Implicit characterizations of FPTIME and NC revisited

Karl-Heinz Niggl, Henning Wunderlich

Ulmer Informatik-Berichte
Nr. 2009-08
Juli 2008
Implicit characterizations of $\text{FPTIME}$ and $\text{NC}$ 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 $\text{FPTIME}$ and $\text{NC}$ 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 paid to the complexity classes $\text{FPTIME}$ and $\text{NC}$, 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 $\text{FPTIME}$ and $\text{NC}$.

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 $\text{NC}$, which might be of help to resolve the open problem [2] of characterizing $\text{NC}$ through higher types.

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

1. For every $f$ in $\text{Cob}$ one can construct a function $f'(w; \vec{x})$ in $\text{BC}$, called simulation of $f$, and a polynomial $p_f$, called witness for $f$, such that $f(\vec{x}) = f'(w; \vec{x})$ whenever $|w| \geq p_f(|\vec{x}|)$.

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

$^*$Technische Universität Ilmenau, Fakultät für Informatik und Automation, 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 $\text{SN}(g)(\vec{x}, \vec{y}; \vec{z})$, 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:

$$f(\vec{x}) = \text{SN}(f')(W_p(\vec{x};), \vec{x};)$$

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

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

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

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

then a simulation $f'$ can be constructed the correctness of which is immediate from its definition. So let 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 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. Furthermore, note that while bcase$(\ ; x, y, z_0, z_1)$ is provably indefinable in BC, the function case$(\ ; x, z_0, z_1)$ is obviously in BC', since case$(\ ; x, z_0, z_1) = \text{bcase}(\ ; x, z_0, z_0, z_1)$.

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 [23], 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 $\mu$-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.

As a technical consequence, in BC' we don’t have to bother with defining the functions “PARITY,” “T,” “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 $BC' = Cob$, thereby demonstrating the step-by-step comparison technique. Recalling Clote’s function algebra, $CLO$, in Section 4 and 5, two variants, $CLO'$ and $CLO''$, are considered, and a proof of $CLO' = CLO = CLO''$ 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 \textit{number-theoretic}, that is, functions of the form $f: \mathbb{N}^n \to \mathbb{N}$. For unary functions $f$ and numbers $k$, $f^k$ denotes the $k$th 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 $\text{bin}(x)$, can be simulated by $0$ (viewed as $1$-ary function) and the binary successors $S_0, S_1$ which correspond to the operations of extending binary representations by a new lowest order bit.

- $S_0(x) = 2 \cdot x$ (operation $\text{bin}(x) \mapsto \text{bin}(x)0$ for $x \neq 0$)
- $S_1(x) = 2 \cdot x + 1$ (operation $\text{bin}(x) \mapsto \text{bin}(x)1$)

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

- $f(0, \bar{x}) = g(\bar{x})$
- $f(S_i(y), \bar{x}) = h_i(y, \bar{x}, f(y, \bar{x}))$ for $S_i(y) \neq 0$.

Observe that $\text{bin}(y) = b_{l-1} \ldots b_0 = \epsilon$ implies $y = S_{b_l}(\ldots S_{b_{l-1}}(0) \ldots)$. Thus, for recursion on notation, the recourse is from $b_{l-1} \ldots b_0$ to $b_{l-1} \ldots b_1$ to $\ldots$ to
Theorem 2.1 (12) computable functions are precisely the functions denable in Cob (in the binary length of the input). Cobham showed that the polynomial-time is, the functions computable (in binary) on a Turing machine in polynomial time ≤ 1 Π where

\[ C_\text{PR} \]

were the first to give a purely functional characterization of Cobham's well-known function algebra.

\[ C_\text{PR} \]

is bounded by some function already constructed. 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 \( |f(\vec{x})| \leq b_f(|\vec{x}|) \).

While the latter is a necessary condition for all functions in \( \text{FPTIME} \), 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). \( \text{Cob} = \text{FPTIME} \)

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 Simons [27] and Leivant [16, 17], Bellantoni and Cook [4] were the first to give a purely functional characterization of \( \text{FPTIME} \) 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 \( g(\vec{x}; \vec{a}); h_0(u, \vec{x}; \vec{a}, \vec{v}), \) and \( h_1(u, \vec{x}; \vec{a}, \vec{v}) \), denoted by \( f = \text{srn}(g, h_0, h_1) \), if for all \( y, \vec{x}, \vec{a} \),

\[
\begin{align*}
f(0, \vec{x}; \vec{a}) &= g(\vec{x}; \vec{a}) \\
f(S_i(y), \vec{x}; \vec{a}) &= h_i(y, \vec{x}; \vec{a}, f(y, \vec{x}; \vec{a})) & \text{ for } S_i(y) \neq 0.
\end{align*}
\]

Enforcing the above principle when composing functions of given ones, a function \( f(\vec{x}; \vec{a}) \) is defined by safe composition from functions \( g(\vec{a}; \vec{v}); h(\vec{x}; \vec{a}) \), and
\( \jmath(x; y) \), denoted by \( f = \text{scomp}(g, \hat{h}, \jmath) \), if for all \( x, a \),

\[ f(x; a) = g(\hat{h}(x); \jmath(x; a)). \]

Of course, now all initial functions must be written in a ramified form, too. These are the functions \( 0, s_0(;) y, s_1(;) y, \pi_i^{n, m}(\vec{x}; \vec{y}), p(;) y \), and case( ; \( x, y, z \)), where the latter is defined in Section 1. The function \( p(;) y \) is the ramified form of the binary predecessor \( P \) satisfying \( P(x) = \lceil \frac{x}{2} \rceil \), 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 \( 1 \leq i \leq n+m \), 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 \( \vec{x} = x_1, \ldots, x_l \), \( \vec{y} = y_1, \ldots, y_l \), with \( n := l+m \), and \( \vec{z} = z_{l+1}, \ldots, z_s \) with \( s := n+r \). Then by scomp we obtain

\[ \text{SN}(f)(\vec{x}, \vec{y}; \vec{z}) = f(\pi_1^{0, 0}(\vec{x}, \vec{y}; \vec{z}), \ldots, \pi_l^{0, 0}(\vec{x}, \vec{y}; \vec{z}), \ldots, \pi_s^{r, r}(\vec{x}, \vec{y}; \vec{z}), \ldots). \]

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 \( \hat{h}(\vec{x}; \vec{y}; \vec{z}) \) and \( \jmath(\vec{x}; \vec{y}; \vec{z}) \) 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: \( C_a^n(\vec{x}; \vec{y}) = 0 \), and inductively for \( 2 \cdot a + i \geq 1 \), \( C_{2a+i}^n(\vec{x}; \vec{y}) = s_i(;) C_a^n(\vec{x}; \vec{y}) \). As a consequence, every constant \( a \) may occur in the right-hand side of any defining equation according to scomp or ssn.

