Two function algebras defining functions in NC k boolean circuits

Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion, Isabel Oitavem

Two function algebras defining functions in NC^k boolean circuits

Guillaume Bonfante[a], Reinhard Kahle[c], Jean-Yves Marion[a], Isabel Oitavem[b,c]

[a] *LORIA - BP239, 615, rue du jardin botanique - 54506 Villers-lès-Nancy - France*
[b] *CMAF - Avenida Professor Gama Pinto, 2 - Lisboa - Portugal*
[c] *FCT-UNL - Monte de Caparica - Caparica Portugal*

Abstract

We describe the functions computed by boolean circuits in NC^k by means of functions algebra for $k \geq 1$ in the spirit of implicit computational complexity. The whole hierarchy defines NC. In other words, we give a recursion-theoretic characterization of the complexity classes NC^k for $k \geq 1$ without reference to a machine model, nor explicit bounds in the recursion schema. Actually, we give two equivalent description of the classes NC^k, $f \geq 1$. One is based on a tree structure à la Leivant [Lei98], the other is based on words. This latter puts into light the role of computation of pointers in circuit complexity. We show that transducers are a key concept for pointer evaluation.

Keywords: Boolean circuits, NC^k, parallel computation class, transducers

The core of *Implicit computational complexity* is to provide description of complexity classes which are independent from the notion of time or of space related to the underlying machine's definition. For instance, polynomial time complexity has been thoroughly examined under these terms considered Cobham-Edmonds's thesis that polynomial time is the class of feasible functions (see [Cob62, Edm65, Gol08]). Doing so, some of the key concept of implicit computational complexity have been introduced. For instance, Harold Simmons [Sim88] justifies the equivalence of some recursive schema with primitive recursion. Daniel Leivant in [Lei91], Stephen Cook and Steve Bellantoni in [BC92] brought to light the notion of ramification in recursion theory. Based on logics, there are two main directions: one is based on the Curry-Howard paradigm, see Girard's Light Linear Logic [Gir98], the other is known as Descriptive Complexity, illustrated by Immer-

Email address: bonfante@loria.fr (Guillaume Bonfante)

man's characterization of polynomial time (see [Imm98]).

In this paper, we characterize functions computed by uniform boolean circuits of polylogarithmic depth and polynomial time. More precisely, we will describe properly each layers of the hierarchy NC^k for all $k > 0$. This is, to our knowledge, the first exact characterization of each layer of the hierarchy by function algebra over infinite domains in implicit complexity. The classes NC^k were firstly described based on circuits. NC^k is the class of functions accepted by uniform boolean circuit families of depth $O(\log^k n)$ and polynomial size with bounded fan-in gates, where n is the length of the input—see [BDG90] or [Imm98]. In [Ruz81], Ruzzo identifies NC^k with the classes of languages recognized by alternating Turing machines (in short ATMs) in time $O(\log^k n)$ and space $O(\log n)$.

Compared to say polynomial time Turing Machine, computation with uniform boolean circuits rely on a description of inputs by pointers[1]; moreover, the machine stores only such pointers (this is the space bound). The second ingredient is parallelism, which is reflected by a tree of computation of depth (poly-)logarithmic with respect to the inputs. We will propose two different solutions to cope with these features.

If one embeds words into well-balanced binary trees, one gets structurally a) trees of logarithmic depth with respect to the size of inputs, which–by means of a tiering mechanism–may serve as a basis for time iteration, b) any sub-term of the tree is a window on a sub-word of the input, that is a pointer on the input and c) structural induction on trees fit with the tree-like nature of alternating computing. Daniel Leivant benefited from these three salient aspects in his description of NC by means of so-called ramified tree recurrence. His schema is parametrized by a number k and he proves $\textsc{rsr}_k \subseteq \mathrm{NC}^k \subseteq \textsc{rsr}_{k+2}$ for all $k > 1$, missing however the exact delineation of the hierarchy. His ideas have been reworked by Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion and Isabel Oitavem in [BKMO08] where mutual in place recursion (MIP) is introduced. An other source of inspiration of [BKMO08] was the description of NC^1, that is ALOGTIME, by Daniel Leivant and Jean-Yves Marion [LM00].

Based on a variant of ramified recurrence, Steve Bloch characterized ALOGTIME. Ramified recurrence over trees was introduced by Daniel Leivant in [Lei93], then reconsidered by Steve Bellantoni in [Bel95] where he gives a characterization of alternating poly-log functions.

Using trees however leads to (at least) two issues. First, computations are done on well-balanced tree, thus not on a free algebra. Hence, we are not talking of an intrinsic property of the recursion schema, but on a property of the schema for some

[1] As for Random Access Machines, a sequential model of computation.

restricted subset of trees. Second, since trees serve both for inputs and pointers, the schema do not reveal the realm of pointer computations. For that reason, we provide a second description of the hierarchy based on two other recursion schema, one is called *rational bitwise equations* (RBE). This schema describes basic functions as a two step process: first, transducers rely some input bits together, second a finite map is applied on these latter bits. The key ingredient is that inputs bits– that is pointers– are computed by transducers, thus involving a very weak form of induction. The second schema is time iteration, which corresponds to a ramified version of primitive recursion on pointers. Based on some word recurrence schema, we mention here the work of Clote on the class NC and the hierarchy which appeared in [Clo90]. Compared to our proposition, Clote needs explicit bounds on the recursion schema, thus violating one of the "rules" of implicit computational complexity.

Taking a view based on logics, boolean circuits where addressed by Immermann in terms of Descriptive Complexity, see for instance [Imm98]. They have been considered by Mogbil et al in [MR07].

The paper is based originally on the conference contribution [BKMO08]. Compared to that, recursion schema are extended to arbitrary word signatures. And the proof has been cleaned. More importantly, the equivalence between MIP and RBE is new. Our thesis is that this result enforces the definition of MIP, showing its robustness, justifying its form. Second point, the new recursion schema called RBE open a new window on computation on pointers in implicit computational complexity.

1. Preliminaries

For some set X, we define $\mathfrak{P}(X) = \{U \mid U \subseteq X\}$ of subsets of X. The set 1 is an arbitrary singleton set. It is clear that $1 \times X$ is isomorphic to X. Thus, a sequence λ indexed by $1 \times X$ will be presented as $(\lambda_x)_{x \in X}$.

As we will come back to it later, we recall that, given a semi-ring $(A, 0, +, 1, \times)$ and two finite sets P and Q, a matrix of dimension $P \times Q$ on A is a table data $m = (m_{p,q})_{(p,q) \in P \times Q}$ whose entries are in A. The set of such matrices is written $A^{P \times Q}$. Matrices of equal dimensions can be summed. Given m and n of dimension $P \times Q$, $m + n = (m_{p,q} + n_{p,q})_{p,q}$. Matrices are multiplied according to the usual rules. Given $m = (m_{p,q})_{(p,q) \in P \times Q}$ and $n = (n_{q,r})_{(q,r) \in Q \times R}$, we set $m \times n$ —of dimension $P \times R$— defined by its components $(m \times n)_{p,r} = \sum_{q \in Q} m_{p,q} \times n_{q,r}$.

Given a matrix $m \in A^{P \times Q}$, $m_{p,q}$ denotes the entry at position p, q. Sometimes, when indices become too heavy, the entry is denoted $m[p, q]$.

To end with general notations, all along, sequences $x_1, \ldots, x_k$ are written $\vec{x}$ when the context makes it clear.

1.1. Words, words, words

Given some *alphabet* Σ of *letters*, the set Σ^* denotes the set of words over Σ. The empty string is written ϵ and $w.w'$ denotes the concatenation of two words w and w'. The length of a word is written $|w|$. The set $\mathbb{W}$ denotes words in $\{0,1\}^*$. Given some $n \geq 0$, $\mathbb{W}_n$ denotes its subset of words of size n, whereas $\mathbb{W}_{\leq n}$ denote the subset of words w such that $|w| \leq n$.

A language on Σ is a subset $L \subseteq \Sigma^*$. Given two languages L_1, L_2 on Σ, $L_1 + L_2 \triangleq L_1 \cup L_2$, $L_1 \cdot L_2 \triangleq \{w \cdot u \mid w \in L_1, u \in L_2\}$. We define $L_1^0 = \{\epsilon\}$ and $L_1^{i+1} = L_1^i \cdot L_1$. Finally, $L_1^* = \bigcup_{i \in \mathbb{N}} L_1^i$. The rational expressions $Rat(\Sigma)$ are defined by the grammar:

$$Rat(\Sigma) ::= 0 \mid 1 \mid a \mid Rat(\Sigma) + Rat(\Sigma) \mid Rat(\Sigma) \cdot Rat(\Sigma) \mid Rat(\Sigma)^*$$

Rational expressions are interpreted inductively: $[\![0]\!] = \emptyset$, $[\![1]\!] = \{\epsilon\}$, $[\![a]\!] = \{a\}$ for all $a \in \Sigma$, $[\![L + M]\!] = [\![L]\!] + [\![M]\!]$, $[\![L \cdot M]\!] = [\![L]\!] \cdot [\![M]\!]$ and $[\![L^*]\!] = [\![L]\!]^*$. The interpretations of rational expressions are the regular languages. In the sequel, we will not use anymore brackets. The contexts will make clear what we are talking about.

We recall that the 5-tuple $\langle \mathfrak{P}(\Sigma^*), 0, +, 1, \cdot \rangle$ is a semi-ring. Indeed, $\langle \mathfrak{P}(\Sigma^*), 0, + \rangle$ is a commutative monoid, $\langle \mathfrak{P}(\Sigma^*), 1, \cdot \rangle$ is a monoid, the product $\cdot$ distributes over $+$ and 0 is nullary: $0 \cdot L = 0$ for all $L \in \mathfrak{P}(\Sigma^*)$. Finally, $\langle Rat(\Sigma), 0, +, 1, \cdot \rangle$ is a sub-semi-ring of $\langle \mathfrak{P}(\Sigma^*), 0, +, 1, \cdot \rangle$.

1.2. Les arbres ne voyagent que par leur bruit

The MIP-recursion schema is defined on trees. In this section, we show how we relate computations over trees to computations over words.

Given some alphabet Σ, the tree algebra $\mathbb{T}_\Sigma$ is generated by the 0-ary constructors $a \in \Sigma$ and a binary constructor $\star$. In other words, $\mathbb{T}_\Sigma$ can be seen as the set of binary trees whose leaves are labeled by letters $a \in \Sigma$. Let $|t|$ denote the size of the tree t, that is the number of constructors defining t, $\mathsf{H}(t)$ corresponds to the usual notion of height. We take the convention that $\mathsf{H}(a) = 0$ for $a \in \Sigma$. We say that a tree t is *perfectly balanced* if it has exactly $2^{\mathsf{H}(t)}$ leaves.

Given some alphabet Σ, let $\mathrm{Pow}_2(\Sigma) = \{w \in \Sigma^* \mid |w| = 2^k \text{ with } k \geq 0\}$. Let us suppose for a while that the letter # does not belong to Σ. Any word $w \in \Sigma^*$ can be padded by k extra letters # to the two's power length: $w_\# = w\#^k$ with $k \geq 0$ such that $|w_\#| = 2^{\lfloor \log_2(|w|) \rfloor + 1}$. One may observe that $|w_\#| \leq 2|w|$. Thus, the encoding has no cost up to a linear factor.

A function $\phi : (\Sigma^*)^k \to \Sigma^*$ is represented by $\phi' : \mathrm{Pow}_2(\Sigma \cup \#)^k \to \mathrm{Pow}_2(\Sigma \cup \#)$ iff for all $w_1, \ldots, w_k$, $\phi'(w_{1\#}, \ldots, w_{k\#}) = \phi(w_1, \ldots, w_k)_\#$. Now that we related functions over Σ^* to functions over $\mathrm{Pow}_2(\Sigma \cup \#)$, to avoid heavy notations and

considerations, we simply forget the padding letter # and we restrict our attention to functions over $\mathrm{Pow}_2(\Sigma)$ for some suitable alphabet Σ.

For any alphabet Σ, any word $w \in \mathrm{Pow}_2(\Sigma)$ can be encoded as a perfectly balanced tree $\widehat{w} \in \mathbb{T}_\Sigma$ of size $2|w| - 1$. A function $\phi : \mathrm{Pow}_2(\Sigma)^k \to \mathrm{Pow}_2(\Sigma)$ is represented by a function $f : \mathbb{T}(\Sigma)^k \to \mathbb{T}(\Sigma)$ iff for all $w_1, \ldots, w_k \in \mathrm{Pow}_2(\Sigma)$,

$$\widehat{\phi(w_1, \ldots, w_k)} = f(\widehat{w_1}, \ldots, \widehat{w_k}).$$

Given a non-empty (enumerable) set of variables $\mathcal{X}$, we denote by $\mathbb{T}_\Sigma(\mathcal{X})$ the term-algebra of binary trees whose leaves are labeled by letters in Σ or by variables from $\mathcal{X}$. If t, u denote some terms and x is a variable, the term $t[x \leftarrow u]$ denotes the substitution of x by u in t. Then, $t[x \leftarrow u, y \leftarrow v] = t[x \leftarrow u][y \leftarrow v]$. All along, we take care to avoid variables clashes. That is, in the example above, y is supposed not to be an occurrence of u neither x an occurrence of v. When we have a collection I of variable substitutions, we use the notation $t[(x_w \leftarrow u_w)_{w \in I}]$. Again, we will avoid conflicts of variables.

We now introduce some convenient notations, used extensively all along the paper.

Notation 1. Given a set of variables $\mathcal{X} = (x_w)_{w \in \mathbb{W}}$, we define a family of perfectly balanced trees that we call tree patterns $(\mathrm{t}_i)_{i \in \mathbb{N}}$ in $\mathbb{T}_\Sigma(\mathcal{X})$ where each leaf is labeled by a distinct variable:

$$\begin{aligned} \mathrm{t}_0 &= x_\epsilon \\ \mathrm{t}_{i+1} &= \mathrm{t}_i[(x_w \leftarrow x_{0w})_{w \in \mathbb{W}_i}] \star \mathrm{t}_i[(x_w \leftarrow x_{1w})_{w \in \mathbb{W}_i}] \end{aligned}$$

Observe that the index in a variable of some tree pattern indicates the path from the root to it. For example, in $\mathrm{t}_2 = (x_{00} \star x_{01}) \star (x_{10} \star x_{11})$, x_{10} denotes the first child of the second child of t_2. The use of the t's and substitutions makes notations very short. For instance, $\mathrm{t}_2[(x_w \leftarrow f_w(x_w))_{w \in \mathbb{W}_2}] = (f_{00}(x_{00}) \star f_{01}(x_{01})) \star (f_{10}(x_{10}) \star f_{11}(x_{11}))$. This notation is particularly useful to define "big-step" recursion equations as in:

$$f((x_{00} \star x_{01}) \star (x_{10} \star x_{11})) \quad = \quad (f(x_{00}) \star f(x_{01})) \star (f(x_{10}) \star f(x_{11}))$$

which we shall write: $f(\mathrm{t}_2) = \mathrm{t}_2[(x_w \leftarrow f(x_w))_{w \in \mathbb{W}_2}]$

1.3. Finite state transducers

Entre a àrvore e o vê-la
Onde está o sonho?

