Implicit characterizations of FPTIME and NC revisited

Karl-Heinz Niggl, Henning Wunderlich

Ulmer Informatik-Berichte

Nr. 2008-09
Juli 2008

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm

Implicit characterizations of **FPTIME** and **NC** revisited

Karl-Heinz Niggl* Henning Wunderlich†

July 8, 2008

Abstract

Various simplified or improved, and partly corrected well-known implicit characterizations of the complexity classes **FPTIME** and **NC** are presented. Primarily, the interest is in simplifying the required simulations of various recursion schemes in the corresponding (implicit) framework, and in developing those simulations in a more uniform way, based on a step-by-step comparison technique, thus consolidating groundwork in implicit computational complexity.

1 Introduction

In implicit computational complexity, much attention has been payed to the complexity classes **FPTIME** and **NC**, e.g. see [2, 4, 6, 7, 9, 10, 15, 18, 19, 24, 26]. This paper presents simplified or improved, and partly corrected well-known implicit characterizations of the complexity classes **FPTIME** and **NC**.

The core of the present research is to simplify the required simulations of various (bounded) recursion schemes in the corresponding (implicit) framework, and moreover, to develop those simulations in a more uniform way, based on a step-by-step comparison technique. Furthermore, we establish a new ground type function algebraic characterization of **NC**, which might be of help to resolve the open problem [2] of characterizing **NC** through higher types.

The starting point is a simplified proof that the functions of Cobham's class, **Cob** [12], characterizing **FPTIME** is contained in the function algebra **BC** of Bellantoni and Cook [4]. That every function f of Cobham's class can be simulated in **BC** rests on three findings:

(S1) For every f in **Cob** one can construct a function $f'(w;\ \vec{x})$ in **BC**, called *simulation of* f, and a polynomial p_f, called *witness* for f, such that

$$f(\vec{x}) = f'(w;\ \vec{x}) \text{ whenever } |w| \geq p_f(|\vec{x}|).$$

(S2) For every polynomial $p(\vec{x})$ one can construct a function $W_p(\vec{x};\ )$ in **BC**, called *length-bound* on p, such that $|W_p(\vec{x};\ )| \geq p(|\vec{x}|)$.

*Technische Universität Ilmenau, Fakultät für Informatik und Automatisierung, Institut für Theoretische Informatik, Helmholtzplatz 1, D-98684 Ilmenau, e-mail: `niggl@tu-ilmenau.de`

†Universität Ulm, Fakultät für Ingenieurwissenschaften und Informatik, Institut für Theoretische Informatik, Oberer Eselsberg, D-89069 Ulm, e-mail: `Henning.Wunderlich@uni-ulm.de`

(S3) Every function $g(\vec{x};\ \vec{y},\vec{z})$ in **BC** can be written as $\mathbf{SN}(g)(\vec{x},\vec{y};\ \vec{z})$, called *safe-to-normal property.*

Thus, by use of **(S1)**, **(S2)**, **(S3)**, and safe composition, the proof that every f in **Cob** can be simulated in **BC** is then concluded as follows:

$$f(\vec{x}) = \mathbf{SN}(f')(W_{p_f}(\vec{x};\ ),\vec{x};\ )$$

In each simulation, we will concentrate on the crucial statement corresponding to **(S1)**. As for **(S1)** above, all cases are obvious, except for the case where f is defined by bounded recursion on notation, and here a difficult simulation and proof was given in [4]. The difficulty mainly arises because of an unnatural choice of a *case* function defined as

$$\mathrm{case}(\ ;x,even,odd) := \begin{cases} even & \text{if } x \text{ is even} \\ odd & \text{if } x \text{ is odd.} \end{cases}$$

When replacing function case by the function bcase (for binary case), that is,

$$\mathrm{bcase}(\ ;x,zero,even,odd) := \begin{cases} zero & \text{if } x = 0 \\ even & \text{if } x > 0 \text{ and } x \text{ is even} \\ odd & \text{if } x > 0 \text{ and } x \text{ is odd} \end{cases}$$

then a simulation f' can be constructed the correctness of which is immediate from its definition. So let $\mathbf{BC}'$ be **BC** where case is replaced with bcase.

Note that both case and bcase (as well as the binary predecessor function p) could be defined by recursion on notation and composition, using projections and the constructor functions 0, s_0, s_1. But in both algebras **BC** and $\mathbf{BC}'$, this is only possible at the cost of introducing normal input positions, and that is why they come as initial functions with safe input positions only. But then we have a choice between case and bcase. We clearly opt for bcase because it is the natural choice. In fact, bcase naturally springs from a “flat” recursion on notation, since that scheme distinguishes – for the recursion argument – the cases *zero*, *nonzero and even* and *nonzero and odd*[1]. Furthermore, note that while $\mathrm{bcase}(\ ;\ x,y,z_0,z_1)$ is provably indefinable in **BC**, the function $\mathrm{case}(\ ;\ x,z_0,z_1)$ is obviously in $\mathbf{BC}'$, since $\mathrm{case}(\ ;\ x,z_0,z_1){=}\mathrm{bcase}(\ ;\ x,z_0,z_0,z_1)$.

To our knowledge, the “simulation method” (**S1**) appears for the first time in the groundwork of Bellantoni and Cook [4]. Since then, it has been applied directly or in adapted form to many characterizations of complexity classes, e.g. the Kálmar-elementary functions and Pspace are treated in [25], in [20], [5] the method is extended to all levels of the Grzegorczyk hierarchy, and in [15] that method is adapted so as to compute all functions at Grzegorczyk level $n{+}2$ by loop programs of μ-measure n.

Roughly speaking, the simulation method consists in separating the “structure” in a recursion from the “growth rate” given with it. Technically, one introduces a single normal parameter, w, to which all given recursion parameters refer to in a “safe” way. It is hard to say what those simulations compute for wrong values of w, however, once w is sufficiently large, and that is where the witness comes into play, all given recursions unfold in the expected way.

[1] As a technical consequence, in $\mathbf{BC}'$ we don’t have to bother with defining the functions “PARITY”, “I”, “V” or “h”, unlike in [4].

Our way of performing the simulation method for various forms of recursion does not change that at all. However, unlike many instances of that method in the literature, we always start off with a clear semantics based on a step-by-step comparison technique such that when implementing the simulation in the given framework, the correctness of the implementation is immediate from the specified semantics. As pointed out above, the right choice of initial functions, such as bcase, will sometimes prove decisive.

Rounding off, the main goal is to propose a step-by-step comparison technique, exemplified at various forms of recursion, so as to perform the simulation method in a way that is easy to grasp and does away with hard going proofs. Thereby, groundwork in implicit computational complexity is revised and consolidated.

The paper is organized as follows. In **Section 2**, all basic notions involved in the design of Cobham's and Bellantoni/Cook's function algebra, **Cob** and **BC**, are introduced and examined. **Section 3** presents a simplified proof of $\mathbf{BC}' = \mathbf{Cob}$, thereby demonstrating the step-by-step comparison technique. Recalling Clote's function algebra, **CLO**, in **Section 4** and **5**, two variants, $\mathbf{CLO}'$ and $\mathbf{CLO}''$, are considered, and a proof of $\mathbf{CLO}' = \mathbf{CLO} = \mathbf{CLO}''$ is presented, using the same technique. In **Section 6** several ramified function algebras are introduced, and, using both the step-by-step comparison technique and the above identities, it is proved that all of them characterize the class **NC**.

2 Preliminaries and some existing function algebras

We assume only basic knowledge about the function algebras and complexity classes studied here. In this section, we introduce to and summarize some basic concepts, and make some stipulations concerning notations used throughout this article.

Albeit describing operations on binary representations, all of the functions under consideration are *number-theoretic*, that is, functions of the form $f\colon \mathbb{N}^n \to \mathbb{N}$. For unary functions f and numbers k, f^k denotes the *kth iterate of* f, inductively defined by $f^0(x) = x$ and $f^{k+1}(x) = f(f^k(x))$.

Binary representations of natural numbers x, denoted by $\mathrm{bin}(x)$, can be simulated by 0 (viewed as 0-ary function) and the *binary successors* $\mathrm{S}_0, \mathrm{S}_1$ which correspond to the operations of extending binary representations by a new lowest order bit.

$$
\begin{aligned}
\mathrm{S}_0(x) &= 2\cdot x && \text{(operation } \mathrm{bin}(x) \mapsto \mathrm{bin}(x)0 \text{ for } x \neq 0)\\
\mathrm{S}_1(x) &= 2\cdot x + 1 && \text{(operation } \mathrm{bin}(x) \mapsto \mathrm{bin}(x)1)
\end{aligned}
$$

This "data structure" gives rise to a canonical recursion scheme: A function f is defined by *recursion on notation* from functions g, h_0, h_1, denoted by $f = \mathrm{RN}(g, h_0, h_1)$, if for all $y, \vec{x}$,

$$
\begin{aligned}
f(0, \vec{x}) &= g(\vec{x})\\
f(\mathrm{S}_i(y), \vec{x}) &= h_i(y, \vec{x}, f(y, \vec{x})) \qquad \text{for } \mathrm{S}_i(y) \neq 0.
\end{aligned}
$$

Observe that $\mathrm{bin}(y) = b_{l-1} \dots b_0 \neq \epsilon$ implies $y = \mathrm{S}_{b_0}(\mathrm{S}_{b_1}(\dots \mathrm{S}_{b_{l-1}}(0) \dots))$. Thus, for recursion on notation, the recourse is from $b_{l-1} \dots b_0$ to $b_{l-1} \dots b_1$ to $\dots$ to

$b_{l-1} = \mathrm{S}_1(0)$, and finally from $\mathrm{S}_1(0)$ to 0. So one needs $|y|$ recursive calls of f when computing $f(y, \vec{x})$, where $|y| = \lceil \log_2(y+1) \rceil$ is the *binary length of* y.

A function f is defined by *bounded recursion on notation* from functions g, h and bound B, denoted by $f = \mathrm{BRN}(g, h, B)$, if $f = \mathrm{RN}(g, h)$ and $f \le B$.

Finally, f is defined by ordinary *composition* from functions $h, g_1, \ldots, g_l$, denoted by $f = \mathrm{COMP}(h, g_1, \ldots, g_l)$, if it satisfies $f(\vec{x}) = h(g_1(\vec{x}), \ldots, g_l(\vec{x}))$.

We use Clote's [11] notation to specify function algebras, $[\mathcal{X}; \text{OP}]$, denoting the smallest set of functions containing the functions specified in $\mathcal{X}$ and closed under the operations listed in OP.

Each function algebra is either purely *number-theoretical* or *ramified*. A typical example of the former is Cobham's [12] well-known function algebra

$$\mathbf{Cob} := [0, \mathrm{S}_0, \mathrm{S}_1, \Pi, \#;\ \mathrm{COMP}, \mathrm{BRN}]$$

where Π denotes the set of all *projections* Π^n_i satisfying $\Pi^n_i(x_1, \ldots, x_n) = x_i$, for $1 \le i \le n$, and where $\#$, called *smash function*, satisfies $\#(x, y) = 2^{|x| \cdot |y|}$.

