Counting Classes and the Fine Structure between NC^1 and L

Samir Datta[1], Meena Mahajan[2], B. V. Raghavendra Rao[3], Michael Thomas[4],
and Heribert Vollmer[4]

[1] Chennai Mathematical Institute, India. sdatta@cmi.ac.in
[2] The Institute of Mathematical Sciences, Chennai, India. meena@imsc.res.in
[3] Universität des Saarlandes, Saarbrücken, Germany. bvrr@cs.uni-sb.de
[4] Leibniz Universität, Hannover, Germany, {thomas,vollmer}@thi.uni-hannover.de

Abstract. The class NC^1 of problems solvable by bounded fan-in circuit families of logarithmic depth is known to be contained in logarithmic space L, but not much about the converse is known. In this paper we examine the structure of classes in between NC^1 and L based on counting functions or, equivalently, based on arithmetic circuits. The classes PNC^1 and $\mathsf{C_=NC}^1$, defined by a test for positivity and a test for zero, respectively, of arithmetic circuit families of logarithmic depth, sit in this complexity interval. We study the landscape of Boolean hierarchies, constant-depth oracle hierarchies, and logarithmic-depth oracle hierarchies over PNC^1 and $\mathsf{C_=NC}^1$, provide complete problems, obtain the upper bound L for all these hierarchies, and prove partial hierarchy collapses—in particular, the constant-depth oracle hierarchy over PNC^1 collapses to its first level PNC^1, and the constant-depth oracle hierarchy over $\mathsf{C_=NC}^1$ collapses to its second level.

1 Introduction

The class NC^1 occupies a special place in the study of complexity classes inside P, owing to its robustness and multiple characterizations. It is defined as the class of languages accepted by families of circuits of polynomial size and logarithmic depth using bounded fan-in Boolean gates. By uniform NC^1 we mean the subclass where the circuit families have succinct descriptions: given a (unary) size parameter, the circuit for that size from the family can be "easily" computed. Various notions of uniformity give rise to the same class of languages, also coinciding with the class of languages accepted by logarithmic-time alternating machines $\mathsf{ALOGTIME}$. Other characterizations of NC^1 include polynomial-sized formulas, bounded-width branching programs, bounded-width circuits and programs over finite monoids.

It is known that all NC^1 languages can be accepted in logarithmic space L, but it is not known whether this containment is strict. However, there are very few candidates for languages in L not known to be in NC^1, and all these candidates lie in classes defined using the natural counting classes associated with

Supported in part by the Indian DST and the German DAAD.

NC^1, namely, $\#\mathsf{NC}^1$ and GapNC^1. The former counts "proving sub-circuits" in an NC^1 circuit (see Section 2 for formal definitions); the latter is its closure under subtraction. It is not yet known whether these functions can be evaluated in NC^1, although the best upper bound is very very close (an $O(\log^*)$ factor in depth). It is known that functions in $\#\mathsf{NC}^1$ and GapNC^1 can be evaluated in function logarithmic space FL; thus languages definable by applying simple predicates to such functions are also in L. The natural choices of predicates are a test for zero and a test for positivity, giving rise to the language classes $\mathsf{C}_=\mathsf{NC}^1$ and PNC^1 sitting between NC^1 and L. (There are also predicates testing for zero modulo a fixed prime; the resulting language classes are already known to coincide with NC^1.) A nice survey of these classes can be found in [1].

It is not clear how much structure is there between NC^1 and L if the classes are distinct. We attempt to explore the structure between NC^1 and L, based on hierarchies of language classes built upon $\mathsf{C}_=\mathsf{NC}^1$ and PNC^1. For a complexity class $\mathcal{C}$, there are three standard ways of defining the hierarchies above $\mathcal{C}$: the Boolean hierarchy $\mathsf{BH}(\mathcal{C})$, the constant-depth hierarchy using oracle gates $\mathsf{AC}^0(\mathcal{C})$, and the NC^1-oracle-gate hierarchy $\mathsf{NC}^1(\mathcal{C})$, with $\mathsf{BH}(\mathcal{C}) \subseteq \mathsf{AC}^0(\mathcal{C}) \subseteq \mathsf{NC}^1(\mathcal{C})$.

Our results: As a first step in our study, we describe the oracle hierarchies in terms of arithmetic circuits augmented with test gates. These are the arithmetic-Boolean circuits defined in [14]; with size and depth restrictions as in NC^1, and with test gates for "$= 0$?" or "> 0?", we obtain the classes $\mathsf{a}\text{-}\mathsf{NC}^1_=$ and $\mathsf{a}\text{-}\mathsf{NC}^1_>$. We observe that if each path in the circuit has $O(1)$ test gates, then $\mathsf{a}\text{-}\mathsf{NC}^1_=$ and $\mathsf{a}\text{-}\mathsf{NC}^1_>$ coincide with $\mathsf{AC}^0(\mathsf{C}_=\mathsf{NC}^1)$ and $\mathsf{AC}^0(\mathsf{PNC}^1)$ respectively (Proposition 3.4). However, there is a subtlety in similarly characterizing $\mathsf{NC}^1(\mathsf{C}_=\mathsf{NC}^1)$ and $\mathsf{NC}^1(\mathsf{PNC}^1)$. We introduce a syntactic restriction on the arithmetic-Boolean circuits giving rise to a reasonable definition, and show that (1) the classes so defined coincide with $\mathsf{NC}^1(\mathsf{C}_=\mathsf{NC}^1)$ and $\mathsf{NC}^1(\mathsf{PNC}^1)$ (Proposition 3.5), and (2) as expected, are indeed contained in L (Theorem 3.9). On the other hand, without this restriction, the best upper bound we can show for the arithmetic circuit hierarchy is the complexity class TC^1 (Theorem 3.10), which subsumes L and even nondeterministic logspace NL, but is contained in NC^2.

Next, we show that the constant-depth hierarchy over PNC^1 (and hence also the Boolean hierarchy) collapses to PNC^1 (Theorem 4.1). We adapt the proof of [11], where an analogous result for PL is shown. One difficulty in the adaptation is showing the required normal form for GapNC^1 circuits. We use the equivalent characterization of GapNC^1 as arithmetic bounded-width branching programs $\mathsf{GapBWBP}$, and establish the normal form here. Another difficulty is computing an exponential sum; we use the notion of read-once certified circuits and read-once exponential sums, introduced in [10], to to carry the proof through.

Finally, we examine the hierarchies over $\mathsf{C}_=\mathsf{NC}^1$. Since $\mathsf{C}_=\mathsf{NC}^1$ is not even known to be closed under complementation, we do not expect a collapse all the way down. Our first result is a characterization of the Boolean hierarchy over $\mathsf{C}_=\mathsf{NC}^1$ as the class of languages described by checking feasibility of small systems of linear equations, where the coefficients themselves are GapNC^1-computable functions of the input word (Theorem 5.5). Our second result is that the constant-depth hierarchy over $\mathsf{C_{=}NC^1}$ collapses to a class slightly weaker than the second level (Theorem 5.9). Both these results appear as analogues of known results [2] for the corresponding logarithmic-space class $\mathsf{C_{=}L}$, but require substantially different proofs.

Also, unlike in the case of PL and $\mathsf{C_{=}L}$, our results do not seem to go through for the $\mathsf{NC^1}$-hierarchies over $\mathsf{PNC^1}$ and $\mathsf{C_{=}NC^1}$.

2 Background

For any language L, χ_L denotes its characteristic function: $\chi_L(x) := 1$ if $x \in L$, $\chi_L(x) := 0$ if $x \notin L$.

Boolean circuits and language classes: We denote by L the class of languages accepted by deterministic logarithmic-space Turing machines.

We consider Boolean circuits with internal gates labelled $\vee$, $\wedge$, or $\neg$. By $\mathsf{NC^1}$ we denote the class of languages which can be accepted by a family $\{C_n\}_{n\geq 0}$ of Boolean circuits of polynomial size whose depth is bounded by $O(\log n)$, with each gate having constant fan-in. The class $\mathsf{AC^0}$ denotes the set of languages accepted by a Boolean circuit family $\{C_n\}_{n\geq 0}$ of polynomial size and constant depth, with unbounded fan-in. Without loss of generality, we can assume that negation gates appear only at the leaves, and that the $\mathsf{AC^0}$ and $\mathsf{NC^1}$ circuits are actually formulas: every gate has out-degree one. An $\mathsf{NC^0}$ circuit is a Boolean circuit, or formula, of constant size, with each gate having constant fan-in. We denote by $\mathsf{AC^0}_k$ (respectively) $\mathsf{NC^0}_k$ the polynomial size (respectively, constant size) circuit families of depth at most k.

By $\mathsf{TC^0}$ and $\mathsf{TC^1}$ we denote the class of languages decided by circuit families of polynomial size and constant (respectively, logarithmic) depth, where each gate is either a negation gate or an unbounded fan-in majority gate: it outputs 1 if and only if more than half of its inputs are 1. Integer addition and multiplication are known to be in $\mathsf{TC^0}$.

A branching program (BP for short) is a layered acyclic graph G with edges labelled by constants (0 or 1) or literals, and with two special vertices s and t. It accepts an input x if there is an $s \rightsquigarrow t$ path where each edge is labelled by a true literal or the constant 1; we call such a path an *accepting path* on input x. BWBP denotes the class of languages that can be accepted by families of polynomial size bounded width branching programs $\{G_n\}_{n\geq 0}$, where the graph G_n considers n variables. It is known that BWBP equals $\mathsf{NC^1}$ ([4]). Restricted to uniform circuits (with appropriate notions of uniformity, see for instance [13]), it is known that $\mathsf{NC^1} = \mathsf{BWBP} \subseteq \mathsf{L}$.

Proposition 2.1 (Known containments).
$\mathsf{AC^0} \subseteq \mathsf{TC^0} \subseteq \mathsf{NC^1} = \mathsf{BWBP} \subseteq \mathsf{L} \subseteq \mathsf{TC^1} \subseteq \mathsf{DSPACE}(\log^2 n) \cap \mathsf{P}$.

Arithmetic circuit classes: For the purposes of this paper, an arithmetic circuit is a circuit where the gates are labelled from the set $\{+, \times, -1, 0, 1, x_1, \ldots, x_n\}$. The gates $+$ and $\times$ are the addition and multiplication operations over $\mathbb{Z}$. Such a circuit computes a function $f : \{0,1\}^n \rightarrow \mathbb{Z}$.

An $\mathsf{a\text{-}NC}^1$ circuit family $\{C_n\}_{n\geq 0}$ is a family of bounded fan-in arithmetic circuits where for each n, C_n is of size polynomial in n, depth logarithmic in n, and computes a function $f_n : \{0,1\}^n \to \mathbb{Z}$. The family computes the function $f : \{0,1\}^* \to \mathbb{Z}$ where $f(x) := C_{|x|}(x)$. GapNC^1 is the class of functions computed by $\mathsf{a\text{-}NC}^1$ circuit families. The analogous arithmetic class for constant-depth unbounded fan-in circuits is denoted by $\mathsf{a\text{-}AC}^0$.

