From determinism, non-determinism and alternation to recursion schemes for P, NP and Pspace

Isabel Oitavem

CMAF, Universidade de Lisboa and
DM, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829-516 Caparica, Portugal
oitavem@fct.unl.pt

Abstract

Our goal is to approach the classes of computational complexity *P*, *NP*, and *Pspace* in a recursion-theoretic manner. Here we emphasize the connection between the structure of the recursion schemes and the underlying models of computation.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.1.1 Models of Computation, F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes

Keywords and phrases Computational complexity, Recursion schemes, P, NP, Pspace

Digital Object Identifier 10.4230/LIPIcs.CSL.2013.24

Category Invited Talk

1 Introduction

P, *NP* and *Pspace* are well-known classes of computational complexity that can be described following different approaches. Here we describe them in a machine independent manner, using recursion schemes, which turn the known inclusions $P \subseteq NP \subseteq Pspace$ obvious. This work contributes to a better understanding of the involved classes, but no separation result is foreseen.

Recursion-theoretic approaches lead to classes of functions instead of predicates (or boolean functions). Therefore, instead of *P* and *Pspace* we reach the classes *FPtime* and *FPspace*. As a class of functions corresponding to *NP* we choose $FPtime \cup NP$, and we adopt the notation *FNP*.

Our strategy is, as always in recursion-theoretic contexts, to start with a set of initial functions — which should be basic from the complexity point of view — and to close it under composition and recursion schemes. The recursion schemes can be bounded or unbounded depending on the chosen approach. In the first case we consider the Cobham characterization of *FPtime* [3], in the second case we consider the Bellantoni-Cook characterization of *FPtime* [2]. In both cases we work over $\mathbb{W}$, instead of $\mathbb{N}$, where $\mathbb{W}$ is interpreted over the set of 0-1 words. ϵ stands for the empty word, and S_0 and S_1 stand for concatenation, respectively, with 0 and 1. Therefore, as initial functions one considers ϵ, S_0, S_1, P (binary predecessor) and C (case distinction).

We look to these three classes of complexity — *FPtime*, *FNP* and *FPspace* — as resulting from three different models of computation — deterministic, non-deterministic and alternating Turing machines (as described in [1]) — and imposing the same resource constraint, polynomial time. Thus the adopted recursion schemes should somehow reflect

the "increasing" computational power of the computation model. For *FNP*, besides the calibration of the recursion schemes, we have an additional problem since one is dealing with a class which, in principle, is not closed under composition (because *NP* is, in principle, not closed under negation).

2 Bounded recursion schemes

Bounded recursion schemes are recursion schemes where the length of the outputs are, at every step, bounded. In the cases we treat here, the bound of the lengths is polynomial. The bounds are functions explicitly definable from ϵ, S_0, S_1, string concatenation and string product (corresponding to the smash function of Buss). The bound is, at each step of the recursion, imposed via truncation. We use $x_{|y}$ to denote x truncated to the length of y.

2.1 FPtime

The bounded recursion scheme for *FPtime* described below is based on Cohbam's work [C64] and reproduces the sequential structure of deterministic computations. We denote it by *bounded recursion over* $\mathbb{W}$ *(BR)*:

$$\begin{aligned} f(\epsilon, \bar{x}) &= g(\epsilon, \bar{x}) \\ f(y0, \bar{x}) &= h(y0, \bar{x}, f(y, \bar{x}))_{|t(y0,\bar{x})} \\ f(y1, \bar{x}) &= h(y1, \bar{x}, f(y, \bar{x}))_{|t(y1,\bar{x})} \end{aligned}$$

Notice that, for instance, the definition of $f(11)$ by *BR* (based on g and h and t) leads to $h(11, h(1, g(\epsilon)))$ (t is omitted), which corresponds to the sequence

$$\begin{array}{c} h \\ | \\ h \\ | \\ g \end{array}$$

2.2 FPspace

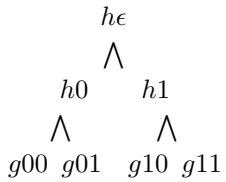
It is well-known that: a function f (over $\mathbb{W}$) is computable in polynomial space if, and only if, f is bitwise computable by an alternating Turing machine in polynomial time, and the length of the outputs of f is polynomial in the length of the inputs.

Alternating Turing machines lead to trees of computation. Therefore, the corresponding recursion scheme, instead of a sequential structure, has a tree structure. It is defined analogously to BR, but we double the recursive call and we distinguish them from each other via a pointer (denoted by p).