The idea in the design of **Cob** is that recursion on notation can be used to define new functions in the class as long as they are bounded by functions already defined. That this actually allows one to define in **Cob** functions of any polynomial length is due to the presence of the initial function $\#$. In fact, one easily verifies that for every function f in **Cob** there exists a *polynomial length bound on* f, that is, a polynomial b_f satisfying $|f(\vec{x})| \le b_f(|\vec{x}|)$.

While the latter is a necessary condition for all functions in **FPTIME**, that is, the functions computable (in binary) on a Turing machine in polynomial time (in the binary length of the input), Cobham showed that the polynomial-time computable functions are precisely the functions definable in **Cob**.

Theorem 2.1 ([12]). **Cob = FPTIME**

From a programming point of view, function algebras like **Cob** are not practically appealing because they cannot be used as a construction kit: Whenever a recursion is performed, one is prompted with a proof that the computed function is bounded by some function already constructed.

Building on work of Simmons [27] and Leivant [16, 17], Bellantoni and Cook [4] were the first to give a purely functional characterization of **FPTIME** that does away with the "explicit" reference to the growth rate of functions defined by (BRN) in Cobham's class. In fact, this "explicit" reference can be made "implicit" by ensuring the following **principle (P-BC)**: *Computed values in recursions must not control other recursions* (cf. [21], [23]).

That principle led to the well-known function algebra **BC** [4] which actually can be used as a *construction kit*, since all restrictions are of purely syntactical nature. In **BC**, each function is written in the form $f(\vec{x};\ \vec{y})$, thus bookkeeping the *normal* input positions, $\vec{x}$, which may control a recursion, and those (*safe*), $\vec{y}$, which do not. This simple bookkeeping allows us to implement (P-BC): A function $f(y, \vec{x};\ \vec{a})$ is defined by *safe recursion* from $g(\vec{x};\ \vec{a}), h_0(u, \vec{x};\ \vec{a}, v)$, and $h_1(u, \vec{x};\ \vec{a}, v)$, denoted by $f = \mathrm{srn}(g, h_0, h_1)$, if for all $y, \vec{x}, \vec{a}$,

$$\begin{aligned} f(0, \vec{x};\ \vec{a}) &= g(\vec{x};\ \vec{a}) \\ f(\mathrm{S}_i(y), \vec{x};\ \vec{a}) &= h_i(y, \vec{x};\ \vec{a}, f(y, \vec{x};\ \vec{a})) \qquad \text{for } \mathrm{S}_i(y) \ne 0. \end{aligned}$$

Enforcing the above principle when composing functions of given ones, a function $f(\vec{x};\ \vec{a})$ is defined by *safe composition* from functions $g(\vec{u};\ \vec{v}), \vec{h}(\vec{x};\ )$, and

$\vec{j}(\vec{x};\ \vec{a})$, denoted by $f = \mathrm{scomp}(g, \vec{h}, \vec{j})$, if for all $\vec{x}, \vec{a}$,

$$f(\vec{x};\ \vec{a}) = g(\vec{h}(\vec{x};\ ); \vec{j}(\vec{x};\ \vec{a})).$$

Of course, now all *initial functions* must be written in a ramified form, too. These are the functions 0, $\mathrm{s}_0(\ ; y)$, $\mathrm{s}_1(\ ; y)$, $\pi_i^{n,m}(\vec{x};\ \vec{y})$, $\mathrm{p}(\ ; y)$, and $\mathrm{case}(\ ; x, y, z)$, where the latter is defined in Section 1. The function $\mathrm{p}(\ ; y)$ is the ramified form of the *binary predecessor* P satisfying $\mathrm{P}(x) = \lfloor \frac{x}{2} \rfloor$, and thus corresponds to the operation of chopping off the lowest order bit, if any.

Note that the projections $\pi_i^{n,m}(x_1, \ldots, x_n;\ x_{n+1}, \ldots, x_{n+m}) = x_i$, for $1 \le i \le n+m$, are the only initial functions with normal input positions. It is their presence that is in charge of the *safe-to-normal property*, **(S3)**, stated in Section 1. To see this, let $f(\vec{x};\ \vec{y}, \vec{z})$ be in **BC**, say $\vec{x} = x_1, \ldots, x_l$, $\vec{y} = x_{l+1}, \ldots, x_n$ with $n := l+m$, and $\vec{z} = x_{n+1}, \ldots, x_s$ with $s := n+r$. Then by scomp we obtain

$$\mathbf{SN}(f)(\vec{x}, \vec{y};\ \vec{z}) = f(\pi_1^{n,0}(\vec{x}, \vec{y};\ ), \ldots, \pi_l^{n,0}(\vec{x}, \vec{y};\ );\ \pi_{l+1}^{n,r}(\vec{x}, \vec{y};\ \vec{z}), \ldots, \pi_s^{n,r}(\vec{x}, \vec{y};\ \vec{z})).$$

In particular, this shows that normal variables may occur in any safe position in the right-hand side of any defining equation according to scomp.

Furthermore, note that both $\vec{h}(\vec{x};\ )$ and $\vec{j}(\vec{x};\ \vec{a})$ in scheme scomp may be empty function lists. Thus, all *n-ary constant functions* $C_a^n(\vec{x};\ \vec{y}) = a$ can be defined in **BC**: $C_0^n(\vec{x};\ \vec{y}) = 0$, and inductively for $2 \cdot a + i \ge 1$, $C_{2a+i}^n(\vec{x};\ \vec{y}) = s_i(\ ;\ C_a^n(\vec{x};\ \vec{y}))$. As a consequence, every constant a may occur in the right-hand side of any defining equation according to scomp or srn.

Altogether, the function algebra **BC** can be stated as

$$\mathbf{BC} := [0, \mathrm{s}_0, \mathrm{s}_1, \pi, \mathrm{p}, \mathrm{case};\ \mathrm{scomp}, \mathrm{srn}]$$

where π denotes the set of all ramified projections.

This function algebra is a prominent example of a *ramified algebra*, and as done here, for the remainder we will adopt the convention that ramified versions of functions written in capital letters, like S_i, P or BIT, are written in small letters, like s_i, p or bit, and if not explicitly stated otherwise, we tacitly assume that they have safe input positions only.

The benefit of ramification can be seen by the fact, verified by a straightforward induction on the structure of functions in **BC**, that for every function $f(\vec{x};\ \vec{y})$ there exists a *poly-max length bound*, that is, a polynomial q_f satisfying

$$|f(\vec{x}; \vec{y})| \le q_f(|\vec{x}|) + \max(|\vec{y}|).$$

Using this **poly-max length bounding**, every recursion in **BC** can be written as bounded recursion in Cobham's class, implying $\mathbf{BC} \subseteq \mathbf{Cob}$. The converse holds by simulating the functions of **Cob** in **BC**, and that brings us back to the main topic of the present research.

Theorem 2.2 ([4]). **BC = FPTIME**

Rounding off this section, we prove property **(S2)** stated in Section 1. First note that the *shift-left function* $\mathrm{shl}(x;\ y) = 2^{|x|} \cdot y$ is defined by srn as follows:

$$\begin{aligned} \mathrm{shl}(0;\ y) &= \pi_1^{0,1}(\ ; y) \\ \mathrm{shl}(\mathrm{S}_i(x);\ y) &= s_0(\ ; \mathrm{shl}(x;\ y)) \quad \text{for } \mathrm{S}_i(x) \ne 0 \end{aligned}$$

As $2^{(|x|+1)\cdot|y|} = 2^{|y|}\cdot 2^{|x|\cdot|y|}$, the *smash function* $\#(x,y;\ ) = 2^{|x|\cdot|y|}$ is defined by

$$\begin{aligned}\#(0,y;\ ) &= 1\\ \#(\mathrm{S}_i(x),y;\ ) &= \mathrm{shl}(\pi_2^{2,0}(x,y;\ );\ \#(x,y;\ )) \qquad \text{for } \mathrm{S}_i(x)\neq 0.\end{aligned}$$

Now, to prove **(S2)**, we proceed by induction on the structure of polynomials $p(\vec{x})$ in $\mathbb{N}[\vec{x}]$. If $p(x_1,\dots,x_n)$ is x_i or c, then $W_{x_i}(\vec{x};\ ) := \mathrm{shl}(\pi_i^{n,0}(\vec{x};\ );\ 1)$ and $W_c(\vec{x};\ ) := C^n_{2^c}(\vec{x};\ )$, respectively, will do. Otherwise $p(\vec{x})$ is $p_1(\vec{x})\circ p_2(\vec{x})$ with $\circ\in\{+,\cdot\}$, and using $x+y, x\cdot y \le (x+1)\cdot(y+1)$ and $|2^x| = x+1$, we inductively define the required function $W_p(\vec{x};\ )$ by safe composition as follows:

$$W_p(\vec{x};\ ) := \#(s_1(\ ;W_{p_1}(\vec{x};\ )), s_1(\ ;W_{p_2}(\vec{x};\ ));\ )$$

3 The variant $\mathbf{BC}'$ and the step-by-step comparison technique

In this section, we will give a simplified proof of $\mathbf{BC}' = \mathbf{Cob}$, for the following variant $\mathbf{BC}'$ of Bellantoni and Cook's function algebra (cf. Section 1 for bcase).

$$\mathbf{BC}' := [0, \mathrm{s}_0, \mathrm{s}_1, \pi, \mathrm{p}, \mathrm{bcase};\ \mathrm{scomp}, \mathrm{srn}]$$

Theorem 3.1. $\mathbf{BC}' = \mathbf{FPTIME}$.

Proof. $\boxed{\mathbf{Cob}\subseteq\mathbf{BC}'}$ Following the simulation method (**S1**) stated above, we only consider the crucial case $f = \mathrm{BRN}(g,h_0,h_1,B)$, assuming inductively simulations $g', h_0', h_1' \in \mathbf{BC}'$ and witnesses p_g, p_{h_0}, p_{h_1}. As usual, the witness for f is defined by $p_f(y,\vec{x}) := (p_{h_0}+p_{h_1})(y,\vec{x},b_f(y,\vec{x}))+p_g(\vec{x})+2y+1$ for some polynomial length bound b_f on f. Thus, by monotonicity of polynomials, we have that $(*)$ $|w|\ge p_{h_j}(|\mathrm{P}^i(y),\vec{x},f(\mathrm{P}^i(y),\vec{x})|)$ whenever $|w|\ge p_f(|y,\vec{x}|)$. Now, for any $y,i\in\mathbb{N}$, let

$$y\{i\} := \mathrm{P}^i(y)$$

be the *y-section* up to i. That is, for given $y = (b_{l-1}\cdots b_0)_2$ with $\mathrm{bin}(y) = b_{l-1}\cdots b_0$, we have $y\{i\} = (b_{l-1}\cdots b_i)_2$, and $y\{i\} \bmod 2 = b_i$ for $i<|y|$. Thus, by unfolding the recursion we obtain the following steps:

$$\begin{array}{llr}
f(y,\vec{a}) = h_{y\{0\} \bmod 2}(y\{1\},\vec{a}, & & \text{step } 1\\
\quad\ddots & & \vdots\\
\qquad h_{y\{i\dot{-}1\} \bmod 2}(y\{i\},\vec{a}, & & \text{step } i\\
\qquad\quad\ddots & & \vdots\\
\qquad\qquad h_{y\{|y|\dot{-}1\} \bmod 2}(y\{|y|\},\vec{a}, & & \text{step } |y|\\
\qquad\qquad\qquad g(\vec{a}))\cdots)\cdots) & & \text{step } |y|+1
\end{array}$$