Altogether, the function algebra BC can be stated as

\[
\text{BC} := [0, s_0, s_1, \pi, p, \text{case; scomp}, \text{ssn}]
\]

where \( \pi \) 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

\[ |f(\vec{x}; \vec{y})| \leq q_f(|\vec{x}|) + \max(|\vec{y}|). \]

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

**Theorem 2.2** ([4]). \( \text{BC} = \text{FPTIME} \)

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

\[
\text{shl}(0; y) = \pi_1^{0, 1}(; y), \quad \text{shl}(S_i(x); y) = s_0(; \text{shl}(x; y)) \quad \text{for } S_i(x) \neq 0
\]
Following the simulation method, the smash function \( \#(x, y) = 2^{\lfloor |x|/2 \rfloor} \cdot 2^{\lfloor |y|/2 \rfloor} \) is defined by

\[
\#(0, y; ) = 1 \\
\#(S_i(x), y; ) = \text{shl}(2, 0)(x, y; ); \quad \#(x, y; ) \quad \text{for } S_i(x) \neq 0.
\]

Now, to prove (S2), we proceed by induction on the structure of polynomials \( p(\vec{x}) \) in \( \mathbb{N}[\vec{x}] \). If \( p(x_1, \ldots, x_n) \) is \( x_i \) or \( c \), then \( W_{x_i}(\vec{x}; ) := \text{shl}(\pi_{x_i}^0(\vec{x}; ) \cdot 1) \) and \( W_C(\vec{x}; ) := C_{x_i}^0(\vec{x}; ) \), respectively, will do. Otherwise \( p(\vec{x}) \) is \( p_1(\vec{x}) \circ p_2(\vec{x}) \) with \( \circ \in \{ +, \cdot \} \), and using \( x + y, x \cdot y \leq (x + 1) \cdot (y + 1) \) and \( 2^\lfloor n \rfloor = x + 1 \), 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 BC' and the step-by-step comparison technique

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

\( \text{BC}' := [0, s_0, s_1, \pi, p, \text{bcase}; \text{scomp}, \text{srn}] \)

**Theorem 3.1. BC' = FPTIME.**

**Proof.** \( \text{Cob} \subseteq \text{BC}' \) Following the simulation method (S1) stated above, we only consider the crucial case \( f = \text{BRN}(g, h_0, h_1, B) \), assuming inductively simulations \( g', h_0', h_1' \in \text{BC}' \) and witnesses \( p_g, p_{h_0}, p_{h_1} \). As usual, the witness for \( f \) is defined by \( p_f(y; \vec{x}) := (p_{g_0} + p_{h_1})(y, \vec{x}, b_f(y, \vec{x})); p_f(\vec{x}; ) + y + 1 \) for some polynomial length bound \( b_f \) on \( f \). Thus, by monotonicity of polynomials, we have that \( \binom{(*)}{|w| 

\leq p_{h_1}( |P^1(y; \vec{x}), f(P^1(y; \vec{x}))| \text{ whenever } |w| 

\geq p_f(|y, \vec{x}|) \). Now, for any \( y, i \in \mathbb{N} \), let

\[
y(i) := P^i(y)
\]

be the \( y \)-section up to \( i \). That is, for given \( y = (b_{i-1} \cdots b_0) \) with \( \text{bin}(y) = b_{i-1} \cdots b_0 \), we have \( y(i) = (b_{i-1} \cdots b_1) \), and \( y(i) \mod 2 = b_i \) for \( i < |y| \). Thus, by unfolding the recursion we obtain the following steps:

\[
f(y, \vec{a}) = h_y[0 \mod 2] (y[1], \vec{a}), \quad \text{step } 1
\]

\[
h_y[i \mod 2] (y[i], \vec{a}), \quad \text{step } i
\]

\[
h_y[|y| - 1 \mod 2] (y[|y|], \vec{a}, g(\vec{a}) \cdots), \quad \text{step } |y|
\]

\[
h_y[|y| + 1 \mod 2] (y[|y| + 1], \vec{a}), \quad \text{step } |y| + 1
\]

We will define a simulation \( f' \in \text{BC}' \) by

\[
f'(w; y, \vec{a}) := \hat{f}(w, w; y, \vec{a})
\]
where \( f := \text{srn}(0, \hat{h}, \hat{h}) \) is defined by safe recursion from the zero function and some \( \hat{h} \in \text{BC}' \). Again, unfolding the recursion yields the following \( \hat{f} \)-steps:

\[
\hat{h}(\text{P}^i, w; y, \vec{a}) \quad \text{step } i
\]

Thus, for \( f(y, \vec{a}) = \hat{f}(w, y, \vec{a}) \) 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:

\[
\hat{h}(\text{P}^i(w), y, \vec{a}, v) = h'(w; y, \vec{a}, v) \quad \text{in steps } 1 \leq i \leq |y|
\]

\[
\hat{h}(\text{P}^{i+1}(w), y, \vec{a}, v) = g'(w; \vec{a}) \quad \text{in step } |y| + 1
\]

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

\[
Y(\text{P}^i(w), y) := \odot(Y(S_1(\text{P}^i(w)), y)) = \text{P}^{(|w| - |\hat{w}|)}(y) = y(|w| - |\hat{w}|)
\]

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

\[
Y(S_1(\text{P}^i(w)), y) = y(i - 1) > 0 \quad \text{for } 1 \leq i \leq |y|.
\]

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

\[
\hat{h}(w; y, \vec{a}, v) := \text{bcase} \left( Y(S_1(\hat{w}), y), g'(w; \vec{a}) \right)
\]

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

\[
\hat{h}(\text{P}^i(w), y, \vec{a}, v) = \text{bcase} \left( v, g'(w; \vec{a}), T_0, T_1 \right)
\]

and \( \hat{h}(\text{P}^{i+1}(w), y, \vec{a}, v) = \text{bcase} \left( 0, 0, g'(w; \vec{a}), \ldots, \right) = g'(w; \vec{a}) \).