An arithmetic branching program is a BP B where edges are labelled by literals or constants from the set $\{-1,0,1\}$. For an $s \rightsquigarrow t$ path P, let $wt(P(a))$ denote the product of all the edge labels in P under the assignment a. Then the function computed by B is defined as follows:

$$\text{for all } a \in \{0,1\}^n \qquad f(a) := \sum_{P \text{ is an } s \rightsquigarrow t \text{ path in } B} wt(P(a))$$

An $\mathsf{a\text{-}BWBP}$ family $\{B_n\}_{n\geq 0}$ is a family of arithmetic branching programs of polynomial size and bounded width. $\mathsf{GapBWBP}$ is the class of functions computed by $\mathsf{a\text{-}BWBP}$ program families.

For a Boolean (no edge labelled -1) BP B and an input assignment a, let $\#[s \rightsquigarrow t](a)$ denote the the number of $s \rightsquigarrow t$ paths in B under the assignment a. $\#\mathsf{BWBP}$ is the class of functions : $\{0,1\}^* \to \mathbb{N}$ computed by BWBP. The class $\mathsf{DiffBWBP}$ is the closure of $\#\mathsf{BWBP}$ under finite subtractions; $\mathsf{DiffBWBP} = \{f - g \mid f, g \in \#\mathsf{BWBP}\}$.

The above three classes coincide:

Proposition 2.2 ([6]). $\mathsf{GapNC}^1 = \mathsf{GapBWBP} = \mathsf{DiffBWBP}$.

We will often use the following equivalent form for GapNC^1 functions: for any GapNC^1 function f, there is a BWBP B with start node s, two target nodes t_1 and t_2, and $f(a) = \#[s \rightsquigarrow t_1](a) - \#[s \rightsquigarrow t_2](a)$. We say that B *gap-represents* the function f.

It is known that NC^1 circuits can be made unambiguous [9]. In terms of arithmetic circuits, this yields:

Proposition 2.3. *Let L be any AC^0 (or NC^1) language. Then there is an $\mathsf{a\text{-}AC}^0$ ($\mathsf{a\text{-}NC}^1$, respectively) circuit family C that does not use the constant -1 such that for each string w, $C(w) = \chi_L(w)$.*

The classes $\mathsf{C_{=}NC}^1$ and PNC^1, central to this paper, are defined as follows.

$$\mathsf{C_{=}NC}^1 := \left\{ L \in \{0,1\}^* \;\middle|\; \begin{array}{l}\text{for some } f \in \mathsf{GapNC}^1, \text{ for all } x \in \{0,1\}^*, \\ x \in L \text{ if and only if } f(x) = 0.\end{array} \right\}$$

$$\mathsf{PNC}^1 := \left\{ L \in \{0,1\}^* \;\middle|\; \begin{array}{l}\text{for some } f \in \mathsf{GapNC}^1, \text{ for all } x \in \{0,1\}^*, \\ x \in L \text{ if and only if } f(x) > 0.\end{array} \right\}$$

Proposition 2.4 ([6]).

1. $\mathsf{NC}^1 \subseteq \mathsf{C_{=}NC}^1 \subseteq \mathsf{PNC}^1 \subseteq \mathsf{L}$.
2. $\mathsf{C_{=}NC}^1$ *is closed under union and intersection.*

3. PNC^1 *is closed under union, intersection and complementation.*

Arithmetic-Boolean circuits: Let a *test gate for "=0?"* (respectively *">0?"*) be a unary gate that outputs 1 if its input is equal to 0 (respectively greater than 0) and 0 otherwise. Define an $\mathsf{a\text{-}NC}^1_=$ *circuit* (respectively $\mathsf{a\text{-}NC}^1_>$ *circuit*) to be an arithmetic circuit of logarithmic depth and polynomial size over Boolean input gates, binary +- and ×-gates, constants -1, 0 and 1 as well as test gates for "=0?" (respectively ">0?"). From the definitions, it follows that

Proposition 2.5. *A language L is in $\mathsf{C_=NC}^1$ (or PNC^1) if and only if χ_L can be computed by an $\mathsf{a\text{-}NC}^1_=$ (respectively $\mathsf{a\text{-}NC}^1_>$) circuit family in which each circuit has exactly one test gate appearing as the output gate.*

Read-once certificates: Let B be a branching program on variables $X = \{x_1, \ldots, x_n\} \cup Y = \{y_1, \ldots, y_m\}$. B is said to be *read-once* certified in Y if there are indices $i_0 = 1 < i_1 < i_2 < \ldots < i_m$ such that variable y_j appears only between layer i_{j-1} and i_j. By specialising the arguments in [10] to the counting classes, we can compute exponential sums over the variables in Y efficiently.

Proposition 2.6 ([10]). *Let $f(X,Y)$ be a function computed by an $\mathsf{a\text{-}BP}$ B of size s and width w, read-once certified in Y. Let g be the function defined as $g(X) := \sum_{e\in\{0,1\}^m} f(X,e)$. Then g can be computed by an $\mathsf{a\text{-}BP}$ of size $\mathsf{poly}(s)$ and width $O(w^2)$. Hence, if $f \in \mathsf{GapBWBP}$, then $g \in \mathsf{GapBWBP}$.*

Miscellaneous We denote by $\mathcal{C}_1 \cdot \mathcal{C}_2$ a circuit which can be split horizontally into two parts, with the top part being a circuit of type $\mathcal{C}_1$, and all its inputs being either circuit inputs (literals or constants) or circuits of type $\mathcal{C}_2$. We denote by $[\mathcal{C}]$ an oracle gate for a language in $[\mathcal{C}]$. Thus $[\mathcal{C}] \cdot \mathsf{AC}^0$ is the class of all languages accepted by $\mathsf{AC}^0(\mathcal{C})$ oracle circuits such that each circuit has a single oracle gate at the output, and each input bit to the oracle gate is the output of an AC^0 sub-circuit.

3 Hierarchies: Definitions and Upper Bounds

Among the simplest is the Boolean hierarchy, which characterizes the languages expressible as Boolean combinations of any constant number of languages from respectively $\mathsf{C_=NC}^1$ or PNC^1.

Definition 3.1 (The Boolean Hierarchy). *Let $\mathcal{C}$ be a complexity class. The* Boolean hierarchy over $\mathcal{C}$ *is defined as the set of languages L for which there exists an NC^0 circuit C with k inputs and $A_1, \ldots, A_k \in \mathcal{C}$ such that for all $x \in \{0,1\}^*$,*

$$x \in L \Longleftrightarrow C(\chi_{A_1}(x), \chi_{A_2}(x), \ldots, \chi_{A_k}(x)) = 1$$

We denote this class of languages by $\mathsf{NC}^0 \cdot \mathcal{C}$ or $\mathsf{BH}(\mathcal{C})$.

Remark 3.2. A more standard way of defining the Boolean hierarchy is to define the levels $\mathsf{BH}_0(\mathcal{C}) := \mathcal{C}$, $\mathsf{BH}_i(\mathcal{C}) := \{L_1 \triangle L_2 \mid L_1, L_2 \in \mathsf{BH}_{i-1}(\mathcal{C})\}$, and then take the union $\bigcup_{i>0} \mathsf{BH}_i(\mathcal{C})$. If $\mathcal{C}$ is closed under union and intersection, then these definitions coincide with each other and with the definition of $\mathsf{NC}^0 \cdot \mathcal{C}$ above ([8]).

The other way of defining hierarchies is via oracle queries. As shown in [3] (see also [2]), nesting queries above a base machine is equivalent to adding oracle gates in an AC^0 circuit. And it also often turns out to be equivalent to adding oracle gates in an NC^1 circuit. We present the oracle-circuit definitions below.

Let L be any language. An $\mathsf{AC}^0(L)$ *circuit family* is a sequence $\{C_n\}_{n\geq 0}$ of AC^0circuits containing additional oracle gates for L of unbounded fan-in. Similarly, an $\mathsf{NC}^1(L)$ *circuit family* is a sequence $\{C_n\}_{n\geq 0}$ of NC^1 circuits with additional oracle gates for L of unbounded fan-in such that oracle gates of fan-in m account for depth $\lceil \log m \rceil$.

Definition 3.3 (The AC^0 and the NC^1 Hierarchy). *Let $\mathcal{C}$ be a complexity class. Then $\mathsf{AC}^0(\mathcal{C})$ (respectively $\mathsf{NC}^1(\mathcal{C})$) is defined to comprise those problems decidable by an $\mathsf{AC}^0(L)$ (respectively $\mathsf{NC}^1(L)$) circuit family for some $L \in \mathcal{C}$.*

We now characterize the hierarchies using Arithmetic-Boolean Circuits. From Proposition 2.5, we know that $\mathsf{C_=NC}^1$ and PNC^1 have equivalent arithmetic-Boolean circuits. It is natural to ask whether there are equivalent such circuits for the hierarchies above these classes. For the AC^0 hierarchy, this is easy to see; we show below that $\mathsf{AC}^0(\mathsf{C_=NC}^1)$ and $\mathsf{AC}^0(\mathsf{PNC}^1)$ can be characterized using arithmetic-Boolean circuits. We need the notion of nesting depth: in a circuit C, the nesting depth of gates of a type t is the largest number k such that some path from the output to a leaf of C goes through exactly k gates of type t.

Proposition 3.4. *$\mathsf{AC}^0(\mathsf{C_=NC}^1)$ (respectively $\mathsf{AC}^0(\mathsf{PNC}^1)$) equals the class of languages decidable by $\mathsf{a\text{-}NC}^1_=$ (respectively $\mathsf{a\text{-}NC}^1_>$) circuit families such that the nesting depth of test gates is bounded by a constant and the output gate of each circuit is a test gate.*

It is tempting to believe that dropping the requirement on nesting depth of test gates will characterize $\mathsf{NC}^1(\mathsf{C_=NC}^1)$ and $\mathsf{NC}^1(\mathsf{PNC}^1)$. This, however, is not the case. The conversion from left to right ($\mathsf{NC}^1(\mathsf{C_=NC}^1)$ to $\mathsf{a\text{-}NC}^1_=$) goes through, but for the converse, the requisite depth bound does not follow. We describe a certain condition under which we can obtain an exact characterization.

Let C be an $\mathsf{a\text{-}NC}^1_=$ circuit (respectively $\mathsf{a\text{-}NC}^1_>$ circuit) with n inputs and let $g_1, \ldots, g_m$ enumerate all of its test gates. Denote by S_i the maximal connected sub-circuit of C rooted at g_i that consists of $+$, $\times$-gates and the constants -1, 0, 1; these are the "blobs" in the proof of Proposition 3.4. As the depth of C is logarithmic in the number of its inputs, we may without loss of generality assume that $S_1, \ldots, S_m$ induce a partition of the non-input gates of C. Thus any path from the output to a leaf in C goes through a chain of these blobs. There can be $O(\log n)$ blobs on any such chain, and the logarithm of the size of a blob can be as large as $\theta(\log n)$, and this causes the problem in replicating the above proof. We "define away" the problem: We say that C has the *small-blob-chains* property if for every path π from the root of C to an input gate or a constant,

$$\sum_{g_i \text{ occurs in } \pi} \log |S_i| \in O(\log n).$$

Now we can characterize exactly the NC^1 hierarchies above $\mathsf{C_=NC}^1$ and PNC^1.