We will define a simulation $f'\in\mathbf{BC}'$ by

$$f'(w;\ y,\vec{a}) := \hat{f}(w,w;\ y,\vec{a})$$

where $\hat{f} := \mathrm{srn}(0, \hat{h}, \hat{h})$ is defined by safe recursion from the zero function and some $\hat{h} \in \mathbf{BC}'$. Again, unfolding the recursion yields the following $\hat{f}$-steps:

$$\begin{array}{rl}
\hat{f}(w, w;\ y, \vec{a}) = \hat{h}(\mathrm{P}^1(w), w;\ y, \vec{a}, & \text{step } 1 \\
\ddots & \vdots \\
\hat{h}(\mathrm{P}^i(w), w;\ y, \vec{a}, & \text{step } i \\
\ddots & \vdots \\
\hat{h}(\mathrm{P}^{|y|}(w), w;\ y, \vec{a}, & \text{step } |y| \\
\hat{h}(\mathrm{P}^{|y|+1}(w), w;\ y, \vec{a}, & \text{step } |y|+1 \\
\cdots (0)) \cdots) \cdots) & \text{step } > |y|+1
\end{array}$$

Thus, for $f(y, \vec{a}) = \hat{f}(w, w;\ y, \vec{a})$ whenever $|w| \geq p_f(|y, \vec{a}|)$, using the I.H. for g, h_0, h_1 – recall $(*)$ – a stepwise comparison yields the following requirements:

$$\begin{array}{rll}
\hat{h}(\mathrm{P}^i(w), w;\ y, \vec{a}, v_i) = & h'_{y\{i \dot{-} 1\} \bmod 2}(w;\ y\{i\}, \vec{a}, v_i) & \text{in steps } 1 \leq i \leq |y| \\
\hat{h}(\mathrm{P}^{|y|+1}(w), w;\ y, \vec{a}, v_{|y|+1}) = & g'(w;\ \vec{a}) & \text{in step } |y|+1
\end{array}$$

where $v_i := f(\mathrm{P}^i(y), \vec{a})$ for $i = 1, \ldots, |y|+1$. Now, defining $\ominus(u;\ v) := \mathrm{P}^{|u|}(v)$ by (srn), and hence $| \ominus (u;\ v)| = |v| \dot{-} |u|$, by safe composition we obtain the following y-section implementation in $\mathbf{BC}'$.

$$Y(\hat{w}, w;\ y) := \ominus(\mathbf{SN}(\ominus)(\hat{w}, w;\ );\ y) = \mathrm{P}^{|w| \dot{-} |\hat{w}|}(y) = y\{|w| \dot{-} |\hat{w}|\}$$

In fact, for *sufficiently large* w, that is, for $|w| \geq p_f(|y, \vec{a}|)$, one has that

$$Y(\mathrm{P}^i(w), w;\ y) = \begin{cases} y\{i\} & \text{if } i \leq |y| \\ 0 & \text{if } |y| \leq i \leq |w| \end{cases}$$
$$Y(\mathrm{S}_1(\mathrm{P}^i(w)), w;\ y) = y\{i \dot{-} 1\} > 0 \quad \text{for } 1 \leq i \leq |y|.$$

Thus, using function bcase above, function $\hat{h}$ can be defined in $\mathbf{BC}'$ as follows:

$$\begin{array}{rl}
\hat{h}(\hat{w}, w;\ y, \vec{a}, v) := \mathrm{bcase}(\ ; & Y(\mathrm{s}_1(\ ;\hat{w}), w;\ y), \\
& g'(w;\ \vec{a}), \\
& h'_0(w;\ Y(\hat{w}, w;\ y), \vec{a}, v), \\
& h'_1(w;\ Y(\hat{w}, w;\ y), \vec{a}, v))
\end{array}$$

To see this, for steps $1 \leq i \leq |y|$ (and w sufficiently large), we obtain as required, with $T_b := h'_b(w;\ y\{i\}, \vec{a}, v_i)$,

$$\begin{aligned}
\hat{h}(\mathrm{P}^i(w), w;\ y, \vec{a}, v_i) &= \mathrm{bcase}(\ ; y\{i \dot{-} 1\}, g'(w;\ \vec{a}), T_0, T_1) \\
&= h'_{y\{i \dot{-} 1\} \bmod 2}(w;\ y\{i\}, \vec{a}, v_i) \qquad \text{as } y\{i \dot{-} 1\} > 0,
\end{aligned}$$

and $\hat{h}(\mathrm{P}^{|y|+1}(w), w;\ y, \vec{a}, v_{|y|+1}) = \mathrm{bcase}(\ ; 0, g'(w;\ \vec{a}), \cdots, \cdots) = g'(w;\ \vec{a})$.

The converse $\mathbf{BC}' \subseteq \mathbf{Cob}$ follows by a straightforward induction on the structure of $f(\vec{x};\ \vec{a})$ in $\mathbf{BC}'$, using polymax length bounding to turn any safe recursion on notation into a bounded recursion in $\mathbf{Cob}$ (cf. [4] or [20], [22]). □

4 Clote's function algebra CLO and its variant CLO′

In this section, we first recall Clote's [10, 11] function algebra, **CLO**, that characterizes the class **NC** of functions computable by uniform circuit families of polynomial size and poly-logarithmic depth. Then we consider a variant $\mathbf{CLO}'$ due to Bellantoni [3], and prove that these classes coincide.

To define **CLO**, we need two more schemes and the function BIT satisfying $\mathrm{BIT}(m,i)=b_i$ if $\mathrm{bin}(m)=b_{l-1}\ldots b_0$ and $i<l$, and $\mathrm{BIT}(m,i)=0$ otherwise.

A function f is defined by *weak bounded recursion on notation* from functions g,h_0,h_1,B, denoted by $f:=\mathrm{WBRN}(g,h_0,h_1,B)$, if it satisfies $f(y,\vec{a})=F(|y|,\vec{a})$, for $F=\mathrm{BRN}(g,h_0,h_1,B)$.

Furthermore, a function f is defined by *concatenation recursion on notation* from functions g,h_0,h_1, denoted by $f:=\mathrm{CRN}(g,h_0,h_1)$, if for all $y,\vec{a}$,

$$\begin{aligned} f(0,\vec{a}) &= g(\vec{a}) \\ f(\mathrm{S}_i(y),\vec{a}) &= \mathrm{S}_{h_i(y,\vec{a}) \bmod 2}(f(y,\vec{a})) \quad \text{for } \mathrm{S}_i(y)\neq 0. \end{aligned}$$

Clote [10, 11] was the first to give a function-algebraic characterization of **NC** through his algebra

$$\mathbf{CLO} := [0,\mathrm{S}_0,\mathrm{S}_1,\Pi,|\cdot|,\mathrm{BIT},\#;\ \mathrm{COMP},\mathrm{CRN},\mathrm{WBRN}].$$

Theorem 4.1 ([10, 11])**.** **NC** $=$ **CLO**

In [3, p. 73] Bellantoni pointed out that the same class is obtained when replacing scheme (WBRN) with the following streamlined variant.

Definition 4.2. A function f is defined by WBRN' from functions g,h,B, denoted by $f:=\mathrm{WBRN}'(g,h,B)$, if for all $y,\vec{a}$,

$$\begin{aligned} f(0,\vec{a}) &= g(\vec{a}) \\ f(y,\vec{a}) &= h(y,\vec{a},f(\mathrm{H}(y),\vec{a})) \quad \text{for } y\neq 0 \\ f(y,\vec{a}) &\leq B(y,\vec{a}) \end{aligned}$$

where the *half function* H is defined by $\mathrm{H}(m) := \lfloor m/2^{\lceil |m|/2\rceil}\rfloor$.

The behavior of function H can be easily expressed on binary representations:

$$\begin{aligned} \mathrm{H}((b_{2n-1}\cdots b_0)_2) &= (b_{2n-1}\cdots b_n)_2 && \text{even length} \\ \mathrm{H}((b_{2n}\cdots b_0)_2) &= (b_{2n}\cdots b_{n+1})_2 && \text{odd length} \end{aligned}$$

In fact, defining the class $\mathbf{CLO}'$ by

$$\mathbf{CLO}' := [0,\mathrm{S}_0,\mathrm{S}_1,\Pi,|\cdot|,\mathrm{BIT},\#;\ \mathrm{COMP},\mathrm{CRN},\mathrm{WBRN}']$$

one obtains the following result.

Theorem 4.3. $\mathbf{CLO} = \mathbf{CLO}'$

As the proof sketch in [3, footnote on p. 73] of either inclusion is wrong[2], we give a proof of the above theorem – the first one according to our knowledge –, using the above step-by-step comparison technique.

[2] Any $f=\mathrm{WBRN}(g,h_0,h_1,B)$ is claimed to be identical to $f' := \mathrm{WBRN}'(g,h',B)$, where $h'(x,\vec{v},z) := h_{|x| \bmod 2}(|x|-1,\vec{v},z)$. But, for example, $f(5,\vec{v}) = F(|5|,\vec{v}) = F(\mathrm{S}_1(\mathrm{S}_1(0)),\vec{v}) = h_1(1,\vec{v},h_1(0,\vec{v},g(\vec{v})))$, while $f'(5,\vec{v})=h'(5,\vec{v},h'(1,\vec{v},g(\vec{v})))=h_{|5| \bmod 2}(|5|-1,\vec{v},h_{|1| \bmod 2}(|1|-1,\vec{v},g(\vec{v})))=h_1(2,\vec{v},h_1(0,\vec{v},g(\vec{v})))$.
For the converse, any $f'=\mathrm{WBRN}'(g,h,B)$ is claimed to be definable by $f(u,\vec{v}) := \hat{f}(u,u,\vec{v})$, where $\hat{f} := \mathrm{WBRN}(g,h_0,h_1,B)$, and $h_i(u,x,\vec{v},z) := h(\mathrm{E}(u,x),\vec{v},z)$, with $\mathrm{E}(u,x)=x \bmod 2^u$, being the low-order u bits of x, assuming $u\leq |x|$. But, e.g., $f'(5,\vec{v}) = h(5,\vec{v},h(1,\vec{v},g(\vec{v})))$, while $f(5,\vec{v}) = \hat{f}(5,5,\vec{v}) = \hat{F}(|5|,5,\vec{v}) = \hat{F}(\mathrm{S}_1(\mathrm{S}_1(0))) = h(\mathrm{E}(1,5),\vec{v},h(\mathrm{E}(0,5),\vec{v},g(\vec{v}))) = h(1,\vec{v},h(0,\vec{v},g(\vec{v})))$.

The key observation is that the recursion depths of both schemes WBRN and WBRN$'$ are identical, and hence step-by-step simulations are possible. To see this, we first define the *half norm* of y, denoted by $\|y\|_H$, that represents the recursion depth of an WBRN$'$ instance at y.

$$\|y\|_H := \min\{k \in \mathbb{N} \mid \mathrm{H}^k(y) = 0\}$$

As $|(|y|)|$ represents the recursion depth of an WBRN instance at y, the above claimed equality on recursion depth then follows by the next lemma.

