Towards an implicit characterization of NC^k

G. Bonfante[1], R. Kahle[2], J.-Y. Marion[1], and I. Oitavem[3]

[1] Loria - INPL, 615, rue du Jardin Botanique, BP-101, 54602 Villers-lès-Nancy, France, {Jean-Yves.Marion|Guillaume.Bonfante}@loria.fr
[2] Dept. Matemática, Universidade de Coimbra, Apartado 3008, 3001-454 Coimbra, Portugal, and CENTRIA, UNL, 2829-516 Caparica, Portugal, kahle@mat.uc.pt
[3] CMAF, Universidade de Lisboa, Av. Prof. Gama Pinto, 2, 1649-003 Lisboa, Portugal, and DM, UNL, 2829-516 Caparica, Portugal, isarocha@ptmat.fc.ul.pt

Abstract. We define a hierarchy of term systems T^k by means of restrictions of the recursion schema. We essentially use a pointer technique together with tiering. We prove $T^k \subseteq NC^k \subseteq T^{k+1}$, for $k \geq 2$. Special attention is put on the description of T^2 and T^3 and on the proof of $T^2 \subseteq NC^2 \subseteq T^3$. Such a hierarchy yields a characterization of NC.

1 Introduction

The present work enters the field of Implicit Complexity. By Implicit, we mean that focus is done on algorithms rather than functions. In particular, the question of how to compute a function with respect to of one of its algorithms arise.

Apart from its theoretical interest, the present study has some practical consequences. Since the tiering condition we consider on programs is decidable, the term systems can be used for static analysis of programs, that is for certification of bounds on the time/space usage of programs. In an other context, such an approach has been established on the theoretical study of Hofmann [Hof99,Hof02] which found applications in the Embounded Project[1].

The present approach of Implicit Complexity is in the vein of Bellantoni-Cook/Leivant, that is, we use some tiering discipline. Since the seminal papers of Simmons [Sim88], Bellantoni-Cook [BC92], and Leivant-Marion [LM93], the approach has shown to be fruitful. For instance, see Mairson and Neergaard [Nee04] who propose a nice characterization of LOGSPACE by means of a tiering discipline. Another branch of Implicit Complexity use logic, see for instance [Hof99,BM04] for recent work on the subject. We also mention the work of Niggl which covers the crucial issue of imperative programming [Nig05].

In this paper, we try to shape the form of recursion that corresponds to NC^k. In other words, we are working towards an *implicit characterization* of

Research supported by the joint french-portuguese project *Theorie et langages de programmation pour les calculs à ressources bornées/Teorias e linguagens de programação para computações com recursos limitados* from *EGIDE – GRICES*.
The last author was also partially supported by the project POCI/MAT/61720/2004 of FCT.

[1] http://www.embounded.org/

the complexity classes NC^k, $k \in \mathbb{N}$. We discuss the term systems T^k such that $T^k \subseteq NC^k \subseteq T^{k+1}$, for $k \geq 2$. NC^k is the class of languages accepted by uniform boolean circuit families of depth $O(\log^k n)$ and polynomial size with bounded gates; and $NC = \bigcup_k NC^k$.

To motivate the definition of our system we look to NC^k from the point of view of *Alternating Turing Machines* (ATMs). The relation is established by the following theorem.

Theorem 1 (Ruzzo, [Ruz81]). *Let $k \geq 1$. For any language $L \subseteq \{0,1\}^*$, L is recognized by an ATM in $O(\log^k n)$ time and $O(\log n)$ space iff it is in (uniform) NC^k.*

The underlying intuition for our implicit approach is to use ramified recursion to capture the *time* aspect and recursion with *pointers* to capture the *space* aspect. We use linear recursion in the sense of [LM00] in order to stratify the degree of the polylogarithmic time, and recursion with pointers as in [BO04]. We define term systems T^k allowing k ramified recursion of which the lowest one is equipped with pointers.

We work in a sorted context, in the vein of Leivant [Lei95]. For T^k we use $k+1$ tiers:

- tier 0 with no recursion;
- tier 1 for recursion with pointers to capture the *space* aspect;
- tiers 2 to k for ramified recursions which deal with the *time* aspect.

There are two implicit characterization of NC^1, one by Leivant-Marion using *linear ramified recurrence with substitution* [LM00], and one by Bloch [Blo94] in a Bellantoni and Cook [BC92] recursion setting. Clote [Clo90], using bounded recursion schemes, gives a machine-independent characterization of NC^k. Leivant's approach to NC^k [Lei98] is machine and resource independent, however, it is not sharp. It consists of term systems RSR^k for *ramified schematic recurrence*. RSR^k characterizes NC^k only within *three* levels:

$$RSR^k \subseteq NC^k \subseteq RSR^{k+2}, \quad k \geq 2.$$

Our term systems reduce the unsharpness of the characterization of NC^k to *two* levels:

$$T^k \subseteq NC^k \subseteq T^{k+1}, \quad k \geq 2.$$

As related work we mention here [LM95] where alternating computations was captured by mean of ramified recursion with parameter substitution. For NC there exists also an implicit characterization by use of higher type functions in [AJST01].