Proposition 3.5. $\mathsf{NC}^1(\mathsf{C_{=}NC}^1)$ *(respectively* $\mathsf{NC}^1(\mathsf{PNC}^1)$*) equals the class of languages decidable by* $\mathsf{a\text{-}NC}^1_=$ *(respectively* $\mathsf{a\text{-}NC}^1_>$*) circuit families with the small-blob-chains property in which the output gate of each circuit is a test gate.*

There exist arithmetic-Boolean circuits violating the small-blob-chains property. (See, for instance, the example in the Appendix.) Hence, dropping the small-blob-chains property from the circuits in Proposition 3.5 leads to presumably different class of languages. We denote these classes by AH, for arithmetic hierarchy, defined analogously to the classes figuring in Propositions 2.5 and 3.5.

Definition 3.6 (Arithmetic Circuit Hierarchies over $\mathsf{C_{=}NC}^1$ and PNC^1). *A language L is said to be in* $\mathsf{AH}(\mathsf{C_{=}NC}^1)$ *(or* $\mathsf{AH}(\mathsf{PNC}^1)$*) if and only if χ_L can be computed by an* $\mathsf{a\text{-}NC}^1_=$ *(respectively* $\mathsf{a\text{-}NC}^1_>$*) circuit family such that in each circuit, the output gate is a test gate.*

The following chain of inclusions holds.

Observation 3.7.

$$\begin{array}{ccccccccc}
\mathsf{C_{=}NC}^1 & \subseteq & \mathsf{BH}(\mathsf{C_{=}NC}^1) & \subseteq & \mathsf{AC}^0(\mathsf{C_{=}NC}^1) & \subseteq & \mathsf{NC}^1(\mathsf{C_{=}NC}^1) & \subseteq & \mathsf{AH}(\mathsf{C_{=}NC}^1) \\
\rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} & & \rotatebox{-90}{$\subseteq$} \\
\mathsf{PNC}^1 & \subseteq & \mathsf{BH}(\mathsf{PNC}^1) & \subseteq & \mathsf{AC}^0(\mathsf{PNC}^1) & \subseteq & \mathsf{NC}^1(\mathsf{PNC}^1) & \subseteq & \mathsf{AH}(\mathsf{PNC}^1).
\end{array}$$

Remark 3.8. We can also augment the $\mathsf{a\text{-}NC}^1_=$ and $\mathsf{a\text{-}NC}^1_>$ circuits in Definition 3.6 by allowing oracle gates, with $\lceil \log(\text{fan-in}(g)) \rceil$ charged to the depth of each such gate g. Since, without loss of generality, we deal with languages over a binary alphabet, the inputs to the oracle gate must be Boolean inputs. But the circuit computes arithmetic values, except at test gates. Thus, we will require that all the inputs to an oracle gate are either Boolean circuit inputs (literals or the constants 0,1, but not -1) or the outputs of test gates. It can be shown that allowing $\mathsf{C_{=}NC}^1$ oracle gates in $\mathsf{a\text{-}NC}^1_=$ circuits, or PNC^1 oracle gates in $\mathsf{a\text{-}NC}^1_>$ circuits, with this condition, does not add to the power of the circuit families beyond $\mathsf{AH}(\mathsf{C_{=}NC}^1)$ and $\mathsf{AH}(\mathsf{PNC}^1)$ respectively.

We now show some upper bounds. We first establish that the AC^0 and the NC^1 hierarchies over $\mathsf{C_{=}NC}^1$ and PNC^1 are contained in L. By the containments depicted in Observation 3.7, it suffices to show this bound for $\mathsf{NC}^1(\mathsf{PNC}^1)$.

Theorem 3.9. $\mathsf{NC}^1(\mathsf{PNC}^1) \subseteq \mathsf{L}$.

We give (in the appendix) two proofs of this theorem; one works directly with the oracle circuit, and the second works with the $\mathsf{a\text{-}NC}^1_=$ circuit. By Proposition 3.5, $\mathsf{AH}(\mathsf{PNC}^1)$ differs from $\mathsf{NC}^1(\mathsf{PNC}^1)$ only in the small-blob-chains property. In the absence of this property, the recursive simulation in the second proof of Theorem 3.9 yields only a $O(\log^2 n)$ space bound. Also, since the log-space evaluation of each blob may not be read-once in its inputs, each blob may have to be evaluated several times. So we cannot obtain a polynomial time bound for the recursive procedure. However, using a bottom-up evaluation, we can show that $\mathsf{AH}(\mathsf{PNC}^1)$ circuits can be evaluated in TC^1.

Theorem 3.10. $\mathsf{AH}(\mathsf{PNC}^1) \subseteq \mathsf{TC}^1$.

4 The PNC^1 hierarchy collapses

In this section we show that the constant-depth PNC^1 hierarchy, $\mathsf{AC}^0(\mathsf{PNC}^1)$, collapses to the base level.

Theorem 4.1. $\mathsf{AC}^0(\mathsf{PNC}^1) = \mathsf{PNC}^1$.

Proof. Since PNC^1 is closed under complementation, and since unbounded fan-in $\vee$ and $\wedge$ functions are in NC^1 and hence in PNC^1, we can assume without loss of generality that the $\mathsf{AC}^0(\mathsf{PNC}^1)$ circuit has only oracle gates. Theorem 4.2 below shows how to collapse two adjacent levels of PNC^1 oracle gates into one. Applying this repeatedly gives the desired result. □

The rest of this section is devoted to proving Theorem 4.2:

Theorem 4.2. $[\mathsf{PNC}^1] \cdot [\mathsf{PNC}^1] = \mathsf{PNC}^1$.

We adapt the techniques of [11] to the case of constant width branching programs. Also, as in [11], we use the polynomial technique developed earlier ([5, 7]) to show closure properties of the complexity class PP. A new ingredient we need is read-once certified circuits and exponential sums, from [10].

4.1 Overview of the Collapse Argument

Consider a language L in $[\mathsf{PNC}^1] \cdot [\mathsf{PNC}^1]$. Then there is a language $H \in \mathsf{PNC}^1$ and a circuit family $\{C_n\}$ accepting L where each C_n has depth 2 and has only oracle gates for H. That is, the output gate g is an oracle gate whose inputs are themselves oracle gates or literals or constants. Without loss of generality, we can assume that in fact all inputs to g are outputs of oracle gates. Let g have fan-in t. On input x, its inputs are $\chi_H(Y_1), \chi_H(Y_2), \ldots, \chi_H(Y_t)$, where each Y_i is a projection (re-ordering of bits) of the input x.

Let f be the GapNC^1 function witnessing that $H \in \mathsf{PNC}^1$. Then there is a $\mathsf{a\text{-}BWBP}$ family computing f. The idea is to consider the $\mathsf{a\text{-}BWBP}$ B for inputs of length t, say $y_1, \ldots, y_t$, and try to replace each edge labeled y_i by a copy of the $\mathsf{a\text{-}BWBP}$ on Y_i. However, since Y_i is the input to an oracle gate, we want a 0-1 value for the sign of $f(Y_i)$, not the value of $f(Y_i)$ itself. If the sign function can be computed by a suitable polynomial function, then we can apply this function to each $f(Y_i)$ to get another GapNC^1 function. Unfortunately, the sign function cannot be represented in this fashion. However, it can be approximated by rational functions (ratios of polynomials); this approximation was first used in [5], and later in [7] and [11]. We follow the presentation from [11]. For completeness, we describe the polynomials in the Appendix.

To show that using such approximations is valid, we require that B satisfies a certain condition: All paths should have equal susceptibility to error, so as to not change the overall outcome. In particular, since a y_i edge label corresponds to using the output of an oracle gate, and since different oracle gates can have different errors, we will require that each path has exactly the same multiset of edge labels, independent of the input. This is a strong normal form. Such a normal form was required to collapse $\mathsf{AC}^0(\mathsf{PL})$ to PL, and was shown in [11]. We show a corresponding normal form for a-BWBPs in Lemma 4.3.

Finally, we need to show that there is a GapBWBP function h which has the same sign as the value of the a-BWBP B with the rational approximations in place. In [11], the analogous result is shown by describing an appropriate probabilistic log-space machine. In the GapBWBP setting, things are a bit more complicated since we have only $O(1)$ storage. We get around this by using the notion of exponential sums over read-once certified circuits, introduced in [10]. The GapBWBP family computing the desired h is described in Section 4.2, completing the proof of Theorem 4.2.

4.2 Some Details of the Proof

We introduce a notation here. A node v in a BP B is called a *nondeterministic* node if there is an input assignment for which v has two out-edges labelled 1. We show the following normal form for branching programs computing functions in GapNC^1; this is analogous to Lemma 3.1 in [11] for PL and #L functions.

Lemma 4.3. *Let f be a function in GapNC^1. Then there exists a branching program Q of width $O(1)$ such that*

1. *Q has a single start node s and two terminal nodes t_1 and t_2;*
2. *every path originating from s ends at either t_1 or t_2 and nowhere else;*
3. *any path of Q on any given input x contains exactly q nondeterministic nodes;*
4. *every edge is labelled by a literal y_i or $\neg y_i$;*
5. *on any input y, Q has exactly 2^q paths, originating from s, where $q = q(n) \leq \mathsf{poly}(n)$;*
6. *$f = \#[s \rightsquigarrow t_1] - \#[s \rightsquigarrow t_2]$.*

We use the following characterization of the class PNC^1.

Proposition 4.4. *A language L belongs to PNC^1 if and only if there is a function $f \in \mathsf{GapNC}^1$ such that if $x \in L$ then $f(x) \geq 1$ and if $x \notin L$ then $f(x) \leq -1$.*

We now complete the proof of Theorem 4.2. Let $L \in [\mathsf{PNC}^1] \cdot [\mathsf{PNC}^1]$. As described in Section 4.1, there is a GapNC^1 function f and a circuit family accepting L such that for any word x, $f(x) \neq 0$ and $x \in L \Leftrightarrow f(b_1, \ldots, b_t) > 0$, where $b_i = \chi_H(Y_i)$ and so $b_i = 1$ if $f(Y_i) > 0$; $b_i = 0$ otherwise. Each query string Y_i is obtained from x by a projection and is an oracle query at the bottom layer; b_i is the oracle reply.