As we will refer to Sakarovitch's book [Sak09] on automata theory, we use his notations. However, since we will only use the same alphabet Σ both for input datas and output datas, we mention only one alphabet in definitions.

Definition 2. A transducer is a 5-tuple $\mathcal{M} = \langle Q, \Sigma, E, I, F\rangle$ made of a set of *states* Q, an alphabet Σ, a subset $I \subseteq Q$ of *initial* states, $F \subseteq Q$ the *final* states and a transition relation $E \subseteq Q \times \Sigma^* \times \Sigma^* \times Q$.

A transducer $\mathcal{M} = \langle Q, \Sigma, E, I, T\rangle$ induces a relation $E^* \subseteq Q \times \Sigma^* \times \Sigma^* \times Q$ defined by: $(q, \epsilon, \epsilon, q) \in E^*$ for all $q \in Q$ and $(q, w \cdot w', u \cdot u', q') \in E^*$ iff $(q, w, u, q'') \in E^*$ and $(q'', w', u', q') \in E$. The relation E^* induces itself a relation $[\mathcal{M}] \subseteq \Sigma^* \times \Sigma^*$ defined as follows: $(w, u) \in [\mathcal{M}]$ iff $(q, w, u, q') \in E^*$ with $q \in I$ and $q' \in F$. A relation $R \subseteq \Sigma^* \times \Sigma^*$ is qualified as rational if there is a transducer $\mathcal{M}$ such that $[\mathcal{M}] = R$.

When a rational relation is actually a partial function, $[\mathcal{M}](w)$ denotes the unique u such that $(w, u) \in [\mathcal{M}]$ if such a u exists. Otherwise, we write $[\mathcal{M}](w) = \perp$.

In the sequel, to present transition relations E, we use the notation $q \xrightarrow{w,u}_E q'$ rather than $(q, w, u, q') \in E$.

A transducer $\mathcal{M}$ is represented by the *matrix-triple* (λ, μ, ν) of dimension R iff $\lambda \in Rat(\Sigma)^{1\times R}$, for all $a \in \Sigma : \mu(a) \in Rat(\Sigma)^{R\times R}$, $\nu \in Rat(\Sigma)^{R\times 1}$ and for all words $w = a_1 \cdots a_n \in \Sigma^*$, $(w, u) \in [\mathcal{M}]$ iff

$$u \in \lambda \times \mu(a_1) \times \cdots \times \mu(a_n) \times \nu.$$

Actually, μ extends as a monoid morphism $\mu(a_1 \cdots a_n) = \mu(a_1) \times \cdots \mu(a_n)$ so that the relation above is read $u \in \lambda \times \mu(w) \times \nu$. We recall that

Proposition 3. *Any transducer $\mathcal{M}$ is represented by a matrix-triple (λ, μ, ν).*

For sequential transducers[2], the matrix representation is even simpler.

Definition 4. A sequential transducer is a 7-tuple $\mathcal{M} = \langle Q, \Sigma, \delta, \eta, i, I, T\rangle$ such that:

- $\langle Q, A, \delta, i, T\rangle$ is a deterministic automaton,
- I is a word in Σ^*,
- the output function η is a function $Q \times \Sigma \to \Sigma^*$,
- T is a partial function $Q \to \Sigma^*$ on some domain $F \subseteq Q$ called the set of final states.

[2]We use the terminology of Sakarovitch. They are sometimes qualified as sub-sequential in the literature.

The functions δ and η extends to words by:

$$\begin{array}{lcllcl} \delta(q,\epsilon) & = & q & \delta(q,w.a) & = & \delta(\delta(q,w),a) \\ \eta(q,\epsilon) & = & \eta & \eta(q,w.a) & = & \eta(q,w)\cdot\eta(\delta(q,w),a) \end{array}$$

The function computed by the sequential transducer is defined for all words $w \in \Sigma^*$: $[\mathcal{M}](w) = I \cdot \eta(i,w) \cdot T(\delta(i,w))$.

A co-sequential transducer is a 6-tuple $\mathcal{M}$ as above but interpreted differently: $[\mathcal{M}](w) = \overleftarrow{I \cdot \eta(i,\overleftarrow{w}) \cdot T(\delta(i,\overleftarrow{w}))}$ with $\overleftarrow{a_1 \cdots a_n} = a_n \cdots a_1$. That is a co-sequential transducer can be seen as a sequential transducer, but with input read from right to left (and so are written outputs).

Functions computed by sequential transducers and co-sequential transducers are rational. Moreover, they have interesting matrix representations. For sequential transducers, the representation is a triple (λ,μ,ν) such that λ is a vector with only one non zero entry which is in Σ^* (thus a singleton language to be compared with $Rat(\Sigma)$ for transducers in general), for all $a \in \Sigma$, $\mu(a)$ is a row-monomial matrix (any line contains at most one non zero entry) with entries in Σ^* and ν is a vector whose entries are in Σ^*. For co-sequential transducers, λ is a vector over Σ^*, for all a, $\mu(a)$ is a column-monomial (columns contain at most one non zero entry) matrix on Σ^*, and ν has only one non zero coefficient in Σ^*.

Theorem 5 (Elgot and Mezei)**.** *Any rational function is the composition of a sequential function*[3] *and a co-sequential function*[4].

The matrix representation of the composition of the two transducers is interesting. Let (λ,μ,ν) be the representation of a sequential transducer $\mathcal{M}$ and (η,κ,χ) be a representation of the co-sequential transducer $\mathcal{N}$. The composition of the two transducers is represented by block matrices (ζ,π,ω) with:

- $\zeta \in (\Sigma^*)^{1\times(Q\times R)}$, $\zeta_{q,r} = (\eta \times \kappa(\lambda_q))_r$,
- For all $a \in \Sigma$, $\pi(a) \in \Sigma^{(Q\times R)\times(Q\times R)}$ with $\pi(a)_{(q,r),(s,t)} = (\kappa(\mu(a)_{q,s}))_{r,t}$,
- $\omega \in (\Sigma^*)^{(Q\times R)\times 1}$, $\omega_{q,r} = (\kappa(\nu_q) \times \chi)_r$.

Grouping the matrix by blocks of dimension R, that is considering the matrix of dimension $Q\times Q$ whose entries are $\Pi_{q,s} = (\pi_{q,r,s,t})_{r,t} \in (\Sigma^*)^{R\times R}$, one observes that $(\Pi_{q,s})_{q,s}$ is row-monomial and that for all q, s, the matrix $\Pi_{q,s}$ is column-monomial. Such matrices are called semi-monomial.

[3] that is a function computed by a sequential transducer.

[4] Idem.

Simple calculations show that the vector ζ has at most R non zero entries. Moreover, these entries are indexed (q_0, r) with q_0 the unique index of λ with a non zero entry and $r \in R$. For ω, the non-zero entries have indices (q, r_0) with $q \in Q$ and r_0 the unique index of χ with a non zero entry.

2. The Mutual In Place Recursion Schema

All along in this section, we suppose given some alphabet $\Sigma \ni \{\mathbf{0}, \mathbf{1}, \#\}$, where # serves for padding. Computations are done via their well-balanced tree encoding seen in Section 1.2

Definition 6. The set of *basic functions* is $\mathcal{B} = \Sigma \cup \{\star, (\pi_i^j)_{i \leq j}, \mathsf{cond}, \mathsf{d}_0, \mathsf{d}_1\}$ where Σ and $\star$ are the constructors of the algebra $\mathbb{T}_\Sigma$, d_0 and d_1 are the destructors, cond is a conditional, π_i^j are the projections. Destructors and conditional are defined as follows:

$$\begin{aligned}
\mathsf{d}_0(c) = \mathsf{d}_1(c) &= c, && c \in \Sigma \\
\mathsf{d}_0(t_0 \star t_1) &= t_0, \qquad \mathsf{d}_1(t_0 \star t_1) = t_1, \\
\mathsf{cond}(c, x_a, \ldots, x_z, x_\star) &= x_c, && c \in \Sigma \\
\mathsf{cond}(t_0 \star t_1, x_a, \ldots, x_z, x_\star) &= x_\star.
\end{aligned}$$

with $\{a, \ldots, z\} = \Sigma$.

The set of basic functions closed by composition is called the set of *explicitly defined functions* (edf). If the output of an edf is in Σ, then we say that the function is *atomic*. A function is said to be a *generalized destructor* if it is obtained by compositions of the projections and destructors. Let $w \in \mathbb{W}^*$, d_w denotes the identity function if $w = \epsilon$. Otherwise, $\mathsf{d}_{w.b} = \mathsf{d}_b \circ \mathsf{d}_w$. Given a generalized destructor $f(\vec{x})$, an immediate observation shows that there is an index i and a word w such that for all $\vec{x}$, $f(\vec{x}) = \mathsf{d}_w(x_i)$.

Definition 7. MIP is the set of functions obtained by closure of the set $\mathcal{B}$ under composition and mutual in place recursion (MIP). INC^k is the closure of the set $\mathcal{B}$ under composition, mutual in place recursion (MIP), explicit structural recursion (ESR), and time iteration (TI) for k.

The mentioned schemes are described below.

Theorem 8. *For $k \geq 1$, the set of functions over words represented in INC^k is exactly the set of functions computed by circuits in NC^k.*

The proof of the theorem is a direct consequence of Proposition 32 and Proposition 34 coming in Section 4 and 5.

2.1. Mutual In-Place recursion

The functions $(f_i)_{i\in I}$ (with the set I finite) are defined by *mutual in place recursion* (MIP) if they are defined by a set of equations, with $i, j, l \in I$ and $c \in \Sigma$, of the form

$$f_i(t_0 \star t_1, \vec{u}) = f_j(t_0, \vec{\sigma}_{i,0}(t_0 \star t_1, \vec{u})) \star f_l(t_1, \vec{\sigma}_{i,1}(t_0 \star t_1, \vec{u})) \quad (1)$$

$$f_i(c, \vec{u}) = g_{i,c}(c, \vec{u}) \quad (2)$$

where $\vec{\sigma}_{i,0} = (\sigma_{i,0,1}, \ldots, \sigma_{i,0,m})$ and $\vec{\sigma}_{i,1} = (\sigma_{i,1,1}, \ldots, \sigma_{i,1,n})$ are sequences of generalized destructors and the functions $g_{i,c}$ are atomic functions.

Example 9. The following function turns the leaves of its argument to some fixed constant $c \in \{\mathbf{0}, \mathbf{1}, \bot\}$:

$$\mathsf{const}_c(t_0 \star t_1) = \mathsf{const}_c(t_0) \star \mathsf{const}_c(t_1)$$
$$\mathsf{const}_c(c') = c \qquad c' \in \{\mathbf{0}, \mathbf{1}, \bot\}$$

Example 10. Using $\mathbf{0}$ as false and $\mathbf{1}$ as true, we can compute the bitwise-or of their labels using MIP-recursion, as follows:

$$\mathsf{or}(t_0 \star t_1, u_0 \star u_1) = \mathsf{or}(t_0, u_0) \star \mathsf{or}(t_1, u_1)$$
$$\mathsf{or}(\mathbf{0}, b) = b \text{ for } b \in \{\mathbf{0}, \mathbf{1}\}$$
$$\mathsf{or}(\mathbf{1}, b) = \mathbf{1} \text{ for } b \in \{\mathbf{0}, \mathbf{1}\}$$

Actually, all "bitwise boolean formula" of several balanced trees of the same size can be written in a similar manner. In Section 3, we provide a precise equivalence between MIP and a class of bitwise functions.

All functions defined mutually by the MIP-schema share their first argument, the one which serves for recursion. Then, given a sequence of trees $\vec{t}$, a function defined by MIP-recursion outputs on $\vec{t}$ a tree which has the shape of t_0, the first element of the sequence, with possibly different leaves.

The remaining of the section is devoted to some equivalent description of MIP-functions. These variations show that the schema is actually quite robust.

Proposition 11. *In the MIP schema, Equation 2 may equivalently be replaced by equation of the form:*

$$f_i(c, \vec{u}) = g_i(c, \vec{u}) \quad (3)$$

with g_i atomic.

Indeed, the equations $f_i(c, \vec{u}) = g_{i,c}(\vec{u})$ are equivalent to by $f(c, \vec{u}) = \mathsf{cond}(c, g_{i,a}(\vec{u}), \ldots, g_{i,z}(\vec{u}), \bot)$ which is atomic as long as all the $g_{i,c}$ are atomic.

Proposition 12. *In the definition of functions defined by* MIP*, one may suppose that the functions $g_{i,c}(x_1, \ldots, x_k)$ have the* `if-then-else` *form, that is $g(x_1, \ldots, x_k) =$*

$$\begin{array}{ll} \textit{if } x_1 = a_{1,1} \wedge x_2 = a_{1,2} \wedge \cdots \wedge x_k = a_{1,k} & \textit{then} \quad b_1 \\ \textit{else if } \ldots & \\ \textit{else if } x_1 = a_{i,1} \wedge x_2 = a_{i,2} \wedge \cdots \wedge x_k = a_{i,k} & \textit{then} \quad b_i \\ \textit{else if } \ldots & \\ \textit{else } b_0 & \end{array}$$

where the $(a_{i,1}, \ldots, a_{i,k})_i$ range in all combination of Σ^k and $b_i \in \Sigma$ for $i \leq |\Sigma|^k$.

The `if-then-else` form can be seen as a generalized conditional working simultaneously on all arguments. The proposition states that one may transform simultaneous pattern matching to its corresponding nested form.