The structure of the paper is briefly as follows. In Section 2, we present the term systems and state some preliminary results. In Section 3, we describe the upper bounds, that is the way of compiling the term systems in terms of circuits. Section 4 is devoted to the lower bound.

2 The term systems T^k

The term systems T^k are formulated in a $k+1$-sorted context, over the tree algebra $\mathbb{T}$. The algebra $\mathbb{T}$ is generated by 0, 1 and $*$ (of arities 0, 0 and 2, respectively), and we use infix notation for $*$. As usual, we introduce three additional constants: L for the left destructor, R for the right destructor and C for the conditional. They are defined as follows: $\mathrm{L}(0) = 0$, $\mathrm{L}(1) = 1$, $\mathrm{L}(u * v) = u$, $\mathrm{R}(0) = 0$, $\mathrm{R}(1) = 1$, $\mathrm{R}(u * v) = v$, and $\mathrm{C}(0, x, y, z) = x$, $\mathrm{C}(1, x, y, z) = y$, $\mathrm{C}(u * v, x, y, z) = z$.

Following notation introduced by Leivant in [Lei95], we consider $k+1$ copies of $\mathbb{T}$. Therefore, we formally have $k+1$ copies of the constructors $\mathbb{T}$, and $k+1$ sorts of variables ranging over the different tiers. As usual, we separate different tiers by semicolons.

As initial functions of T^k one considers the constructors, destructors, conditional and projection functions over the $k+1$ tiers. T^k is closed under sorted composition over $k+1$-tiers — $f(\boldsymbol{x}_k; \ldots; \boldsymbol{x}_0) = h(\boldsymbol{g}_k(\boldsymbol{x}_k; \ldots;); \ldots; \boldsymbol{g}_0(\boldsymbol{x}_k; \ldots; \boldsymbol{x}_0))$ — and k schemes of sorted recursion as described below.

We start by define the recursion schemes of T^2 and T^3 and describe only then the general case of T^k. In what follows one should notice that, whenever f is defined by recursion with step function h, h itself cannot be defined by recursion over the same tier as f is defined. Therefore, in T^k we allow, at most, k *(step-)nested* recursions. However, in the base cases the function g can be defined by a further recursion over the same tier as f is defined, i.e., no restriction is imposed on the number of recursions constructed on top of each other (*base-nested* recursions).

2.1 The term system T^2

According to the underlying idea, the characterization of NC^2 requires three tiers:

- Tier 0: no recursion.
- Tier 1: recursion with pointers (space tier).
- Tier 2: recursion without pointers, modifying tier 1 (time tier).

Tier 1 recursion (with pointers)

$$\begin{aligned} f(; p, 0, \boldsymbol{x}; \boldsymbol{w}) &= g(; p, 0, \boldsymbol{x}; \boldsymbol{w}), \\ f(; p, 1, \boldsymbol{x}; \boldsymbol{w}) &= g(; p, 1, \boldsymbol{x}; \boldsymbol{w}), \\ f(; p, u * v, \boldsymbol{x}; \boldsymbol{w}) &= h(; ; \boldsymbol{w}, f(; p * 0, u, \boldsymbol{x}; \boldsymbol{w}), f(; p * 1, v, \boldsymbol{x}; \boldsymbol{w})). \end{aligned}$$

Tier 2 recursion

$$\begin{aligned} f(0, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}) &= g(\boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}), \\ f(1, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}) &= g(\boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}), \\ f(u * v, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}) &= h(; \boldsymbol{x}; \boldsymbol{w}, f(u, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w})). \end{aligned}$$

Since h can use the variables of lower tier, $\boldsymbol{x}$, to recurse on, we can nest recursions.

In the tier 2 recursion, the recursion input is only used as a counter. In particular, the recursion only takes the height of the tree $u * v$ into account. Therefore, it might be more natural to rewrite this scheme in form of a successor recursion: If one uses in the following scheme the expression $u + 1$ as some kind of schematic variable for $u * v$, where v is arbitrary, and 0 as a schematic expression for 1 or the actual 0 (note that $f(1, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w})$ is defined as the same as $f(0, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w})$), the scheme can be written as follows:

Tier 2 recursion (successor notation)

$$\begin{aligned} f(0, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}) &= g(\boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}), \\ f(u+1, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}) &= h(; \boldsymbol{x}; \boldsymbol{w}, f(u, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w})). \end{aligned}$$

Later on, in the course of a definition of a function using successor notation, $x + 1$ can be read as an abbreviation of $x * 1$, and the schematic notation is in accordance with the actual definition in terms of trees.

