mathtt{p}_1(1, \mathtt{x}, \mathtt{x})$$

$$\mathtt{p}_1(\mathtt{x}_1, \mathtt{x}_2, \mathtt{x}_3) = \text{if } \mathtt{x}_1 = 0 \text{ then } \mathtt{x}_3 \text{ else } \mathtt{p}_1(\mathtt{x}_1 + 1, \mathtt{x}_2, \mathtt{x}_2 \cdot \mathtt{x}_3),$$

computes the Frobenius automorphism $x \mapsto x^p$ when interpreted over a field of characteristic p, and computes the everywhere diverging function when interpreted over a field of characteristic 0.

A recursive program is *tail recursive* in case it has the form

$$\mathtt{p}_0(\vec{\mathtt{x}}_0) = \mathtt{p}_1(G(\bar{\mathtt{x}}_0))$$

$$\mathtt{p}_1(\vec{\mathtt{x}}_1) = \text{if } \tau(\vec{\mathtt{x}}_1) \text{ then } o(\vec{\mathtt{x}}_1) \text{ else } \mathtt{p}(F(\vec{\mathtt{x}}_1))$$

where G, τ, o, and F are *explicit* terms, i.e., generated by the grammar

$$M := \mathtt{x}_n \mid \phi(M_1, \ldots, M_n) \mid \text{if } M_0 \text{ then } M_1 \text{ else } M_2.$$

The Frobenius-computing program is, then, tail recursive.

References

[1] Siddharth Bhaskar. *Recursion versus Tail Recursion over Abstract Structures.* PhD thesis, UCLA, 2015.

[2] Siddharth Bhaskar. Recursion versus tail recursion over finite fields. *Journal of Logical and Algebraic Methods in Programming*, 94:68–90, January 2018.

[3] Friedman and Mansfield. Algorithmic procedures. *Transactions of the American Mathematical Soceity*, 332:297–312, 1992.

[4] Neil Jones. Logspace and ptime characterized by programming languages. *Theoretical Computer Science*, 228(1):151–174, 1999.

[5] N. A. Lynch and E. K. Blum. A difference in expressive power between flowcharts and recursion schemes. *Mathematical Systems Theory*, 12(1):205–211, 1979.

[6] John McCarthy. A basis for a mathematical theory of computation. In *Proceedings of the Western Joint Computer Conference*, pages 225–238, 1961.

[7] Y. N. Moschovakis. Abstract recursion and intrinstic complexity. Lecture Notes in Logic (to appear), 2018.

[8] M. S. Paterson and C. E. Hewitt. Comparative schematology. In *Proc. Rec. ACM Conference on Concurrent Systems and Parallel Computation*, pages 119–127, 1970.