Bounded tree recursion over $\mathbb{W}$*(BTR)*, also called *bounded recursion with pointers*:

$$\begin{aligned} f(p, \epsilon, \bar{x}) &= g(p, \epsilon, \bar{x}) \\ f(p, y0, \bar{x}) &= h(p, y0, \bar{x}, f(p0, y, \bar{x}), f(p1, y, \bar{x}))_{|t(p,y0,\bar{x})} \\ f(p, y1, \bar{x}) &= h(p, y1, \bar{x}, f(p0, y, \bar{x}), f(p1, y, \bar{x}))_{|t(p,y1,\bar{x})} \end{aligned}$$

If $f(\epsilon, 11)$ is defined by *BTR* on its second input based on g, h and t, then (omitting, once more, the bound t) one obtains $h(\epsilon, h(0, g(00), g(01)), h(1, g(10), g(11)))$. The corresponding tree is



The mentioned input is the pointer, and it gives the address from the root of the tree to the current node. The tree structure of BTR is clear. It is also clear that if h and g do not depend on their first input (the pointer), then the tree structure collapses to a sequential one. Therefore, BTR trivially extends BR.

More about this characterization of *FPspace* can be found in [4].

2.3 FNP

Non-deterministic Turing machines can be seen, simultaneously, as an extension of the concept of deterministic Turing machines and a restriction of alternating Turing machines. Thus our goal is, also simultaneously, to extend BR and restrict BTR in an appropriated way.

We would like to do it via a single recursion scheme, however so far that was not achieved. This issue is also related with the restricted form of composition one may have in *FNP*.

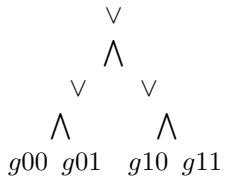
What we describe here is a recursion scheme which should be taken in addition to BR. We call it $TR[\vee]$ because it results from BTR by fixing the step function h — h is the disjunction of its last two inputs (the recursive calls). More precisely, *disjunctive tree recursion over* $\mathbb{W}$ ($TR[\vee]$) is the scheme:

$$\begin{aligned} f(p,\epsilon,\bar{x}) &= g(p,\epsilon,\bar{x}) \\ f(p,y0,\bar{x}) &= \vee(f(p0,y,\bar{x}),f(p1,y,\bar{x})) \\ f(p,y1,\bar{x}) &= \vee(f(p0,y,\bar{x}),f(p1,y,\bar{x})), \end{aligned}$$

where $\vee(u,v)$ returns 1 if at least one of its inputs ends with 1, and 0 otherwise.

Notice that there is no need of imposing bounds — a single bit is returned at every step of the recursion (with possible exception of the base level, where g is computed).

Let us look at our example once more. If $f(\epsilon,11)$ is defined by $TR[\vee]$ based on g, then one has $\vee(\epsilon,\vee(0,g(00),g(01)),\vee(1,g(10),g(11)))$, which corresponds to the tree



Therefore one gets a tree structure as before, but only the addresses of the leaves are available. All internal nodes have the same (disjunctive) label. The parallel with non-deterministic Turing machines is obvious.

Notice that if, in $TR[\vee]$, g does not depend on the pointer, then the scheme loses his tree structure. However, since the step function is fixed (it is $\vee$) this scheme does not extend BR. As mentioned above, $TR[\vee]$ is taken in addition to BR.

See [5] for more about this characterization of *FNP*.

3 Final considerations

With a simple example one is able to illustrate the structure of the recursion schemes used to describe the classes of computational complexity *FPtime*, *FNP* and *FPspace*. The connection between the structure of the recursion and the underlying model of computation is of interest and it might deserve some further thoughts. Some work is being developed concerning the levels of the polynomial hierarchy of time.

FPtime, *FNP* and *FPspace* are reached in a recursion-theoretic manner by successively "extending" the characterization of *FPtime* given in 1964 by Cobham. That is achieved by introducing pointers in the recursion schemes. Recursion with pointers can be understood as a restrict form of recursion with substitution. Leivant and Marion have work in this direction.

What is here stated using recursion schemes with bounds can be done in other frameworks. The polynomial bounds explicitly address the resource constraint of the studied complexity classes. There exist several ways of enriching the syntax, in order to build in the classes some internal control on the growth of the functions terms. This can be done, for instance, via ranks (which measure the syntactical complexity of the functions terms), distinguishing sorts of variables (Leivant style), or sorts of input-positions (Bellantoni-Cook style).

Acknowledgements. I want to thank the funding of the projects PTDC/MAT/104716/2008 and PEst-OE/MAT/UI0209/2011, from Fundação para a Ciência e a Tecnologia.

References

1 J. L. Balcázar, J. Díaz, J. Gabarró, *Structural Complexity I and II*, Springer-Verlag, (1990)
2 S. Bellantoni and S. Cook, *A new recursion-theoretic characterization of Polytime functions*, Computational Complexity, vol. 2 (1992), pp. 97–110.
3 A. Cobham, *The intrinsic computational difficulty of functions*, Proc. of the 1964 International Congress for Logic, Methodology, and the Philosophy of Science, ed. Y. Bar-Hillel, North Holland, Amsterdam (1965), pp. 24–30.
4 I. Oitavem, *Characterizing Pspace with pointers*, Mathematical Logic Quarterly, vol. 54 (2008), no. 3, pp. 317–323.
5 I. Oitavem, *A recursion-theoretic approach to NP*, Annals of Pure and Applied Logic, vol. 162 (2011), no. 8, pp. 661–666.