2.2 The term system T^3

According to the underlying idea, T^3 has one more tier than T^2 for recursion (without pointers). Note that the recursion for the tiers 1 and 2 differ from those for T^2 only by the extra semicolon needed for the additional tier separation.

Tier 1 recursion (with pointers)

$$\begin{aligned} f(;; p, 0, \boldsymbol{x}; \boldsymbol{w}) &= g(;; p, 0, \boldsymbol{x}; \boldsymbol{w}), \\ f(;; p, 1, \boldsymbol{x}; \boldsymbol{w}) &= g(;; p, 1, \boldsymbol{x}; \boldsymbol{w}), \\ f(;; p, u * v, \boldsymbol{x}; \boldsymbol{w}) &= h(;;; \boldsymbol{w}, f(;; p * 0, u, \boldsymbol{x}; \boldsymbol{w}), f(;; p * 1, v, \boldsymbol{x}; \boldsymbol{w})). \end{aligned}$$

Tier 2 recursion

$$\begin{aligned} f(; 0, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}) &= g(; \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}), \\ f(; 1, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}) &= g(; \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}), \\ f(; u * v, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}) &= h(;; \boldsymbol{x}; \boldsymbol{w}, f(; u, \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w})). \end{aligned}$$

Tier 3 recursion

$$\begin{aligned} f(0, \boldsymbol{z}; \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}) &= g(\boldsymbol{z}; \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}), \\ f(1, \boldsymbol{z}; \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}) &= g(\boldsymbol{z}; \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}), \\ f(u * v, \boldsymbol{z}; \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}) &= h(; \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w}, f(u, \boldsymbol{z}; \boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w})). \end{aligned}$$

Again the recursion schemes for time can be rewritten in successor notation.

Tier 2 recursion (successor notation)

$$f(;0,\boldsymbol{y};\boldsymbol{x};\boldsymbol{w}) = g(;\boldsymbol{y};\boldsymbol{x};\boldsymbol{w}),$$
$$f(;u+1,\boldsymbol{y};\boldsymbol{x};\boldsymbol{w}) = h(;;\boldsymbol{x};\boldsymbol{w},f(;u,\boldsymbol{y};\boldsymbol{x};\boldsymbol{w})).$$

Tier 3 recursion (successor notation)

$$f(0,\boldsymbol{z};\boldsymbol{y};\boldsymbol{x};\boldsymbol{w}) = g(\boldsymbol{z};\boldsymbol{y};\boldsymbol{x};\boldsymbol{w}),$$
$$f(u+1,\boldsymbol{z};\boldsymbol{y};\boldsymbol{x};\boldsymbol{w}) = h(;\boldsymbol{y};\boldsymbol{x};\boldsymbol{w},f(u,\boldsymbol{z};\boldsymbol{y};\boldsymbol{x};\boldsymbol{w})).$$

2.3 The term systems T^k

The extension of the definition of T^2 and T^3 to arbitrary k, $k \geq 4$ is straightforward. In each step from $k-1$ to k we have to add another *time tier*, adapting the notation of the existing recursion schemes to the new number of tiers and adding one more nested recursion for the new tier k of the form:

Tier k recursion

$$f(0,\boldsymbol{x_k};\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w}) = g(\boldsymbol{x_k};\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w})$$
$$f(1,\boldsymbol{x_k};\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w}) = g(\boldsymbol{x_k};\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w})$$
$$f(u*v,\boldsymbol{x_k};\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w}) = h(;\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w},f(u,\boldsymbol{x_k};\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w})$$

In successor notation this scheme reads as follows:

Tier k recursion (successor notation)

$$f(0,\boldsymbol{x_k};\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w}) = g(\boldsymbol{x_k};\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w})$$
$$f(u+1,\boldsymbol{x_k};\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w}) = h(;\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w},f(u,\boldsymbol{x_k};\boldsymbol{x_{k-1}};\ldots;\boldsymbol{x_1};\boldsymbol{w})$$

2.4 The term system T^1

T^1 is just the restriction of T^2 to two tiers only, and the single recursion scheme:

Recursion of T^1

$$f(p,0,\boldsymbol{x};\boldsymbol{w}) = g(p,0,\boldsymbol{x};\boldsymbol{w})$$
$$f(p,1,\boldsymbol{x};\boldsymbol{w}) = g(p,1,\boldsymbol{x};\boldsymbol{w})$$
$$f(p,u*v,\boldsymbol{x};\boldsymbol{w}) = h(;\boldsymbol{w},f(p*0,u,\boldsymbol{x};\boldsymbol{w}),f(p*1,v,\boldsymbol{x};\boldsymbol{w})).$$

Lemma 2. $T^1 \subseteq NC^1$.