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

\section{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 $\text{CLO}'$ due to Bellanoni [3], and prove that these classes coincide.

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

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

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

\[
\begin{align*}
    f(0, \bar{a}) &= g(\bar{a}) \\
    f(S_i(y), \bar{a}) &= S_{h_i(y, \bar{a}) \mod 2}(f(y, \bar{a})) & \text{for } S_i(y) \neq 0.
\end{align*}
\]

Clote [10, 11] was the first to give a function-algebraic characterization of $\text{NC}$ through his algebra

$$\text{CLO} := [0, S_0, S_1, \Pi, | |, \text{BIT}, \#; \text{COMP}, \text{CRN}, \text{WBRN}].$$

**Theorem 4.1** ([10, 11]). $\text{NC} = \text{CLO}$

In [3, p. 73] Bellanoni pointed out that the same class is obtained when replacing scheme ($\text{WBRN}$) with the following streamlined variant.

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

\[
\begin{align*}
    f(0, \bar{a}) &= g(\bar{a}) \\
    f(y, \bar{a}) &= h(y, \bar{a}, f(H(y, \bar{a}))) & \text{for } y \neq 0 \\
    f(y, \bar{a}) &\leq B(y, \bar{a})
\end{align*}
\]

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

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

\[
\begin{align*}
    H((b_{2n-1} \ldots b_0)_2) &= (b_{2n-1} \ldots b_n)_2 & \text{even length} \\
    H((b_{2n} \ldots b_0)_2) &= (b_{2n} \ldots b_{n+1})_2 & \text{odd length}
\end{align*}
\]

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

$$\text{CLO}' := [0, S_0, S_1, \Pi, | |, \text{BIT}, \#; \text{COMP}, \text{CRN}, \text{WBRN}']$$

one obtains the following result.

**Theorem 4.3.** $\text{CLO} = \text{CLO}'$

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 $f = \text{WBRN}(g, h, h_1, B)$ is claimed to be identical to $f' := \text{WBRN}'(g, h', B)$, where $h'(x, \bar{v}, z) := h_{|x| \mod 2(|x| - 1, \bar{v}, z)}$. But, for example, $f(5, \bar{v}) = F([5], \bar{v}) = F(S_1(1(0)), \bar{v}) = h_1(1, \bar{v}, h_1(0, \bar{v}, g(\bar{v})))$, while $f'(5, \bar{v}) = h'(5, \bar{v}, h'(1, \bar{v}, g(\bar{v}))) = h_{|5| \mod 2(|5| - 1, \bar{v}, h_{|1| \mod 2(|1| - 1, \bar{v}, g(\bar{v}))}) = h_{1}(2, \bar{v}, h_{1}(0, \bar{v}, g(\bar{v}))).$

For the converse, any $f' := \text{WBRN}'(g, h, B)$ is claimed to be definable by $f(u, \bar{v}) := f'(u, u, \bar{v})$, where $f := \text{WBRN}(g, h_0, h_1, B)$, and $h_1(u, x, \bar{v}, z) := h(E(u, x), \bar{v}, z)$, with $E(u, x) = x \mod 2^u$, being the low-order $u$ bits of $x$, assuming $u \leq |x|$. But, e.g., $f'(5, \bar{v}) = h(5, \bar{v}, h(1, \bar{v}, g(\bar{v})))$, while $f(5, \bar{v}) = f(5, 5, \bar{v}) = F([5], \bar{v}) = F(S_1(1(0))) = h(E(1, 5, \bar{v}, h(E(0, 5), \bar{v}, g(\bar{v})))) = h(1, \bar{v}, h(0, \bar{v}, g(\bar{v}))).$
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 \( y \).

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

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
\[
\|y\|_H = ||(y)||
\]
(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 \( |H(y)| = n \) by definition, and we obtain
\[
\|y\|_H = \|H(y)\|_H + 1 \overset{(4.H.1)}{=} ||(H(y))|| + 1 = |n| + 1 = |2n + i| = ||(y)||.
\]

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 \( \text{MSP}(m, n) = [\frac{m}{2^n}] = P^n(m) \),

(b) function DROP, satisfying \( \text{DROP}(m, n) = [\frac{m}{2^n}] = P^n(m) \),

(c) the binary predecessor, \( P \), satisfying \( \text{P}(m) = [\frac{m}{2}] \).

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

(e) the binary conditional, CASE, satisfying \( \text{CASE}(x, y, z) = \text{case}(x, y, z) \),

(f) and function half, \( H \), satisfying \( H(m) = \lfloor m/2^{[m/2]} \rfloor \).

**Proof.** As for part (a), observe that MSP can be defined by (CRN), since
\[
\text{MSP}(0, n) = 0 \\
\text{MSP}(S_b(m), n) = S_{\text{BIT}(S_b(m), n)}(\text{MSP}(m, n))
\]
for \( S_b(m) \neq 0 \). Thus, both parts (b) and (c) follow from (a), since
\[
\text{DROP}(m, n) = \text{MSP}(m, |n|) \\
\text{P}(m) = \text{MSP}(m, 1).
\]

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

As well, since \( ||x|| = 0 \Leftrightarrow x = 0 \), we obtain \( \text{COND} = \text{WBRN′}(g, h, b) \in \text{CLO′} \).

Now, part (e) follows from (d), since \( \text{CASE}(x, y, z) = \text{COND}(\text{BIT}(x, 0), y, z) \), and finally, (f) follows from parts (a) – (e), since
\[
H(m) = \text{CASE}(|m|, \text{DROP}(m, P(|m|)), \text{DROP}(m, P(|S_1(m)|))).
\]
Proof. It suffices to consider any $f := \text{WBRN}(y, h_0, h_1, B)$ in CLO, assuming $g, h_0, h_1, B \in \text{CLO}'$. We shall give a direct simulation $f' \in \text{CLO}'$ of $f$, that is, $f(y, \vec{a}) = f'(y, \vec{a})$ for all $y, \vec{a}$, where