Replace each b_i by a variable y_i and apply Lemma 4.3 to get a polynomial size branching program Q, with three special nodes s, t_1, and t_2, computing $f(Y)$ on t-bit inputs via the gap $f = \#[s \rightsquigarrow t_1] - \#[s \rightsquigarrow t_2]$. Note that for every layer k of Q, there is a variable $u_k \in Y$ such that the edges from layer k to layer $k+1$ are labelled from the set $\{u_k, \neg u_k\}$. Note that all the u_k need not be distinct. Henceforth, we denote by y_k and Y_k the variable at layer k of Q and the corresponding query string, respectively. Without loss of generality we can assume that every layer is a nondeterministic layer. Let Q have p layers. Then every pair of bit-strings w, u, each of length p, uniquely represents a path in the BP Q, by considering the ith bit w_i of w as the query answer at the ith layer and ith bit u_i of u as the nondeterministic choice. For w, u, with $|w| = |u| = p$, define the Boolean function $e(x, w, u)$ as follows: $e(x, w, u) = 1$ if and only if the path of Q represented by the strings w and u on input x is an accepting path (that is, it terminates at t_1). Now define the following functions:

$$\begin{aligned} T(x) &:= \sum_{u,w \in \{0,1\}^p} e(x, w, u)\tilde{S}(x, w), \\ a(x) &:= \sum_{u,w \in \{0,1\}^p} e(x, w, u)\tilde{\alpha}(x, w), \text{ and} \\ h(x) &:= 4a(x) - 2^{p+1}\beta(x). \end{aligned}$$

Here, $\tilde{S}(x, w)$, $\tilde{\alpha}(x, w)$ and $\beta(x)$ are Ogihara's polynomials (see Appendix).

As shown in [11] (see Lemma 6.4), $T(x) = a(x)/\beta(x)$. Using the properties of $\tilde{S}$ we have:

Lemma 4.5. *[11] If $x \in L$ then $T(x) > 2^{p-1}$, and if $x \notin L$ then $T(x) < 2^{p-1}$. Hence, $x \in L$ if and only if $h(x) \geq 0$.*

Now it suffices to prove the following; see the appendix for a complete proof.

Lemma 4.6. $h(x) \in \mathsf{GapBWBP}{=}\mathsf{GapNC}^1$.

5 The Hierarchy above $\mathsf{C}_=\mathsf{NC}^1$

Since we do not even know if $\mathsf{C}_=\mathsf{NC}^1$ is closed under complementation, we cannot hope for a direct collapse of the hierarchies above $\mathsf{C}_=\mathsf{NC}^1$ all the way down to $\mathsf{C}_=\mathsf{NC}^1$. However, we show here two partial collapses. For the analogous class $\mathsf{C}_=\mathsf{L}$, it has been shown in [2] that the hierarchy collapses to $\mathsf{L}^{\mathsf{C}_=\mathsf{L}}$, and that testing feasibility of systems of linear equations FSLE is complete for this class. At the level of NC^1, we show that the analogous situation splits into two counterparts. We define an appropriate non-trivial notion of constant-dimension FSLE and show that it is complete for the Boolean hierarchy over $\mathsf{C}_=\mathsf{NC}^1$, $\mathsf{BH}(\mathsf{C}_=\mathsf{NC}^1)$. We then show that the constant-depth hierarchy over $\mathsf{C}_=\mathsf{NC}^1$, $\mathsf{AC}^0(\mathsf{C}_=\mathsf{NC}^1)$, collapses to a certain level within the hierarchy that we denote $\mathsf{AC}^0 \cdot \mathsf{C}_=\mathsf{NC}^1$; this is contained in the second level of the hierarchy.

5.1 The Boolean Hierarchy above $\mathsf{C}_=\mathsf{NC}^1$

Definition 5.1. *For any $k \in \mathbb{N}$, and any class $\mathcal{C}$ of functions from words to integers, the language class $\mathsf{FSLE}^k[\mathcal{C}]$ is defined as follows: A language L belongs to the class $\mathsf{FSLE}^k[\mathcal{C}]$ if there are functions $A_{ij} \in \mathcal{C}$ for $1 \leq i, j \leq k$ and a vector $b \in \mathbb{Z}^k$ such that for each $w \in \{0,1\}^*$, $w \in L$ if and only if the system $Az = b$ of linear equations in k variables z_j, where $A_{ij} = A_{ij}(w)$, has a feasible solution over the rationals. The class $\mathsf{FSLE}^{bdd}[\mathcal{C}]$ is the union of $\mathsf{FSLE}^k[\mathcal{C}]$ over all k.*

Proposition 5.2. *The following two containments hold:*

$$\mathsf{coC_=NC^1} \subseteq \mathsf{FSLE^1[GapNC^1]}, \qquad \mathsf{C_=NC^1} \subseteq \mathsf{FSLE^2[GapNC^1]}.$$

We prove something stronger, by showing that $\mathsf{FSLE}^{bdd}[\mathsf{GapNC}^1]$ can express conjunctions and negations.

Lemma 5.3. $\mathsf{NC^0}_d \cdot \mathsf{C_=NC^1} \subseteq \mathsf{FSLE}^{3(2^{d+1}-1)}[\mathsf{GapNC}^1]$.

And we establish a converse as well, with somewhat different parameters. The proof uses the fact that to check feasibility, the ranks of finitely many sub-matrices need to be computed.

Lemma 5.4. $\mathsf{FSLE}^k[\mathsf{GapNC}^1] \subseteq \mathsf{NC^0}_{3k} \cdot \mathsf{C_=NC^1}$.

From Lemmas 5.3 and 5.4, we have shown the following:

Theorem 5.5. $\mathsf{NC^0} \cdot \mathsf{C_=NC^1} = \mathsf{FSLE}^{bdd}[\mathsf{GapNC}^1]$.

5.2 The $\mathsf{AC^0}$ Hierarchy above $\mathsf{C_=NC^1}$

We now show the collapse of the constant-depth hierarchy over $\mathsf{C_=NC^1}$, that is, we prove $\mathsf{AC^0(C_=NC^1)} = \mathsf{AC^0}_3 \cdot [\mathsf{C_=NC^1}]$. First we set up some notation.

Let $\mathsf{AC^0}_k(\mathcal{C})$ denote the class of languages accepted by $\mathsf{AC^0}$ oracle circuits, where the oracle gates are for a language in $\mathcal{C}$, and where on any root-to-leaf path, the number of oracle gates encountered is at most k. (This is in analogy with $\mathsf{AC^0}_k$ denoting depth-k $\mathsf{AC^0}$ circuits.) Then, $\mathsf{AC^0}_k(\mathcal{C})$ is exactly $\mathsf{AC^0} \cdot [\mathcal{C}] \cdot \mathsf{AC^0} \ldots$ (k times) $\ldots [\mathcal{C}] \cdot \mathsf{AC^0}$. In particular, when $\mathcal{C} = \mathsf{C_=NC^1}$, using notation from Proposition 3.4 we can see that $\mathsf{AC^0}_k(\mathsf{C_=NC^1})$ equals $\mathsf{a\text{-}NC^1_=}$ circuits where the nesting depth of the test gates is at most k.

Proposition 5.6. $[\mathsf{C_=NC^1}] \cdot \mathsf{AC^0} = \mathsf{C_=NC^1}$ *and* $[\mathsf{coC_=NC^1}] \cdot \mathsf{AC^0} = \mathsf{coC_=NC^1}$.

The heart of our collapse result is the following lemma, stating that two adjacent levels of $\mathsf{coC_=NC^1}$ oracle gates can be combined into one.

Lemma 5.7. $[\mathsf{coC_=NC^1}] \cdot [\mathsf{coC_=NC^1}] \subseteq \mathsf{AC^0} \cdot [\mathsf{coC_=NC^1}]$. *In particular, the* $\mathsf{AC^0}$ *circuitry is of depth 3, with an* OR *of* AND*s and some negations at the leaves.*

The result follows immediately from Lemma 5.8 below.

Lemma 5.8. *Let* $h : \{0,1\}^t \longrightarrow \{0,1\}$, $f_1, f_2, \ldots, f_t : \{0,1\}^n \longrightarrow \{0,1\}$ *be functions in* GapNC^1, *where for all* w, $f_i(w) \geq 0$. *Then for some* $T \in t^{O(1)}$, *there exist* GapNC^1 *functions* $g_1, g_2, \ldots, g_T : \{0,1\}^n \longrightarrow \{0,1\}$ *and an* $\mathsf{AC^0}$ *circuit* H *on* T *inputs such that, for all* $w \in \{0,1\}^n$,

$$h(b_1, b_2, \ldots, b_t) \neq 0 \Longleftrightarrow H(d_1, d_2, \ldots, d_T) = 1$$

where $$b_i := \begin{cases} 1 & \text{if } f_i(w) \neq 0, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{and} \qquad d_j := \begin{cases} 1 & \text{if } g_j(w) \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

To establish this, we define appropriate symmetric polynomials such that when evaluated at the values $f_i(w)$, a simple predicate involving them reveals the value of h. We then use the fact that the symmetric polynomials are efficiently computable over fields.

Using Proposition 5.6 and Lemma 5.7, we get our collapse result.

Theorem 5.9. *The* AC^0 *hierarchy over* $\mathsf{C}_=\mathsf{NC}^1$ *collapses to its first level, requiring a single layer of oracle gates and a depth-3 circuit above it,*

$$\mathsf{AC}^0(\mathsf{C}_=\mathsf{NC}^1) = \mathsf{AC}^0 \cdot [\mathsf{C}_=\mathsf{NC}^1] = \mathsf{AC}^0 \cdot [\mathsf{coC}_=\mathsf{NC}^1] = \mathsf{AC}^0{}_3 \cdot [\mathsf{C}_=\mathsf{NC}^1].$$

References

[1] E. Allender. Arithmetic circuits and counting complexity classes. In Jan Krajicek, editor, *Complexity of Computations and Proofs*, Quaderni di Matematica Vol. 13, pages 33–72. Seconda Universita di Napoli, 2004. An earlier version appeared in the Complexity Theory Column, SIGACT News 28, 4 (Dec. 1997) pp. 2-15.

[2] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8(2):99–126, 1999.

[3] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO Theoretical Information and Applications*, 30:1–21, 1996. Conference version in *Proc. 9th IEEE Structure in Complexity Theory Conference* (1994), 267–278.

[4] D. A. Barrington. Bounded-width polynomial size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[5] R. Beigel, N. Reingold, and D. A. Spielman. PP is closed under intersection. *Journal of Computer and System Sciences*, 50(2):191–202, 1995.

[6] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998. Preliminary version in Proceedings of the 11th IEEE Conference on Computational Complexity, 1996, 12–21.

[7] L. Fortnow and N. Reingold. PP is closed under truth-table reductions. *Inf. Comput.*, 124(1):1–6, 1996.

[8] J. Köbler, U. Schöning, and K. W. Wagner. The difference and truth-table hierarchies for NP. *Theoretical Informatics and Applications*, 21(4):419–435, 1987.

[9] K.-J. Lange. Unambiguity of circuits. *Theor. Comput. Sci.*, 107(1):77–94, 1993.

[10] M. Mahajan and B. V. Raghavendra Rao. Small-space analogues of Valiant's classes. In *FCT, LNCS vol. 5699*, pages 250–261, 2009.