Proof. Let us give the key argument of the proof. Writing $\mathsf{H}(t)$, the height of a term t, it is the case that arguments in tier 1 encountered along the computation have all linear height in the height of the inputs. As a consequence, for a given function, any branch of recursion *for this symbol* is done in logarithmic time in the size of the inputs. We conclude by induction on the definition of functions: a branch of the computation will involve only finitely many such function symbols, and so, can be simulated in NC^1.

By a straightforward induction on the definition of functions, one proves the key fact, that is arguments have linear height in the height of the input. In other words, when computing $f(x_1, \ldots, x_k; \boldsymbol{w})$, for all subcalls of the form $h(x'_1, \ldots, x'_n; \boldsymbol{w'})$, then $\forall x'_i : \mathsf{H}(x') \leq O(\sum_{i \leq k} \mathsf{H}(x_i))$.

Since a function of T^k which has no arguments in tiers greater than 1 can be defined also in T^1 we have the immediate corollary:

Lemma 3. *A function in T^k using only arguments in tier 1 and tier 0 is definable in NC^1.*

As an *ad hoc* designation, in the following, we call NC^1 *function* to a function using only argument in tier 1 and tier 0.

For NC^1 functions we have simultaneous recursion:

Lemma 4. *If $f_1, \ldots, f_n$ are defined by simultaneous recursion over tier 1, then they are definable in T^1.*

The proof is a straightforward adaptation of the corresponding proposition for the system $\mathbf{ST}_{\mathbb{T}}$, characterizing NC, in [Oit04, Proposition 5].

3 The upper bound of T^k

For the upper bound we model the computations of T^k by circuits. We start with the exemplary case of T^2.

Theorem 5.

$$T^2 \subseteq NC^2.$$

Proof. Let $f(\boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w})$ be a function of T^2. We will show that there is a circuit in NC^2 which computes $f(\boldsymbol{y}; \boldsymbol{x}; \boldsymbol{w})$.

The proof is done by induction on the definition of the function. The base cases and composition are straightforward. Recursion for tier 1 only leads to NC^1 functions (Lemma 3). Therefore, we have to consider only the case that a function defined by recursion for tier 2.

We consider first the case where we have only one argument in tier 2. It will serve as a paradigm for the more general case.

Base case: f has only one argument of tier 2, i.e. it is defined by the following scheme:

$$f(0; \boldsymbol{x}; \boldsymbol{w}) = g(; \boldsymbol{x}; \boldsymbol{w}),$$
$$f(u+1; \boldsymbol{x}; \boldsymbol{w}) = h(; \boldsymbol{x}; \boldsymbol{w}, f(u; \boldsymbol{x}; \boldsymbol{w})).$$

where g are h already defined. By Lemma 3 they are both in NC^1. That means there are (uniform) circuits G and H both of polynomial size and $O(\log(n))$ height that compute g and h. The circuit which computes f on $(y; \boldsymbol{x}; \boldsymbol{w})$ is given in figure 1 (with $n = |y|$) .

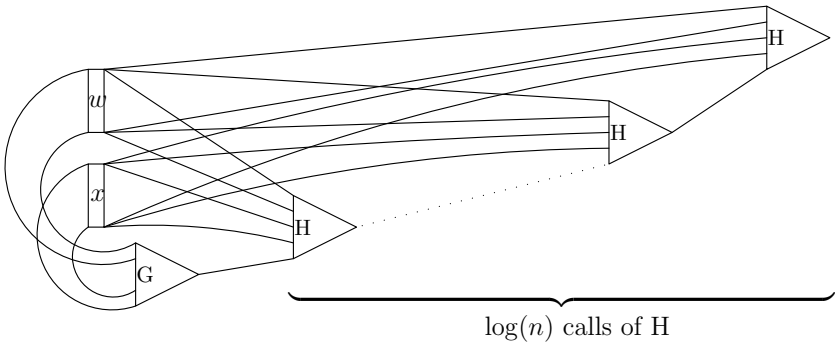

Fig. 1. The circuit F computing f

First of all, observe that the circuit is uniform. Now, the size of this circuit is $|G| + \sum_{i<|n|=\log(n)} |H|$. As the height of the tree is bounded by $O(\log(n))$, the number of H circuits is logarithmic. Since H and G circuits are of polynomial size, it is also the case of F. We end by noting that the height of the circuit is $O(\log(n)) \times O(\log(n)) = O(\log^2(n))$ since it is tree of height $O(\log(n))$ of circuit of height $O(\log(n))$. As a consequence, the circuit is in NC^2.