Lemma 4.4 (Half Norm)**.** *For any $y \in \mathbb{N}$, one has*

$$(0) \qquad \|y\|_H = |(|y|)|$$

(and so we just write $||y||$ for $\|y\|_H$).

Proof. We proceed by course-of-values induction. As $\|0\|_H = 0 = |(|0|)|$, consider any $y > 0$, say $|y| = 2n+i$, $i \in \{0,1\}$. Then $|\mathrm{H}(y)| = n$ by definition, and we obtain $||y||_H = ||\mathrm{H}(y)||_H + 1 \overset{\text{(I.H.)}}{=} |(|\mathrm{H}(y)|)| + 1 = |n| + 1 = |2n+i| = |(|y|)|$. □

Further facilitating the proof structure, we provide some auxiliary functions.

Lemma 4.5 (Auxiliary functions)**.** *All of the following functions belong to both* **CLO** *and* **CLO**$'$*:*

(a) *the* most significant part, MSP*, satisfying* $\mathrm{MSP}(m,n) = \lfloor \frac{m}{2^n} \rfloor = \mathrm{P}^n(m)$,

(b) *function* DROP*, satisfying* $\mathrm{DROP}(m,n) = \lfloor \frac{m}{2^{|n|}} \rfloor = \mathrm{P}^{|n|}(m)$,

(c) *the* binary predecessor, P*, satisfying* $\mathrm{P}(m) = \lfloor \frac{m}{2} \rfloor$,

(d) *the* unary conditional, COND*, satisfying* $\mathrm{COND}(x,y,z) := \begin{cases} y & \text{if } x = 0 \\ z & \text{else,} \end{cases}$

(e) *the* binary conditional, CASE*, satisfying* $\mathrm{CASE}(x,y,z) = \mathrm{case}(\,;x,y,z)$,

(f) *and function* half, H*, satisfying* $\mathrm{H}(m) = \lfloor m/2^{\lceil |m|/2 \rceil} \rfloor$.

Proof. As for part (a), observe that MSP can be defined by (CRN), since

$$\begin{aligned} \mathrm{MSP}(0,n) &= 0 \\ \mathrm{MSP}(\mathrm{S}_b(m),n) &= \mathrm{S}_{\mathrm{BIT}(\mathrm{S}_b(m),n)}(\mathrm{MSP}(m,n)) \end{aligned}$$

for $\mathrm{S}_b(m) \neq 0$. Thus, both parts (b) and (c) follow from (a), since

$$\begin{aligned} \mathrm{DROP}(m,n) &= \mathrm{MSP}(m,|n|) \\ \mathrm{P}(m) &= \mathrm{MSP}(m,1). \end{aligned}$$

As for (d), first define function $F := \mathrm{BRN}(g,h,h,b)$ from both **CLO** and **CLO**$'$ functions $g(y,z) = y$, $h(x,y,z,v) = z$, and $b(x,y,z) = 2^{|z|} \cdot y + z$, where b can be defined by (CRN). Then we already have $F = \mathrm{COND}$. Thus, as $|x| = 0 \Leftrightarrow x = 0$, we can use (WBRN) to define $\mathrm{COND}(x,y,z) = F(|x|,y,z)$ as a function in **CLO**. As well, since $||x|| = 0 \Leftrightarrow x = 0$, we obtain $\mathrm{COND} = \mathrm{WBRN}'(g,h,b) \in$ **CLO**$'$.

Now, part (e) follows from (d), since $\mathrm{CASE}(x,y,z) = \mathrm{COND}(\mathrm{BIT}(x,0),y,z)$, and finally, (f) follows from parts (a) – (e), since

$$\mathrm{H}(m) = \mathrm{CASE}(|m|, \mathrm{DROP}(m, \mathrm{P}(|m|)), \mathrm{DROP}(m, \mathrm{P}(|\mathrm{S}_1(m)|))).$$

□

Proof. $\boxed{\mathbf{CLO} \subseteq \mathbf{CLO}'}$. It suffices to consider any $f := \mathrm{WBRN}(g, h_0, h_1, B)$ in **CLO**, assuming $g, h_0, h_1, B \in \mathbf{CLO}'$. We shall give a *direct simulation* $f' \in \mathbf{CLO}'$ of f, that is, $f(y, \vec{a}) = f'(y, \vec{a})$ for all $y, \vec{a}$, where

$$f'(y, \vec{a}) := \hat{f}(y, y, \vec{a}) \text{ with } \hat{f} := \mathrm{WBRN}'(\hat{g}, \hat{h}, \hat{B})$$

for some $\hat{g}, \hat{h}, \hat{B} \in \mathbf{CLO}'$. Here, the *y-section* is defined by

$$y\{i\} := \mathrm{P}^i(|y|). \tag{1}$$

Referring to (0), suppose that $|y| = (b_{||y|| \dot{-} 1} \cdots\ b_0)_2$. Then $y\{i\} = (b_{||y|| \dot{-} 1} \cdots b_i)_2$, and $y\{i\} \bmod 2 = b_i$ for $i < ||y||$. Therefore, by unfolding the recursions we obtain the following steps in comparison:

$$\begin{array}{lll}
f(y, \vec{a}) = F(|y|, \vec{a}) & \stackrel{!}{=} f'(y, \vec{a}) & \text{steps} \\
\quad = h_{b_0}(y\{1\}, \vec{a}, & = \hat{h}(\mathrm{H}^0(y), y, \vec{a}, & 1 \\
\qquad \ddots & \qquad \ddots & \vdots \\
\qquad h_{b_{i \dot{-} 1}}(y\{i\}, \vec{a}, & \quad \hat{h}(\mathrm{H}^{i \dot{-} 1}(y), y, \vec{a}, & i \\
\qquad\quad \ddots & \qquad\quad \ddots & \vdots \\
\qquad\quad h_{b_{||y|| \dot{-} 1}}(y\{||y||\}, \vec{a}, & \qquad \hat{h}(\mathrm{H}^{||y|| \dot{-} 1}(y), y, \vec{a}, & ||y|| \\
\qquad\qquad g(\vec{a})) \cdots) \cdots) & \qquad\quad \hat{g}(y, \vec{a})) \cdots) \cdots) & ||y|| + 1
\end{array}$$

Thus, a stepwise comparison yields the requirement

$$\hat{h}(\mathrm{H}^{i \dot{-} 1}(y), y, \vec{a}, v) = h_{y\{i \dot{-} 1\} \bmod 2}(y\{i\}, \vec{a}, v) \quad \text{in steps } 1 \le i \le ||y|| \tag{2}$$

and step $||y|| + 1$ implies that $\hat{g}$ can be defined by $\hat{g}(y, \vec{a}) := g(\vec{a})$.

By (1) the y-section implementation in $\mathbf{CLO}'$ (below) we need this time is

$$Y(w, y) := \mathrm{P}^{||y|| \dot{-} ||w||}(|y|) = y\{||y|| \dot{-} ||w||\}.$$

As (0) implies $||\mathrm{H}^i(y)|| = ||y|| \dot{-} i$, we conclude that

$$Y(\mathrm{H}^i(y), y) = y\{i\} \quad \text{for } i \le ||y||. \tag{3}$$

Thus, the required function $\hat{h}$ satisfying (2) can be defined by

$$\begin{aligned}
\hat{h}(w, y, \vec{a}, v) &:= h_{Y(w,y) \bmod 2}(Y(H(w), y), \vec{a}, v) \\
&= \mathrm{CASE}(Y(w, y), h_0(Y(\mathrm{H}(w), y), \vec{a}, v), h_1(Y(\mathrm{H}(w), y), \vec{a}, v)).
\end{aligned}$$

In fact, (2) is true of $\hat{h}$, since (3) implies for $i \le ||y||$:

$$\begin{aligned}
\hat{h}(\mathrm{H}^{i \dot{-} 1}(y), y, \vec{a}, v) &= h_{Y(\mathrm{H}^{i \dot{-} 1}(y), y) \bmod 2}(Y(\mathrm{H}^i(y), y), \vec{a}, v) \\
&= h_{y\{i \dot{-} 1\} \bmod 2}(y\{i\}, \vec{a}, v)
\end{aligned}$$

For $\hat{h} \in \mathbf{CLO}'$, it remains to define in $\mathbf{CLO}'$ function $Y(w, y) = \mathrm{P}^{||y|| \dot{-} ||w||}(|y|)$. First we define by (WBRN$'$) a function $\ominus'$ satisfying $||\ominus'(w, y)|| = ||y|| \dot{-} ||w||$.

$$\begin{aligned}
\ominus'(0, y) &:= y \\
\ominus'(w, y) &:= \mathrm{H}(\ominus'(\mathrm{H}(w), y)) \quad \text{for } w \neq 0
\end{aligned}$$

To see this, observe inductively that for $w \neq 0$, $|| \ominus' (w, y)|| = ||\mathrm{H}(\ominus'(\mathrm{H}(w), y))||$ $= || \ominus' (\mathrm{H}(w), y)|| \dot{-} 1 = (||y|| \dot{-} ||\mathrm{H}(w)||) \dot{-} 1 = (||y|| \dot{-} (||w|| \dot{-} 1)) \dot{-} 1 = ||y|| \dot{-} ||w||$, as $||w|| \geq 1$. Note that the outmost use of $\mathrm{H} \in \mathbf{CLO'}$ in the above definition is not part of the (WBRN$'$) scheme. Now, we conclude the required definition of the y-section implementation in $\mathbf{CLO'}$ as follows:

$$Y(w, y) := \mathrm{MSP}(|y|, \ominus'(w, y))$$

To complete the definition of $\hat{f}$, it still remains to define a bound $\hat{B} \in \mathbf{CLO'}$, and here we run into a problem. To see this, first observe that one can show:

$$||w|| \leq ||y|| \implies \hat{f}(w, y, \vec{x}) = F(Y(w, y), \vec{x}) \leq B(Y(w, y), \vec{x}) \tag{4}$$

But $Y(w, y) = |y|$ whenever $||w|| \geq ||y||$, hence $\hat{h}(w, y, \vec{a}, v) = h_{|y| \bmod 2}(\mathrm{P}(|y|), \vec{a}, v)$, which in turn implies that $\hat{f}(w, y, \vec{a})$ is obtained by iterating $||w|| \dot{-} (||y|| - 1)$ times function $h_{|y| \bmod 2}(\mathrm{P}(|y|), \vec{a}, \cdot)$ on $f(y, \vec{a})$. Thus, we cannot guarantee that $\hat{f}$ can be bounded by a function in $\mathbf{CLO'}$. To resolve that problem, by use of the functions $\mathrm{COND}, \ominus'$ (both in $\mathbf{CLO'}$) and $|\cdot|$, we simply modify $\hat{h}$ such that it returns 0 whenever $||w|| \dot{-} ||y|| > 0$. Thus by (4), setting $\hat{B}(w, y, \vec{x}) := B(Y(w, y), \vec{x})$ will do.