[11] M. Ogihara. The PL hierarchy collapses. *SIAM J. Comput.*, 27(5):1430–1437, 1998.

[12] I. Tzamaret. *Studies in Algebraic and Propositional Proof Complexity.* PhD thesis, Tel Aviv University, 2008.

[13] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach.* Springer-Verlag New York Inc., 1999.

[14] J. von zur Gathen and G. Seroussi. Boolean circuits versus arithmetic circuits. *Information and Computation*, 91(1):142–154, 1991.

6 Appendix

Proposition 3.4. $\mathsf{AC}^0(\mathsf{C}_=\mathsf{NC}^1)$ *(respectively* $\mathsf{AC}^0(\mathsf{PNC}^1)$*) equals the class of languages decidable by* $\mathsf{a}\text{-}\mathsf{NC}^1_=$ *(respectively* $\mathsf{a}\text{-}\mathsf{NC}^1_>$*) circuit families such that the nesting depth of test gates is bounded by a constant and the output gate of each circuit is a test gate.*

Proof. We will consider the case $\mathsf{C}_=\mathsf{NC}^1$ only, the proof for PNC^1 is completely analogous.

For the direction from left to right, let $L \in \mathsf{C}_=\mathsf{NC}^1$ and denote by f the function witnessing this fact (that is, $x \in L \iff f(x) = 0$ for all x). Let C be an unambiguous $\mathsf{AC}^0(L)$ circuit with n inputs (without loss of generality we may assume that C is unambiguous. From C, construct an arithmetic circuit C' as follows:

- Replace each $\wedge$-gate with a $\times$-gate.
- Replace each $\neg$-gate with input x with the sub-circuit $1 + (-1 \times x)$.
- Replace each $\vee$-gate with inputs x_1 and x_2 with the sub-circuit $x_1 + ((1 + (-1 \times x_1)) \times x_2)$.
- Replace each oracle gate with inputs $x_1, \ldots, x_m$ with a test gate whose input is the arithmetic circuit that computes $f(x_1 \cdots x_m)$.

It holds that $C'(x) \in \{0, 1\}$ and $C'(x) = C(x)$ for all possible inputs x. The size of C' is clearly polynomial in the size of C. Its depth is $O(\log n)$ owing to the replacement of an oracle gate by a test gate atop an $\mathsf{a}\text{-}\mathsf{NC}^1$ circuit; however, the nesting gate of test gates is at most the depth of C and hence a constant. Thus, the circuit D deciding whether $C'(x) + (-1)$ is equal to zero is the desired $\mathsf{a}\text{-}\mathsf{NC}^1_=$ circuit.

For the converse direction, let C denote an $\mathsf{a}\text{-}\mathsf{NC}^1_=$ circuit with a test gate at the output and $O(1)$ nesting depth of test gates. Without loss of generality, we can assume that the circuit is a formula: every gate has out-degree 1. If we cut all the edges leading out of a test gate, the circuit breaks up into blobs, each of which is an arithmetic circuit with a test gate at the output. In particular, let $g_1, \ldots, g_m$ enumerate all test gates in C, and denote by S_i the maximal connected sub-circuit of C rooted at g_i that consists of $+$, $\times$-gates and the constants -1, 0, 1. Then each S_i computes some function $f_i \colon \{0, 1\}^m \to \{0, 1\}$. By the structure of the circuit, using Proposition 2.5, we see that these functions f_i are all characteristic functions of $\mathsf{C}_=\mathsf{NC}^1$ languages. Replacing S_i with an oracle gate for the corresponding language yields a Boolean circuit C' of polynomial size comprising input gates and oracle gates for $\mathsf{C}_=\mathsf{NC}^1$ only. The depth of this circuit is the nesting depth of test gates in C and hence a constant. This is the required $\mathsf{AC}^0(\mathsf{C}_=\mathsf{NC}^1)$ circuit. □

Remark 6.1. A small technicality in the above proof: in going from right to left, different oracle gates appear to query different languages in $\mathsf{C}_=\mathsf{NC}^1$. However, these can all be replaced by queries to any one language that is complete for $\mathsf{C}_=\mathsf{NC}^1$ under projections.

Proposition 3.5. $\mathsf{NC}^1(\mathsf{C_=NC}^1)$ *(respectively* $\mathsf{NC}^1(\mathsf{PNC}^1)$*) equals the class of languages decidable by* $\mathsf{a\text{-}NC}^1_=$ *(respectively* $\mathsf{a\text{-}NC}^1_>$*) circuit families with the small-blob-chains property in which the output gate of each circuit is a test gate.*

Proof. We will consider the case $\mathsf{C_=NC}^1$ only, the proof for PNC^1 is completely analogous. The constructions are exactly as in the proof of Proposition 3.4, only the analysis is different.

For the direction from left to right, let C' be the circuit obtained from the unambiguous $\mathsf{NC}^1(L)$ circuit C. The size of C' is clearly polynomial in the size of C. As in C any oracle gate of fan-in m accounts for depth $\log m$ and the depth of C bounded by $O(\log n)$, we obtain that the depth C' remains bounded by $O(\log n)$ and also satisfies the small-blob-chain property. Thus, the circuit D deciding whether $C'(x) + (-1)$ is equal to zero is the desired $\mathsf{a\text{-}NC}^1_=$ circuit.

For the converse direction, let C' be the Boolean circuit constructed from C; it is of polynomial size comprising input gates and oracle gates for $\mathsf{C_=NC}^1$ only. From the small-blob-chains property, we finally obtain that for each path π' starting in the root of C' and leading to an input gate or a constant, $\sum_{g\in\pi'} \log(\text{fan-in}(g)) = \sum_{S\in\pi} O(\log|S|) \in O(\log n)$, where π is the corresponding path in C going through blob S instead of oracle gate g. Thus the depth of C' remains bounded by $O(\log n)$. □

Dropping the small-blob-chains property Consider, for example, any circuit family $\{C_n\}_{n\geq 0}$ such that C_n contains n input gates, $\log n$ test gates, and the input to each of these test gates is a binary gate g with inputs i_1 and i_2 such that the sub-circuits computing i_1 and i_2 are disjoint, the sub-circuit computing i_1 contains $\geq n$ gates, and the sub-circuit computing i_2 is rooted at a test gate (see Fig. 1). Then the path starting in the root of circuit C_n and proceeding to the right ancestor of every binary gate crosses all maximal sub-circuits consisting of all but test gates rooted at test gates that consist of +-, ×-gates as well as the constants −1, 0, 1. Thus,

$$\sum_{S_i \text{ occurs in } \pi} \log|S_i| \geq \log n \cdot \log n \notin O(\log n).$$

Theorem 3.9. $\mathsf{NC}^1(\mathsf{PNC}^1) \subseteq \mathsf{L}$.

Proof. Let L be in $\mathsf{NC}^1(\mathsf{PNC}^1)$ and suppose without loss of generality that $L \subseteq \{0,1\}^\star$. Then there exists a language B in PNC^1 and an $\mathsf{NC}^1(B)$ circuit family $\{C_n\}_{n\geq 0}$ such that $C_{|x|}(x) = 1$ if and only if $x \in L$. As apparent from the proof of Proposition 3.5, we may without loss of generality assume that C_n contains oracle gates and input gates only.

Given x, we now simulate the circuit $C_{|x|}$ in a top-down way. At any (oracle) gate for B, we simulate the PNC^1 circuit for B, using the fact that PNC^1 is known to be in L (Proposition 2.4). If this simulation requests an input g, we recursively simulate the sub-circuit of C rooted at g to obtain this bit.

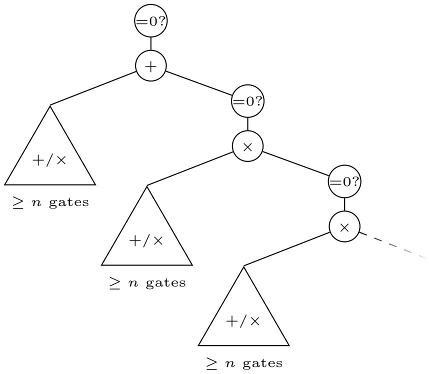

Fig. 1. A circuit violating the small-blob-chains property.

The amount of space required by this approach is $O(\log n)$ for the paths in C plus the space needed for the simulations. As $\mathsf{PNC}^1 \subseteq \mathsf{L}$, each simulation requires space logarithmic in the fan-in of the oracle gate. Let $s(g)$ denote the space needed to simulate the circuit associated with the (oracle) gate g, assuming that its inputs are explicitly available. Then, for each path π starting in the root of C and leading to an input gate or a constant, we have $\sum_{g\in\pi} s(g) = \sum_{g\in\pi} O(\log(\text{fan-in}(g))) \in O(\log n)$. Thus the entire recursive simulation requires $O(\log n)$ space, and so we conclude that $\mathsf{NC}^1(\mathsf{PNC}^1)$ is contained in L. □

Recall from Proposition 3.5 that $\mathsf{NC}^1(\mathsf{PNC}^1)$ can be described in terms of $\mathsf{a}\text{-}\mathsf{NC}^1_=$ circuits with the small-blob-chains property. The proof above can be restated assuming that the equivalent $\mathsf{a}\text{-}\mathsf{NC}^1_=$ circuit family is given, rather than the oracle circuit. We describe this restatement here, primarily to highlight why it does not work for $\mathsf{AH}(\mathsf{PNC}^1)$.

Proof (Alternative proof of Theorem 3.9). Let C be the $\mathsf{a}\text{-}\mathsf{NC}^1_>$circuit with test gates placed so as to satisfy the small-blob-chains property. Let it have n input gates, and let $x \in \{0,1\}^n$. We use the facts that PNC^1 is known to be in L (Proposition 2.4), and that PNC^1 is characterized by $\mathsf{a}\text{-}\mathsf{NC}^1_>$ circuits with a single test gate at the output (Proposition 2.5).

As in the proof of Proposition 3.4, we assume that the circuit is a formula, and cut all the edges leading out of test gates, so that the circuit breaks up into blobs, each of which is an arithmetic circuit with a test gate at the output. If $g_1, \ldots, g_m$ enumerate all test gates in C, and S_i the maximal connected sub-circuit of C rooted at g_i with no test gate other than g_i, then each S_i computes $f_i = \chi_{L_i}$ for some $L_i \in \mathsf{PNC}^1$, and so each f_i is computable in L.

Given x, we evaluate the circuit in a top-down way, starting with the topmost blob and carrying out the logspace evaluation of the associated function. When this evaluation needs a bit that is an input to this blob and is an output of another test gate, we recursively starts a fresh logspace computation of the required function. When this computation terminates, we resume the earlier computation.

At any stage, the computation traces out a path, or a chain, through the blobs in C, with the current computation focussing on the leaf of the chain. The space requirement per blob is logarithmic in the size of the blob. By the small-blob-chains propery, we conclude that the overall space requirement is $O(\log n)$. □

Theorem 3.10. $\mathsf{AH}(\mathsf{PNC}^1) \subseteq \mathsf{TC}^1$.