HIGHER ARITIES: f has $\ell + 1$ argument in tier 2, i.e. it is defined by the following scheme:

$$f(0, y_1, \ldots, y_\ell; \boldsymbol{x}; \boldsymbol{w}) = f'(y_1, \ldots, y_\ell; \boldsymbol{x}; \boldsymbol{w}),$$
$$f(u+1, y_1, \ldots, y_\ell; \boldsymbol{x}; \boldsymbol{w}) = h(; \boldsymbol{x}; \boldsymbol{w}, f(u, y_1, \ldots, y_\ell; \boldsymbol{x}; \boldsymbol{w})).$$

where f' and h are already defined. We already know that h is definable in NC^1.

Suppose that the rule for f' is

$$f'(u+1, y_2, \ldots, y_\ell; \boldsymbol{x}; \boldsymbol{w}) = h'(; \boldsymbol{x}; \boldsymbol{w}, f(u, y_2, \ldots, y_\ell; \boldsymbol{x}; \boldsymbol{w}))$$

There is a circuit H' that computes h' which is in NC^1. As a base case, we have:

$$f'(0, y_2, \ldots, y_\ell; \boldsymbol{x}; \boldsymbol{w}) = f''(y_2, \ldots, y_\ell; \boldsymbol{x}; \boldsymbol{w})$$

and f'' will itself call h'' and f''', etc. After ℓ steps, we get f^ℓ an NC^1 function as in the base case. Let us call it g as above.

The circuitry that computes f is analogous to that given in figure 1. But, in that case, the circuit for F is made of a first layer of height $O(\log(n))$ of H circuit with one leaf (the base case) which is formed by a second layer of a tree

of height $O(\log(n))$ of H' circuit, and so on. The circuit remains uniform and its size is of the form $\sum_{i<|y_0|} |H| + \sum_{i<|y_1|} |H'| + \cdots + \sum_{i<|y_\ell|} |H^{(\ell)}| + |G|$.

What is the height of the circuit? The first layer has height $O(\log(n)) \times O(\log(n))$ as in the base case. The second layer has also height $O(\log(n)) \times O(\log(n))$ for the same reason. More generally, the height of the tree is $\ell \times O(\log(n)^2) = O(\log(n)^2)$.

Concerning the size of the circuit. The first layer is formed of $\log(n)$ circuits of size $2^{O(\log(n))}$, that is bounded by a polynomial. Actually, all layers have polynomial size. Since there is only a finite number of such layers, there is a polynomial number of circuit of polynomial size.

Following the scheme of this proof, the result can be extended to arbitrary $k \geq 2$ and together with Lemma 2 we have:

Theorem 6. *For every $k \geq 1$:*

$$T^k \subseteq NC^k.$$

4 The (unsharp) lower bound

We adopt the description of NC^k in terms of ATMs, cf. Theorem 1. Here ATMs are assumed to have only one tape. Each machine has a finite number of internal states and each state is classified as either *conjunctive*, *disjunctive*, *oracle*, *accepting* or *rejecting*. Oracle, accepting or rejecting states are *halting states*. Conjunctive or disjunctive states are *action states*. Outputs are single bits — no output device is required. A *configuration* is composed by the tape contents together with the internal state of the machine.

As Leivant in [Lei98] we describe the operational semantics of an ATM M as a two stage process: Firstly, generating an input-independent computation tree; secondly, evaluating that computation tree for a given input. A binary tree T of configurations is a *computation tree* (of M) if each non-leaf of T spawns its children configurations. A computation tree of M is generated as follows: when in a configuration with an action state, depending on the state and bit read, it spawns a pair of successor configurations. These are obtained from the parent by changing the read bit, or/and changing the internal state of the machine. We will be interested in configuration trees which have the initial configuration of M as a root. Each computation tree T maps binary representation of integers (inputs) to a value in $\{0, 1, \perp\}$, where $\perp$ denotes "undefined" — in our term systems $\perp$ will be represented by $0 * 0$. This map is defined accordingly points 1 and 2, below.

1. If T is a single configuration with state q then:
   (a) if q is an accepting [rejecting] state, the returned value is 1 [respectively, 0];
   (b) if q is an action state, the returned value is $\perp$;

(c) if q is an oracle state (i, j), where i is a symbol of the machine's alphabet — 0 or 1 — and j ranges over the number of oracles, the returned value is 1 or 0 depending on whether the nth bit of the jth oracle is i or not, where n is the integer binary represented by the portion of the tape to the right of the current head position.