Proof. By induction on the structure of atomic functions. The result is almost immediate for functions defined with the conditional, projections and compositions. The problem comes from the use of destructors. Consider for instance, $g_{i,c}(x, \vec{y}) = \mathsf{cond}(\mathsf{d}_0(x), e_a, \ldots, e_z, e_\star)$ with $e_a, \ldots, e_\star$ some (atomic) expressions. It cannot be rewritten as above (indeed, one cannot perform a test on a sub-tree with an `if-then-else` function). In that case, we replace the definitions of the f_i function as follows: we add to f_i a new argument t_{k+1} which will be made equal to $\mathsf{d}_0(t_1)$. To do that, replace any definition: $f_j(t_0 \star t_1, \vec{u}) = f_i(t_0, \vec{\sigma}_{i,0}(t_0 \star t_1, \vec{u})) \star f_l(t_1, \vec{\sigma}_{i,1}(t_0 \star t_1, \vec{u}))$ with $j \neq i \neq l$ by

$$f_j(t_0 \star t_1, \vec{u}) = f_i(t_0, \vec{\sigma}_{i,0}(t_0 \star t_1, \vec{u}), \mathsf{d}_0(\sigma_{1,0}(t_0 \star t_1, \vec{u}))) \star f_l(t_1, \vec{\sigma}_{i,1}(t_0 \star t_1, \vec{u}))$$

and replace the definition of $g_{i,c}(x, \vec{y})$ by

$$g_{i,c}(x, \vec{y}, y_{k+1}) = \mathsf{cond}(y_{k+1}, e_a, \ldots, e_z, e_\star).$$

A definition of the form $f_j(t_0 \star t_1, \cdots) = f_l(\cdots) \star f_i(\ldots)$ is modified in the same way. And for the definition of f_i itself, simply forget the last argument: $f_i(t_0 \star t_1, \vec{u}, t_{k+1}) = f_j(t_0, \vec{\sigma}_{i,0}(t_0 \star t_1, \vec{u})) \star f_l(t_1, \vec{\sigma}_{i,1}(t_0 \star t_1, \vec{u}))$. Observe that the new definition of $g_{i,c}$ does not refer to the destructor anymore so that we come back to the preceding case. We let the reader convince himself that the trick mentioned above applies to any atomic function. □

Proposition 13. *Any* MIP*-function on some alphabet Σ can be computed (via some encoding) by a* MIP*-function over the booleans $\mathbb{B} = \{\mathbf{0}, \mathbf{1}\}$.*

Proof. Let $\Sigma = \{c_1, \ldots, c_k\}$. We encode the c_i's as well-balanced trees $e_i \in \mathbb{T}_{\mathbb{B}}$ of height H for some $H > 0$. More generally, $e(c)$ denote the encoding of constant c and e extends homomorphically: $e(t_0 \star t_1) = e(t_0) \star e(t_1)$.

We suppose given some MIP-equations for some functions $(f_i)_{i \in I}$ defined on the alphabet Σ. To enhance the readability of the proof, we suppose that the f_i's have exactly two arguments. Generalization to $k > 2$ arguments is tedious but not difficult. In other words, we suppose given equations:

$$f_i(t_0 \star t_1, u) = f_j(t_0, \sigma_{i,0}(t_0 \star t_1, u)) \star f_\ell(t_1, \sigma_{i,1}(t_0 \star t_1, u)) \tag{4}$$

$$f_i(c, u) = g_i(c, u) \tag{5}$$

First point, think of the g_i in `if-then-else` form. Then, consider $g'_i(x_0, x_1)$ defined as follows:

if $x_0 = e_1 \wedge x_1 = e_1$	then	$e(g_i(c_1, c_1))$
else if $\ldots$		
else if $x_0 = e_i \wedge x_1 = e_j$	then	$e(g_i(c_i, c_j))$
else if $\ldots$		
else $e(g_i(\bot \star \bot, \bot))$		the default value

g'_i is an explicitly defined function. It is not atomic since it outputs trees, but for any word u of length H, the function $g'_{i,u} = \mathsf{d}_u \circ g'_i$ is both explicitly defined and atomic.

Now the encoding of atomic functions is done, to see the trouble for the induction step, suppose that $\Sigma = \{0, 1, 2, \bot\}$ is encoded $\mathbf{0} \star \mathbf{0}, \mathbf{0} \star \mathbf{1}, \mathbf{1} \star \mathbf{0}, \mathbf{1} \star \mathbf{1}$. Then $e(0 \star 2) = (\mathbf{0} \star \mathbf{0}) \star (\mathbf{1} \star \mathbf{0})$. In the induction schema, there is no way to distinguish the $\star$ at the root of $e(0 \star 2)$–which corresponds to an inductive step in the initial schema– from the $\star$ in the sub term $(\mathbf{1} \star \mathbf{0})$–which corresponds to a base case. In other words, one cannot stop induction when arguments are the encoding of some constants. To cope with that issue, the main trick is to keep a delayed index and to add "delayed" side arguments with respect to current arguments. The side variables serve to recover encoded constants.

Let us introduce notations. $\iota(i, 0)$ denotes the index j of Equation 4 and $\iota(i, 1)$ index ℓ. As previously observed, for all $i \in I$, $b \in \{0, 1\}$, since $\sigma_{i,b}$ is a generalized destructor, there is a word $\alpha(i, b) \in \{0, 1\}^*$ and $c(i, b) \in \{0, 1\}$ such that $\sigma_{i,b}(x_0, x_1) = \mathsf{d}_{\alpha(i,b)}(x_{c(i,b)})$.

Let $M = \max\{|\alpha(i, b)| \mid i \in I, b \in \{0, 1\}\}$. We introduce functions $f_{i,i',w,v_1,\ldots,v_k}$ indexed as following: $i, i' \in I$, $w \in \{0, 1\}^{\leq H}$, $v_1, \ldots, v_k \in \{0, 1\} \times \{0, 1\}^{\leq M}$ and $k = |w|$. Observe that there are finitely many indices. All these functions have arity

4. And they are designed so that $f_i(t,u) = f_{i,i,\epsilon,[]}(t,u,t,u)$. Let the equations:

$$\begin{aligned} f_{i,i',w,v_1,\ldots,v_k}(t_0 \star t_1, u, t', u') = \\ f_{j,i',w\cdot 0,v_1,\ldots,v_k,(c(i,0),\alpha(i,0))}(t_0, \mathsf{d}_{\alpha(i,0)}(u_{c(i,0)}), t', u') \quad \star \\ f_{\ell,i',w\cdot 1,v_1,\ldots,v_k,(c(i,1),\alpha(i,1))}(t_0, \mathsf{d}_{\alpha(i,1)}(u_{c(i,1)}), t', u') \end{aligned}$$

when $|w| \leq H-1$. In the equation, u_0 denotes $t_0 \star t_1$ and $u_1 = u$. For $|w| = H$, let:

$$\begin{aligned} f_{i,i',b\cdot w',(c,\alpha),v_2,\ldots,v_n}(t_0 \star t_1, u, t', u') = \\ f_{j,\iota(i',b),w'\cdot 0,v_2,\ldots,v_n,(c(i,0),\alpha(i,0))}(t_0, \mathsf{d}_{\alpha(i,0)}(u_{c(i,0)}), \mathsf{d}_b(t'), \mathsf{d}_\alpha(u'_c)) \quad \star \\ f_{\ell,\iota(i',b),w'\cdot 1,v_2,\ldots,v_n,(c(i,1),\alpha(i,1))}(t_0, \mathsf{d}_{\alpha(i,1)}(u_{c(i,1)}), \mathsf{d}_b(t'), \mathsf{d}_\alpha(u'_c)) \end{aligned}$$

where u'_0 denotes t' and u'_1 denotes u'. The base case is treated as follows:

$$f_{i,i',w,v_1,v_2,\ldots,v_n}(c,u,t',u') = g'_{i',w}(t',u')$$

One will observe that a) any primed element correspond to a delay of H steps in the computation and thus b), when reaching the base case, the arguments i' correspond to the index of the basic function to be called and t' and u' exactly point on the two encoded constants. Thus the result. □

Lemma 14. We suppose given a (finite) family $(n_i)_{i\in I}$ of integers, and a family $(f_i)_{i\in I}$ of functions satisfying equations of the form:

$$f_i(\mathrm{t}_{n_i}, \vec{u}) = \mathrm{t}_{n_i}[(x_w \leftarrow f_{\mathrm{p}(i,w)}(x_w, \vec{\sigma}_{i,w}(\vec{u})))_{w\in\mathbb{W}_{n_i}}], \tag{6}$$

$$f_i(\mathrm{t}_m[(x_w \leftarrow c_w)_{w\in\mathbb{W}_m}], \vec{u}) = \mathrm{t}_m[(x_w \leftarrow g_{i,w,c_w}(\vec{u}))_{w\in\mathbb{W}_m}], 0 \leq m < n_i, \tag{7}$$

where p is a finite mapping from $I \times \mathbb{W}$ to I, $c_w \in \Sigma$, $\vec{\sigma}_{i,w}$ are vectors of $\star$-free explicitly defined functions, and $(g_{i,w,c_w})_{i\in I, w\in\mathbb{W}, c_w\in\Sigma}$ are explicitly defined boolean functions. Then, the functions $(f_i)_{i\in I}$ are MIP-definable.

One may note that the equations above specify the functions only for well balanced trees. Since we use this Lemma only for such trees, we do not care with the values for other inputs given by the proof below.

Proof. In an equation such as Equation (6), we call n_i the level of the definition of f_i. The proof is by induction on the maximal level of the functions $N = \max_{i\in I} n_i$. If $N = 1$, then the equations correspond to usual MIP-equations.

Suppose now $N > 1$. For all the indices i such that f_i has level N, we replace its definitional equations by:

$$\begin{aligned}
f_i(t_0 \star t_1, \vec{u}) &= f_{i\bullet 0}(t_0, \vec{u}) \star f_{i\bullet 1}(t_1, \vec{u}) \\
f_{i\bullet w}(t_0 \star t_1, \vec{u}) &= f_{i\bullet w0}(t_0, \vec{u}) \star f_{i\bullet w1}(t_1, \vec{u}), && (1 < |w| < N-1) \\
f_{i\bullet w}(t_0 \star t_1, \vec{u}) &= f_{\mathrm{p}(i,w0)}(t_0, \vec{\sigma}_{i,w0}(\vec{u})) \star f_{\mathrm{p}(i,w1)}(t_1, \vec{\sigma}_{i,w1}(\vec{u})), && (|w| = N-1) \\
f_{i\bullet w}(c, \vec{u}) &= g_{i,w,c}(\vec{u}), && (1 \leq |w| < N) \\
f_i(c, \vec{u}) &= g_{i,\epsilon,c}(\vec{u})
\end{aligned}$$

where the indices $i \bullet w$ are fresh. One may observe that the level of each of these functions is 1. We end by induction. □

The following Lemma is easy to verify:

Lemma 15. Suppose that $f \in (f_i)_{i\in I}$ is defined by MIP-recursion. Then, any function $g(t, \vec{u}) = f(t, \vec{\sigma}(t, \vec{u}))$ where the $\vec{\sigma}$ are generalized destructors can be defined by MIP-recursion.

2.2. Explicit Structural Recursion

Functions defined by MIP are non-size increasing. The growth rate of basic functions is an affine function of their inputs. Even functions defined by time iteration stick to the growth rate of their basic case. The ESR-schema serves to get bigger growth rates. In particular, it will be used to construct trees of height $O(\log n)$, corresponding to polynomial size growth rate, see the following Lemma. When characterizing implicitly small classes of complexity, one prevents the step function, h, to be itself defined by recursion of its critical arguments. This may be achieved by imposing some tiering discipline. Here, functions are explicitly defined, that is do not involve any kind of recursion.

Definition 16. *Explicit structural recursion* (ESR) is the following scheme:

$$\begin{aligned}
f(t_0 \star t_1, \vec{u}) &= h(f(t_0, \vec{u}), f(t_1, \vec{u})) \\
f(c, \vec{u}) &= g(c, \vec{u}) && c \in \Sigma
\end{aligned}$$

where h and g are explicitly defined.

Example 17. `left_most`(t) which outputs the leftmost leaf of a tree t is defined by ESR:

$$\begin{aligned}
\texttt{left_most}(t_0 \star t_1) &= \pi_1(\texttt{left_most}(t_0), \texttt{left_most}(t_1)) \\
\texttt{left_most}(c) &= c \\
\pi_1(x, y) &= x
\end{aligned}$$

Lemma 18. Given two natural numbers α_0 and α_1, there is a function f defined by ESR such that for any tree t, $\mathsf{H}(f(t)) = \alpha_1\mathsf{H}(t) + \alpha_0$. Moreover, given $c \in \Sigma$, one may suppose that all leaves of $f(t)$ are equal to c.

Proof. The proof is immediate, taking f defined by explicit structural recursion with $h = h_{\alpha_1}$ and $g(x) = h_{\alpha_0}(c, c)$ where $h_1(w_0, w_1) = w_0 \star w_1$ and $h_i(w_0, w_1) = h_{i-1}(w_0, w_1) \star h_{i-1}(w_0, w_1)$ for $i > 1$. □

2.3. Time iteration

The following scheme allows us to iterate MIP-definable functions. It serves to capture the time aspect of functions definable in NC^k. The scheme depends on the parameter k used for the stratification.

Definition 19. Given $k \geq 1$, a function f is defined by *k-time iteration* (k-TI) from the function h which is MIP-definable and the function g if:

$$\begin{aligned}
f(t_1' \star t_1'', t_2, \ldots, t_k, s, \vec{u}) &= h(f(t_1', t_2, \ldots, t_k, s, \vec{u}), \vec{u}) \\
f(c_1, t_2' \star t_2'', t_3, \ldots, t_k, s, \vec{u}) &= f(s, t_2', t_3 \ldots, t_k, s, \vec{u}) \\
&\vdots \\
f(c_1, \ldots, c_{i-1}, t_i' \star t_i'', t_{i+1}, \ldots, t_k, s, \vec{u}) &= f(c_1, \ldots, c_{i-2}, s, t_i', t_{i+1}, \ldots, t_k, s, \vec{u}) \\
&\vdots \\
f(c_1, \ldots, c_k, s, \vec{u}) &= g(s, \vec{u})
\end{aligned}$$

where $c_1, \ldots, c_k \in \Sigma$.

Notice that if (k-TI) would allow the function h to be, for instance, $\star$ then, by the following lemma, we would obviously violate the space constraint of the classes NC^k. Informally, (k-TI) enables us to iterate $O(\log^k n)$ times functions which do not increase the space needs; as remarked above, MIP-definable functions are such ones.

Lemma 20. Given a MIP-definable function h, a function g and constants β_1 and β_0, there is a function f defined by k-TI such that for all perfectly balanced trees t

$$f(t, \vec{u}) = \underbrace{h(\ldots h}_{\beta_1(\mathsf{H}(t))^k + \beta_0 \text{ times}}(g(t, \vec{u}), \vec{u})\ldots).$$

Proof. The proof follows the lines of Lemma 18. □

3. Rational Bitwise Equations

In this section, we describe Rational Bitwise Equations. Two features are noticeable. First, RBE-equations do not involve the tree structure seen for MIP-recursion: the functions are directly defined on words, avoiding the encoding. Second, function definitions do not rely on a recursive schema on data, but on a (as weak as transducers are) schema on pointers.

Given a non empty word $w = a_0 \dots a_n \in \Sigma^*$, consider some word $p \in \{0,1\}^*$. We define

$$w[p] = \begin{cases} a_0 & \text{if } p = \epsilon \text{ and } w = a_0 \\ a_0 \cdots a_{\lfloor n/2 \rfloor}[q] & \text{if } p = 0 \cdot q \text{ and } n \geq 1 \\ a_{\lfloor n/2 \rfloor + 1} \cdots a_n[q] & \text{if } p = 1 \cdot q \text{ and } n \geq 1 \\ \bot & \text{otherwise.} \end{cases}$$

In other words, $w[p]$ denotes the p-th bit of w. The representation is compatible with MIP's tree encoding: $w[p] = \mathsf{d}_p(\widehat{w})$ for all path of size k and words w of size 2^k for some $k \geq 0$. We define $\Pi(w)$ to be the set of valid addresses, that is $\Pi(w) = \{p \in \{0,1\}^* \mid w[p] \neq \bot\}$.