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

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

$$y(i) := P^i(||y||).$$

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

$$f(y, \vec{a}) = F(||y||, \vec{a}) = h_{b_0}(y(1), \vec{a}),$$

$$\vdots$$

$$h_{b_{||y||-1}}(y(i), \vec{a}), h_{b_{||y||-2}}(y(i||y||), \vec{a}), \ldots,$$

$$\vdots$$

$$h_{b_0}(y(1), \vec{a}), h_0(y(1), \vec{a}), h_1(y(1), \vec{a}), \ldots,$$

Thus, a stepwise comparison yields the requirement

$$\hat{h}(H^{i-1}(y), y, \vec{a}, v) = h_{y(i-1) \mod 2}(y(i), \vec{a}, v) \quad \text{in steps } 1 \leq i \leq ||y||$$

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 CLO (below) we need this time is

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

As (0) implies $||H^i(y)|| = ||y|| - i$, we conclude that

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

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

$$\hat{h}(w, y, \vec{a}, v) := h_{Y(w, y) \mod 2}(Y(H(w), y), \vec{a}, v)$$

$$= \text{CASE}(Y(w, y), h_0(Y(H(w), y), \vec{a}, v), h_1(Y(H(w), y), \vec{a}, v)).$$

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

$$\hat{h}(H^{i-1}(y), y, \vec{a}, v) = h_{Y(H^{i-1}(y), y) \mod 2}(Y(H^i(y), y), \vec{a}, v)$$

$$= h_{y(i-1) \mod 2}(y(i), \vec{a}, v)$$

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

$$\odot'(0, y) := y$$

$$\odot'(w, y) := H(\odot'(H(w), y)) \quad \text{for } w \neq 0$$

10
To see this, observe inductively that for \( w \neq 0 \), \( ||\circ'(w,y)|| = ||H(\circ'(H(w),y))|| = ||\circ'(H(w),y)|| + 1 = (||y|| + ||H(w)||) + 1 = (||y|| + ||w|| + 1) + 1 = ||y|| + ||w||. \)

As \( ||w|| \geq 1 \). Note that the outmost use of \( H \in \text{CLO}' \) in the above definition is not part of the (WBRN') scheme. Now, we conclude the required definition of the \( y \)-section implementation in \( \text{CLO}' \) as follows:

\[
Y(w,y) := \text{MSP}(|y|, \circ'(w,y))
\]

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

\[
\ ||| \ \implies \ \hat{f}(w,y,\vec{x}) = F(Y(w,y),\vec{x}) \leq B(Y(w,y),\vec{x})
\]

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

\( \text{[CLO}^\prime \subseteq \text{CLO]} \) It suffices to consider any \( f := \text{WBRN'}(g,h,B) \), assuming inductively \( g,h,B \in \text{CLO} \). Accordingly, the \( y \)-section we need is defined by

\[
y(i) := H^{i+1}(y).
\]

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

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

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

\[
\begin{align*}
\hat{f}(y,\vec{a}) & = h(Y(\{1\},\vec{a})), \quad \hat{f}(y,\vec{a}) = \hat{F}(|y|,y,\vec{a}) \quad \text{steps} \\
& = h(P(|y|),y,\vec{a}), \quad 1 \\
& \cdots \\
& h(y(i),\vec{a}), \quad \hat{h}(P^i(|y|),y,\vec{a}), \quad i \\
& \cdots \\
& h(y(|y||),\vec{a}), \quad \hat{h}(P(|y||)(|y|),y,\vec{a}), \quad ||y|| \\
& \hat{g}(\vec{a}) \cdots ) \cdots ) \quad \hat{g}(y,\vec{a}) \cdots ) \cdots ) \quad ||y|| + 1
\end{align*}
\]

Thus, a stepwise comparison yields the requirement

\[
\hat{h}(P^i(|y|),y,\vec{a},v) = h(y(i),\vec{a},v) \quad \text{in steps} 1 \leq i \leq ||y||
\]

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 \( \text{CLO} \) we need this time is

\[
Y(w,y) := H^{|y| - (|w| + 1)}(y) = y(||y|| - |w|)
\]

In fact, since \( |P^i(|y||)| = ||y|| - i \), we conclude from (7) that

\[
Y(P^i(|y||),y) = y(i) \quad \text{for} \quad i \leq ||y||.
\]
Thus, we obtain the required function \( \hat{h} \in \text{CLO} \) by setting
\[
\hat{h}(w, y, \vec{a}, v) := h(Y(w, y), \vec{a}, v)
\]
provided that function \( Y \) is definable in \( \text{CLO} \). To see that, using \( H, \text{DROP} \in \text{CLO} \), and \(|w| < |x| \Leftrightarrow |S_1(w)| \leq |x| \Leftrightarrow \text{DROP}(S_1(w), x) = P^{|x|}(S_1(w)) = 0 \), we first define by (BRN) a function \( G \) in \( \text{CLO} \), satisfying \( G(x, y, w) = H^{|x| \cdot |w|}(y) \).

\[
G(0, y, w) := y
\]
\[
G(S_b(x), y, w) := \text{COND} \left( \text{DROP}(S_1(w), x), H(G(x, y, w)), y \right)
\]
for \( S_b(x) \neq 0 \). Then define \( \hat{Y}(x, y, w) := G(|x|, y, w) = H^{|x| \cdot |w|}(y) \) by (WBRN), and conclude the \( y \)-section implementation in \( \text{CLO} \) by setting
\[
Y(w, y) := \hat{Y}(y, y, S_1(w)).
\]
To complete the definition of \( \hat{f} \), it remains to define a bound \( \hat{B} \in \text{CLO} \), and again we run into a problem. To see this, first observe that one can show:
\[
|w| \leq ||y|| \implies \hat{F}(w, y, \vec{x}) = f(Y(w, y), \vec{x}) \leq B(Y(w, y), \vec{x})
\]
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 \(|w| \cdot (||y|| - 1)\) 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 \( \text{CLO} \). To resolve this problem, we use the functions \( \text{COND}, |\cdot| \) and \( G' \) below (all of which are in \( \text{CLO} \)) to modify \( \hat{h} \) such that it returns 0 whenever \(|w| = ||y|| > 0 \), and by (8) setting \( \hat{B}(w, y, \vec{x}) := B(Y(w, y), \vec{x}) \) then will do.

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

5 Variant \( \text{CLO}'' \) 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 \( \text{NC} \), building on ideas and techniques presented in [6].