Proof. We perform a straightforward bottom-up Boolean evaluation of the arithmetic circuit; that is, we evaluate bit representations of the values carried on the wires. Since C is a formula, all intermediate values have polynomial-sized bit representations, so the $+$ and $\times$ gates can be replaced by TC^0 sub-circuits. The test gates can also be trivially replaced by appropriate TC^0 sub-circuits. This yields a log depth circuit with majority gates, that is, a TC^1 circuit. □

Lemma 4.3. *Let f be a function in GapNC^1. Then there exists a branching program Q of width $O(1)$ such that*

1. *Q has a single start node s and two terminal nodes t_1 and t_2;*
2. *every path originating from s, ends at either t_1 or t_2 and nowhere else;*
3. *any path of Q on any given input x contains exactly q nondeterministic nodes;*
4. *every edge is labelled by a literal y_i or $\neg y_i$;*
5. *on any input y, Q has exactly 2^q paths, originating from s, where $q = q(n) \leq \mathsf{poly}(n)$;*
6. *$f = \#[s \rightsquigarrow t_1] - \#[s \rightsquigarrow t_2]$.*

Proof. From Proposition 2.2, there is a BP P of width $w = O(1)$ and size $s = \mathsf{poly}(n)$, with nodes s, t_1 and t_2 such that $f = \#[s \rightsquigarrow t_1] - \#[s \rightsquigarrow t_2]$. The edge labels in P are literals or the constant 1. We modify P so that

- Every node has out-degree 0,1,or 2.
- For each layer k, there is an index i_k such that, edges from layer k to $k+1$ are labelled from the set $\{1, y_{i_k}, \neg y_{i_k}\}$.

This can be achieved by doing necessary staggering of the program P. Copy all the nodes of a layer into new nodes, and then implement the edges according to their labels, taking one variable at a time. Repeat this process for all the layers. This ensures that every edge in a particular layer is labelled by a single variable, its negation or a constant. This will double the width and increase the size by a factor of wn. In a similar way, we can ensure that the out-degree of every node is 0,1 or 2. The resulting BP P' will have width bounded by $O(4w)$ and size $O(w^2n^2 \cdot s)$, where $s = size(P)$.

We create a new line (a path) called the "zero-gap" line starting from s. This line remains a single path until the last layer and forks out to both t_1 and t_2, with all the edges being labelled by 1. Note that this line produces a zero-gap. To meet condition 2 in the lemma, for every node v, if v has an out-edge labelled y_i (respectively $\neg y_i$) and no out-edge labelled $\neg y_i$ (respectively y_i), then add an edge labelled $\neg y_i$ (respectively y_i) to the zero-gap line. Note that this process ensures condition 2 without changing the gap-function.

Recall that a node in P' is nondeterministic if it has two out-going edges labelled 1 on at least one input assignment to y (for example, a node with two outgoing edges labelled y_i and 1 respectively). A layer is called nondeterministic if at least one of the nodes in that layer is nondeterministic. In order to ensure that for every input y, the total number of paths originating from s remains the same, we make all the nodes in every nondeterministic layer nondeterministic, by adding necessary paths to the zero-line as follows: Consider a deterministic node v in a nondeterministic layer. Let y_i be the allowed variable label for this layer. There are two possibilities: v has two out-edges, one labelled by y_i and the other labelled by $\neg y_i$ or v has a single out-edge labelled by 1. In both the cases, we add an edge to the zero-gap line, with label 1. In the case when v is a node already in the zero-gap line, we just add a parallel edge. Again, this construction does not alter the gap function.

Finally we eliminate the constant 1 on edge labels: replace an edge labelled 1 by parallel edges, one labelled y_i and the other labelled $\neg y_i$, where y_i is the variable at this layer.

Let Q denote the resulting BP. Let q be the number of nondeterministic layers of Q. Since we have two choices at every nondeterministic layer and only one choice at deterministic layers, on any input y, the number of distinct paths originating from s is exactly 2^q. As we have not changed the gap values in the whole process, this proves the lemma. □

Lemma 4.6. $h(x) \in \mathsf{GapBWBP}$

Proof. Since $\mathsf{GapBWBP}$ is closed under taking polynomially bounded sums and products, it follows easily that $\alpha(x, i, w_i)$ and $\beta(x, i)$ are in $\mathsf{GapBWBP}$. We now show that $a(x) \in \mathsf{GapBWBP}$.

First we show that $e(x, W, U)$ can be computed by a constant width branching program which is read-once certified in the variables W and U. Let r be the width of Q; note that $r = O(1)$. We build a Boolean circuit C which is “read-once certified” in W and U (that is, there exists a partition of C into sub-ciruits $C_1, \ldots, C_{|U|+|W|}$ such that for all inputs from $U \cup W$ there exists an index i such that all wires from the input gate lead to the sub-circuit C_i), has width $O(r)$, and computes $e(x, W, U)$. We proceed layer by layer. At the ith stage, C computes the index of the node v at layer i which is a part of the unique path in Q represented by x, W and U. Note that this index requires $O(\log r)$ bits and hence C has those many gates as the output of this stage. Now the $(i+1)$th stage computes the index of the node v' at layer $i+1$, which is uniquely defined given the index of v and bits w_i and u_i. Given the bit representation of the index of v, the index of v' depends only on w_i and u_i hence can be computed by a

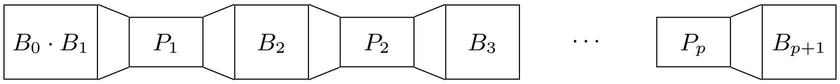

Fig. 2. Composing B_is and P_is

circuitry of size (hence width) $O(r)$. After the final stage, the circuit outputs 1 if and only if the index corresponds to the node t_1 at the last layer. Clearly C computes $e(x, W, U)$ and is of size $O(r \cdot \mathsf{size}(Q))$ and width $O(r)$. Also from the description of C, it is clear that C is read-once certified in W as well as U.

Using a standard subset construction (see [4], Section 5), we obtain a BP B computing $e(x, W, U)$ so that:

- $\mathsf{width}(B) = 2^{O(r)}$ and $\mathsf{size}(B) = 2^{O(r)}\mathsf{size}(C)$; and
- B is read-once certified in both W and U.

Let B_0 be the initial part of B that depends on none of the variables from W an U. Let B_i be the part that depends only on x, w_i, and u_i. Now we construct a BP that gap-represents the product $e(x, W, U)\tilde{\alpha}(x, W)$ and is read-once certified in the variables W and U. Let P_i be a constant width BP gap representing $\alpha(x, i, W_i)$. We position the programs B_is and P_is as shown in the Figure 2 to obtain the BP B'. Clearly B' gap-represents $e(x, W, U)\tilde{\alpha}(x, W)$ and is read-once certified in Ws and Us. The size of B' is bounded by $\mathsf{size}(B) + \sum \mathsf{size}(B_i)$, and its width is $\max\{\mathsf{width}(B), \mathsf{width}(P_i)\}$; these are bounded by $2^{O}(r)\mathsf{size}(Q)\mathsf{poly}(p)$ and $2^{O(r)}$ respectively.

Notice that $a(x)$ is exactly the read-once exponential sum of $e(x, W, U)\tilde{\alpha}(x, W)$ over the variables from W and U. Thus, applying Proposition 2.6 to B', we get a BP B'' of size $\mathsf{poly}(\mathsf{size}(B'))$ and width $(\mathsf{width}(B'))^4$ such that B'' gap-represents $a(x)$. The width blows up by a power of 4 (since we need to do the read-once exponential sum closure twice, once for the variables from W and then for those from U), but remains $O(1)$. Hence we obtain a $\mathsf{GapBWBP}$ for $a(x)$. □

Rational Approximations for the Sign: Ogihara's Polynomials

In the following, we define the functions that will be needed in the construction. We follow the same notations from [11].

Definition 6.2. *[11]*

$$\begin{aligned}
\mathcal{P}_m(z) &:= (z-1)\prod_{i=1}^{m}(z-2^i)^2, \\
\mathcal{Q}_m(z) &:= -(\mathcal{P}_m(z)+\mathcal{P}_m(-z))), \\
\mathcal{R}_{m,k}(z) &:= \left(\frac{2\mathcal{P}_m(z)}{\mathcal{Q}_m(z)}\right)^{2k}, \\
\mathcal{S}_{m,k}(z) &:= \left(1+\mathcal{R}_{m,k}(z)\right)^{-1}, \\
\mathcal{A}_{m,k}(z) &:= \mathcal{Q}(z)^{2k}, \text{ and} \\
\mathcal{B}_{m,k}(z) &:= \mathcal{Q}_m(z)^{2k}+(2\mathcal{P}_m(z))^{2k}.
\end{aligned}$$

The following properties of $\mathcal{S}_{m,k}$ are proved in [11]:

Proposition 6.3. *[11] For $m,k \geq 1$, and for every z, the following holds:*

1. $\mathcal{S}_{m,k}(z) = \frac{\mathcal{A}_{m,k}(z)}{\mathcal{B}_{m,k}(z)}$,
2. $1 \leq z \leq 2^m \Rightarrow 1-2^{-k} \leq \mathcal{S}_{m,k} \leq 1$, *and*
3. $-2^m \leq z \leq -1 \Rightarrow 0 \leq \mathcal{S}_{m,k}(z) \leq 2^{-k}$.

Let f be a function from strings to non-zero integers, and let $\mu = \mu(|x|)$ be such that for all strings x, $|f(x)| \leq 2^{\mu(|x|)}$. Let B' be a function mapping a string x to a sequence $Y_1,\ldots,Y_p$ of p Boolean strings for some p, and let $\kappa = 2p+1$. For $i \in [1,p]$, define the following functions:

$$\begin{aligned}
S(x,i,1) &:= \mathcal{S}_{\mu,\kappa}(f(Y_i)), \\
S(x,i,0) &:= 1-\mathcal{S}_{\mu,\kappa}(f(Y_i)) \\
\alpha(x,i,1) &:= \mathcal{A}_{\mu,\kappa}(f(Y_i)), \\
\alpha(x,i,0) &:= \mathcal{B}_{\mu,\kappa}(f(Y_i))-\mathcal{A}_{\mu,\kappa}(f(Y_i)), \text{ and} \\
\beta(x,i) &:= \mathcal{B}_{\mu,\kappa}(f(Y_i)).
\end{aligned}$$

For $w \in \{0,1\}^p$, define

$$\begin{aligned}
\tilde{S}(x,w) &:= \prod_{i=1}^{p} S(x,i,w_i), \\
\tilde{\alpha}(x,w) &:= \prod_{i=1}^{p} \alpha(x,i,w_i), \text{ and} \\
\tilde{\beta}(x) &:= \prod_{i=1}^{p} \beta(x,i).
\end{aligned}$$

Let H be the language defined as $\{y \mid f(y) > 0\}$. Then

Lemma 6.4. *[11]*

1. *If $w_i = \chi_H(Y_i)$, then $1-p2^{-\kappa} \leq \tilde{S}(x,w) \leq 1$ and $0 \leq \tilde{S}(x,w) \leq 2^{-\kappa}$.*

2. $\tilde{S}(x,w) = \frac{\tilde{\alpha}(x,w)}{\tilde{\beta}(x,w)}$.