2. If T is not a single configuration, then the root configuration has a conjunctive or a disjunctive state. We define the value returned by T to be the conjunction, respectively the disjunction, of the values returned by the immediate subtrees.

Conjunctive and disjunctive states may diverge, indicated by the "undetermined value" $\perp$; one understands $0 \vee \perp = \perp$, $1 \vee \perp = 1$, $0 \wedge \perp = 0$, $1 \wedge \perp = \perp$.

Theorem 7.

$$NC^2 \subseteq T^3.$$

Proof. The proof runs along the lines of the proof of [BO04, Lemma 5.1].

Let M be a ATM working in $O(\log^2 n)$ time and $O(\log n)$ space. Let us say that, for any input $\boldsymbol{X}$, M runs in time $T_M = t_0{}^{\ulcorner}\boldsymbol{x}^{\urcorner} + t_1$ and space $S_M = s_0{}^{\ulcorner}\boldsymbol{x}^{\urcorner} + s_1$, where $\boldsymbol{x}$ is a minimal balanced tree corresponding to $\boldsymbol{X}$, as in [BO04].

The proof is now based on the idea of *configuration trees*. A configuration tree for M contains as paths all possible configuration codes of M.[2] A configuration tree *for time* t will code on the leafs the values at level t in the bottom-up labeling of the computation tree.

Now, the proof can be split in several steps.

Coding Configurations. A configuration of M is given by a sequence of triples which encode the content of the tape together with information about the position of the head, and, in addition, a encoding of the current state. Padding the tapes with blanks we can assume that we have fixed tape length l. To code the three symbols 0, 1 and blank we will use two bits, $(0, 1)$, $(1, 1)$, and $(0, 0)$ respectively. Now a triple $x_i = (a_i, b_i, c_i)$ in the sequence $x_0, \ldots, x_l$ codes the symbol of cell l by a_l and b_l, and c_l is 1 only for the position of the head at the current state and 0 for all other cells. Finally we add a code w for the current state at the end of this sequence, such that a configuration is uniquely determine by the bit string $x_0, \ldots, x_l, w$ which has a fixed length for all configurations. In the sequent by configuration we mean the path containing the configuration code as described above.

The label^0 function. Given a configuration p and an input x, the function $\textsc{label}^0(;;p,x;)$ returns 1, 0 or $0*0$ depending on the configuration and the input x: 1 if the configuration leads to the acceptance of x, 0 if it leads to rejection, and $0 * 0$ if p is a non-halting configuration. $\textsc{label}^0(;;p,x;)$ can be defined by composition and simultaneous recursion over tier 1. Since simultaneous tier 1 recursion can be simulated in T^1 (lemma 4), $\textsc{label}^0$ is a NC^1 function.

[2] In fact, such a tree will have a lot of branches which do not represent configurations; but these branches will not disturb.

Configuration trees. The configuration tree of time 0 is a perfect balanced tree of hight $3s_0{\ulcorner}\boldsymbol{x}{\urcorner} + 3s_1 + m$, labeled by 0, 1, or $0 * 0$, according to $\textsc{label}^0$, where m is the length needed to represent w (the code of the state). Its branches "contain" all possible configurations. A branch p is labeled by 1 if p accepts x, by 0 if p rejects x, and by $0 * 0$ otherwise (i.e. if p has an action state or if it is not a configuration).

We define $\textsc{ct}^0_{3s_0,3s_1+m}$ by meta-induction on the second index with a side-induction on the first index in the base case. Note that this definition requires space recursion, i.e., recursion in tier 1. It calls in the base case $\textsc{label}^0$, which also needs a space recursion. Since this is in the base case, we do not need (step-)nested recursion here.

$$\textsc{ct}^0_{0,0}(;;p,u,x;) = \textsc{label}^0(;;p,x;), \tag{1}$$

$$\textsc{ct}^0_{a+1,0}(;;p,0,x;) = \textsc{ct}^0_{a,0}(;;p,x,x;), \tag{2a}$$

$$\textsc{ct}^0_{a+1,0}(;;p,1,x;) = \textsc{ct}^0_{a,0}(;;p,x,x;), \tag{2b}$$

$$\textsc{ct}^0_{a+1,0}(;;p,u*v,x;) = \textsc{ct}^0_{a+1,0}(;;p*0,u,x;) * \textsc{ct}^0_{a+1,0}(;;p*1,v,x;), \tag{2c}$$

$$\textsc{ct}^0_{a,b+1}(;;p,u,x;) = \textsc{ct}^0_{a,b}(;;p*0,u,x;) * \textsc{ct}^0_{a,b}(;;p*1,u,x;). \tag{3}$$

Case (1) and the cases of (2) define $\textsc{ct}^0_{a,0}$ by meta-induction on a. Within this definition, the cases (2a)–(2c) use tier 1 recursion. Finally, case (3) is the induction step for the definition of $\textsc{ct}^0_{a,b}$ by meta-induction on b.