Before defining that variant of \( \text{CLO}' \), 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 \( f := \text{CRN}'(y, h) \), if for all \( y, \vec{a}, \)
\[
f(0, \vec{a}) = g(\vec{a})
\]
\[
f(y, \vec{a}) = S_{h(y, \vec{a}) \mod 2}(f(P(y), \vec{a})) \quad \text{for } y \neq 0.
\]

**Corollary 5.2** (\( h \)-variant). In the context of \( \text{CLO} \) or \( \text{CLO}' \), the \( h \)-variant (CRN') is equivalent to (CRN).
Proof. Given any \( f = \text{CRN}(g, h_0, h_1) \), we obtain \( f = \text{CRN}'(g, h) \) for
\[
h(w, \bar{a}) := \text{CASE}(w, h_0(P(w), \bar{a}), h_1(P(w), \bar{a})).
\]

Conversely, given any \( f = \text{CRN}'(g, h) \), we have \( f = \text{CRN}(g, h_0, h_1) \) where
\[
h_0(w, \bar{a}) := h(S_0(w), \bar{a})
\]
\[\square\]

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

Definition 5.3. A function \( f \) is defined by the \( g \)-variant of \( \text{CRN}' \) from function \( h \), denoted by \( f := \text{CRN}''(h) \), if for all \( y, \bar{a} \),
\[
f(0, \bar{a}) = 0
\]
\[
f(y, \bar{a}) = S_{h(y, \bar{a})} \mod 2(f(P(y), \bar{a})) \quad \text{for} \quad y \neq 0.
\]

In fact, defining the class \( \text{CLO}'' \) by
\[
\text{CLO}'' := [0, S_0, S_1, \Pi, |\cdot|, \#; \text{COMP}, \text{CRN}'', \text{WBRN}']
\]
one ends up with the same class of functions. In [4, p. 77] \( \text{CRN} \) is simulated by the ramified \( g \)-variant of \( \text{CRN} \) (ramified \( \text{CRN}' \)). As this construction is wrong\(^3\), we give a proof in the corresponding unramified setting.

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

Proof. As \( \text{CRN}''(h) = \text{CRN}'(0, h) \), the inclusion \( \subseteq \) follows from Corollary 5.2.

\( \text{CLO}' \subseteq \text{CLO}'' \) By Corollary 5.2 it suffices to consider any function \( f := \text{CRN}'(g, h) \), assuming inductively that \( g, h \in \text{CLO}' \). Accordingly, the \( g \)-section is defined by
\[
y[i] := P^i(y)
\]
and by unfolding the recursion, we obtain the following steps:
\[
f(y, \bar{a}) = S_{h(y, \bar{a})} \mod 2\left(\begin{array}{c}
\vdots \\
S_{h(y, \bar{a})} \mod 2\left(\begin{array}{c}
\vdots \\
\vdots \\
S_{h(y, \bar{a})} \mod 2\left(\begin{array}{c}
\vdots \\
\vdots \\
\vdots \\
\vdots
\end{array}\right) \cdots \right) \cdots \right) \cdots
\end{array}\right)
\]

\( \text{To see this, consider the function } f = \text{CRN}(0, C_1, C_1) \text{ satisfying } f(u; ) = 2^{|u|}. \) It is claimed that for sufficiently large \( w \), \( f(u; ) = f'(w; u) := f(w; u, u) \), where \( h'(w; u) := \text{case}(c; u, C_0(w, p(u)); C_1(w, p(u))) = C_2(w; u) = 1 \), and \( f(w; 0, u) := 0 \), and \( f(w; c, u) := S_{\text{case}(c; |c| \leq |w|, h'(w; u, u) \mod 2, \text{bit}(u, \text{bit}(w, |c| - h'(w; u)), f(w, C, u)) = 1} \), for \( c \neq 0 \). But \( f(1) = 1 \), while e.g. for \( |w| = 3 \) we have \( f'(w; 1) = f(w; w, 1) = S_{\text{case}(c; 3 \leq |w|, 1, 0) \cdot S_{\text{case}(c; 2 \leq |w|, 1, 0)} \cdot S_{\text{case}(c; 1 \leq |w|, 1, 0)} \cdot 0) = S_0(S_0(S_1(0))) \neq 4 \). In general, if \( f(w, \bar{u}) = 2^{b_{w-1} \ldots b_0} \), then for sufficiently large \( w \), \( f'(w; y, \bar{v}) = 2^{b_{w-1} \ldots b_0(|w| - 1)/2} \).

3To see this, consider the function \( f = \text{CRN}(0, C_1, C_1) \) satisfying \( f(u; ) = 2^{|u|} \). It is claimed that for sufficiently large \( w \), \( f(u; ) = f'(w; u) := f(w; u, u) \), where \( h'(w; u) := \text{case}(c; u, C_0(w, p(u)); C_1(w, p(u))) = C_2(w; u) = 1 \), and \( f(w; 0, u) := 0 \), and \( f(w; c, u) := S_{\text{case}(c; |c| \leq |w|, h'(w; u, u) \mod 2, \text{bit}(u, \text{bit}(w, |c| - h'(w; u)), f(w, C, u)) = 1} \), for \( c \neq 0 \). But \( f(1) = 1 \), while e.g. for \( |w| = 3 \) we have \( f'(w; 1) = f(w; w, 1) = S_{\text{case}(c; 3 \leq |w|, 1, 0) \cdot S_{\text{case}(c; 2 \leq |w|, 1, 0)} \cdot S_{\text{case}(c; 1 \leq |w|, 1, 0)} \cdot 0) = S_0(S_0(S_1(0))) \neq 4 \). In general, if \( f(w, \bar{u}) = 2^{b_{w-1} \ldots b_0} \), then for sufficiently large \( w \), \( f'(w; y, \bar{v}) = 2^{b_{w-1} \ldots b_0(|w| - 1)/2} \).
To achieve a step-by-step simulation with respect to CRN$''$(̃$h$) for some ̃$h$, we just express $g(̃a)$ as further steps of ̃$h$ that will be performed after the above $|y|$ steps. The simple idea is that any $z=(b_{n-1} \ldots b_0)_2$ can be written as

$$z = S_{b_n}(\ldots(S_{b_{n-1}}(S_0^b(0)) \ldots)) \quad \text{for any } k \in \mathbb{N}. $$

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

$$g(̃a) = S_{\text{BIT}(g(̃a),0)}(\ldots) \quad \text{step } |y| + 1$$

$$S_{\text{BIT}(g(̃a),|g(̃a)|-1)}(\ldots) \quad \text{step } |y| + |g(̃a)|$$

$$S_0(\ldots)$$

$$L \quad \text{step } |y| + |g(̃a)| + k$$

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

$$\geq |y| + |g(̃a)|$$

steps.