Proposition 5.2. *The following two containments hold:*

$$\begin{aligned} \mathsf{coC_{=}NC^1} &\subseteq \mathsf{FSLE}^1[\mathsf{GapNC}^1], \\ \mathsf{C_{=}NC^1} &\subseteq \mathsf{FSLE}^2[\mathsf{GapNC}^1]. \end{aligned}$$

Proof. Given $g = g(x_1, \dots, x_n) \in \mathsf{GapNC}^1$, and a word $a \in \{0,1\}^n$, consider the system of equations $Az = b$, where

1. $A := (g(a)), b := (1)$, and z consists of a single variable z_1. Clearly, the system is feasible if and only if $g(a) \neq 0$.
2. $A := \begin{pmatrix} g(a) & 0 \\ 1 & 0 \end{pmatrix}, b := \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, and z consists of two variables. Clearly, the system is feasible if and only if $g(a) = 0$.

□

Lemma 5.3. $\mathsf{NC^0}_d \cdot \mathsf{C_{=}NC^1} \subseteq \mathsf{FSLE}^{3(2^{d+1}-1)}[\mathsf{GapNC}^1]$.

Proof. We use the following constructions:

Claim. If $(A_1, b_1), (A_2, b_2)$ are $\mathsf{FSLE}^{k_1}[\mathsf{GapNC}^1]$- and $\mathsf{FSLE}^{k_2}[\mathsf{GapNC}^1]$-instances respectively, then we can construct an instance of $\mathsf{FSLE}^{k_1+k_2}[\mathsf{GapNC}^1]$-instance (A, b) which is feasible if and only if both $(A_1, b_1), (A_2, b_2)$ are feasible.

Proof. Let

$$A := \begin{pmatrix} A_1 & O_1 \\ O_2 & A_2 \end{pmatrix}, \quad b := \begin{pmatrix} b_1 \\ b_2 \end{pmatrix},$$

where O_1, O_2 are all zeroes matrices of appropriate dimensions ($k_1 \times k_2$ and $k_2 \times k_1$ respectively). □

Claim. $\mathsf{FSLE}^k[\mathsf{GapNC}^1] \subseteq$ co-$\mathsf{FSLE}^{k+1}[\mathsf{GapNC}^1]$

Proof. We need to show that given an $\mathsf{FSLE}^k[\mathsf{GapNC}^1]$-instance (A, b), we can construct an $\mathsf{FSLE}^{k+1}[\mathsf{GapNC}^1]$-instance (A', b') such that (A, b) is feasible if and only if (A', b') is infeasible. We essentially mimic the construction in [2] to achieve this. We set

$$A' := \begin{pmatrix} A^T & O \\ b^T & 0 \end{pmatrix}, \quad b' := \begin{pmatrix} O \\ 1 \end{pmatrix},$$

where O is a $k \times 1$ column vector of all zeroes. For a proof of correctness see [2]. □

Assume that the NC^0 circuit consists of $\wedge$- and $\vee$-gates, with $\neg$-gates ocurring only at the base level. This is to fix the notion of depth for such circuits.

We proceed by induction on the depth d of the NC^0circuit. For the purposes of the construction, we eliminate all internal $\vee$ gates, replacing them by equivalent sub-circuits using $\wedge$ and $\neg$.

The base case $d = 0$ is considered in Proposition 5.2—notice that we do not include negation gates in our depth computation.

Depending on what the top gate is, we use one or both of the preceding claims. Notice that the dimension of the $\mathsf{FSLE}^{bdd}[\mathsf{GapNC}^1]$-instance constructed at most doubles at an $\wedge$-gate. To simulate an $\vee$-gate, we need a negation, an $\wedge$, and then one more negation, so the the dimension of the $\mathsf{FSLE}^{bdd}[\mathsf{GapNC}^1]$-instance goes from k to $2k + 3$.

Hence, we conclude that a depth-d NC^0 circuit over $\mathsf{C_{=}NC}^1$ or $\mathsf{coC_{=}NC}^1$ translates to an $\mathsf{FSLE}^{bdd}[\mathsf{GapNC}^1]$-instance of dimension at most $2\times 3(2^d - 1) + 3 = 3(2^{d+1} - 1)$. □

Lemma 5.4. $\mathsf{FSLE}^k[\mathsf{GapNC}^1] \subseteq \mathsf{NC}^0{}_{3k} \cdot \mathsf{C_{=}NC}^1$.

Proof. Let us fix some notation first. Given a $k \times k$ square matrix A, for $S, T \subseteq [k]$, let $A_{S,T}$ denote the square submatrix of A with exactly the rows in S and columns in T. Also denote by b_S the column vector of b with only the entries indexed in S. Also denote by $[A|_i b]$ the square matrix formed by replacing the i-th column of A by b. Thus $[A_{S,T}|_t b_S]$ for some $t \in T$ is the matrix formed by replacing the t-th column of $A_{S,T}$ by b_S.

Let $Az = b$ describe the system of linear equations on an input word, and let $r = \mathsf{rank}(A)$. We observe the following.

Proposition 6.5. *The following are equivalent.*

1. *For every $S, T \subseteq [k]$ such that $|S| = |T| = r$ and $rank(A_{S,T}) = r$, and for every $j \notin S$, $\det([A_{S',T} b_{S'}]) = 0$, where $S' = S \cup \{j\}$.*
2. *For some $S, T \subseteq [k]$ such that $|S| = |T| = r$ and $rank(A_{S,T}) = r$, and for every $j \notin S$, $\det([A_{S',T} b_{S'}]) = 0$, where $S' = S \cup \{j\}$.*
3. *The system $Az = b$ is feasible.*

Proof. $1 \Rightarrow 2$: Obvious.

$2 \Rightarrow 3$: Let S, T be the subsets with the given property. Since $\mathsf{rank}(A_{S,T}) = r$, $A_{S,T}$ is full-rank, and so the sub-system $A_{S,T} z_T = b_S$ has a unique solution z_T. Now extend this to a solution by setting $z = 0$ outside T. Clearly, this satisfies the equations indexed by S. By the condition $\det([A_{S',T} b_{S'}]) = 0$ for $j \notin S$ and $S' = S \cup \{j\}$, it also satisfies every equation $j \notin S$. So it is a feasible solution.

$3 \Rightarrow 1$: Let z be a feasible solution, and let S, T be any pair of subsets of $[k]$ of size r such that $A_{S,T}$ is non-singular. If z is 0 outside T, then $A_{S,T} z_T = b_S$ and for each $j \notin S$, $A_{\{j\},T} z_T = b_j$. But $A_{\{j\},T}$ is expressible uniquely as a linear combination of the rows of $A_{S,T}$. It follows that b_j is the same combination of b_S, and so $\det([A_{S',T} b_{S'}]) = 0$.

But we can assume without loss of generality that z is 0 outside T. This is because the columns indexed by T span all the columns of A. So if for $j \notin T$, $z_j \neq 0$, then we can set it to 0 and adjust the values of z in the T part to account for the contribution of the jth column. □

So now, to check if the system $Az = b$ is feasible, we check condition (2) of Proposition 6.5. We claim that this can be checked in $\mathsf{NC^0}_{k+1} \cdot \mathsf{C_=NC^1}$. To see this, note that A is a matrix of $O(1)$ size. So determinants of all its sub-matrices can be computed by $O(1)$-size arithmetic formulas taking the $\mathsf{GapNC^1}$ function values as inputs. In terms of the input word, these are themselves $\mathsf{GapNC^1}$ functions. Thus, in particular, using the closure properties of $\mathsf{C_=NC^1}$ (Proposition 2.4), computing the rank of A or a submatrix of A is in $\mathsf{C_=NC^1}$.

Now, the condition (2) is a disjunction over 2^{2k} conditions, one for each $S, T \subseteq [k]$. Each condition is of the form

$$[|S| = |T| = \mathsf{rank}(A) = \mathsf{rank}(A_{S,T})] \Rightarrow \bigwedge_{j \notin S; S' = S \cup \{j\}} [\det([A_{S',T} b_{S'}]) = 0]$$

The whole condition can thus be expressed as an $\mathsf{NC^0}$ circuit of depth at most $2k + O(\log k) \leq 3k$. □

Proposition 5.6.

$$[\mathsf{C_=NC^1}] \cdot \mathsf{AC^0} = \mathsf{C_=NC^1}$$
$$[\mathsf{coC_=NC^1}] \cdot \mathsf{AC^0} = \mathsf{coC_=NC^1}$$

Proof. The inlcusion from right to left is obvious. We prove the left-to-right inclusion for $\mathsf{C_=NC^1}$; the proof for $\mathsf{coC_=NC^1}$ is identical. So let A be a language in $[\mathsf{C_=NC^1}] \cdot \mathsf{AC^0}$. At each length n, there is an $\mathsf{AC^0}$ oracle circuit C with a single oracle gate g at the top. Let r be the number of input wires to g. On input $x = x_1x_2\cdots x_n$, let these wires carry the values $z_i(x)$ for $i = 1, \ldots, r$. By Proposition 2.3, there are arithmetic $\mathsf{a\text{-}AC^0}$ circuits and hence $\mathsf{a\text{-}NC^1}$ circuits Z_i such that for all x, $Z_i(x) = z_i(x)$. By Proposition 2.5, there is an $\mathsf{a\text{-}NC^1_=}$ circuit D_g with a single test gate at the output such that $D_g(z_1(x), \ldots, z_r(x))$ is the bit computed by the oracle gate g. Replacing the inputs $z_i(x)$ in D_g by the circuits Z_i gives an $\mathsf{a\text{-}NC^1_=}$ circuit D'_g computing the same function as C. Since D'_g has a single test gate at the top, by Proposition 2.5, it computes the characteristic funciton of a language in $\mathsf{C_=NC^1}$. □

Lemma 5.8. *Let $h : \{0,1\}^t \longrightarrow \{0,1\}$, $f_1, f_2, \ldots, f_t : \{0,1\}^n \longrightarrow \{0,1\}$ be functions in $\mathsf{GapNC^1}$, where $\forall w : f_i(w) \geq 0$. Then for some $T \in t^{O(1)}$, there exist $\mathsf{GapNC^1}$ functions $g_1, g_2, \ldots, g_T : \{0,1\}^n \longrightarrow \{0,1\}$ and an $\mathsf{AC^0}$ circuit H on T inputs such that, for all $w \in \{0,1\}^n$,*

$$h(b_1, b_2, \ldots, b_t) \neq 0 \Longleftrightarrow H(d_1, d_2, \ldots, d_T) = 1$$

where $b_i := \begin{cases} 1 & \text{if } f_i(w) \neq 0, \\ 0 & \text{otherwise,} \end{cases}$ *and* $d_j := \begin{cases} 1 & \text{if } g_j(w) \neq 0, \\ 0 & \text{otherwise.} \end{cases}$

Proof. Let C be an $\mathsf{a\text{-}NC}^1$ circuit computing h. Without loss of generality, assume that C is a formula (all gates have out-degree 1), that is is layered with alternating $+$- and $\times$-layers, and that the underlying graph is a complete binary tree where every root-to-leaf path is of length exactly $2d$.