Now, we define the initial configuration tree $\textsc{ct}^0$ as follows.

$$\textsc{ct}^0(;;x;) = \textsc{ct}^0_{3s_0,3s_1+m}(;;0,x,x;).$$

Notice, that $\textsc{ct}^0$ is a NC^1 function.

The idea is now to update this configuration tree along the time the machine is running.

The label^{+1} function. One can define a function $\textsc{label}^{+1}$ which for a configuration p and a configuration tree z, returns 0, 1 or $0{*}0$ according as configuration p is rejecting, accepting or undetermined, using the labels of the successor configurations of p in z.

$\textsc{label}^{+1}$ uses simultaneous tier 1 recursion.

Note, that $\textsc{label}^{+1}$ is a NC^1 function.

The update function. The aim of the function $\textsc{ct}^{+1}$ is to update a configuration tree for time t to the configuration tree at time $t{+}1$. Here, we need the space recursion with step function $*$ and base function $\textsc{label}^{+1}$, in order to build a copy of the given configuration tree where the leaves are updated according to $\textsc{label}^{+1}$.

We define $\textsc{ct}^{+1}_{a,b}$ in analogy to $\textsc{ct}^{0}_{a,b}$ by meta-induction on a and b.

$$\begin{aligned}
\textsc{ct}^{+1}_{0,0}(;;p,u,x;z) &= \textsc{label}^{+1}(;;p,x;z)\\
\textsc{ct}^{+1}_{a+1,0}(;;p,0,x;z) &= \textsc{ct}^{+1}_{a,0}(;;p,x,x;z),\\
\textsc{ct}^{+1}_{a+1,0}(;;p,1,x;z) &= \textsc{ct}^{+1}_{a,0}(;;p,x,x;z),\\
\textsc{ct}^{+1}_{a+1,0}(;;p,u*v,x;z) &= \textsc{ct}^{+1}_{a+1,0}(;;p*0,u,x;z) * \textsc{ct}^{+1}_{a+1,0}(;;p*1,v,x;z),\\
\textsc{ct}^{+1}_{a,b+1}(;;p,u,x;z) &= \textsc{ct}^{+1}_{a,b}(;;p*0,u,x;z) * \textsc{ct}^{+1}_{a,b}(;;p*1,u,x;z).
\end{aligned}$$

The update of a configuration tree for time t is the configuration tree for time $t+1$. For a given configuration tree z, such an update can be performed by the function $\textsc{ct}^{+1}$:

$$\textsc{ct}^{+1}(;;x;z) = \textsc{ct}^{+1}_{3s_0,3s_1+m}(;;0,x,x;z).$$

Note, that $\textsc{ct}^{+1}$ is a NC^1 function.

The iteration. The iteration function iterates the update function $t_0{\ulcorner}\boldsymbol{x}{\urcorner}^2 + t_1$ times.

We define it by use of two auxiliary functions $\textsc{it}^1$ and $\textsc{it}^2$. $\textsc{it}^1$ iterates the update function $a{\ulcorner}\boldsymbol{x}{\urcorner}$ times; $\textsc{it}^2$ iterate then $\textsc{it}^1$ ${\ulcorner}\boldsymbol{x}{\urcorner}$ times and add b more iterations of the update function.

For a given natural number n, let $\textsc{ct}^{+n}(;;x;z)$ be the $\textsc{ct}^{+1}$ function composed with itself n times. Thus, in an inductive definition of $\textsc{ct}^{+n}$ we have that $\textsc{ct}^{+(n+1)}(;;x;z)$ is defined as $\textsc{ct}^{+1}(;;x;\textsc{ct}^{+n}(;;x;z))$.

– $\textsc{it}^1$ is defined by recursion in tier 2.

$$\begin{aligned}
\textsc{it}^1(;0;x;z) &= z,\\
\textsc{it}^1(;y+1;x;z) &= \textsc{ct}^{+t_0}(;;x;\textsc{it}^1(;y;x;z)).
\end{aligned}$$

– $\textsc{it}^2$ is defined by recursion in tier 3:

$$\begin{aligned}
\textsc{it}^2(0;y;x;z) &= \textsc{ct}^{+t_1}(;;x;z),\\
\textsc{it}^2(u+1;y;x;z) &= \textsc{it}^1(;y;x;\textsc{it}^2(u;y;x;z)).
\end{aligned}$$

Note, that $\textsc{it}^1$ and $\textsc{it}^2$ are the only non NC^1 functions needed in this proof.