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

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

where $W$ is any CLO$''$ function satisfying $|W(y, ̃a)| \geq |y| + |g(̃a)|$. For example, setting $W(y, ̃a) := \#(S_1(y), S_1(g(̃a)))$ will do, since

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

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

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

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

$$Y(̃w, w, y) = P^{|w| - |̃w|}(y).$$

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

$$P^i(y) = Y(P^i(w), w, y)$$

$$i < |y| \iff Y(P^i(w), w, y) > 0$$

$$i - |y| = |\text{DROP}(\text{DROP}(w, P^i(w)), y)|$$

The latter follows from $|w| - (|w| - i) = i$ for $i \leq |w|$, and $|\text{DROP}(m, n)| = |P^{|n|}(m)| = |m| - |n|$, implying $|\text{DROP}(w, P^i(w))| = i$ for $i \leq |w|$. Altogether, as $P^i(w)$ acts as ̃$w$ in $f(̃w, w, y, ̃a)$, the required function ̃$h$ satisfying (10) can be defined in CLO$''$ by

$$\hat{h}(w, w, y, ̃a) := \text{COND}(Y(̃w, w, y), \text{BIT}(g(̃a), |\text{DROP}(\text{DROP}(w, ̃w), y)|), h(Y(̃w, w, y, ̃a)))$$
and the $y$-section implementation $Y$ satisfying (11) is definable in CLO'' since
\[ Y(\hat{w}, w, y) = P^{1-w-|\hat{w}|}(y) = \text{DROP}(y, \text{DROP}(w, \hat{w})). \]
To see that $\hat{h}, Y \in \text{CLO}''$, just recall the proof of Lemma 4.5, and observe that
the definition of function MSP is, in fact, by CRN'' in CLO''. As a consequence,
the given definitions of both functions DROP and COND show that they belong
to CLO'', too. Thus, we obtain $Y, h \in \text{CLO}''$ as claimed.  

6 Embeddings

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

- $\text{2CLO} := [0, s_0, s_1, \pi, \text{len}, \text{bit}, \#_{\text{Bel}}, \text{case}; \text{scomp}, \text{scrn}, \text{slr}]
- \text{2NC} := [0, s_0, s_1, \pi, \text{len}, \text{bit}, \#_{\text{Bel}}, \text{case}, \text{half}, \text{drop}; \text{scomp}, \text{scrn}', \text{slr}]
- \text{2NC'} := [0, s_0, s_1, \pi, \text{len}, \text{bit}, \text{sm}, \#_{\text{AJST}}, \text{case}, \text{half}, \text{drop}; \text{scomp}, \text{scrn}', \text{slr}]
- \text{2NC''} := [0, s_0, s_1, \pi, \text{len}, \text{sm}, \#_{\text{AJST}}, \text{bcase}, \text{msp}; \text{scomp}, \text{scrn}'', \text{slr}]

To explain the new components, a function $f(y, \vec{x}; \vec{a})$ is defined by safe loga-
ritmic 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 $f = \text{scrn}(y, h)$, if for all $y, \vec{x}, \vec{a},$

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

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

\[ f(\vec{x}; y, \vec{a}) = S_{h(\vec{x}, y, \vec{a}) \mod 2}(f(\vec{x}; P(y), \vec{a})). \]

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

These function algebras should be contrasted with those of Bloch [8], namely
$\text{sc(BASE)} := [\text{BASE}; \text{scomp}, \text{safe DCR}]$ characterizing NC$^1$; and $\text{vsc(BASE)} := [\text{BASE}; \text{scomp}, \text{very safe DCR}]$ 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 NC is obtained, because scrn is neces-
sary to reach any level NC$^k$ of the NC hierarchy.

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

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

\[
\begin{align*}
  f(0, \vec{x}; \vec{a}) &= 0 \\
  f(y, \vec{x}; \vec{a}) &= S_{h(\vec{x}, y, \vec{a}) \mod 2}(f(P(y), \vec{x}; \vec{a})) \quad \text{for } y \neq 0.
\end{align*}
\]
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 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.**  
\( 2CLO \subseteq 2NC \subseteq 2NC' \subseteq 2NC'' \subseteq CLO'' \subseteq 2CLO \)

**Proof.**  
\( 2CLO \subseteq 2NC \) 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 \( 2NC' \subseteq 2NC'' \).

\( 2NC \subseteq 2NC' \) It suffices to define function \( \#Bel(w; a, b) \) in \( 2NC' \). As \( \left| P(2^x) \right| = x \) and \( p(\ ; x) = \text{drop}(\ ; x, s_1(\ ; 0)) \), hence \( p \in 2NC' \), this follows from

\[
\#Bel(w; a, b) = 2^{a+|b|} \mod 2^{|w|^2} = \text{sm}(\ ; p(\ ; \#AJST(w)), a, b).
\]

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

\[
\begin{align*}
\text{bit}(\ ; m, n) &= \left\lfloor \frac{m}{2^n} \right\rfloor \mod 2 = \text{case}(\ ; \text{msp}(\ ; m, n), 0, s_1(\ ; 0)) \\
\text{drop}(\ ; m, n) &= \left\lfloor \frac{m}{2^n} \right\rfloor \mod 2^n = \text{msp}(\ ; m, \text{len}(\ ; n)) \\
\text{half}(\ ; m) &= \left\lfloor \frac{m}{2^{\lfloor \log_2(m) \rfloor}} \right\rfloor \\
&= \text{case}(\ ; \text{len}(\ ; m), \\
&\quad \text{drop}(\ ; m, p(\ ; \text{len}(\ ; m))))).
\end{align*}
\]

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

To resolve this problem, we will construct for every function $f(\vec{x}; \vec{a})$ in $2NC'$ a simulation $f'(w; \vec{x}, \vec{a})$ in $2NC''$, 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 $2NC''$, all cases are obvious or standard, except for the case $f = \text{scrn}'(h)$ with $h \in 2NC'$. The I.H. yields a simulation $h' \in 2NC''$ 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 $f' \in 2NC''$ of $f$ by