Without loss of generality, assume that each f_i is non-negative everywhere. (If this not the case, use the function f_i^2 instead of f_i; this will not change the zero-test and hence will not change b_i. And f_i^2 is also in GapNC^1.)

Let $\mathcal{F}$ denote the set of functions $\{f_1, \ldots, f_t\}$. For each $i \in [t]$, let $\mathcal{F}_i$ denote the set of functions $\mathcal{F} \setminus \{f_i\}$. Also, for each $w \in \{0,1\}^n$, let $\mathcal{F}(w)$ and $\mathcal{F}_i(w)$ denote the set (or possibly multiset) of values taken by the functions in $\mathcal{F}$ and $\mathcal{F}_i$ on the input w.

Over any set Y of l variables, the symmetric polynomials S_l^k for $0 \leq k \leq l$ are defined as follows.

$$S_l^k(Y) := \sum_{S \subseteq Y; |S|=k} \prod_{y \in S} y$$

For each $i, k \in [t]$, define the function

$$F_i^k(w) := f_i(w) S_{k-1}^{t-1}(\mathcal{F}_i(w)).$$

Let C_k be the circuit obtained from C by replacing the constant 1 with the value $S_k^t(\mathcal{F})$, and replacing the constant -1 with the value $-S_k^t(\mathcal{F})$. Now define the predicate Π_k as follows:

$$\Pi_k(w) \equiv \left[S_t^k(\mathcal{F}(w)) \neq 0 \ \wedge \ S_t^{k+1}(\mathcal{F}(w)) = 0 \ \wedge \ C_k(F_1^k(w), \ldots, F_t^k(w)) \neq 0 \right]$$

Claim. For all $w \in \{0,1\}^n$,

$$h(b_1, b_2, \ldots, b_t) \neq 0 \Longleftrightarrow \exists k \in \{0, 1, \ldots, t\} : \Pi_k(w).$$

Proof. Fix $w \in \{0,1\}^n$ and let $k_w = r$ denote the number of functions f_i that evaluate to a non-zero value at w. By the properties of symmetric polynomials, and the fact that each $f_i(w)$ is non-negative, we can see that $S_t^k(\mathcal{F}(w)) \neq 0$ is false exactly when $k > r$, and $S_t^{k+1}(\mathcal{F}(w)) = 0$ is false exactly when $k < r$. So Π_k is false whenever $k \neq r$, and

$$\exists k \in \{0, 1, \ldots, t\} : \Pi_k(w) \equiv \Pi_r(w) \equiv C_r(F_1^r(w), \ldots, F_t^r(w)) \neq 0.$$

But at $k = r$, for each $i \in [t]$,

$$F_i^k(w) = f_i(w) S_{k-1}^{t-1}(\mathcal{F}_i(w)) = \begin{cases} \prod_{f_j(w) \neq 0} f_j(w) & \text{if } f_i(w) \neq 0, \\ 0 & \text{if } f_i(w) = 0. \end{cases}$$

Let $G(w)$ denote $\prod_{f_j(w) \neq 0} f_j(w)$; if no $f_j(w)$ is non-zero, then $G(w) := 1$. Thus $G(w) \neq 0$. Then $F_i^r(w) = b_i G(w)$ for each $i \in [t]$. Since C was in normal layered form as a complete binary tree with $\times$-depth d, and since in C_r we replace each constant ± 1 by $\pm G(w)$, and since we evaluate C_r replacing each each bit b_i by the value $b_i G(w)$, we see that $C_r(F_1^r(w), \ldots, F_t^r(w)) = [G(w)]^{2^d} C(b_1, \ldots, b_t)$. Thus $C_r(F_1^r(w), \ldots, F_t^r(w)) \neq 0 \Leftrightarrow C(b_1, \ldots, b_t) \neq 0$, proving the claim. $\square$

Claim. For each $k \in \{0, 1, \ldots, t\}$, there exist GapNC^1 functions N_t^k and D_t^k such that $S_t^k(\mathcal{F}(w)) = N_t^k(w)/D_t^k(w)$.

Proof. The symmetric polynomials can be computed efficiently over fields. Since we are dealing with integers, we need to carry the numerators and denominators separately. The functions N_t^k and D_t^k do just this.

For completeness, we describe the functions explicitly, following notation from [12]. Consider the symmetric polynomial $S_l^k(Y)$ over some set Y of l variables. Then $\prod_{i=1}^{l}(y_i + z) = \sum_{k=0}^{l} S_l^k(Y)z^k$. Given values to Y, this polynomial in z can be computed by interpolation through values at any $l+1$ points. So fix any $l+1$ distinct constants, without loss of generality they can be $0, 1, \ldots, l$. Let B be the $(l+1) \times (l+1)$ Vandermonde matrix where for $0 \le i, j \le l$, $B_{ij} := i^j$. Then

$$\begin{bmatrix} \\ B \\ \\ \end{bmatrix} \cdot \begin{bmatrix} S_l^0(Y) \\ S_l^1(Y) \\ \vdots \\ S_l^l(Y) \end{bmatrix} = \begin{bmatrix} \prod_{i=1}^{l} y_i \\ \prod_{i=1}^{l}(y_i + 1) \\ \vdots \\ \prod_{i=1}^{l}(y_i + l) \end{bmatrix}.$$

Since B can be precomputed, and since the vector on the right-hand side is easy to compute for any given values to the variables in Y, the vector of symmetric polynomials can be obtained. In particular,

$$S_l^k(Y) = \sum_{j=0}^{l} B_{kj}^{-1} \prod_{i=1}^{l}(y_i + j)$$

Each entry of B^{-1} can be written as a rational function with $\det(B) = \prod_{0 \le i < j \le l}(j - i)$ in the denominator. So, as long as the variables in Y take integer values, the function $\det(B)S_l^k(Y)$ is integral.

Thus to compute $S_t^k(\mathcal{F}(w))$, set $l = t$ in the argument above. Then $D_t^k(w) = \det(B)$, and $N_t^k(w) = \sum_{j=0}^{l} \det(B)B_{kj}^{-1} \prod_{i=1}^{l}(f_i(w) + j)$. The terms $\det(B)$ and $\det(B)B_{kj}^{-1}$ are constants that can be precomputed and can be expressed via $\mathsf{a}\text{-}\mathsf{NC}^1$ circuits. Since each f_i is in GapNC^1, the computation of N_t^k is also in GapNC^1. □

Claim. For each k, there is a GapNC^1 circuit G_k such that $G_k(w) = 0$ if and only if $C_k(F_1^k(w), \ldots, F_t^k(w)) = 0$.

Proof. We do local surgery on the circuit C_k to carry at every gate g a rational value as a pair of values corresponding to the numerator N_g and the denominator D_g . At a leaf labeled by a constant c, set $N_g = c$, $D_g = 1$. At a leaf labeled $F_1^k(w)$, set N_g and D_g as the output gates of the corresponding circuits described in the previous claims. The gate $g = g_1 + g_2$ has the obvious implementation: $D_g = D_{g_1} \times D_{g_2}$, and $N_g = N_{g_2} \times D_{g_1} + N_{g_1} \times D_{g_2}$. Similarly at gate $g = g_1 \times g_2$, set $D_g = D_{g_1} \times D_{g_2}$, and $N_g = N_{g_1} \times N_{g_2}$. In the resulting circuit, label the wire carrying N_g for the output gate of C_k as the output gate. □

Now we can complete the proof of the lemma. The required GapNC^1 functions are those computed by the circuits N_t^k and G_k for each k. The AC^0 circuit H implements the check

$$\bigvee_{k=0}^{t} \left[N_t^k(\mathcal{F}(w)) \neq 0 \ \wedge\ N_t^{k+1}(\mathcal{F}(w)) = 0 \ \wedge\ G_k(w) \neq 0\right].$$

□

Theorem 5.9. *The AC^0 hierarchy over $\mathsf{C}_=\mathsf{NC}^1$ collapses to its first level, requiring a single layer of oracle gates and a depth-3 circuit above it,*

$$\mathsf{AC}^0(\mathsf{C}_=\mathsf{NC}^1) = \mathsf{AC}^0 \cdot [\mathsf{C}_=\mathsf{NC}^1] = \mathsf{AC}^0 \cdot [\mathsf{coC}_=\mathsf{NC}^1] = \mathsf{AC}^0{}_3 \cdot [\mathsf{C}_=\mathsf{NC}^1].$$

Proof. Let A be a language in $\mathsf{AC}^0(\mathsf{C}_=\mathsf{NC}^1)$; since the AC^0 circuitry is allowed to use negation gates, equivalently A is in $\mathsf{AC}^0(\mathsf{coC}_=\mathsf{NC}^1)$. Then there is a constant k such that $A \in \mathsf{AC}^0{}_k(\mathsf{coC}_=\mathsf{NC}^1)$. The circuit for A thus has the form

$$\mathsf{AC}^0 \cdot [\mathsf{coC}_=\mathsf{NC}^1] \cdot \mathsf{AC}^0 \cdot \ldots \cdot [\mathsf{coC}_=\mathsf{NC}^1] \cdot \mathsf{AC}^0.$$

Using Proposition 5.6, we can absorb all except the topmost AC^0 circuitry into the oracle gates, giving a circuit of the form

$$\mathsf{AC}^0 \cdot \underbrace{[\mathsf{coC}_=\mathsf{NC}^1] \cdot [\mathsf{coC}_=\mathsf{NC}^1] \cdot \cdots \cdot [\mathsf{coC}_=\mathsf{NC}^1]}_{(k \text{ times})}.$$

Using Lemma 5.7, we replace the bottom two oracle layers by a sub-circuit of the form $\mathsf{AC}^0 \cdot \mathsf{coC}_=\mathsf{NC}^1$. Then using Proposition 5.6 again, we abosrb this new AC^0 circuitry into the oracle gate layer above it to get a circuit of the form

$$\mathsf{AC}^0 \cdot \underbrace{[\mathsf{coC}_=\mathsf{NC}^1] \cdot [\mathsf{coC}_=\mathsf{NC}^1] \cdot \cdots \cdot [\mathsf{coC}_=\mathsf{NC}^1]}_{(k-1 \text{ times})}.$$

Repeating this process another $k-2$ times gives the desired circuit of the form $\mathsf{AC}^0 \cdot [\mathsf{coC}_=\mathsf{NC}^1]$. Since $\mathsf{coC}_=\mathsf{NC}^1$ contains AC^0, it can be written in the form $[\mathsf{coC}_=\mathsf{NC}^1] \cdot [\mathsf{coC}_=\mathsf{NC}^1]$. Now using Lemma 5.7 again, we can replace the top oracle gate by a depth-3 AC^0 circuit. □