Definition 21 (Rational Bitwise Equation)**.** A function $f : \mathbb{W}^k \to \mathbb{W}$ is computed by rational bitwise equations iff for all $w_1, \dots, w_k \in \mathbb{W}$, the word $w = f(w_1, \dots, w_k)$ verifies for all path $p \in \Pi(w)$,

$$w[p] = h(\phi_0(p), w_{e_1}[\phi_1(p)], \dots, w_{e_m}[\phi_m(p)]). \tag{8}$$

with

- $e_1, \dots e_m \leq k$,
- $\phi_1, \dots, \phi_m$ some rational functions on $\{0,1\}^*$,
- ϕ_0 is a rational function ranging in a finite domain Q.
- and h a finite map $Q \times (\Sigma \cup \{\bot\})^m \to \Sigma$.

Example 22. Restricted to vectors (that is words) of equal length, the bitwise disjunction and the bitwise conjunction are bitwise rational. Indeed, the functions $\underline{0} : p \mapsto 0$ and $Id : p \mapsto p$ are rational. And one may observe that the bitwise disjunction is defined by RBE:

$$\mathsf{or}(w_1, w_2)[p] = h(\underline{0}(p), w_1[Id(p)], w_2[Id(p)])$$

with $h : (x, a, b) \mapsto a \vee b$ and $p \in \Pi(w_1)$.

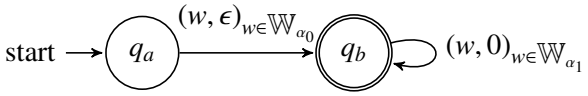
In the definition above, the size of w is not specified. Nevertheless, it is not arbitrary. Indeed, let us suppose for the discussion, that the functions ϕ_i verify $|\phi_i(p)| \geq A \times (|p| + 1)$ for some $A > 0$ (see[5]), then, there is a minimal size $\ell = O(\log(\max\{|w_i| \mid i \leq k\}))$ such that for all[6] path $|p| \geq \ell$, $|\phi_i(p)| > \mathsf{H}(w_{e_i})$, which in turns means that $w_{e_i}[\phi_i(p)] = \perp$. Then, the byte corresponding to path p of the output does not depend anymore on the inputs. It is then natural to define the size of the output to be $2^{\ell-1} = \max\{|w_i| \mid i \leq k\}^{O(1)}$. That is outputs have polynomial size.

Nevertheless, in what follows, in order to avoid inextricable technicalities about size, we will restrict transducers to be (A, B)-length preserving. A transducer is said to be (A, B)-length preserving whenever $|\phi(p)| = \frac{1}{A}(|p| - B)$ for all path p and some $A > 0$, $B \in \mathbb{Z}$. In other words, $|p| = A \times |\phi(p)| + B$, which in turns means that the height of outputs of the RBE-induced function is $A \times \max\{|w_i| \mid i \leq k\}$, which finally, means that outputs have size $B \times \max\{|w_i| \mid i \leq k\}^A$.

Definition 23. A function computed by RBE is said to be (A,B)-length preserving for some $A, B > 0$ iff the underlying transducers of functions $\phi_1, \ldots, \phi_k$ are themselves (A, B)-length preserving. It is said to be length-preserving if $A = B = 1$.

For a length-preserving function f, we have $|f(w_1, \ldots, w_k)| = \max\{|w_i| \mid i \leq k\}$. To end this discussion, consider the function as defined by Lemma 18, it is computed by a α_0, α_1-length preserving RBE function.

Example 24. Let the transducer $\mathcal{M}$:



Let $h(x, y) = c$ for some arbitrary letter c. Consider the function defined by RBE from $(Id, \mathcal{M}, h)$. Suppose $[\mathcal{M}](p) = a^n$ denote some byte of some input word w. Then, its antecedent is a path $p \in \mathbb{W}_{\alpha_0+\alpha_1\times n}$. Thus, for all input w and all path $|p| = \alpha_0 + \alpha_1\mathsf{H}(w)$, its value is $f(w)[p] = c$. That is $f(w) = c^{\alpha_1|w|+\alpha_0}$.

Theorem 25. *Length-preserving functions computed by RBE are exactly functions computable by* MIP.

[5]Suppose now the contrary, there are words u, v, w such that for all $k \geq 0$: $\phi_i(u \cdot v^k \cdot w) = \phi_i(u \cdot w)$. And $e_i[\phi_i(u \cdot v^k \cdot w)] = \perp$ for all inputs $\mathsf{H}(w_i) \geq |\phi_i(u \cdot w)|$. In that case, the output on bytes indexed $u \cdot v^k \cdot w$ does not depend on the input for sufficiently large inputs. It is reasonable to consider ϕ_i as pathological.

[6]An other possibility is to use an existential quantifier: for some path, etc, etc. The conclusion is the same.

Proof. The theorem is a direct consequence of Proposition 28 and Proposition 30 respectively proved in sections 3.2 and 3.3. □

3.1. Strict ramified recurrence at work: characterizing the $(\mathrm{NC}^k)_{k\geq 1}$

According to Theorem 25, MIP and RBE have the same computational power; it is then natural to extend RBE to all the layers of the hierarchy $\mathrm{NC}^k, k \geq 1$ as we did for MIP. In the case of INC^k, one of the key features of time iteration is that the height of a tree amounts to the logarithm of the size of inputs, thus turning structural recursion into the expected logarithm. In the absence of trees, we introduce the length function $\mathrm{len} : \mathbb{W} \to \mathbb{N}$ on which time iteration is performed.

To get each layer of the hierarchy, we need to iterate $O(\log(n)^k)$-times the basic schema. To do that, we essentially follows the lines of Marion's strict ramification method as described in [Mar09].

Computations involve two sorts, pointers $\mathbb{P}$ and datas which is the set $\mathbb{W}$ of words on some alphabet Σ. Pointers are represented by words over the alphabet $\{A, B\}$ supposed to be distinct from Σ. Recursion is done only on pointers and to control it, we introduce a tiering mechanism. That is we suppose to have a copy of $\mathbb{P}$ for each $n \in \mathbb{N}$, next denoted $\mathbb{P}_n$, composed of letters A_n, B_n and empty word ϵ_n.

On data, the set of basic word functions $\mathcal{B}_{\mathsf{RBE}}$ is composed of the following set. First, any (C,D)-length-preserving function defined by RBE is in $\mathcal{B}_{\mathsf{RBE}}$. Second, for each $n \geq 0$, set $\mathrm{len}_n : \mathbb{W} \to \mathbb{P}_n$ to be the function which maps a word to the binary representation of its length in $\mathbb{P}_n$ is in $\mathcal{B}_{\mathsf{RBE}}$. Observe that $|\mathrm{len}_n(w)| = \log(|w|)$.

On pointers,we say that a function $f : \mathbb{P}_k \times \mathbb{P}_{i_1} \cdots \times \mathbb{P}_{i_n} \times \mathbb{W}^m \to \mathbb{W}$ is defined by blind[7] strictly ramified recursion of rank k from the functions $h : \mathbb{P}_{i_1} \times \cdots \times \mathbb{P}_{i_n} \times \mathbb{W}^m \to \mathbb{W}$ and $g : \mathbb{P}_{i_1} \times \cdots \times \mathbb{P}_{i_n} \times \mathbb{W}^{m+1} \to \mathbb{W}$ if:

$$\begin{aligned} f(\epsilon_k, \vec{q}, \vec{w}) &= h(\vec{q}, \vec{w}) \\ f(A_k(p), \vec{q}, \vec{w}) &= g(\vec{q}, f(p, \vec{q}, \vec{w}), \vec{w}) \\ f(B_k(p), \vec{q}, \vec{w}) &= g(\vec{q}, f(p, \vec{q}, \vec{w}), \vec{w}) \end{aligned}$$

and $k > i_1 > \ldots > i_n$. In case $\vec{q}$ is empty, g is supposed to be length preserving.

Definition 26. Let RBE^k be the closure of basic word functions in $\mathcal{B}_{\mathsf{RBE}}$ by (typed) composition and i-recursion (with $i \leq k$) typed $\mathbb{W}^k \to \mathbb{W}$.

Theorem 27. *Functions in* RBE^k *are exactly functions in* NC^k *for* $k > 0$.

The theorem is proved in Section 6.

[7] In the schema, we do not make the distinction between A's and B's.

3.2. From MIP *to RBE*

Proposition 28. *Any function defined by* MIP*-recursion is defined by a (length-preserving) function in RBE.*

Proof. Consider a function defined by MIP. That is there is a family $(f_i)_{i\in I}$ with equations:

$$f_i(t_0 \star t_1, \vec{u}) = f_j(t_0, \vec{\sigma}_{i,0}(t_0 \star t_1, \vec{u})) \star f_l(t_1, \vec{\sigma}_{i,1}(t_0 \star t_1, \vec{u})) \tag{9}$$

$$f_i(c, \vec{u}) = g_i(c, \vec{u}) \tag{10}$$

Adding dummy arguments to functions, we can suppose without loss of generality that all functions have a common arity, say $k+1$. Moreover, without loss of generality, we can suppose that $I = \{1, \dots, m\}$ and that f_1 is the function we want to define by RBE. Finally, as in Proposition 12, the functions $g_{i,c}$ are supposed to be in `if-then-else` form.

Let us introduce notations. Let $r(i,b)$ be the index of the b-th sub call of f_i, $e(i,b,j)$ and $\alpha(i,b,j)$ be such that Equation 9 is read:

$$f_i(t_0, t_1, \dots, t_k) = f_{r(i,0)}(\mathsf{d}_0(t_0), \mathsf{d}_{\alpha(i,0,1)}(t_{e(i,0,1)}), \dots, \mathsf{d}_{\alpha(i,0,k)}(t_{e(i,0,k)})) \star \\ f_{r(i,1)}(\mathsf{d}_1(t_0), \mathsf{d}_{\alpha(i,1,1)}(t_{e(i,1,1)}), \dots, \mathsf{d}_{\alpha(i,1,k)}(t_{e(i,1,k)}))$$

In what follows, we consider the computation of $f_1(\widehat{w_0}, \dots, \widehat{w_k})$ for some words $w_0, \dots, w_k$. Let $t_i = \widehat{w_i}$ for all $i \le k$. For all path p in t_0, in the computation of $f_1(t_0, \dots, t_k)$, there is a recursive sub-call $f_j(\mathsf{d}_p(t_0), u_1, \dots, u_k)$ for some $j \in I$ and the u_i's some sub-trees of the t_i's. With the notation above, let $\iota(p) \in I$ be the index j of the function f_j and $\pi_\ell(p), \xi_\ell(p)$ be such that $u_\ell = \mathsf{d}_{\pi_\ell(p)}(t_{\xi_\ell(p)})$ with $1 \le \ell \le k$. One may observe that $\iota(\epsilon) = 1$, $\xi_\ell(\epsilon) = \ell$ and $\pi_\ell(\epsilon) = \epsilon$ with $1 \le \ell \le k$.

Our simulation splits in two parts. First, the rational function ϕ_0 will serve to compute the function ι. Second, the other rational functions will compute the functions π_ℓ for $\ell \le k$. The *state transducer* is $S = \langle 0 \cup I \times I, \{0,1\}, \sigma, \{0\}, F\rangle$ with $F = \{(j,j) \mid j \in I\}$ and σ:

$$\begin{aligned} 0 &\xrightarrow{(\epsilon,\ell)}_\sigma (1,\ell) && \text{for all } \ell \in I \\ (j,\ell) &\xrightarrow{(b,\epsilon)}_\sigma (r(j,b),\ell) && \text{for all } j \in I, b \in \{0,1\}, \ell \in I \end{aligned}$$

Some observations. First, S is functional. Second, $0 \xrightarrow{(p,q)}_{\sigma^*} (j,\ell) \Rightarrow q = \ell$ and third, it verifies

$$[S](p) = \ell \text{ iff } 0 \xrightarrow{(p,\ell)}_{\sigma^*} (\ell,\ell) \tag{11}$$

Fourth, with the preceding notations, the sub-call corresponding to some word p verifies

$$\iota(p) = [S\,](p). \tag{12}$$

That is $[S\,]$ computes the function ι. It is proved by induction on the length of p. For $p = \epsilon$, by Equation 11, $[S\,](\epsilon) = 1 = \iota(\epsilon)$. Otherwise, $p = q \cdot b$. Let us run the state transducer: $0 \xrightarrow{q,\ell}_{\sigma^*} (j,\ell) \xrightarrow{b,\epsilon}_{\sigma} (r(j,b),\ell)$. Then, $\iota(q \cdot b) = r(\iota(q),b) = r(j,b)$, the first equality is by definition of ι, the second equality is by Induction Hypothesis together with Equation 11.

To compute the functions π_ℓ, we introduce the following family of transducers. Given $0 \le e \le k$, let the transducer $T_e = \langle I \times \{0,\dots,k\}, \{0,1\}, \delta, \{(1,e)\}, \emptyset\rangle$ with δ defined by:

$$(j,d) \quad \xrightarrow{(b,\alpha(j,b,d'))}_{\delta} (r(j,b),d') \quad \text{for all } d' \text{ s.t. } e(j,b,d') = d$$

By induction on the length of the paths, one proves (A) that for all path p, if $(1,e) \xrightarrow{(p,q)}_{\delta^*} (j,d)$, then $j = \iota(p)$.

And (B), $(1,e) \xrightarrow{(p,q)}_{\delta^*} (j,d)$ iff the d-th argument of $f_{\iota(p)}(\mathsf{d}_p(t_0),u_1,\dots,u_k)$ is $\mathsf{d}_q(t_e)$. In other words,

$$u_d = \mathsf{d}_{\pi_d(p)}(t_{\xi_d(p)}) = \mathsf{d}_q(t_e). \tag{13}$$

This is proved by induction on the length of p. The case $p = \epsilon$ is degenerate. Indeed, it corresponds to $(1,e) \xrightarrow{(\epsilon,\epsilon)}_{\delta^*} (1,e)$ and $\pi_e(\epsilon) = \epsilon$ and $\xi_e(\epsilon) = e$. Otherwise, suppose $(1,e) \xrightarrow{(p,q)}_{\delta^*} (j,d)$. By observation (A), the sub-call corresponding to the path p refers to the equation of f_j: $f_j(\vec{u}) = f_{r(j,0)}(u_0^0,\dots,u_0^k) \star f_{r(j,1)}(u_1^0,\dots,u_1^k)$ with $u_b^\ell = \mathsf{d}_{\alpha(j,b,\ell)}(u_{e(j,b,\ell)})$ for all $b \in \{0,1\}$ and for all $\ell \le k$. Given the derivation $(1,e) \xrightarrow{(p,q)} (j,d) \xrightarrow{(b,\alpha(j,b,d'))} (r(j,b),d')$ with $b \in \{0,1\}$ and d' such that $d = e(j,b,d')$, let us consider:

$$\begin{array}{rcll} u_b^{d'} & = & \mathsf{d}_{\alpha(j,b,d')}(u_{e(j,b,d')}) & \text{by definition of } \alpha \text{ and } e \\ & = & \mathsf{d}_{\alpha(j,b,d')}(u_d) & \text{since } d = e(j,b,d') \\ & = & \mathsf{d}_{\alpha(j,b,d')}(\mathsf{d}_p(t_e)) & \text{by Induction Hypothesis} \\ & = & \mathsf{d}_{q\cdot\alpha(j,b,d')}(t_e). & \end{array}$$

Let $T_{e,j,d}$ be the transducer T_e but with a unique terminal state (j,d). With Equation 13, it is clear that $[T_{e,j,d}](p)$ is defined iff the d-th argument of the call corresponding to the path p is $\mathsf{d}_{[T_{e,j,d}](p)}(t_e)$.