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

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

$$y[i] := P^i(y)$$

and by unfolding the recursions we obtain the following steps:

$$f(y, \vec{x}; \vec{a}) = S_{h(y(0), \vec{x}, \vec{a}) \mod 2}(\ldots, S_{h(y(1), \vec{x}, \vec{a}) \mod 2}(\ldots, S_{h(y(|w|-1), \vec{x}, \vec{a}) \mod 2}(\ldots, S_{h(y(|w|), \vec{x}, \vec{a}) \mod 2}(0) = S_{h(w; y, \vec{x}, \vec{a}) \mod 2}(0) \mid y)$$

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:

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

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

$$Y(; w, \hat{w}, y) = P_{|w|-|\hat{w}|}(y) = \text{drop}(; y, \text{drop}(; w, \hat{w})).$$

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

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

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

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

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

We will construct for every $f \in \text{CLO}''$ a simulation $f'(w; \vec{x})$ in $2\text{CLO}$, and a (polynomial) witness $p_f$ such that $f(\vec{x}) = f'(w; \vec{x})$ whenever $|w| \geq p_f(|\vec{x}|)$.

If $f$ is $0, S_0, S_1, \pi_i^{n,m}, \cdot, | \cdot |$ or BIT, then we can define $f'$ directly in $2\text{CLO}$ using safe composition and projection. If $f$ is # then $\#(x, y) = \text{sm}(w; x, y)$ for $|w| \geq |x| + |y| + 1$, since $a \mod b = a \iff a < b$.

The cases $(\text{COMP}), (\text{WBRN}')$ are fairly standard, leaving the case $f = \text{CRN}''(h)$ with $h \in \text{CLO}''$. Here we can proceed as in the case $\text{scrn}'(h)$ of $2\text{NC}'' \subseteq 2\text{NC}''$, because in $2\text{CLO}$ function $\text{msp}(.; m, n)$ can be defined by $(\text{scrn})$ from 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 $\text{drop}(.; m, n)$ in $2\text{CLO}$.

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

**Corollary 6.3.** $\text{NC} = [0, S_0, S_1, \pi, \text{len}, \text{sm}, \#_{\text{AJST}}, \text{bcase}, \text{msp}; \text{scomp}, \text{scrn}''', \text{slr}]$

**References**


URL http://eiche.theoinf.tu-ilmenau.de/~niggl/


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 Künnemann, 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

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
V. Arvind, J. Köbler, R. Schuler
On Helping and Interactive Proof Systems

Christian Kalus, Peter Dadam
Incorporating record subtyping into a relational data model

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

Friedrich von Henke, Harald Rueß
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

Construction and Deduction Methods for the Formal Development of Software

Axel Dold
Formalisierung schematischer Algorithmen

Johannes Köbler, Osamu Watanabe
New Collapse Consequences of NP Having Small Circuits

Rainer Schuler
On Average Polynomial Time

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

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

Alfred Lupper
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

Robert Regn
Verteilte Unix-Betriebssysteme

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

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

Oleg Verbitsky
On the Largest Common Subgraph Problem

Uwe Schöning
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

Harry Buhrman, Thomas Thierauf
The Complexity of Generating and Checking Proofs of Membership

Rainer Schuler, Tomoyuki Yamakami
Structural Average Case Complexity

Klaus Achatz, Wolfram Schulte
Architecture Independent Massive Parallelization of Divide-And-Conquer Algorithms
<table>
<thead>
<tr>
<th>No.</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>95-06</td>
<td>Structure in Average Case Complexity</td>
<td>Christoph Karg, Rainer Schuler</td>
</tr>
<tr>
<td>95-07</td>
<td>ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen</td>
<td>P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe</td>
</tr>
<tr>
<td>95-08</td>
<td>Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik</td>
<td>Jürgen Kehrer, Peter Schulthiss</td>
</tr>
<tr>
<td>95-09</td>
<td>On Sets Turing Reducible to P-Selective Sets</td>
<td>Hans-Jörg Burtschick, Wolfgang Lindner</td>
</tr>
<tr>
<td>95-10</td>
<td>Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper</td>
<td>Boris Hartmann</td>
</tr>
<tr>
<td>95-12</td>
<td>Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists</td>
<td>Klaus Achatz, Wolfram Schulte</td>
</tr>
<tr>
<td>95-13</td>
<td>Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes</td>
<td>Andrea Mößle, Heiko Vogler</td>
</tr>
<tr>
<td>96-01</td>
<td>Formal Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE</td>
<td>Ercüment Canver, Jan-Tecker Gayen, Adam Moik</td>
</tr>
<tr>
<td>96-02</td>
<td>Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class</td>
<td>Bernhard Nebel</td>
</tr>
<tr>
<td>96-03</td>
<td>An Introduction to TkGofer</td>
<td>Ton Vullinghs, Wolfram Schulte, Thilo Schwinn</td>
</tr>
<tr>
<td>96-04</td>
<td>Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering</td>
<td>Thomas Beuter, Peter Dadam</td>
</tr>
<tr>
<td>96-05</td>
<td>Verification of a Prolog Compiler - First Steps with KIV</td>
<td>Gerhard Schellhorn, Wolfgang Ahrendt</td>
</tr>
<tr>
<td>96-06</td>
<td>Satisfiability Problems</td>
<td>Manindra Agrawal, Thomas Thierauf</td>
</tr>
<tr>
<td>96-07</td>
<td>A nonadaptive NC Checker for Permutation Group Intersection</td>
<td>Vikraman Arvind, Jacobo Torán</td>
</tr>
<tr>
<td>96-08</td>
<td>An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction</td>
<td>David Cyrluk, Oliver Möller, Harald Rueß</td>
</tr>
<tr>
<td>96-09</td>
<td>Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen</td>
<td>Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte</td>
</tr>
</tbody>
</table>
Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß  
Formalizing Fixed-Point Theory in PVS

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

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

Klaus Achatz, Helmut Partsch  
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

Jochen Messner  
Pattern Matching in Trace Monoids

Wolfgang Lindner, Rainer Schuler  
A Small Span Theorem within P

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

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

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

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

Manfred Reichert, Peter Dadam  
ADEPT^flex - Supporting Dynamic Changes of Workflows Without Loosing Control