$\boxed{\mathbf{CLO'} \subseteq \mathbf{CLO}}$ It suffices to consider any $f := \mathrm{WBRN'}(g, h, B)$, assuming inductively $g, h, B \in \mathbf{CLO}$. Accordingly, the *y-section* we need is defined by

$$y\{i\} := \mathrm{H}^{i \dot{-} 1}(y). \tag{5}$$

Again, we will give a *direct simulation* $f' \in \mathbf{CLO}$ of f (see above), where

$$f'(y, \vec{a}) := \hat{f}(y, y, \vec{a}) \text{ with } \hat{f} := \mathrm{WBRN}(\hat{g}, \hat{h}, \hat{h}, \hat{B})$$

for some $\hat{g}, \hat{h}, \hat{B} \in \mathbf{CLO}$. By unfolding the recursions, we obtain the following steps:

$$\begin{array}{lll}
f(y, \vec{a}) & \stackrel{!}{=} \hat{f}(y, y, \vec{a}) = \hat{F}(|y|, y, \vec{a}) & \text{steps} \\
\quad = h(y\{1\}, \vec{a}, & = \hat{h}(\mathrm{P}^1(|y|), y, \vec{a}, & 1 \\
\quad\quad \ddots & \quad \ddots & \vdots \\
\quad\quad h(y\{i\}, \vec{a}, & \quad \hat{h}(\mathrm{P}^i(|y|), y, \vec{a}, & i \\
\quad\quad\quad \ddots & \quad\quad \ddots & \vdots \\
\quad\quad\quad h(y\{||y||\}, \vec{a}, & \quad\quad \hat{h}(\mathrm{P}^{||y||}(|y|), y, \vec{a}, & ||y|| \\
\quad\quad\quad\quad g(\vec{a})) \cdots) \cdots) & \quad\quad\quad \hat{g}(y, \vec{a})) \cdots) \cdots) & ||y|| + 1
\end{array}$$

Thus, a stepwise comparison yields the requirement

$$\hat{h}(\mathrm{P}^i(|y|), y, \vec{a}, v) = h(y\{i\}, \vec{a}, v) \quad \text{in steps } 1 \leq i \leq ||y|| \tag{6}$$

and again, step $||y||+1$ shows that $\hat{g}$ can be defined by $\hat{g}(y, \vec{a}) := g(\vec{a})$.

By (5), (6) the y-section implementation in $\mathbf{CLO}$ we need this time is

$$Y(w, y) := \mathrm{H}^{||y|| \dot{-} (|w|+1)}(y) = y\{||y|| \dot{-} |w|\} \tag{7}$$

In fact, since $|\mathrm{P}^i(|y|)| = ||y|| \dot{-} i$, we conclude from (7) that

$$Y(\mathrm{P}^i(|y|), y) = y\{i\} \quad \text{for } i \leq ||y||.$$

Thus, we obtain the required function $\hat{h} \in \mathbf{CLO}$ by setting

$$\hat{h}(w, y, \vec{a}, v) := h(Y(w, y), \vec{a}, v)$$

provided that function Y is definable in $\mathbf{CLO}$. To see that, using $\mathrm{H}, \mathrm{DROP} \in \mathbf{CLO}$, and $|w| < |x| \Leftrightarrow |\mathrm{S}_1(w)| \leq |x| \Leftrightarrow \mathrm{DROP}(\mathrm{S}_1(w), x) = \mathrm{P}^{|x|}(\mathrm{S}_1(w)) = 0$, we first define by (BRN) a function G in $\mathbf{CLO}$, satisfying $G(x, y, w) = \mathrm{H}^{|x| \dot{-} |w|}(y)$.

$$\begin{aligned} G(0, y, w) &:= y \\ G(\mathrm{S}_b(x), y, w) &:= \mathrm{COND}(\mathrm{DROP}(\mathrm{S}_1(w), x), \mathrm{H}(G(x, y, w)), y) \end{aligned}$$

for $\mathrm{S}_b(x) \neq 0$. Then define $\tilde{Y}(x, y, w) := G(|x|, y, w) = H^{||x|| \dot{-} |w|}(y)$ by (WBRN), and conclude the y-section implementation in $\mathbf{CLO}$ by setting

$$Y(w, y) := \tilde{Y}(y, y, \mathrm{S}_1(w)).$$

To complete the definition of $\hat{f}$, it remains to define a bound $\hat{B} \in \mathbf{CLO}$, and again we run into a problem. To see this, first observe that one can show:

$$(8) \qquad |w| \leq ||y|| \implies \hat{F}(w, y, \vec{x}) = f(Y(w, y), \vec{x}) \leq B(Y(w, y), \vec{x})$$

But $Y(w, y) = y$ whenever $|w| \geq ||y||$, hence $\hat{h}(w, y, \vec{a}, v) = h(y, \vec{a}, v)$, which in turn implies that $\hat{f}(w, y, \vec{a})$ is obtained by iterating $|w| \dot{-} (||y|| - 1)$ times function $h(y, \vec{a}, \cdot)$ on $f(y, \vec{a})$. Thus, we cannot guarantee that $\hat{f}$ can be bounded by a function in $\mathbf{CLO}$. To resolve this problem, we use the functions COND, $|\cdot|$ and G' below (all of which are in $\mathbf{CLO}$) to modify $\hat{h}$ such that it returns 0 whenever $|w| \dot{-} ||y|| > 0$, and by (8) setting $\hat{B}(w, y, \vec{x}) := B(Y(w, y), \vec{x})$ then will do.

As for the required function $G' \in \mathbf{CLO}$ satisfying $|G'(y, w)| = |w| \dot{-} ||y||$, first observe that the unramified version of $\ominus$, that is, $\ominus(u, v) = \mathrm{P}^{|u|}(v)$, can be defined by (BRN) from $\mathbf{CLO}$ functions. Thus, applying (WBRN) to $\ominus$ yields the $\mathbf{CLO}$ function $G'(y, w) = \ominus(|y|, w)$, satisfying $G'(y, w) = \mathrm{P}^{||y||}(w)$. $\square$

5 Variant $\mathbf{CLO}''$ of CLO

In this section, we consider another variant of Clote's function algebra that appears in the literature ([1], [2]), the main goal being to give a higher type characterization of **NC**, building on ideas and techniques presented in [6].

Before defining that variant of $\mathbf{CLO}'$, first observe that one obtains the same class when replacing scheme (CRN) with the following *h-variant* that unlike (CRN) uses a single step function (h), and where nonzero recursion arguments are not decremented in h.

Definition 5.1. A function f is defined by the *h-variant of* CRN from functions g, h, denoted by $f := \mathrm{CRN}'(g, h)$, if for all $y, \vec{a}$,

$$\begin{aligned} f(0, \vec{a}) &= g(\vec{a}) \\ f(y, \vec{a}) &= \mathrm{S}_{h(y, \vec{a}) \bmod 2}(f(\mathrm{P}(y), \vec{a})) \quad \text{for } y \neq 0. \end{aligned}$$

Corollary 5.2 (h-variant)**.** *In the context of* $\mathbf{CLO}$ *or* $\mathbf{CLO}'$*, the h-variant* (CRN') *is equivalent to* (CRN).

Proof. Given any $f = \mathrm{CRN}(g, h_0, h_1)$, we obtain $f = \mathrm{CRN}'(g, h)$ for

$$h(w, \vec{a}) := \mathrm{CASE}(w, h_0(\mathrm{P}(w), \vec{a}), h_1(\mathrm{P}(w), \vec{a})).$$

Conversely, given any $f = \mathrm{CRN}'(g, h)$, we have $f = \mathrm{CRN}(g, h_0, h_1)$ where

$$h_b(w, \vec{a}) := h(\mathrm{S}_b(w), \vec{a})$$

□

Unlike the above corollary, the proof of $\mathbf{CLO}' \subseteq \mathbf{CLO}''$ does not come so easy, where $\mathbf{CLO}''$ results from $\mathbf{CLO}'$ by replacing scheme (CRN′) with the *g-variant* obtained from (CRN′) by setting the base function, g, to the zero function.

Definition 5.3. A function f is defined by the *g-variant of* CRN′ from function h, denoted by $f := \mathrm{CRN}''(h)$, if for all $y, \vec{a}$,

$$\begin{aligned} &f(0, \vec{a}) = 0 \\ &f(y, \vec{a}) = \mathrm{S}_{h(y,\vec{a}) \bmod 2}(f(\mathrm{P}(y), \vec{a})) \quad \text{ for } y \neq 0. \end{aligned}$$

In fact, defining the class $\mathbf{CLO}''$ by

$$\mathbf{CLO}'' := [0, \mathrm{S}_0, \mathrm{S}_1, \Pi, |\cdot|, \mathrm{BIT}, \#;\ \mathrm{COMP}, \mathrm{CRN}'', \mathrm{WBRN}']$$

one ends up with the same class of functions. In [4, p. 77] CRN is simulated by the ramified g-variant of CRN (ramified CRN″). As this construction is wrong[3], we give a proof in the corresponding unramified setting.

Theorem 5.4 (g-variant). $\mathbf{CLO}' = \mathbf{CLO}''$

Proof. As $\mathrm{CRN}''(h) = \mathrm{CRN}'(0, h)$, the inclusion "$\supseteq$" follows from Corollary 5.2. $\boxed{\mathbf{CLO}' \subseteq \mathbf{CLO}''}$ By Corollary 5.2 it suffices to consider any function $f := \mathrm{CRN}'(g, h)$, assuming inductively that $g, h \in \mathbf{CLO}''$. Accordingly, the *y-section* is defined by

$$y\{i\} := \mathrm{P}^i(y)$$

and by unfolding the recursion, we obtain the following steps:

$$\begin{array}{ll} f(y, \vec{a}) = \mathrm{S}_{h(y\{0\},\vec{a}) \bmod 2}(& \text{step } 1 \\ \quad \ddots & \quad\vdots \\ \qquad \mathrm{S}_{h(y\{i \dot{-} 1\},\vec{a}) \bmod 2}(& \text{step } i \\ \qquad\quad \ddots & \quad\vdots \\ \qquad\qquad \mathrm{S}_{h(y\{|y| \dot{-} 1\},\vec{a}) \bmod 2}(g(\vec{a}))\cdots)\cdots) & \text{step } |y| \end{array}$$

[3] To see this, consider the function $f = \mathrm{CRN}(0, C_1^1, C_1^1)$ satisfying $f(u;\ ) = 2^{|u|}$. It is claimed that for sufficiently large w, $f(u;\ ) = f'(w;\ u) := \hat{f}(w;\ w, u)$, where $h'(w;\ u) := \mathrm{case}(;\ u, h_0'(w;\ \mathrm{p}(;u)), h_1'(w;\ \mathrm{p}(;u))) = C_1^2(w;\ u) = 1$, and $\hat{f}(w;\ 0, u) := 0$, and $\hat{f}(w;\ c, u) := \mathrm{s}_{\mathrm{case}(;\ |c| \leq |u|, h'(w;\ u \bmod c), \mathrm{bit}(;\ g'(w;\ ), |c - h'(w;\ u)|))}(;\hat{f}(w;\ \mathrm{P}(c), u)) = \mathrm{s}_{\mathrm{case}(;\ |c| \leq |u|, 1, 0)}(;\hat{f}(w;\ \mathrm{P}(c), u))$ for $c \neq 0$. But $f(1) = 1$, while e.g. for $|w| = 3$ we have $f'(w;\ 1) = \hat{f}(w;\ w, 1) = \mathrm{s}_{\mathrm{case}(;\ 3 \leq |1|, 1, 0)}(;\mathrm{s}_{\mathrm{case}(;\ 2 \leq |1|, 1, 0)}(;\mathrm{s}_{\mathrm{case}(;\ 1 \leq |1|, 1, 0)}(;0))) = \mathrm{S}_0(\mathrm{S}_0(\mathrm{S}_1(0))) = 4 \neq 1$. In general, if $f(y, \vec{v}) =_2 b_{l-1} \ldots b_0$, then for sufficiently large w, $f'(w;\ y, \vec{v}) =_2 b_{l-1} \ldots b_0 0^{|w| - |f(y,\vec{v})|}$.