Let h be the function with $1 + k \times m \times k$ arguments next denoted by variable names $x_0, (x_{e,j,f})_{(e,j,f)\in\{1,\dots,k\}\times I\times\{1,\dots,k\}}$, defined as follows:

$$
\begin{array}{rll}
h(\vec{x}) & = & \\
 & \text{if} & (x_0 = 1) \wedge x_{1,1,1} \neq \bot \wedge \cdots \wedge x_{1,1,k} \neq \bot \text{ then } g_1(x_{1,1,1}, \dots, x_{1,1,k}) \\
 & \text{else} & \dots \\
 & \text{else if} & (x_0 = j) \wedge x_{e_1,j,1} \neq \bot \wedge \cdots \wedge x_{e_k,j,k} \neq \bot \text{ then } g_j(x_{e_1,j,1}, \dots, x_{e_k,j,k}) \\
 & \text{else} & \bot
\end{array}
$$

with j ranging in I and the e_i's ranging in $\{1, \dots, k\}$.

Since the g_i's are in `if-then-else` form, the function h is itself defined on a finite domain. With preceding definitions and observations, it is clear that setting $x_0 = [S](p)$, $x_{e,j,d} = \widehat{w_e}[[T_{e,j,d}](p)]$, then $h(\vec{x})$ outputs the p-th bit of the input. Indeed, consider the "succeeding line" in the definition of h defined by $j, e_1, \dots, e_k$. By Equation 12, $j = \iota(p)$ and we fire g_j. Second, for all $\ell \leq k$, the ℓ-th argument of $f_j(\mathsf{d}_p(t_0), \dots)$ is $\mathsf{d}_{[T_{e_\ell,j,\ell}](p)}(t_{e_\ell}) = \mathsf{d}_{\pi_\ell(p)}(t_{\xi_\ell(p)})$. Thus g_j is fired with the expected arguments. □

3.3. From RBE to MIP

In the other direction, the main difficulty comes from the fact that transducers are not deterministic. Otherwise, the solution is immediate. Indeed, consider an RBE equation of some (length-preserving) function f:

$$w[p] = h(\phi_0(p), w_{e_1}[\phi_1(p)], \dots, w_{e_m}[\phi_m(p)])$$

and suppose that $\phi_0, \dots, \phi_m$ are functions computed by deterministic transducers $\mathcal{M}_i = \langle Q_i, \{0,1\}, \delta_i, q_{0,i}, F_i\rangle$, $i \leq m$. By deterministic[8], we mean more precisely that δ_i is a function $Q_i \times \{0,1\} \to \{0,1\}^* \times Q_i$. Furthermore, for the sake of the argument, suppose that $[\phi_0](p)$ is the last state of the run of the transducer on p, that is $[\phi_0](p) = q$ with $(w, q) = \delta_0^*(q_{0,0}, p)$. Then, set $I = Q_0 \times \cdots \times Q_m$ and define

$$
\begin{array}{rcl}
f_{(q_0,\dots,q_m)}(t_0 \star t_1, u_1, \dots, u_m) & = & \\
\quad f_{(q'_0,\dots,q'_m)}(t_0, \mathsf{d}_{p'_1}(u_1), \dots, \mathsf{d}_{p'_m}(u_m)) \star f_{(q''_0,\dots,q''_m)}(t_1, \mathsf{d}_{p''_1}(u_1), \dots, \mathsf{d}_{p''_m}(u_m)) & & \\
f_{(q_0,\dots,q_m)}(c, u_1, \dots, u_m) & = & h(q_0, u_1, \dots, u_m)
\end{array}
$$

[8] We do not want to raise a terminology issue here, the word seems appropriate in the present context.

with $(p'_i, q'_i) = \delta_i(q_i, 0)$ and $(p''_i, q''_i) = \delta(q_i, 1)$, $i \leq m$ and h is considered to be undefined if one of the input is not in Σ. Then, for all $w_1, \ldots, w_m$ $f(w_1, \ldots, w_m) = f_{q_{0,0},\ldots,q_{0,m}}(\widehat{w_{e_1}}, \widehat{w_{e_1}}, \ldots, \widehat{w_{e_m}})$. Thus, f is in MIP.

However, the simulation only works if the δ_i's are deterministic. It is hard skip the issue, since a) the proof of the preceding section relies on non-deterministic transducers, and b) some rational functions are computed by inherently non-deterministic functions. E.g., the function:

$$a_1 \cdots a_n \mapsto \begin{cases} 0^n & \text{if } \forall i \leq n : a_i \neq 0 \\ 1^n & \text{otherwise.} \end{cases}$$

To tame the non-determinism, we use the matrix representation of transducers, thanks to the decomposition of rational functions due to Elgot and Mezei, as described in Section 1.3. We treat apart the transducer $\mathcal{M}_0$, for which the issue can be solved easily as we will see now.

Proposition 29. *Given a transducer $\mathcal{M} = \langle Q, \Sigma, E, I, T\rangle$ computing a function f whose range is the finite set X. There is a deterministic automaton $\mathcal{M}' = \langle R, \Sigma, \delta, i, F\rangle$ and a finite partial map $\Omega : R \to X$ such that for all words $w \in \Sigma^*$, $[\mathcal{M}](w) = x$ if and only if $w \in [\mathcal{M}']$ and moreover for the unique state $i \xrightarrow{w}_{\delta^*} q$, $\Omega(q) = x$.*

Proof. Let L be the maximal size of $x \in X$. We define the automaton $\mathcal{M}'' = \langle \{0\} + Q \times \Sigma^{\leq L}, \Sigma, \delta', 0, F'\rangle$ with $F' = T \times \Sigma^{\leq L}$, and δ' as follows. For all $q_i \in I$, set $0 \xrightarrow{\epsilon} (q_i, \epsilon)$. Suppose that $q \xrightarrow{a,u}_E q'$. If $|x \cdot u| \leq L$, then set $((q, x), a, (q', x \cdot u)) \in \delta'$. Otherwise, the transition is sent to a trash state. From the definition of $\mathcal{M}$, it is clear that succeeding runs of $\mathcal{M}''$ correspond to succeeding runs of $\mathcal{M}$. This automaton is not deterministic, let us consider its determinization by the powerset method restricted to reachable states. Any final state $\mathfrak{F}$ looks like $\{(q_1, x_1), \ldots, (q_n, x_n)\}$. For such a state, notice that there is at most one $x \in X$ such that $x_i = x$, $q_i \in F$. Indeed, wouldn't it be the case, let (q_j, x_j) be an other state with $x_j \neq x_i$ and q_j a final state. Since (q_i, x_i) and (q_j, x_j) are in the same (reachable) powerset state, there is a word w such that $q_0 \xrightarrow{w,x_i}_{E^*} q_i$ and $q_0 \xrightarrow{w,x_j}_{E^*} q_j$ with q_i and q_j some final states. This would imply that f is not functional. Accordingly, we define $\Omega(\mathfrak{F})$ to be the unique such x. If no such x exists (in particular, if $\mathfrak{F}$ is not a final state), let $\Omega(\mathfrak{F}) = \bot$. □

Proposition 30. *Any length-preserving function defined by RBE is computable by a* MIP*-function.*

Proof. Let $f : \mathbb{W}^k \to \mathbb{W}$ be an RBE function defined by equation:

$$\forall p \in \Pi(w) : w[p] = h(\phi_0(p), w_{e_1}[\phi_1(p)], \ldots, w_{e_m}[\phi_m(p)]).$$

with $e_1, \dots e_m \leq k$, h a finite map and $\phi_1, \dots, \phi_m$ some rational functions on $\{0,1\}^*$ respectively computed by the transducer be $\mathcal{M}_0, \dots, \mathcal{M}_m$. Let X be the finite co-domain of ϕ_0.

Applying Proposition 29, let $\mathcal{A}_0 = \langle R, \delta_0, i_\star, F\rangle$ be the deterministic automaton and $\Omega : R \to X$ be the function corresponding to $\mathcal{M}_0$. Thus, for all path $p \in \{0,1\}^*$, $h([\mathcal{M}_0](p), \vec{y}) = h(\Omega(\delta_0(i_\star, p)), \vec{y})$. In the sequel, the computation of ϕ_0 is done via the deterministic automaton $\mathcal{A}_0$.

For each transducer $\mathcal{M}_i$, $1 \leq i \leq m$, let $(\lambda_i, \mu_i, \nu_i)$ of dimension Q_i and $(\eta_i, \kappa_i, \chi_i)$ of dimension R_i be the triples corresponding to the decomposition of $\mathcal{M}_i$ into a sequential transducer and a co-sequential transducer. And let $(\zeta_i, \pi_i, \omega_i)$ be the composition of $(\lambda_i, \mu_i, \nu_i)$ and $(\eta_i, \kappa_i, \chi_i)$ as defined in Section 1.3. Consider a path $p = p_1 \cdots p_k \in \{0,1\}^*$. Computing $\phi_i(p)$ remains to compute $\zeta_i \times \pi_i(p) \times \omega_i = \zeta_i \times \pi_i(p_1) \times \cdots \times \pi_i(p_k) \times \omega_i$. Let

$$\rho_i(p) = \zeta_i \times \pi_i(p) = \zeta_i \times \pi_i(p_1) \times \cdots \times \pi_i(p_k).$$

By induction on the length of p, one sees that the vector $\rho_i(p) \in \{0,1\}^{*\,Q_i \times R_i}$ has at most one index $q_i \in Q_i$ such that $\rho_i[(q_i, r_i)]$ is non zero for some $r_i \in R_i$ (Note A). In other words, to simulate the matrix product, it is sufficient to keep track of the index q_i and the entries $\rho_i(p)[(q_i, r_{i,1})], \dots, \rho_i(p)[(q_i, r_{i_{k_i}})]$, that is an R_i-indexed sequence of paths. But now, observe that the step from $\rho_i(p)$ to $\rho_i(p.b)$ is deterministic– actually encoded in the π_i's matrices. The rest of the section implements this process with MIP.

We set $I = \{0\} \cup R \times Q_1 \times \cdots \times Q_m$. The functions f_0 has arity k serves for parameter initializations. All other functions have arity $1 + |R_1| + \cdots + |R_m|$. The intention is that the state(s) $(q_0, \dots, q_m)$ memorize states with non-zero components and each parameter corresponds to the entries in vectors ρ_i. Let $R_j = \{r_{j,1}, \dots, r_{j,k_j}\}$ for all $j \geq 1$. For all $\vec{q} \in R \times Q_1 \times \cdots \times Q_m$, for all $r_{j,\ell} \in R_j$, $r_{j,\ell}$ denotes $1 + |R_1| + \cdots |R_{j-1}| + \ell$. In other words, in $f_{\vec{q}}(\vec{x})$, think of $\vec{x}$ as $x_0, x_{r_{1,1}}, \dots, x_{r_{m,k_m}}$.

Let us say some few more words on the simulation of f. Our intention is that $f = f_0$. Let us motivate the other functions. Consider the computation of $f_0(t_0, \dots, t_{k-1})$, the sub-call corresponding to the path p will have the shape $f_{q_0,\dots,q_m}(\mathsf{d}_p(t_0), \vec{u})$ such that: $q_0 = \delta_0(i_\star, p)$ and for $i \geq 1$, q_i is precisely the unique index q_i mentioned above (Note A); and, finally, $u_{r_{i,j}} = \mathsf{d}_{\rho_i[(q_i, r_{i,j})]}(t_{e_i})$. Thus, at each steps of the computation, the variable indexed $r_{i,j}$ points to the sub-tree corresponding to the path stored in $\rho_i[(q_i, r_{i,j})]$. At the end, it is sufficient to multiply ρ_i by ω_i to get the unique sub-tree corresponding to the i-th input variable. This is done in Equation 14. Let us define the base functions:

$$g_0(\vec{x}) \quad = \quad h(\Omega(i_\star), \mathsf{d}_{[\mathcal{M}_1](\epsilon)}(x_{e_1}), \dots, \mathsf{d}_{[\mathcal{M}_m](\epsilon)}(x_{e_m}))$$

which is a degenerate case, and for all $(q_0, \ldots, q_m) \in R \times Q_1 \times \cdots \times Q_m$:

$$g_{(q_0,\ldots,q_m)}(\vec{x}) \quad = \quad h(\Omega(q_0), z_1, \ldots, z_m) \tag{14}$$

with $z_1, \ldots, z_m$ defined as follows. For all $i \geq 1$, let $t_i \in R_i$ be the unique—if it exists—index such that $\chi_i[t_i]$ is not zero. If such a t_i exists (actually, when the i-th transducer succeeds on the current input path), we define $z_i = \mathsf{d}_{\omega_i[(q_i,t_i)]}(x_{t_i})$. Otherwise, $z_i = \perp$. Remark that the definition of $g_{(q_0,\ldots,q_m)}$ is by finite case analysis on the matrices (and does not depend on the inputs!), thus the function is atomic. To conclude, we define:

$$f_i(c, \vec{u}) \quad = \quad g_i(c, \vec{u}).$$

with $i \in I$. We come now to the recursive part of the definition of the functions f_i, $i \in I$. For the index 0,

$$f_0(t_0 \star t'_0, t_2, \ldots, t_m) = f_{(q_0^0,\ldots,q_m^0)}(t_0, u^0_{r_{1,1}}, \ldots, u^0_{r_{m,k_m}}) \star f_{(q_0^1,\ldots,q_m^1)}(t'_0, u^1_{r_{1,1}}, \ldots, u^1_{r_{m,k_m}})$$

where $q_0^b = \delta_0(i_\star, b)$ and for all $i \leq m$, q_i^b is the unique index in Q_i such that $\rho_i(b)[(q_i^b, r)] \neq 0$ for some $r \in R_i$. For the u_i's definition, take the convention that $t_1 = t_0 \star t'_0$ and set $u^b_{r_{i,j}} = \mathsf{d}_{\rho_i(b)[(q_i^b, r_{i,j}))]}(t_{e_i})$. For the other indices, let:

$$f_{\vec{q}}(t_0 \star t'_0, \vec{t}) \quad = \quad f_{(q_0^0,\ldots,q_m^0)}(t_0, u^0_{r_{1,1}}, \ldots, u^0_{r_{m,k_m}}) \star f_{(q_0^1,\ldots,q_m^1)}(t'_0, u^1_{r_{1,1}}, \ldots, u^1_{r_{m,k_m}})$$

where $q_0^b = \delta_0(q_0, b)$. For the other indices, consider the vector ρ_i such that $\rho_i[(q, r_{i,j})] = 0$ if $q \neq q_i$ and ϵ otherwise. Let $\rho'_i = \rho_i \times \pi_i(b)$. This vector has a unique component q such that $\rho'_i[(q, r)] \neq 0$ for some $r \in R_i$. We define q_i^b to be this index. We define $u^b_{r_{i,j}} = \mathsf{d}_{\rho'_i[(q_i^b, r_{i,j})]}(t_{r_{i,j}})$.