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

Christian Heinlein  
Grundlagen von Interaktionsausdrücken

Christian Heinlein  
Graphische Repräsentation von Interaktionsausdrücken

Christian Heinlein  
Sprachtheoretische Semantik von Interaktionsausdrücken

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
<table>
<thead>
<tr>
<th>Year</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>98-12</td>
<td>Gerhard Schellhorn</td>
<td>Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers</td>
</tr>
<tr>
<td>98-13</td>
<td>Gerhard Schellhorn, Wolfgang Reif</td>
<td>Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers</td>
</tr>
<tr>
<td>98-14</td>
<td>Mohammad Ali Livani</td>
<td>SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN</td>
</tr>
<tr>
<td>98-15</td>
<td>Mohammad Ali Livani, Jörg Kaiser</td>
<td>Predictable Atomic Multicast in the Controller Area Network (CAN)</td>
</tr>
<tr>
<td>99-01</td>
<td>Susanne Boll, Wolfgang Klas, Utz Westermann</td>
<td>A Comparison of Multimedia Document Models Concerning Advanced Requirements</td>
</tr>
<tr>
<td>99-02</td>
<td>Thomas Bauer, Peter Dadam</td>
<td>Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation</td>
</tr>
<tr>
<td>99-03</td>
<td>Uwe Schöning</td>
<td>On the Complexity of Constraint Satisfaction</td>
</tr>
<tr>
<td>99-04</td>
<td>Ercument Canver</td>
<td>Model-Checking zur Analyse von Message Sequence Charts über Statecharts</td>
</tr>
<tr>
<td>99-05</td>
<td>Johannes Köbler, Wolfgang Lindner, Rainer Schuler</td>
<td>Derandomizing RP if Boolean Circuits are not Learnable</td>
</tr>
<tr>
<td>99-06</td>
<td>Utz Westermann, Wolfgang Klas</td>
<td>Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets</td>
</tr>
<tr>
<td>99-08</td>
<td>Vikraman Arvind, Johannes Köbler</td>
<td>Graph Isomorphism is Low for ZP^NP and other Lowness results</td>
</tr>
<tr>
<td>99-09</td>
<td>Thomas Bauer, Peter Dadam</td>
<td>Efficient Distributed Workflow Management Based on Variable Server Assignments</td>
</tr>
<tr>
<td>2000-02</td>
<td>Thomas Bauer, Peter Dadam</td>
<td>Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT</td>
</tr>
<tr>
<td>2000-03</td>
<td>Gregory Baratoff, Christian Toepfer, Heiko Neumann</td>
<td>Combined space-variant maps for optical flow based navigation</td>
</tr>
<tr>
<td>2000-04</td>
<td>Wolfgang Gehring</td>
<td>Ein Rahmenwerk zur Einführung von Leistungspunktisystemen</td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>2000-05</td>
<td>Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel</td>
<td>Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos</td>
</tr>
<tr>
<td>2000-06</td>
<td>Wolfgang Reif, Gerhard Schellhorn, Andreas Thums</td>
<td>Fehlersuche in Formalen Spezifikationen</td>
</tr>
<tr>
<td>2000-08</td>
<td>Thomas Bauer, Manfred Reichert, Peter Dadam</td>
<td>Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen</td>
</tr>
<tr>
<td>2000-09</td>
<td>Thomas Bauer, Peter Dadam</td>
<td>Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT</td>
</tr>
<tr>
<td>2000-10</td>
<td>Thomas Bauer, Manfred Reichert, Peter Dadam</td>
<td>Adaptables und verteiltes Workflow-Management</td>
</tr>
<tr>
<td>2000-11</td>
<td>Christian Heinlein</td>
<td>Workflow and Process Synchronization with Interaction Expressions and Graphs</td>
</tr>
<tr>
<td>2001-01</td>
<td>Hubert Hug, Rainer Schuler</td>
<td>DNA-based parallel computation of simple arithmetic</td>
</tr>
<tr>
<td>2001-02</td>
<td>Friedhelm Schwenker, Hans A. Kestler, Günther Palm</td>
<td>3-D Visual Object Classification with Hierarchical Radial Basis Function Networks</td>
</tr>
<tr>
<td>2001-03</td>
<td>Hans A. Kestler, Friedhelm Schwenker, Günther Palm</td>
<td>RBF network classification of ECGs as a potential marker for sudden cardiac death</td>
</tr>
<tr>
<td>2001-04</td>
<td>Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm</td>
<td>Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion</td>
</tr>
<tr>
<td>2002-01</td>
<td>Stefanie Rinderle, Manfred Reichert, Peter Dadam</td>
<td>Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata</td>
</tr>
<tr>
<td>2002-02</td>
<td>Walter Guttmann</td>
<td>Deriving an Applicative Heapsort Algorithm</td>
</tr>
<tr>
<td>2003-01</td>
<td>Manfred Reichert, Stefanie Rinderle, Peter Dadam</td>
<td>A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks</td>
</tr>
<tr>
<td>2003-02</td>
<td>Stefanie Rinderle, Manfred Reichert, Peter Dadam</td>
<td>Supporting Workflow Schema Evolution By Efficient Compliance Checks</td>
</tr>
<tr>
<td>2003-03</td>
<td>Christian Heinlein</td>
<td>Safely Extending Procedure Types to Allow Nested Procedures as Values</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-03</td>
<td>Frank Raiser</td>
<td>Semi-Automatic Generation of CHR Solvers from Global Constraint Automata</td>
</tr>
<tr>
<td>2008-04</td>
<td>Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander</td>
<td>Entscheidungsdocumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse</td>
</tr>
<tr>
<td>2008-05</td>
<td>Markus Kalb, Claudia Dittrich, Peter Dadam</td>
<td>Support of Relationships Among Moving Objects on Networks</td>
</tr>
<tr>
<td>2008-06</td>
<td>Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)</td>
<td>WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke</td>
</tr>
<tr>
<td>2008-07</td>
<td>M. Maucher, U. Schöning, H.A. Kestler</td>
<td>An empirical assessment of local and population based search methods with different degrees of pseudorandomness</td>
</tr>
<tr>
<td>2008-08</td>
<td>Henning Wunderlich</td>
<td>Covers have structure</td>
</tr>
<tr>
<td>2008-09</td>
<td>Karl-Heinz Niggl, Henning Wunderlich</td>
<td>Implicit characterization of FPTIME and NC revisited</td>
</tr>
</tbody>
</table>