To achieve a step-by-step simulation with respect to $\mathrm{CRN}''(\hat{h})$ for some $\hat{h}$, we just express $g(\vec{a})$ as further steps of $\hat{h}$ that will be performed after the above $|y|$ steps. The simple idea is that any $z=(b_{l-1}\dots b_0)_2$ can be written as

$$z = \mathrm{S}_{b_0}(\dots(\mathrm{S}_{b_{l-1}}(\mathrm{S}_0^k(0))\dots) \qquad \text{for any } k\in\mathbb{N}.$$

Thus, it is natural to extend the above $|y|$ steps by further $\geq |g(\vec{a})|$ steps:

$$\begin{array}{ll}
g(\vec{a}) = \mathrm{S}_{\mathrm{BIT}(g(\vec{a}),0)}(& \text{step } |y|+1 \\
\quad\ddots & \quad\vdots \\
\qquad \mathrm{S}_{\mathrm{BIT}(g(\vec{a}),|g(\vec{a})|\dot{-}1)}(& \text{step } |y|+|g(\vec{a})| \\
\qquad\quad \mathrm{S}_0(& \text{step } |y|+|g(\vec{a})|+1 \\
\qquad\qquad\ddots & \quad\vdots \\
\qquad\qquad\quad \mathrm{S}_0(0)\cdots))\cdots) & \text{step } |y|+|g(\vec{a})|+k
\end{array}$$

In other words, for the intended bitwise step-by-step simulation we need

$$\geq\ |y|+|g(\vec{a})| \text{ steps.}$$

Of course, exactly $|y|+|g(\vec{a})|$ steps would suffice, but computing that exact value in $\mathbf{CLO}''$ is difficult. Instead, we define a function $\hat{f}(\hat{w},w,y,\vec{a}) = \mathrm{CRN}''(\hat{h})(\hat{w},w,y,\vec{a})$ by recursion on $\hat{w}$, using w as a bound on $|y|+|g(\vec{a})|$, and show that for all $y,\vec{a}$,

$$f(y,\vec{a}) = f'(y,\vec{a}) := \hat{f}(W(y,\vec{a}),W(y,\vec{a}),\vec{a}) \tag{9}$$

where W is any $\mathbf{CLO}''$ function satisfying $|W(y,\vec{a})|\geq|y|+|g(\vec{a})|$. For example, setting $W(y,\vec{a}) := \#(\mathrm{S}_1(y),\mathrm{S}_1(g(\vec{a})))$ will do, since

$$|W(y,\vec{a})| = |2^{(|y|+1)\cdot(|g(\vec{a})|+1)}| \geq |2^{|y|+|g(\vec{a})|}-1| = |y|+|g(\vec{a})|.$$

Now, a bitwise step-by-step simulation w.r.t. (9), with $w := W(y,\vec{a})$, requires

$$\hat{h}(\mathrm{P}^i(w),w,y,\vec{a}) = \begin{cases} h(y\{i\},\vec{a}) & \text{if } i<|y| \\ \mathrm{BIT}(g(\vec{a}),i\dot{-}|y|) & \text{if } |y|\leq i\leq|w|. \end{cases} \tag{10}$$

Observe that $\mathrm{BIT}(g(\vec{a}),i\dot{-}|y|)=0$ for $i\geq|y|+|g(\vec{a})|$. Accordingly, we need a y-section implementation $Y(\hat{w},w,y)$ in $\mathbf{CLO}''$ satisfying

$$Y(\hat{w},w,y) = \mathrm{P}^{|w|\dot{-}|\hat{w}|}(y). \tag{11}$$

Then (11) implies that for $i\leq|w|$:

$$\begin{aligned}
\mathrm{P}^i(y) &= Y(\mathrm{P}^i(w),w,y) \\
i<|y| &\Leftrightarrow Y(\mathrm{P}^i(w),w,y)>0 \\
i\dot{-}|y| &= |\mathrm{DROP}(\mathrm{DROP}(w,\mathrm{P}^i(w)),y)|
\end{aligned}$$

The latter follows from $|w|\dot{-}(|w|\dot{-}i)=i$ for $i\leq|w|$, and $|\mathrm{DROP}(m,n)| = |\mathrm{P}^{|n|}(m)| = |m|\dot{-}|n|$, implying $|\mathrm{DROP}(w,\mathrm{P}^i(w))|=i$ for $i\leq|w|$.

Altogether, as $\mathrm{P}^i(w)$ acts as $\hat{w}$ in $\hat{f}(\hat{w},w,y,\vec{a})$, the required function $\hat{h}$ satisfying (10) can be defined in $\mathbf{CLO}''$ by

$$\begin{aligned}
\hat{h}(\hat{w},w,y,\vec{a},v) := \mathrm{COND}(&Y(\hat{w},w,y), \\
&\mathrm{BIT}(g(\vec{a}),|\mathrm{DROP}(\mathrm{DROP}(w,\hat{w}),y)|), \\
&h(Y(\hat{w},w,y),\vec{a}))
\end{aligned}$$

and the y-section implementation Y satisfying (11) is definable in $\mathbf{CLO}''$, since

$$Y(\hat{w}, w, y) = \mathrm{P}^{|w| \dot{-} |\hat{w}|}(y) = \mathrm{DROP}(y, \mathrm{DROP}(w, \hat{w})).$$

To see that $\hat{h}, Y \in \mathbf{CLO}''$, just recall the proof of Lemma 4.5, and observe that the definition of function MSP is, in fact, by CRN$''$ in $\mathbf{CLO}''$. As a consequence, the given definitions of both functions DROP and COND show that they belong to $\mathbf{CLO}''$, too. Thus, we obtain $Y, \hat{h} \in \mathbf{CLO}''$ as claimed. □

6 Embeddings

In this final section, we consider the following ramified function algebras and prove that they all characterize $\mathbf{NC}$, facilitated by $\mathbf{CLO} = \mathbf{CLO}' = \mathbf{CLO}''$ established in the last two sections.

$$\begin{aligned}
\mathbf{2CLO} &:= [0, \mathrm{s}_0, \mathrm{s}_1, \pi, \mathrm{len}, \mathrm{bit}, \#_{\mathrm{Bel}}, \mathrm{case};\ \mathrm{scomp}, \mathrm{scrn}, \mathrm{slr}] \\
\mathbf{2NC} &:= [0, \mathrm{s}_0, \mathrm{s}_1, \pi, \mathrm{len}, \mathrm{bit}, \#_{\mathrm{Bel}}, \mathrm{case}, \mathrm{half}, \mathrm{drop};\ \mathrm{scomp}, \mathrm{scrn}', \mathrm{slr}] \\
\mathbf{2NC}' &:= [0, \mathrm{s}_0, \mathrm{s}_1, \pi, \mathrm{len}, \mathrm{bit}, \mathrm{sm}, \#_{\mathrm{AJST}}, \mathrm{case}, \mathrm{half}, \mathrm{drop};\ \mathrm{scomp}, \mathrm{scrn}', \mathrm{slr}] \\
\mathbf{2NC}'' &:= [0, \mathrm{s}_0, \mathrm{s}_1, \pi, \mathrm{len}, \mathrm{sm}, \#_{\mathrm{AJST}}, \mathrm{bcase}, \mathrm{msp};\ \mathrm{scomp}, \mathrm{scrn}'', \mathrm{slr}]
\end{aligned}$$

To explain the new components, a function $f(y, \vec{x};\ \vec{a})$ is defined by *safe logarithmic recursion* (the ramified version of (WBRN$'$) defined in Section 4) from functions $g(\vec{x};\ \vec{a})$ and $h(u, \vec{x};\ \vec{a}, v)$, denoted by $f = \mathrm{srn}(g, h)$, if for all $y, \vec{x}, \vec{a}$,

$$\begin{aligned}
f(0, \vec{x};\ \vec{a}) &= g(\vec{x};\ \vec{a}) \\
f(y, \vec{x};\ \vec{a}) &= h(y, \vec{x};\ \vec{a}, f(H(y), \vec{x};\ \vec{a})) \quad \text{for } y \neq 0.
\end{aligned}$$

The scheme (scrn) is the ramified form of (CRN$''$) defined in Section 5, except that the recursion parameter y in $f = \mathrm{scrn}(h)$ is in a safe position:

$$f(\vec{x};\ y, \vec{a}) = \mathrm{S}_{h(\vec{x};\ y, \vec{a}) \bmod 2}(f(\vec{x};\ \mathrm{P}(y), \vec{a}))$$

By contrast, scheme (scrn$'$) is just the ramified version of (CRN$''$), with y being in normal positions only. Finally, the new initial functions satisfy $\#_{\mathrm{Bel}}(w;\ a, b) = 2^{|a| \cdot |b|} \bmod 2^{|w|^2}$, $\mathrm{sm}(w;\ a, b) = 2^{|a| \cdot |b|} \bmod 2^{|w|}$, and $\#_{\mathrm{AJST}}(w;\ ) = 2^{|w|^2}$.

These function algebras should be contrasted with those of Bloch [8], namely sc(BASE) := [BASE; scomp, safe DCR] characterizing $\mathbf{NC}^1$, and vsc(BASE) := [BASE; scomp, very safe DCR] characterizing "alternating polylog time". Here BASE is a large set of initial functions, and the recursion schemes "safe" and "very safe DCR" are similar to the scheme slr. But as scheme scrn is missing in Bloch's algebras, no characterization of $\mathbf{NC}$ is obtained, because scrn is necessary to reach any level $\mathbf{NC}^k$ of the $\mathbf{NC}$ hierarchy.

Furthermore, $\mathbf{2CLO}$ was defined in [3], and $\mathbf{2NC}$ implicitly in [1]. The idea to split the smash function $\#_{\mathrm{Bel}}$ into two parts can be found in [2]; we call this algebra $\mathbf{2NC}'$. The class $\mathbf{2NC}''$, treated in [28], contains fewer base functions, and uses the following variant of safe concatenation recursion on notation $f = \mathrm{scrn}''(h)$.

Definition 6.1. A function f is defined by the *safe g-variant of* CRN$'$ from function h, denoted by $f := \mathrm{scrn}''(h)$, if for all $y, \vec{x}, \vec{a}$,

$$\begin{aligned}
f(0, \vec{x};\ \vec{a}) &= 0 \\
f(y, \vec{x};\ \vec{a}) &= \mathrm{s}_{h(\vec{x};\ y, \vec{a}) \bmod 2}(f(\mathrm{P}(y), \vec{x};\ \vec{a})) \quad \text{for } y \neq 0.
\end{aligned}$$

In contrast to scheme (scrn) in [3], the recursion parameter here appears in a normal position of f – in consistency with the spirit of ramification –, and unlike the scheme in [2], nonzero recursion parameters, y, must be used in a safe position of h, which is more restrictive.