Now, we just have to iterate the $\textsc{ct}^{+1}$ function T_M times, on the initial configuration tree $\textsc{ct}^0$, in order to obtain the configuration tree $\textsc{ct}^{T_M}$:

$$\textsc{ct}^{T_M}(x) = \textsc{it}^2(x;x;x;\textsc{ct}^0(;;x;))$$

Finally, recursing on x we can follow in $\textsc{ct}^{T_M}$ the path corresponding to the initial configuration and we read its label: 0 or 1.

5 Conclusion

Putting together the theorems 5 and 7 one gets:

Theorem 8.

$$T^2 \subseteq NC^2 \subseteq T^3.$$

As stated in the proof of theorem 7, only the functions which are performing the iteration are not NC^1 functions. The higher tiers are used only for the iteration. It is straightforward that one additional tier on top enables us to define an iteration of the update function $\log(n)$ times the length of iteration definable by use of the lower tiers. Together with the extension of the upper bound to T^k, stated in theorem 6, one gets:

Theorem 9. *For $k \geq 2$ we have:*

$$T^k \subseteq NC^k \subseteq T^{k+1}.$$

As a corollary we get another characterization of NC as the union of the the term systems T^k, $k \in I\!N$:

Corollary 10.

$$NC = \bigcup_{k \in I\!N} T^k.$$

References

[AJST01] Klaus Aehlig, Jan Johannsen, Helmut Schwichtenberg, and Sebastiaan Terwijn. Linear ramified higher type recursion and parallel complexity. In R. Kahle, P. Schroeder-Heister, and R. Stärk, editors, *Proof Theory in Computer Science*, volume 2183 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2001.

[BC92] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

[Blo94] S. Bloch. Function-algebraic characterizations of log and polylog parallel time. *Computational Complexity*, 4(2):175–205, 1994.

[BM04] Patrick Baillot and Virgile Mogbil. Soft lambda-calculus: A language for polynomial time computation. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004*, volume 2987 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2004.

[BO04] S. Bellantoni and I. Oitavem. Separating NC along the δ axis. *Theoretical Computer Science*, 318:57–78, 2004.

[Clo90] P. Clote. Sequential, machine independent characterizations of the parallel complexity classes *ALogTIME*, ac^k, nc^k and *nc*. In S. Buss and P. Scott, editors, *Feasible Mathematics*, pages 49–69. Birkhäuser, 1990.

[Hof99] Martin Hofmann. Linear types and non-size-increasing polynomial time computation. In *Symposium on Logic in Computer Science (LICS '99)*, pages 464–473. IEEE, 1999.

[Hof02] Martin Hofmann. The strength of non-size increasing computation. In *POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 260–269. ACM Press, New York, NY, USA, 2002.

[Lei95] Daniel Leivant. Ramifed recurrence and computational complexity I: Word recurrence and poly-time. In P. Clote and J. B. Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1995.

[Lei98] Daniel Leivant. A characterization of NC by tree recursion. In *FOCS 1998*, pages 716–724. IEEE Computer Society, 1998.

[LM93] Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of poly-time. *Fundamenta Informaticae*, 19(1/2):167–184, 1993.

[LM95] Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: substitution and poly-space. In L. Pacholski and J. Tiuryn, editors, *Proceedings of CSL 94*, pages 486–500. LNCS 933, Springer Verlag, 1995.

[LM00] Daniel Leivant and Jean-Yves Marion. A characterization of alternating log time by ramified recurrence. *Theoretical Computer Science*, 236(1–2):192–208, 2000.

[Nee04] Peter Møller Neergaard. A functional language for logarithmic space. In *Prog. Lang. and Systems: 2nd Asian Symp. (APLAS 2004)*, volume 3302 of *LNCS*, pages 311–326. Springer-Verlag, 2004.

[Nig05] Karl-Heinz Niggl. Control structures in programs and computational complexity. *Annals of Pure and Applied Logic*, 133(1-3):247–273, 2005. Festschrift on the occasion of Helmut Schwichtenberg's 60th birthday.

[Oit04] Isabel Oitavem. Characterizing NC with tier 0 pointers. *Mathematical Logic Quarterly*, 50:9–17, 2004.

[Ruz81] W. L. Ruzzo. On uniform circuit complexity. *J. Comp. System Sci.*, 22:365–383, 1981.

[Sim88] H. Simmons. The realm of primitive recursion. *Archive for Mathematical Logic*, 27:177–188, 1988.