From the definition, each step of the computation updates the matrix multiplication as specified above: by induction on the path p, one proves that in the computation of $f_0(t_0, \ldots, t_{k-1})$, the sub-call $f_{\vec{q}}(\mathsf{d}_p(t_0), \vec{u})$ verifies the two properties: a) $q_0 = \delta_0(i_\star, p)$, b) for all $i > 0$, for all $r_{i,j}$, $u_{r_{i,j}} = \mathsf{d}_{\rho_i(p)[q_i, r_{i,j}]}(t_{e_i})$. Thus, $f = f_0$. □

4. Simulation of alternating Turing machines

We introduce alternating random access Turing machines (ARMs) as described in [Lei98] by Leivant, see also [CKS81, Ruz81]. An ARM $M = (Q, q_0, \delta)$ consists of a (finite) set of states Q, one of these, q_0, being the initial state and actions δ to be described now. States are classified as disjunctive or conjunctive, those are called action states, or as accepting, rejecting and reading states. The operational

semantics of an ARM, M, is a two stage process: firstly, generating a computation tree; secondly, evaluating that computation tree for the given input. A configuration $K = (q, w_1, w_2)$ consists of a state q and two work-stacks $w_i \in \mathbb{W}$, $i \in \{1, 2\}$. The initial configuration is given by the initial state q_0 of the machine and two empty stacks.

First, one constructs a computation tree, that is a tree whose nodes are configurations. The root of the computation tree is the initial configuration. Then **(Successor Rule)**, if the state of a node is an action state, depending on the state and on the bits at the top of the work-stacks, one spawns a pair of successor configurations obtained by pushing/popping letters on the work-stacks. The other states are some leaves. The t-time computation tree is the tree obtained by this process until height t.

Without loss of generality, we assume that for each action state q, one of the two successor configurations, let us say the first one, lets the stacks unchanged. And for the second successor configuration, either the first stack or the second one is modified, but not both simultaneously **(Stack Rule)**. We write accordingly the transition function δ for action states: $\delta(q, a, b) = (q', q'', \mathsf{pop}_i)$ with $i \in \{1, 2\}$ means that being in state q with top bits being a and b, the first successor configuration has state q' and stacks unchanged, and the second successor has state q'' and pops one letter on stack i. When we write $\delta(q, a, b) = (q', q'', \mathsf{push}_i(c))$, with $i \in \{1, 2\}$ and $c \in \{0, 1\}$, it is like above but we push the letter c on the top of the stack i.

The evaluation of a finite computation tree T is done as follows **(Evaluation Rule)**. Beginning from the leaves of T until its root, one labels each node (q, w_1, w_2) according to:

- if q is a rejecting (resp. accepting) state, then it is labeled by 0 (resp. 1);
- if q is a c, j-reading state ($c = 0, 1$, $j = 1, 2$), then it is labeled by 0 or 1 according to whether the n'th bit of the input is c, where n is the content read on the j'th stack. If n is too large, the label is $\perp$;
- if q is an action state,
  - if it has zero or one child, it is labeled $\perp$;
  - if it has two children, take the labels of its two children and compute the current label following the convention that $c = (c \vee \perp) = (\perp \vee c) = (c \wedge \perp) = (\perp \wedge c)$ with $c \in \{0, 1, \perp\}$.

The label of a computation tree is the label of the root of the computation tree thus obtained. That is it is the label of the initial configuration.

We say that the machine works in time $f(n)$ if, for all inputs, the $f(n)$-time tree evaluates to 0 or 1 where n is the size of the input. It works in space $s(n)$ if the size of the stacks of all configurations in the configuration tree are bounded by $s(n)$.

Actually, to relate our function algebra to the NC^k, we say that a function is in $\mathrm{ARM}(O(\log^k n, O(\log n))$, for $k \geq 1$ if it is polynomially bounded and bitwise computed by an ARM working in time $O(\log^k n)$ and space $O(\log n)$. Since the computational power of ARM and ATM (Alternating Turing Machine) are identical (see [Lei98]), as a result, we can restate Ruzzo's Theorem as follows:

Theorem 31 (Ruzzo [Ruz81])**.** NC^k *is exactly the set of languages recognized by ARM working in time* $O(\log(n)^k)$ *and space* $O(\log(n))$.

From that, one inclusion (from the right to the left) of our main theorem is a corollary of:

Proposition 32. *Given* $k \geq 1$ *and constants* $\alpha_1, \alpha_0, \beta_1, \beta_0$*, any* ARM *working in space* $\alpha_1 \log(n) + \alpha_0$ *and time* $\beta_1 \log^k(n) + \beta_0$*, where* n *is the length of the input, can be simulated in* INC^k.

Proof. We consider such a machine $M = (Q, q_0, \delta)$. Take $d = \lceil \log(|Q|) \rceil$. We attribute to each state in Q a word $w \in \mathbb{W}_d$ taking the convention that the initial state q_0 has encoding $0 \cdots 0$. From now on, the distinction between the state and its associated word is omitted.

Let us consider the encoding of two stacks $s_1 = a_1 a_2 \cdots a_i \in \mathbb{W}$ and $s_2 = b_1 b_2 \cdots b_j \in \mathbb{W}$ of length less or equal than $\alpha_1 \cdot \log(n) + \alpha_0$:

$$\mathsf{P}(s_1, s_2) = \mathsf{l}(a_1)\mathsf{l}(b_1)\mathsf{l}(a_2) \cdots \mathsf{l}(a_i)\mathsf{l}(\#)\mathsf{l}(b_2) \cdots \mathsf{l}(b_j)\mathsf{l}(\#)\mathsf{l}(\#) \cdots \mathsf{l}(\#)$$

where $\mathsf{l}(\mathtt{0}) = 10, \mathsf{l}(1) = 11$ and $\mathsf{l}(\#) = 00$, in such a way that this word has length exactly $2(\alpha_1 \cdot \log(n) + \alpha_0 + 1)$. The "+1" origins from the extra character # which separates the two (tails of the) stacks. For convenience we use a typewriter font for the encoding l. Then, the encoding of stacks above is written

$$\mathsf{P}(s_1, s_2) = \mathtt{a_1\, b_1\, a_2\, a_3} \cdots \mathtt{a_i} \,\#\, \mathtt{b_2\, b_3} \cdots \mathtt{b_j} \,\#\,\# \cdots \#.$$

To perform the computations for some input of size n, we use a *configuration tree* which is a perfectly balanced tree of height $d + 2(\alpha_1 \cdot \log(n) + \alpha_0 + 1)$. It is a seen as a dictionary whose keys are all configurations with stacks smaller than $\alpha_1 \log(n) + \alpha_0$ and values in $\{0, 1, \perp\}$ are currently computed labels according to the **Evaluation Rules**. Given a configuration $K = (q, w_1, w_2)$, the leaf obtained following the path $q\mathsf{P}(w_1, w_2)$ from the root of the configuration tree is the stored value for that configuration. In other words, given a configuration tree t, the value corresponding to the configuration (q, w_1, w_2) is $\mathsf{d}_{q\mathsf{P}(w_1,w_2)}(t)$.

We describe now the process of the computation. Suppose we are given some input word w of size n. Let $y = \widehat{w}$ be its well-balanced tree encoding. Let us apply Lemma 18 with $c = \bot$. There is a function f_0 defined by explicit structural recursion such that $f_0(y)$ is a well-balanced tree of depth $d + 2(\alpha_1 \mathsf{H}(y) + \alpha_0 + 1) = d + 2(\alpha_1 \log(n) + \alpha_0 + 1)$. In other words, $f_0(y)$ is a configuration tree for w whose labels are set to $\bot$.

Suppose we are given a MIP-definable function $\mathsf{next}(x, y)$ which takes as input a configuration tree x corresponding to the input y and applies on the configuration tree x the **Evaluation Rules**. Should we apply it $\beta_1 \mathsf{H}(y)^k + \beta_0 = \beta_1 \log(n)^k + \beta_0$ times on $f_0(y)$, that we would get an evaluation of all configurations of $f_0(y)$. Lemma 20 fulfills the requirement by providing a function f_1 such that for all y: $f_1(y, y) = \underbrace{\mathsf{next}(\cdots \mathsf{next}}_{\beta_1 \log(n)^k + \beta_0}(f_0(y), y), y) \cdots)$. Recall that the output is stored in the initial configuration, that is the left most branch of $f_1(y, y)$. Applying `left_most` on it outputs the value of the machine M on w.

So, to finish the proof, we have to show that such an update can be done by MIP-recursion. This is the role of next Lemma. □

4.1. Updating call trees

This subsection is devoted to the proof of the Lemma.

Lemma 33. [Update Lemma] There exists a MIP-definable function $\mathsf{next}(x, y)$ which takes as input a configuration tree x corresponding to the input y and applies on the configuration tree x the **Evaluation Rules**.

Proof. The function $\mathsf{next}(x, y)$ works by finite case distinction just calling auxiliary functions. By Lemma 14 it is shown MIP-definable[9,10]:

$$\mathsf{next}(\mathsf{t}_{d+4}, y) = \mathsf{t}_{d+4}[(x_{q\mathsf{ab}} \leftarrow \mathsf{next}_{q,a,b}(x_{q\mathsf{ab}}, \mathsf{t}_{d+4}, y))_{q \in \mathbb{W}_d, \mathsf{a} \in \mathbb{W}_2, \mathsf{b} \in \mathbb{W}_2}]$$

where $\mathsf{next}_{q,a,b}$ are the auxiliary functions. Notice that $x_{q\mathsf{ab}}$ contains the values of all configurations corresponding to state q and top bits a and b. All these configurations share **Successor Rules**, **Stack rules** and **Evaluation Rules**. The second argument t_{d+4} contains the full configuration tree and y is a copy of the (tree encoding of the) machine's input.

[9] Since we apply next on configuration trees, equations for $m < d + 4$ are dummy for the simulation, we do not write them explicitly.

[10] recall that $|Q| = d$ and that 0 and 1 are encoded by two bits, thus the depth $d + 4$ in the pattern tree t_{d+4} (see Notation 1).

The role of the functions $\mathsf{next}_{q,a,b}$ is to update the part of the configuration tree they correspond to. The definition of these auxiliary functions depends on the kind of states (accepting, rejecting, etc) and, for action states, on the top bits of the stacks.

- *Accepting and rejecting states*. We define

$$\mathsf{next}_{q,a,b}(x,t,y) = \mathsf{const}_{\mathbf{1}}(x) \quad \text{if } q \text{ is accepting}$$
$$\mathsf{next}_{q,a,b}(x,t,y) = \mathsf{const}_{\mathbf{0}}(x) \quad \text{if } q \text{ is rejecting.}$$

- *Reading states*. We first provide the definition corresponding to a c, 1-reading state, that is when you expect that the bit denoted by stack 1 is a c. Let $\mathsf{next}_{q,\mathtt{a},\mathtt{b}}(x,t,y) = \mathsf{read}_1(x,\mathsf{d}_a(y))$ if $a \neq \#$, otherwise let $\mathsf{next}_{q,\#,b}(x,t,y) = \mathsf{read}'_1(x,y)$ with:

$$\begin{aligned}
\mathsf{read}_1(\mathrm{t}_2,y) &= (\mathsf{read}'_1(x_{00},y) \star \mathsf{read}_1(x_{01},y)) \star (\mathsf{read}_1(x_{10},\mathsf{d}_0(y)) * \mathsf{read}_1(x_{11},\mathsf{d}_1(y)))\\
\mathsf{read}'_1(\mathrm{t}_2,y) &= \mathrm{t}_2[(x_w \leftarrow \mathsf{read}'_1(x_w,y))_{w\in\mathbb{W}_2}]\\
\mathsf{read}_1(c',y) &= \bot \qquad \text{//still reading first stack}\\
\mathsf{read}'_1(c',y) &= \mathsf{cond}(y,\neg c,c,\bot,\bot)
\end{aligned}$$

The function read_1 is meant to scan the first stack, that is before the first stop marker # met in the configuration tree. After the marker has been read, the input is supposed to be so. All these configurations will share same result, as computed by read'_1. To read on the second stack, that is for c, 2-reading states, we need three functions. The first one, read_2 corresponds to the scanning of the first stack, the second one, read'_2 to the scanning of the second stack and read''_2 to the scanning of the remaining #. $\mathsf{next}_{q,\mathtt{a},\mathtt{b}}(x,t,y) = \mathsf{read}_2(x,\mathsf{d}_b(y))$ if $b \neq \#$, otherwise let $\mathsf{next}_{q,\mathtt{a},\#}(x,t,y) = \mathsf{read}''_2(x,y)$ with:

$$\begin{aligned}
\mathsf{read}_2(\mathrm{t}_2,y) &= (\mathsf{read}'_2(x_{00},y) \star \mathsf{read}_2(x_{01},y)) \star (\mathsf{read}_2(x_{10},y) * \mathsf{read}_2(x_{11},y))\\
\mathsf{read}'_2(\mathrm{t}_2,y) &= (\mathsf{read}''_2(x_{00},y) \star \mathsf{read}_2(x_{01},y)) \star (\mathsf{read}'_2(x_{10},\mathsf{d}_0(y)) \star \mathsf{read}'_2(x_{11},\mathsf{d}_1(y)))\\
\mathsf{read}''_2(\mathrm{t}_2,y) &= \mathrm{t}_2[(x_w \leftarrow \mathsf{read}''_2(x_w,y))_{w\in\mathbb{W}_2}]\\
\mathsf{read}_2(c',y) &= \bot \qquad \text{//still reading first stack}\\
\mathsf{read}'_2(c',y) &= \bot \qquad \text{//still reading second stack}\\
\mathsf{read}''_2(c',y) &= \mathsf{cond}(y,\neg c,c,\bot,\bot)
\end{aligned}$$

- *Action states*. These are the hard cases. To compute the value of such configurations, we need the value of its two successor configurations. The key point is that the transitions of a configuration $(q, a_1 \cdots a_i, b_1 \cdots b_j)$ to its successors are entirely determined by the state q and the two top bits a_1 and b_1 so that $\mathsf{next}_{q,\mathtt{a}_1,\mathtt{b}_1}$ "knows" exactly which transition it must implement. We have to distinguish the four cases where we push or pop an element on one of the two stacks:

1. $\delta(q, \mathtt{a}_1, \mathtt{b}_1) = (q', q'', \mathsf{push}_1(\mathtt{a}_0))$;
2. $\delta(q, \mathtt{a}_1, \mathtt{b}_1) = (q', q'', \mathsf{pop}_1)$;
3. $\delta(q, \mathtt{a}_1, \mathtt{b}_1) = (q', q'', \mathsf{push}_2(\mathtt{b}_0))$;
4. $\delta(q, \mathtt{a}_1, \mathtt{b}_1) = (q', q'', \mathsf{pop}_2)$.

Let us see first how these actions modify the encoding of configurations. So, we suppose the current configuration to be $K = (q, a_1 \cdots a_i, b_1 \cdots b_j)$. By assumption, the stacks of the first successor configuration are not updated **(Stack Rule)**, so that the encoding of the first successor of K is

$$q' \, \mathtt{a}_1 \, \mathtt{b}_1 \, \mathtt{a}_2 \, \mathtt{a}_3 \cdots \mathtt{a}_i \, \# \, \mathtt{b}_2 \, \mathtt{b}_3 \cdots \mathtt{b}_j \, \# \# \cdots \#$$

For the second successor of K, the encoding depends on the four possible actions:

1. $q'' \, \mathtt{a}_0 \, \mathtt{b}_1 \, \mathtt{a}_1 \, \mathtt{a}_2 \, \mathtt{a}_3 \cdots \mathtt{a}_i \, \# \, \mathtt{b}_2 \, \mathtt{b}_3 \cdots \mathtt{b}_j \, \# \cdots \#$
2. $q'' \, \mathtt{a}_2 \, \mathtt{b}_1 \, \mathtt{a}_3 \cdots \mathtt{a}_i \, \# \, \mathtt{b}_2 \, \mathtt{b}_3 \cdots \mathtt{b}_j \, \# \, \# \cdots \#$
3. $q'' \, \mathtt{a}_1 \, \mathtt{b}_0 \, \mathtt{a}_2 \, \mathtt{a}_3 \cdots \mathtt{a}_i \, \# \, \mathtt{b}_1 \, \mathtt{b}_2 \, \mathtt{b}_3 \cdots \mathtt{b}_j \, \# \cdots \#$
4. $q'' \, \mathtt{a}_1 \, \mathtt{b}_2 \, \mathtt{a}_2 \, \mathtt{a}_3 \cdots \mathtt{a}_i \, \# \, \mathtt{b}_3 \cdots \mathtt{b}_j \, \# \, \# \cdots \#$

As for accepting and rejecting states, we will use auxiliary functions $\mathsf{next}_{\circ,1}$, $\mathsf{next}_{\circ,2,b_1}$, $\mathsf{next}_{\circ,3,b_1}$, and $\mathsf{next}_{\circ,4}$, which correspond to the four cases mentioned above (and where $\circ$ is $\wedge$ or $\vee$ according to the state q). Then we use Lemma 15 to show the functions $\mathsf{next}_{q,\mathtt{a}_1,\mathtt{b}_1}$ defined by MIP-recursion.

We come back now to the definition of the four auxiliary functions $\mathsf{next}_{\circ,1}$, $\mathsf{next}_{\circ,2,b_1}$, $\mathsf{next}_{\circ,3,b_1}$, and $\mathsf{next}_{\circ,4}$. The principle of their definition is to follow in parallel the paths of the two successor configurations. We detail the first example. The others are variations on the theme.

1. Consider the case $\delta(q, \mathtt{a}_1, \mathtt{b}_1) = (q', q'', \mathsf{push}_1(\mathtt{a}_0))$, we define the function $\mathsf{next}_{q,\mathtt{a}_1,\mathtt{b}_1}(x, t, y) = \mathsf{next}_{\circ,1}(x, \mathsf{d}_{q'a_1b_1}(t), \mathsf{d}_{q''a_0b_1a_1}(t))$. With respect to the simulation, observe that $\mathsf{next}_{\circ,1}(x, u, v)$ is fed with $(\mathsf{d}_{qa_1b_1}(t), \mathsf{d}_{q'a_1b_1}(t), \mathsf{d}_{qa_0b_1a_1}(t))$ where t is the configuration tree to be updated. For any letters $a_2, \ldots, a_k, b_2, \ldots, b_j$, $\mathrm{lab}(q, a_1 \cdots a_k, b_1 \cdots b_j)$ is $\mathrm{lab}(q', a_1 \cdots a_k, b_1 \cdots b_j) \circ \mathrm{lab}(q'', a_0 \cdot a_1 \cdots a_k, b_1 \cdots b_j)$ where $\circ$ is the conditional corresponding to the state. Thus, to update the configuration tree, one replaces bit $\mathsf{d}_{\mathtt{a}_2 \ldots \mathtt{a}_k \# \mathtt{b}_2 \cdots \mathtt{b}_j \#^m}(x)$ by $\mathsf{d}_{\mathtt{a}_2 \ldots \mathtt{a}_k \# \mathtt{b}_2 \cdots \mathtt{b}_j \#^m}(u) \circ \mathsf{d}_{\mathtt{a}_2 \ldots \mathtt{a}_k \# \mathtt{b}_2 \cdots \mathtt{b}_j \#^{m-1}}(v)$ where m denotes the number of padding blank symbols. The remaining difficulty comes from the fact that v has height shorter by two[11] compared to x and u. Equations below cope with that technical point, forgetting the last # on paths of v. Formally we define $\mathsf{next}_{\circ,1}$ as:

$$\mathsf{next}_{\circ,1}(\mathtt{t}_4, u, v) = \mathtt{t}_4[(x_w \leftarrow \mathsf{next}_{\circ,1}(x_w, \mathsf{d}_w(u), \mathsf{d}_w(v)))_{w \in \mathbb{W}_4}]$$
$$\mathsf{next}_{\circ,1}(\mathtt{t}_2[x_w \leftarrow c_w], u, v) = \mathtt{t}_2[(x_w \leftarrow \mathsf{d}_w(u) \circ v)_{w \in \mathbb{W}_2}]$$

[11] Each letter being encoded by a branch of length 2.

where the c_w are to be taken in $\{0, 1, \bot\}$ and $\circ$ is the conditional corresponding to the state.

2. If $\delta(q, \mathrm{a}_1, \mathrm{b}_1) = (q', q'', \mathsf{pop}_1)$, we define $\mathsf{next}_{q,\mathrm{a}_1,\mathrm{b}_1}(x, t, y) = \mathsf{next}_{\circ,2,\mathrm{b}_1}(x, \mathsf{d}_{q'a_1b_1}(t), \mathsf{d}_{q''(t)})$. In that case, it is the last argument which is the bigger one.

$$\mathsf{next}_{\circ,2,b_1}(\mathrm{t}_2, u, v) = \mathrm{t}_2[(x_w \leftarrow \mathsf{next}'_{\circ,2}(x_w, \mathsf{d}_w(u), \mathsf{d}_{wb_1}(v)))_{w\in\mathbb{W}_2}]$$
$$\mathsf{next}'_{\circ,2}(\mathrm{t}_2, u, v) = \mathrm{t}_2[(x_w \leftarrow \mathsf{next}'_{\circ,2}(x_w, \mathsf{d}_w(u), \mathsf{d}_w(v)))_{w\in\mathbb{W}_2}]$$
$$\mathsf{next}'_{\circ,2}(c, u, v) = u \circ \mathsf{d}_{00}(v)$$

3. If $\delta(q, \mathrm{a}_1, \mathrm{b}_1) = (q', q'', \mathsf{push}_2(\mathrm{b}_0))$, we define $\mathsf{next}_{q,\mathrm{a}_1,\mathrm{b}_1}$ by the equation:

$$\mathsf{next}_{q,\mathrm{a}_1,\mathrm{b}_1}(x, t, y) = \mathsf{next}_{\circ,3,\mathrm{b}_1}(x, \mathsf{d}_{q'a_1b_1}(t), \mathsf{d}_{q''a_1b_0(t)})$$
$$\mathsf{next}_{\circ,3,b_1}(\mathrm{t}_2, u, v) = (\mathsf{next}_{\circ,1}(x_{00}, \mathsf{d}_{00}(u), \mathsf{d}_{00b_1}(v)) \star \mathsf{next}_{\circ,1}(x_{01}, \mathsf{d}_{01}(u), \mathsf{d}_{01b_1}(v))) \star (\mathsf{next}_{\circ,3,b_1}(x_{10}, \mathsf{d}_{10}(u), \mathsf{d}_{10}(v)) \star \mathsf{next}_{\circ,3,b_1}(x_{11}, \mathsf{d}_{11}(u), \mathsf{d}_{11}(v)))$$
$$\mathsf{next}_{\circ,3,b_1}(c, u, v) = \bot$$

4. For the last case, that is $\delta(q, \mathrm{a}_1, \mathrm{b}_1) = (q', q'', \mathsf{pop}_2)$, we use four auxiliary arguments to remind the first letter read on the stack of the second successor.

$$\mathsf{next}_{q,\mathrm{a}_1,\mathrm{b}_1}(x, t, y) = \mathsf{next}_{\circ,4,\epsilon}(x, \mathsf{d}_{q'a_1b_1}(t), \mathsf{d}_{q''00}(t), \mathsf{d}_{q''01}(t), \mathsf{d}_{q''10}(t), \mathsf{d}_{q''11}(t))$$
$$\mathsf{next}_{\circ,4,00}(\mathrm{t}_2, u, v_{00}, v_{01}, v_{10}, v_{11}) = \mathrm{t}_2[(x_w \leftarrow \mathsf{next}'_{\circ,2}(x_w, \mathsf{d}_w(u), v_w))_{w\in\mathbb{W}_2}]$$
$$\mathsf{next}_{\circ,4,v}(\mathrm{t}_2, u, v_{00}, v_{01}, v_{10}, v_{11}) = \mathrm{t}_2[(x_w \leftarrow \mathsf{next}_{\circ,4,w}(x_w, \mathsf{d}_w(u), \mathsf{d}_w(v_{00}), \mathsf{d}_w(v_{01}), \mathsf{d}_w(v_{10}), \mathsf{d}_w(v_{11}))_{w\in\mathbb{W}_2}]$$
$$\mathsf{next}_{\circ,4,v'}(c, u, v_{00}, v_{01}, v_{10}, v_{11}) = \bot$$

with $v \in \{\epsilon, 01, 10, 11\}$ and $v' \in \mathbb{W}_0 \cup \mathbb{W}_2$.

□

5. Compilation of recursive definitions to circuit

This section is devoted to the proof of the Proposition:

Proposition 34. *For $k \geq 1$, any function in INC^k is computable in NC^k.*

We begin with some observations. All along, n denotes the size of the input. First, to simulate theoretic functions in INC^k, we will forget the tree structure and make the computations on the encoded words. Second, due to Proposition 13, these words are supposed to be in $\{0, 1\}^*$.

Third, functions defined by explicit structural recursion can be computed by NC^1 circuits. This is a direct consequence of the fact that explicit structural recursion is a particular case of LRRS-recursion as defined in Leivant and Marion [LM00].

Fourth, by induction on the definition of functions, one proves the key Lemma:

Lemma 35. Given a function $f \in \mathsf{INC}^k$, there are (finitely many) MIP-functions $h_1, \ldots, h_m$ and polynomials $P_1, \ldots, P_m$ of degree smaller than k such that $f(\vec{t}, \vec{u}) = h_1^{P_1(\log(n))}(\cdots h_m^{P_m(\log(n))}(g(\vec{u}))\ldots)$ where g is defined by structural recursion.

Now, the compilation of functions to circuits relies on three main ingredients. First point, we show that each function h_i as above can be computed by a circuit:

1. of fixed height with respect to the input (the height depends only on the definition of the functions),
2. with a linear number of gates with respect to the size of the first input of the circuit (corresponding to the recurrence argument),
3. with the number of output bits equal to the number of input bits of its first argument.

According to 1), we note H the maximal height of the circuits corresponding to the h_i's.

Second point, since there are $\sum_{i=1..m} P_i(\log(n))$ applications of such h_i, we get a circuit of height bounded by $H \times \sum_{i=1..m} P_i(\log(n)) = O(\log^k(n))$. That is a circuit of height compatible with NC^k. Observe that we have to add as a first layer a circuit that computes g. According to our second remark, this circuit has a height bounded by $O(log(n))$, so that the height of the whole circuit is of the order $O(\log^k(n))$.

Third point, the circuits corresponding to g, being in NC^1, have a polynomial number of gates with respect to n and a polynomial number of output bits with respect to n. Observe that the output of g is exactly the recurrence argument of some h_i whose output is itself the first argument of the next h_i, and so on. So that according to item 3) of the first point, the size of the input argument of each of the h_i is exactly the size of the output of g. Consequently, according to item 2) above, the number of circuit gates is polynomial.

Since all constructions are uniform, we get the expected result.

5.1. NC^0 *circuits compute* MIP

In this section, we prove that functions defined by mutual in place recursion can be computed by NC^0 circuits with a linear number of gates with respect to the size of the first argument. Since MIP-functions keep the shape of their first argument, we essentially have to build a circuit for each bit of this argument.

Lemma 36. Explicitly defined atomic functions can be defined without use of $\star$.

By function normalization. Indeed, in the definition of an atomic function, the $\star$ function is necessarily composed with an other outer function and thus can be simplified.

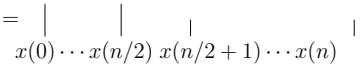
Lemma 37. Explicitly defined atomic functions are in NC^0.

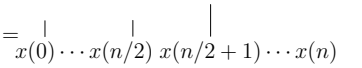
Proof. Consider the following circuits. To stress the fact that circuits are uniform, we put the size of the arguments into the brackets. n corresponds to the size of x, n_0 to the size of x_0 and so on. $x(k)$ for $k \in \mathbb{N}$ corresponds to the k-th bit of the input x. The "long" wires correspond to the outputs. Shorter ones are simply forgotten.

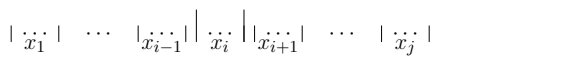
$C_0[n]:$ 

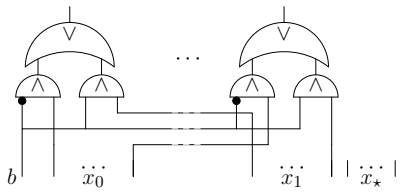
$C_1[n]:$

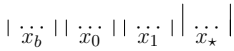
$C_{\mathtt{d}_0}[1] = C_{\mathtt{d}_1}[1] =$ 

$C_{\mathtt{d}_0}[2+n] =$ 

$C_{\mathtt{d}_1}[2+n] =$ 

$C_{\pi_i^j}[n_1, \cdots, n_j]:$ 

$C_{\mathtt{cond}}[1, n_0, n_0, n_\star] =$ 

$C_{\mathrm{cond}}[2+n_b, n_0, n_1, n_\star] =$ 

We see that composing the previous cells, with help of Lemma 36, we can build a circuit of fixed height (with respect to the size of input) for any explicitly defined atomic function. □

5.2. Simulating MIP*'s recursion*

Lemma 38. Any MIP-function can be computed by a circuit of fixed height with respect to the size of the input.

Proof. Let us consider a set $(f_i)_{i\in I}$ of MIP-functions. Write their equations as follows:

$$f_i(t_0 \star t_1, \vec{u}) = f_{p(i,0)}(t_0, \vec{\sigma}_{i,0}(\vec{u})) \star f_{p(i,1)}(t_1, \vec{\sigma}_{i,1}(\vec{u}))$$
$$f_i(c, \vec{u}) = g_i(c, \vec{u})$$

where $p(i,b) \in I$ is an explicit (finite) mapping of the indices, $\vec{\sigma}_{i,0}$ and $\vec{\sigma}_{i,1}$ are vectors of $\star$-free explicitly defined functions and the functions $g_{i,c}$ (and consequently the g_i) are explicitly defined atomic functions.

First, observe that any of these explicitly defined functions g_i can be computed by some circuit B_i of fixed height as seen in Lemma 37. Since I is finite, we call H the maximal height of these circuits $(B_i)_{i\in I}$.

Suppose we want to compute $f(t,\vec{x}) \in (f_i)_{i\in I}$ for some t and $\vec{x}$ which have both size smaller than n. Remember that $|f(t,\vec{x})| = |t|$. So, to any k-th bit of the recurrence argument t, we will associate a circuit computing the corresponding output bit, call this circuit C_k. Putting all the circuits $(C_k)_{k\in\{0,\dots,n-1\}}$ in parallel, we get a circuit that computes all the output bits of f_i, and moreover, this circuit has a height bounded by H. So, the last point is to show that for each k, we may compute uniformly the index i of the circuit B_i corresponding to C_k and among the inputs $t, \vec{x}$ the one plugged into the circuit C_k.

To denote the k-th bit of the input, consider its binary encoding where we take the path in the full binary tree t ending at this k-th bit. Call this path w. Notice first that w itself has logarithmic size with respect to n, the size of t. Next, observe that any sub-tree of the inputs can be represented in logarithmic size by means of its path. More precisely, to represent such sub-trees, we introduce the following data structure. Consider the record type $\mathtt{st} = \{\mathtt{r};\mathtt{w};\mathtt{h}\}$. The field $\mathtt{r}$ says to which input the value corresponds to. $\mathtt{r = 0}$ corresponds to t, $\mathtt{r = 1}$ correspond to x_1 and so on. $\mathtt{w}$ gives the path to the value (in that input). For convenience, we keep its height $\mathtt{h}$. In summary $\{\mathtt{r{=}i;w{=}w';h{=}m}\}$ corresponds to the subtree $\mathtt{d}_{w'}(u_i)$ (where we take the convention that $t = u_0$). We use the '.' notation to refer to a field of a record. We consider then the data structure $\mathtt{val} = \mathtt{st} + \{\mathtt{0}, 1\}$. Variables $\mathtt{u,\ v}$ coming next will be of that "type".

To compute the function $(\sigma_{i,b})_{i\in I, b\in\{0,1\}}$ appearing in the definition of the $(f_i)_{i\in I}$, we compose the programs:

```
zero(u){                  one(u){             pi_i_j(u_1,...u_j){
  return 0;                 return 1;               return u_i;
}                         }                   }
```

```
d0(u){ if(u == 0 || u == 1 || u.h = 0)  return u;
         else  return [r=u.r;w=u.w 0;h= u.h-1];   }
```

```
d1(u){ if(u == 0 || u == 1 || u.h == 0) return u;
         else return [r=u.r;w=u.w 1;h= u.h-1];    }
```

Then given some input bit k, we compute the values of i and the $\vec{u}$ in $g_i(c,\vec{u})$ corresponding to the computation of the k-th bits of the output. Take $d+1$ the

maximal arity of functions in $(f_i)_{i\in I}$. To simplify the writing, we take it (without loss of generality) as a common arity for all functions.

```
G(i,w,u_0,...,u_d){
 //u_0 corresponds to t,
   if(w == epsilon) {
   return(i,u_0,...,u_d);
   }
   else{
   a := pop(w); //get the first letter of w
   w := tail(w); //remove the first letter to w
   switch(i,a){//i in I, a in {0,1}
         case (i1,0):
         v_0 = d_0(u_0);
         foreach 1 <= k <= d:
               v_k = sigma_i1_0_k(u_0,...,u_d);
               //use the sigma defined above
               next_i = p_i1_0;
               //the map p is hard-encoded
         break;
         ...
         case (im,1):
         v_0 = d_1(u_0);
         foreach 1 <= k <= d:
               v_k = sigma_im_1_k(u_0,...,u_d);
               next_i = p_im_1;
         break;
        }
        return G(next_i,w,d_a(u_0),v_1,...,v_d);
   }
}
```

Observe that this program is a tail recursive program. As a consequence, to compute it, one needs only to store the recurrence arguments, that is a finite number of variables. Since the value of these latter variables can be stored in logarithmic space, the computation itself can be performed within the bound. Finally, the program returns the name `i` of the circuit that must be build, a pointer on each of the inputs of the circuit with their size. It is then routine to build the corresponding circuit at the corresponding position `w`. □

6. $\mathbf{RBE}^k = \mathbf{NC}^k$

The proof of Theorem 27 is a direct consequence of Proposition 39 and Proposition 42.

Proposition 39. *Any function in* RBE^k *is computable in* NC^k.

The proof relies on the following two lemmas:

Lemma 40. Given a function f which is a (C, D)-length preserving function in RBE, it is the composition of a function f' which is length-preserving and the constant function on bits $c_{C,D}$ which is (C, D)-length preserving (see Lemma 18):

$$f(x_1, \ldots, x_n) = f'(c(x_1, \ldots, x_n), x_1, \ldots, x_n).$$

Proof. Consider such a function f, that is there are rational functions $\phi_0, \ldots, \phi_k$ and a finite map h such that $f(w_1, \ldots, w_n)[p] = h(\phi_0(p), w_{e_1}[\phi_1(p)], \ldots, w_{e_k}[\phi_k(p)])$. Let f' to be the length-preserving function defined by Equations $f'(w_0, \ldots, w_n)[p] = h(\phi_0, w_{e_1}[\phi_1(p)], \ldots, w_{e_k}[\phi_(p)])$. Let $c_{(C,D)}$ as in Lemma 18. □

Lemma 41. Assume that $f : \mathbb{P}_{i_1} \times \cdots \mathbb{P}^{i_n} \times \mathbb{W}^m \to \mathbb{W}$ is computed by a k-recursion schema. Then, there is a polynomial P of degree k such that $f(p_1, \ldots, p_n, x_1, \ldots, x_m) = h^{P(|p_1|,\ldots,|p_n|)} \circ g(x_1, \ldots, x_k)$ where h is non size increasing and g is (C, D)-length preserving, both being in $\mathcal{B}_{\mathsf{RBE}}$.

By $h^{P(|p_1|,\ldots,|p_n|)} \circ g(x_1, \ldots, x_k)$, we mean

$$h^{P(|p_1|,\ldots,|p_n|)} \circ g(x_1, \ldots, x_k) = \underbrace{h(h(\cdots h(}_{P(|p_1|,\ldots,|p_n|)} g(x_1, \ldots, x_m), x_1, \ldots, x_m) \cdots).$$

Proof. The statement is a rewriting of Lemma 4.8 in Marions's paper [Mar09] in the present context. So is its proof. □

Proof of Proposition 39. The proof relies on the following two observation. First, a function in $\mathcal{B}_{\mathsf{RBE}}$ is computable by a constant height uniform circuit. Indeed, such a function is the composition of a length-preserving function and the constant function (by Lemma 40). Since length-preserving function are in MIP, by Lemma 38, it is computable by a circuit of constant depth. The constant function itself is computable by a constant height circuit, thus the observation holds.

Second, by induction on function definitions, one proves that for all function $f : \mathbb{P}_{i_1} \times \mathbb{P}_{i_k} \times \mathbb{W}^m \to \mathbb{W}$ in RBE^k, then $|f(p_1, \ldots, p_m, x_1, \ldots, x_k)| \leq P(|x_1|, \ldots, |x_k|)$.

Now, consider a function $f : \mathbb{W}^k \to \mathbb{W}$ in RBE^k. Either it is in in $\mathcal{B}_{\mathsf{RBE}}$ and then, by the first observation, it is in $\mathrm{NC}^0 \subseteq \mathrm{NC}^k$ or it is obtained by composition of functions in $\mathcal{B}_{\mathsf{RBE}}$ and the step induction is immediate, or it is a composition of the form

$$f(p_1, \ldots, p_n, x_1, \ldots, x_m) = f'(\mathrm{len}_{i_1}(f_1(x_1, \ldots, x_m)), \ldots, \mathrm{len}_{i_n}(f_n(x_1, \ldots, x_m)), x_1, \ldots, x_m)$$

for some function f' defined by k-recursion and $f_1, \dots, f_m$ some functions in RBE^k. By second observation, for all $1 \leq j \leq n$, there is a polynomial P_j of degree k such that $|f_j(x_1, \dots, x_m)| \leq P_j(|x_1|, \dots, |x_m|)$. Thus, for all $1 \leq j \leq n$, $|\mathrm{len}_{i_j}(f_j(x_1, \dots, x_m))| \leq \sum_{i=1}^{m} a_{i,j}|x_i| + b_j$ for some constants $(a_{i,j})_{i\leq n, j\leq m}$ and $(b_j)_{j\leq n}$.

Composing this inequality with Lemma 41, we can state that there is a polynomial P of degree k such that $f'(x_1, \dots, x_m) = h^{P(|x_1|,\dots,|x_m|)} \circ g(x_1, \dots, x_m)$. The proposition is then a direct consequence of the first observation.

□

Proposition 42. *Any function in* NC^k *is computable in* RBE^k.

Proof. Let us come back to the proof of Proposition 32. It relies on Lemma 18, Lemma 20 and finally Lemma 33. We have seen in Example 24, that the function of Lemma 18 is computable by a (α_0, α_1)-length preserving RBE function. By Proposition 28, it is clear that the update function of Lemma 33 is also computable by a length-preserving function in RBE. To end the proof, it remains to prove a lemma analogous to Lemma 20. □

Lemma 43. Given β_1 and β_0, and RBE^k definable functions such that h is length preserving in RBE^k and g in RBE^k, then there is a function defined by k-recursion in RBE^k such that

$$f(w, \vec{u}) = \underbrace{h(\cdots h(}_{\beta_1|w|^k+\beta_0} g(w, \vec{u}), \vec{u}, \dots, \vec{u}).$$

Proof. Again, the proof is a restatement of a lemma in [Mar09] (namely Lemma 4.2). Given some function $h : \mathbb{W}^{m+1} \to \mathbb{W}$, let h^β be defined as $h^\beta(w, \vec{u}) = \underbrace{h(\cdots h}_{\beta}(w, \vec{u}), \vec{u}) \cdots, \vec{u})$. By induction on k. For $k = 1$, we define $f_1(w, u) = f'_{1,\beta_1,\beta_0}(\mathrm{len}_1(w), \mathrm{len}_0(w), w, \vec{u})$ with $f'_{1,\beta_1,\beta_0}(\epsilon_1, q, w, \vec{u}) = h^{\beta_0}(g(w, \vec{u}), \vec{u}), \cdots, \vec{u})$ and

$$f'_{1,\beta_1,\beta_0}(C(p), q, w, \vec{u}) = h^{\beta_1}(f'_{1,\beta_1,\beta_0}(p, q, w, \vec{u}), \vec{u})$$

with $C \in \{A, B\}$. Note that h^{β_1} is length preserving as long as h is such. And furthermore, $f_{1,\beta_1,\beta_0}(p, q, w, \vec{u}) = h^{\beta_1|p|^1+\beta_0}(g(w, \vec{u}), \vec{u})$.

For $k > 1$, let $f_k(w, \vec{u}) = f'_{k,\beta_1,\beta_0}(\mathrm{len}_k(w), \mathrm{len}_{k-1}(w), w, \vec{u})$ with $f'_{k,\beta_1,\beta_0}(\epsilon_k, q, w, \vec{u}) = h^{\beta_0}(g(w, \vec{u}), \vec{u})$ and

$$f_{k,\beta_1,\beta_0}(C(p), q, w, \vec{u}) = f_{k-1,\beta_1,\beta_0}(q, q, f_{k,\beta_1,\beta_0}(p, q, w, \vec{u}), \vec{u}).$$

□

References

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

[BDG90] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural complexity II*, volume 22 of *EATCS Monographs of Theoretical Computer Science*. Springer, 1990.

[Bel95] Steve Bellantoni. Characterizing parallel time by type 2 recursions with polynomial output length. In Daniel Leivant, editor, *Logic and Computational Complexity*, volume 960, pages 253–268, 1995.

[BKMO08] Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion, and Isabel Oitavem. Recursion Schemata for NCk. In *CSL '08*, volume 5213 of *LNCS*. Springer, 2008.

[CKS81] A. K. Chandra, D. J. Kožen, and L. J. Stockmeyer. Alternation. *Journal ACM*, 28:114–133, 1981.

[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.

[Cob62] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Gir98] J.-Y. Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[Imm98] Neil Immerman. *Descriptive Complexity*. Springer, 1998.

[Lei91] D. Leivant. A foundational delineation of computational feasiblity. In *Proceedings of the Sixth IEEE Symposium on Logic in Computer Science (LICS'91)*, 1991.

[Lei93] Daniel Leivant. Stratified functional programs and computational complexity. In *Twentieth Annual ACM Symposium on Principles of Programming Languages*. ACM Press, New York, NY, USA, 1993.

[Lei98] D. Leivant. A characterization of NC by tree recurrence. In *Foundations of Computer Science 1998*, pages 716–724. IEEE Computer Society, 1998.

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

[Mar09] Jean-Yves Marion. On tiered small jump operators. *Logical Methods in Computer Science*, 5(1), 2009.

[MR07] Virgile Mogbil and Vincent Rahli. Uniform circuits, & boolean proof nets. In Lecture Notes in Computer Science, editor, *LFCS*, volume 4514, pages 401–421, 2007.

[Ruz81] W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–383, 1981.

[Sak09] J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

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