The development of the above variants of **2CLO** was motivated by the wish to achieve a higher type characterization of **NC**. Such characterizations are useful because programs extracted from proofs of their specifications usually use higher type recursion, which easily exceeds the realm of feasible computation. Therefore, however challenging, one would like to guarantee for a reasonable large class of such extracted programs, usually presented as ramified term systems, that they run in polynomial time or even feasibly highly parallel. While showing *correctness* of such systems is hard work, *completeness* is usually obtained by embedding suitable ground type ramified function algebras known to characterize the intended complexity class, e.g. see [13] or [6]. A problem with such higher type systems is that – in order to tame higher type recursion –, they sometimes lead to very restrictive conditions, such as only allowing the use of "non-size-increasing" functions in recursions and limited usage of "previous functionals" in higher type recursions [14]. Note that the present variants of **2CLO**, especially $\mathbf{2NC}''$ with its restricted scheme (scrn$''$), were designed exactly for such situations.

Observe that both properties **(S2)** and **(S3)** (cf. Section 1) hold for any of the above ramified function algebras. In particular, for every function $f(\vec{x};\ \vec{y})$ in any of the above algebras there exists a poly-max length bound (cf. Section 2).

Inspecting the function algebras characterizing **NC** considered so far, we obtain the following embeddings.

Theorem 6.2. $\mathbf{2CLO} \subseteq \mathbf{2NC} \subseteq \mathbf{2NC}' \subseteq \mathbf{2NC}'' \subseteq \mathbf{CLO}'' \subseteq \mathbf{2CLO}$

Proof. $\boxed{\mathbf{2CLO} \subseteq \mathbf{2NC}}$ As the recursion parameter of any scrn(h) is in a safe position, we cannot show directly the required inclusion. However, we can proceed similarly to the proof of $\mathbf{2NC}' \subseteq \mathbf{2NC}''$.

$\boxed{\mathbf{2NC} \subseteq \mathbf{2NC}'}$ It suffices to define function $\#_{\mathrm{Bel}}(w;\ a, b)$ in $\mathbf{2NC}'$. As $|\mathrm{P}(2^x)| = x$ and $\mathrm{p}(\ ;x) = \mathrm{drop}(\ ;x, \mathrm{s}_1(\ ;0))$, hence $\mathrm{p} \in \mathbf{2NC}'$, this follows from

$$\begin{aligned} \#_{\mathrm{Bel}}(w;\ a, b) &= 2^{|a|\cdot|b|} \bmod 2^{|w|^2} \\ &= \mathrm{sm}(\ ;\mathrm{p}(\ ;\#_{\mathrm{AJST}}(w;\ )), a, b). \end{aligned}$$

$\boxed{\mathbf{2NC}' \subseteq \mathbf{2NC}''}$ We must show that the functions bit, half, and drop all are in $\mathbf{2NC}''$, and that any $f = \mathrm{scrn}'(h)$ with $h \in \mathbf{2NC}''$ is contained in $\mathbf{2NC}''$, too. Recalling Lemma 4.5, this is easily obtained for those initial functions, since

$$\begin{aligned} \mathrm{bit}(\ ;m, n) &= \lfloor \tfrac{m}{2^n} \rfloor \bmod 2 = \mathrm{case}(\ ;\mathrm{msp}(\ ;m, n), 0, \mathrm{s}_1(\ ;0)) \\ \mathrm{drop}(\ ;m, n) &= \lfloor \tfrac{m}{2^{|n|}} \rfloor = \mathrm{msp}(\ ;m, \mathrm{len}(\ ;n)) \\ \mathrm{half}(\ ;m) &= \lfloor m/2^{\lceil |m|/2 \rceil} \rfloor \\ &= \mathrm{case}(\ ;\mathrm{len}(\ ;m), \\ &\qquad \mathrm{drop}(\ ;m, \mathrm{p}(\ ;\mathrm{len}(\ ;m))), \\ &\qquad \mathrm{drop}(\ ;m, \mathrm{p}(\ ;\mathrm{len}(\ ;\mathrm{s}_1(\ ;m))))) \end{aligned}$$

where $\mathrm{case}(\ ;x, y, z) = \mathrm{bcase}(\ ;x, y, y, z)$. For the remaining statement, i.e. $f \in \mathbf{2NC}''$ whenever $f = \mathrm{scrn}'(h)$ with $h \in \mathbf{2NC}''$, we run into a problem, since any attempt to define f directly as $\mathrm{scrn}''(\hat{h})$ for some $\hat{h} \in \mathbf{2NC}''$ is tantamount to turning the normal position of h, to which the recursion f passes any nonzero recursion parameter, into a safe position of $\hat{h}$. That cannot work!

To resolve this problem, we will construct for every function $f(\vec{x};\ \vec{a})$ in $\mathbf{2NC}'$ a simulation $f'(w;\ \vec{x},\vec{a})$ in $\mathbf{2NC}''$, and a (polynomial) witness p_f such that

$$f(\vec{x};\ \vec{a}) = f'(w;\ \vec{x},\vec{a}) \text{ whenever } |w| \geq p_f(|\vec{x},\vec{a}|).$$

Building on the above definitions of bit, half, drop in $\mathbf{2NC}''$, all cases are obvious or standard, except for the case $f = \mathrm{scrn}'(h)$ with $h \in \mathbf{2NC}'$. The I.H. yields a simulation $h' \in \mathbf{2NC}''$ with witness p_h. The witness of f is then defined by $p_f(y,\vec{x},\vec{a}) := p_h(y,\vec{x},\vec{a},b_f(y,\vec{x},\vec{a}))+2y+1$ for some polynomial length bound b_f. We'll define a simulation $f' \in \mathbf{2NC}''$ of f by

$$f'(w;\ y,\vec{x},\vec{a}) := \hat{f}(w,w;\ y,\vec{x},\vec{a}) \quad \text{with } \hat{f} := \mathrm{scrn}''(\hat{h})$$

for some $\hat{h}(w;\ \hat{w},y,\vec{x},\vec{a})$ in $\mathbf{2NC}''$. Accordingly, the *y-section* is defined by

$$y\{i\} := \mathrm{P}^i(y)$$

and by unfolding the recursions we obtain the following steps:

$$\begin{array}{lll}
f(y,\vec{x};\ \vec{a}) & \stackrel{!}{=} \hat{f}(w,w;\ y,\vec{x},\vec{a}) & \text{steps} \\
\quad = \mathrm{S}_{h(y\{0\},\vec{x};\ \vec{a}) \bmod 2}(& = \mathrm{S}_{\hat{h}(w;\ w,y,\vec{x},\vec{a}) \bmod 2}(& 1 \\
\qquad \ddots & \quad \ddots & \vdots \\
\qquad \mathrm{S}_{h(y\{i\dot{-}1\},\vec{x};\ \vec{a}) \bmod 2}(& \quad \mathrm{S}_{\hat{h}(w;\ \mathrm{P}^{i\dot{-}1}(w),y,\vec{x},\vec{a}) \bmod 2}(& i \\
\qquad\quad \ddots & \qquad \ddots & \vdots \\
\qquad\quad \mathrm{S}_{h(y\{|y|\dot{-}1\},\vec{x};\ \vec{a}) \bmod 2}(0) & \qquad \mathrm{S}_{\hat{h}(w;\ \mathrm{P}^{|y|\dot{-}1}(w),y,\vec{x},\vec{a}) \bmod 2}(0) & |y| \\
\qquad\qquad \cdots)\cdots) & \qquad\qquad \cdots)\cdots) &
\end{array}$$

Thus, for $f(y,\vec{x};\ \vec{a}) = \hat{f}(w,w;\ y,\vec{x},\vec{a})$ whenever $|w| \geq p_f(|y,\vec{x},\vec{a}|)$, a stepwise comparison, together with the I.H. for h, yields the following requirement:

$$\hat{h}(w;\ \mathrm{P}^i(w),y,\vec{x},\vec{a}) = \begin{cases} h'(w;\ y\{i\},\vec{x},\vec{a}) & \text{if } i < |y| \\ 0 & \text{else.} \end{cases}$$

In the presence of $\mathrm{drop}(\ ;m,n) = \mathrm{P}^{|n|}(m)$ in $\mathbf{2NC}''$, this time the required y-section implementation in $\mathbf{2NC}''$ is definable with safe positions only because

$$Y(\ ;w,\hat{w},y) = \mathrm{P}^{|w|\dot{-}|\hat{w}|}(y) = \mathrm{drop}(\ ;y,\mathrm{drop}(\ ;w,\hat{w})).$$

Indeed, for sufficiently large w, we have for $i \leq |w|$:

$$Y(\ ;w,\mathrm{P}^i(w),y) = \begin{cases} \mathrm{P}^i(y) & \text{if } i < |y| \\ 0 & \text{else.} \end{cases}$$

Since $i < |y| \Leftrightarrow Y(\ ;w,\mathrm{P}^i(w),y) > 0$, function $\hat{h}$ can be defined in $\mathbf{2NC}''$ by

$$\hat{h}(w;\ \hat{w},y,\vec{x},\vec{a}) := \mathrm{cond}(\ ;\ Y(\ ;w,\hat{w},y),\ 0,\ h'(w;\ Y(\ ;w,\hat{w},y),\vec{x},\vec{a}))$$

where $\mathrm{cond}(\ ;x,y,z) = \mathrm{bcase}(\ ;x,y,z,z)$.

$\boxed{\mathbf{2NC}'' \subseteq \mathbf{CLO}''}$ This inclusion is fairly standard, since the functions sm, msp and $\#_{\mathrm{AJST}}$ can be easily defined in $\mathbf{CLO}''$ (for msp, cf. Lemma 4.5), and by forgetting ramification we see inductively that every $f \in \mathbf{2NC}''$ is definable in $\mathbf{CLO}''$. In particular, by poly-max bounding and the fact that for every polynomial p there exists a function $W_p \in \mathbf{CLO}''$ such that $2^{p(|\vec{x}|)} \le W_p(\vec{x})$, every $f = \mathrm{slr}(g, h) \in \mathbf{2NC}''$ can be turned into a $\mathbf{CLO}''$ function $\mathrm{WBRN}'(g, h, W_p)$.

$\boxed{\mathbf{CLO}'' \subseteq \mathbf{2CLO}}$ We will construct for every $f \in \mathbf{CLO}''$ a simulation $f'(w;\ \vec{x})$ in $\mathbf{2CLO}$, and a (polynomial) witness p_f such that

$$f(\vec{x}) = f'(w;\ \vec{x}) \text{ whenever } |w| \ge p_f(|\vec{x}|).$$

If f is $0, \mathrm{S}_0, \mathrm{S}_1, \pi_i^{n,m}, |\cdot|$ or BIT, then we can define f' directly in $\mathbf{2CLO}$ using safe composition and projection. If f is $\#$ then $\#(x, y) = \mathrm{sm}(w;\ x, y)$ for $|w| \ge |x|\cdot|y| + 1$, since $a \bmod b = a \Leftrightarrow a < b$.

The cases (COMP), (WBRN$'$) are fairly standard, leaving the case $f = \mathrm{CRN}''(\ h)$ with $h \in \mathbf{CLO}''$. Here we can proceed as in the case $\mathrm{scrn}'(h)$ of $\mathbf{2NC}' \subseteq \mathbf{2NC}''$, because in $\mathbf{2CLO}$ function $\mathrm{msp}(\ ; m, n)$ can be defined by (scrn) from $\mathrm{bit}(\ ; m, n)$ using safe variables only – recall the recursion equations of MSP in the proof of Lemma 4.5 –, and hence we obtain as above function $\mathrm{drop}(\ ; m, n)$ in $\mathbf{2CLO}$. □

By Theorems 4.1, 4.3, 5.4, and Theorem 6.2 we have established the following new characterization of $\mathbf{NC}$.

Corollary 6.3. $\mathbf{NC} = [0, \mathrm{s}_0, \mathrm{s}_1, \pi, \mathrm{len}, \mathrm{sm}, \#_{\mathrm{AJST}}, \mathrm{bcase}, \mathrm{msp};\ \mathrm{scomp}, \mathrm{scrn}'', \mathrm{slr}]$

References

[1] K. Aehlig, J. Johannsen, H. Schwichtenberg, S. Terwijn, Linear ramified higher type recursion and parallel complexity, Technical Report 17, Mittag-Leffler-Institut (2000).

[2] K. Aehlig, J. Johannsen, H. Schwichtenberg, S. Terwijn, Linear ramified higher type recursion and parallel complexity, in: R. Kahle, P. Schroeder-Heister, R. Stärk (Eds.), Proof Theory in Computer Science, Vol. 2183 of Lecture Notes in Computer Science, Springer, 2001, pp. 1–21.

[3] S. Bellantoni, Predicative recursion and computational complexity, Ph.D. thesis, Department of Computer Science, University of Toronto (1992).

[4] S. Bellantoni, S. Cook, A new recursion theoretic characterization of the polytime functions., Computational Complexity 2 (1992) 97–110.

[5] S. Bellantoni, K.-H. Niggl, Ranking primitive recursions: The low Grzegorczyk classes revisited, SIAM Journal on Computing 29 (2) (2000) 401–415.

[6] S. Bellantoni, K.-H. Niggl, H. Schwichtenberg, Higher type recursion, ramification and polynomial time, Annals of Pure and Applied Logic 104 (2) (2000) 17–30.

[7] S. Bellantoni, I. Oitavem, Separating NC along the delta axis, in: J.-Y. Marion (Ed.), Special issue on Implicit Computational Complexity, Theor. Comput. Sci. 318(1-2): 57-78, Elsevier, 2004.

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

[9] G. Bonfante, R. Kahle, J.-Y. Marion, I. Oitavem, Towards an implicit characterization of NC^k, in: Z. Ésik (Ed.), CSL, Vol. 4207 of Lecture Notes in Computer Science, Springer, 2006, pp. 212–224.

[10] P. Clote, Sequential, machine-independent characterizations of the parallel complexity classes $ALogTIME$, AC^k, NC^k and NC, in: P. J. Scott, S. R. Buss (Eds.), MSI Workshop on Feasable Mathematics, Birkhäuser, 1990, pp. 49–69.

[11] P. Clote, Computation models and function algebras, in: E. R. Griffor (Ed.), Handbook of Computability Theory, Elsevier Science B.V., 1999, pp. 589–681.

[12] A. Cobham, The intrinsic computational difficulty of functions, in: Y. Bar-Hillel (Ed.), Proceedings of the 1964 International Congress for Logic, Methodology, and the Philosophy of Science, North Holland, 1964, pp. 24–30.

[13] M. Hofmann, Type systems for polynomial-time computation, Ph.D. thesis, Technische Universität Darmstadt (1998).

[14] M. Hofmann, Linear types and non-size-increasing polynomial time computation, Inform. and Comput. 183 (1) (2003) 57–85.

[15] L. Kristiansen, K.-H. Niggl, On the computational complexity of imperative programming languages, in: J.-Y. Marion (Ed.), Special issue on Implicit Computational Complexity, Theor. Comput. Sci. 318(1-2): 139–161, Elsevier, 2004.

[16] D. Leivant, Subrecursion and lambda representation over free algebras, in: S. Buss, P. Scott (Eds.), Feasible Mathematics, Perspectives in Computer Science, Birkhäuser-Boston, New York, 1990, pp. 281–291.

[17] D. Leivant, A foundational delineation of computational feasibility, in: Proceedings of the Sixth IEEE Conference on Logic in Computer Science (Amsterdam), IEEE Computer Society Press, Washington D.C., 1991.

[18] D. Leivant, J.-Y. Marion, Lambda calculus characterizations of poly-time, Fundam. Inform. 19 (1/2) (1993) 167–184.

[19] D. Leivant, Ramified recurrence and computational complexity I: Word recurrence and poly-time, in: P. Clote, J. Remmel (Eds.), Feasible Mathematics II, Birkhäuser Boston, 1994, pp. 320–343.

[20] K.-H. Niggl, The μ-measure as a tool for classifying computational complexity, Archive for Mathematical Logic 39 (7) (2000) 515–539.

[21] K.-H. Niggl, A new recursion-theoretic characterization of the polytime functions, by S. Bellantoni and S. Cook. A term rewriting characterization of the polytime functions and related complexity classes, by A. Beckmann and A. Weiermann, Review, The Bulletin of Symbolic Logic 6 (3) (2000) 351–353.

[22] K.-H. Niggl, Control structures in programs and computational complexity, Habilitation thesis, Institut für Theoretische Informatik, Technische Universität Ilmenau (2001).
URL `http://eiche.theoinf.tu-ilmenau.de/~niggl/`

[23] K.-H. Niggl, Control structures in programs and computational complexity, Annals of Pure and Applied Logic 133 (1-3) (2005) 247–273.

[24] K.-H. Niggl, H. Wunderlich, Certifying polynomial time and linear/polynomial space for imperative programs, SIAM Journal on Computing 35 (5) (2006) 1122–1147.

[25] I. Oitavem, New recursive characterizations of the elementary functions and the functions computable in polynomial space, Revista Mathemática de la Universidad Complutense de Madrid 10 (1) (1997) 109–125.

[26] I. Oitavem. *Characterizing NC with tier* 0 *pointers*. Math. Log. Q., 50(1): 9–17, 2004.

[27] H. Simmons, The Realm of Primitive Recursion, Archive for Mathematical Logic 27 (1988) 177–188.

[28] H. Wunderlich, Syntaktische Charakterisierungen effizient berechenbarer Funktionen, Diploma thesis, Institut für Theoretische Informatik, Technische Universität Ilmenau (2003).

Liste der bisher erschienenen Ulmer Informatik-Berichte
Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm
Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

91-02* *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

91-03* *Alfons Geser*
Relative Termination

91-04* *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

91-06* *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

91-07* *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

91-08* *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

92-01* *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

92-02* *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

92-04* *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

92-05* *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

92-06* *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

92-07* *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

92-08* *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

92-09* *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

92-14* *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars:
Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman,Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12 *Gerhard Schellhorn*
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

98-13 *Gerhard Schellhorn, Wolfgang Reif*
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

98-14 *Mohammad Ali Livani*
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

98-15 *Mohammad Ali Livani, Jörg Kaiser*
Predictable Atomic Multicast in the Controller Area Network (CAN)

99-01 *Susanne Boll, Wolfgang Klas, Utz Westermann*
A Comparison of Multimedia Document Models Concerning Advanced Requirements

99-02 *Thomas Bauer, Peter Dadam*
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

99-03 *Uwe Schöning*
On the Complexity of Constraint Satisfaction

99-04 *Ercument Canver*
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

99-05 *Johannes Köbler, Wolfgang Lindner, Rainer Schuler*
Derandomizing RP if Boolean Circuits are not Learnable

99-06 *Utz Westermann, Wolfgang Klas*
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

99-07 *Peter Dadam, Manfred Reichert*
Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik ’99

99-08 *Vikraman Arvind, Johannes Köbler*
Graph Isomorphism is Low for ZPP^{NP} and other Lowness results

99-09 *Thomas Bauer, Peter Dadam*
Efficient Distributed Workflow Management Based on Variable Server Assignments

2000-02 *Thomas Bauer, Peter Dadam*
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

2000-03 *Gregory Baratoff, Christian Toepfer, Heiko Neumann*
Combined space-variant maps for optical flow based navigation

2000-04 *Wolfgang Gehring*
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05 *Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel*
Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06 *Wolfgang Reif, Gerhard Schellhorn, Andreas Thums*
Fehlersuche in Formalen Spezifikationen

2000-07 *Gerhard Schellhorn, Wolfgang Reif (eds.)*
FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification

2000-08 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09 *Thomas Bauer, Peter Dadam*
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Adaptives und verteiltes Workflow-Management

2000-11 *Christian Heinlein*
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01 *Hubert Hug, Rainer Schuler*
DNA-based parallel computation of simple arithmetic

2001-02 *Friedhelm Schwenker, Hans A. Kestler, Günther Palm*
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03 *Hans A. Kestler, Friedhelm Schwenker, Günther Palm*
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04 *Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm*
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02 *Walter Guttmann*
Deriving an Applicative Heapsort Algorithm

2002-03 *Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk*
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01 *Manfred Reichert, Stefanie Rinderle, Peter Dadam*
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
On Dealing With Semantically Conflicting Business Process Changes.

2003-05 *Christian Heinlein*
Dynamic Class Methods in Java

2003-06 *Christian Heinlein*
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

2003-07 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values (Corrected Version)

2003-08 *Changling Liu, Jörg Kaiser*
Survey of Mobile Ad Hoc Network Routing Protocols)

2004-01 *Thom Frühwirth, Marc Meister (eds.)*
First Workshop on Constraint Handling Rules

2004-02 *Christian Heinlein*
Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

2004-03 *Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)*
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

2005-01 *Armin Wolf, Thom Frühwirth, Marc Meister (eds.)*
19th Workshop on (Constraint) Logic Programming

2005-02 *Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven*
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

2005-03 *Walter Guttmann, Markus Maucher*
Constrained Ordering

2006-01 *Stefan Sarstedt*
Model-Driven Development with ACTIVECHARTS, Tutorial

2006-02 *Alexander Raschke, Ramin Tavakoli Kolagari*
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

2006-03 *Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari*
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

2006-04 *Thorsten Liebig*
Reasoning with OWL - System Support and Insights –

2008-01 *H.A. Kestler, J. Messner, A. Müller, R. Schuler*
On the complexity of intersecting multiple circles for graphical display

2008-02	*Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer* Architectural Design of Flexible Process Management Technology
2008-03	*Frank Raiser* Semi-Automatic Generation of CHR Solvers from Global Constraint Automata
2008-04	*Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander* Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse
2008-05	*Markus Kalb, Claudia Dittrich, Peter Dadam* Support of Relationships Among Moving Objects on Networks
2008-06	*Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)* WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke
2008-07	*M. Maucher, U. Schöning, H.A. Kestler* An empirical assessment of local and population based search methods with different degrees of pseudorandomness
2008-08	*Henning Wunderlich* Covers have structure
2008-09	*Karl-Heinz Niggl, Henning Wunderlich* Implicit characterization of FPTIME and NC revisited

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm