Theoretical Computer Science 610 (2016) 24–36

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Positive and negative proofs for circuits and branching programs

Olga Dorzweiler, Thomas Flamm, Andreas Krebs *, Michael Ludwig *

WSI – University of Tübingen, Sand 13, 72076 Tübingen, Germany

A R T I C L E I N F O

Article history:
Received 7 November 2014
Received in revised form 22 August 2015
Accepted 31 August 2015
Available online 3 September 2015

Keywords:
Circuits
Branching programs
Counting
Arithmetization

A B S T R A C T

We extend the # operator in a natural way and derive new types of counting complexity classes. While in the case of $\#\mathcal{C}$ classes (where $\mathcal{C}$ could be some circuit-based class like $\mathbf{NC}^1$) only proofs for acceptance of some input are being counted, one can also count proofs for rejection. The ZAP-$\mathcal{C}$ complexity classes we propose here implement this idea.
We show that in certain cases ZAP-$\mathcal{C}$ lies between $\#\mathcal{C}$ and GAP-$\mathcal{C}$ which could help understanding the relationship between $\#\mathcal{C}$ and GAP-$\mathcal{C}$. In particular we consider ZAP-$\mathbf{NC}^1$ and polynomial size branching programs of bounded and unbounded width. Finally we argue about negative proofs in Turing machines and how those relate to open questions.

1. Introduction

The setting Besides Turing machines, circuits are a well studied model of computation for capturing low level complexity classes. Measures of complexity in circuits include depth and the number of gates, which are roughly speaking an analogue to time and space complexity in Turing machines. When regarding parallels between Turing machines and circuits, a natural question is, what the counterpart to nondeterminism in circuits is. A nondeterministic Turing machine can have more than one accepting computation on some input. In fact, the counterpart to the presence of multiple accepting computations is the presence of multiple proof trees in circuits. A proof tree is a sub-tree of the tree unfolding of a circuit, which is a witness for acceptance of some input word. When looking at the circuit-based characterization of, say **NP**, one can observe that the number of accepting computations and the number of proof trees coincide [16].

A natural question is, how to compute the number of proof trees in a circuit. It can be verified easily that if we move to an arithmetic interpretation of the circuit, it computes precisely the number of proof trees [18]. I.e. we interpret AND as multiplication and OR as addition. This does not work with negation gates, so we assume w.l.o.g. the circuit to be monotone. If we want to address functions counting proof trees in circuits (or equivalently arithmetic circuits), say $\mathbf{NC}^1$ circuits, we write $\#\mathbf{NC}^1$.

In general, it is an open question whether $\#\mathcal{C}$ functions are closed under subtraction (here, the case we are most interested in, is $\mathcal{C} = \mathbf{NC}^1$). This motivated another type of counting complexity approach resulting in GAP-$\mathcal{C}$ classes. Where $\#\mathcal{C}$ functions range over nonnegative integers, GAP-$\mathcal{C}$ functions range over integers. The GAP-$\mathcal{C}$ functions are realized by arithmetic circuits with addition, subtraction, and multiplication gates. By [7,1] we know that GAP-$\mathcal{C}$ functions only require one subtraction gate, which is also the output gate. So GAP-$\mathcal{C}$ functions can be presented as a difference of two $\#\mathcal{C}$ functions,

* Corresponding authors.
E-mail addresses: dorzweiler@informatik.uni-tuebingen.de (O. Dorzweiler), flamm@informatik.uni-tuebingen.de (T. Flamm), krebs@informatik.uni-tuebingen.de (A. Krebs), ludwigm@informatik.uni-tuebingen.de (M. Ludwig).

http://dx.doi.org/10.1016/j.tcs.2015.08.041
0304-3975/© 2015 Elsevier B.V. All rights reserved.

which motivated the naming of GAP-$\mathcal{C}$. That means that each GAP-$\mathcal{C}$ function can be computed by a family of arithmetic circuits only having a single subtraction gate, which is then the output gate.

Boolean circuits can also compute arithmetic functions. Such a circuit has as many output gates as necessary to display the resulting integer in binary representation. We call this the functional version, and for example we get $\mathbf{FNC}^1$ as the functional version of $\mathbf{NC}^1$. Hence, one can ask $\mathbf{FNC}^1 \overset{?}{=} \#\mathbf{NC}^1$ or even $\mathbf{FNC}^1 \overset{?}{=}$ GAP-$\mathbf{NC}^1$. By Jung [11] we know that those classes lie extremely close, but it is still unknown if they coincide.

We have circuits as one major part of this work. But we also consider branching programs (BP). Starting point is the celebrated work of Barrington, who showed that bounded width polynomial size branching programs (BWBP) are equally powerful as $\mathbf{NC}^1$ circuits. In the case of BPs we are also interested in counting. In BPs a witness for acceptance is a path from source to target. By [5], we have that the problem of counting paths can be expressed as matrix multiplication, which is computable in $\#\mathbf{NC}^1$. However we do not know if counting proof trees in $\mathbf{NC}^1$ circuits can be performed in $\#\mathbf{BWBP}$ hence, $\#\mathbf{NC}^1 \overset{?}{=} \#\mathbf{BWBP}$ is an open question.

Contributions on circuits We propose a new type of counting complexity, which fits in between $\#\mathcal{C}$ and GAP-$\mathcal{C}$ very naturally if $\mathcal{C} \in \{\mathbf{AC}^i, \mathbf{NC}^i | i \geq 0\}$. The starting point for our definition is the observation that we can extend the notion of a proof tree. A (now called positive) proof tree is a witness for a word being accepted. If a word is rejected, there are also witnesses: negative proof trees. To our knowledge, negative proof trees haven't been considered before even though the duality of positive and negative proofs is appealing. We call[1] our new counting complexity classes ZAP-$\mathcal{C}$. The functions in ZAP-$\mathcal{C}$ are of the form $\Sigma^* \to \mathbb{Z} \setminus \{0\}$. The image is positive if there are positive proof trees and negative in the case of the existence of negative proof trees.

The most fundamental result concerning ZAP-$\mathcal{C}$ is, how to compute these functions. Similar to $\#\mathcal{C}$, there is an arithmetic interpretation of the circuit which computes exactly the number of positive or negative proofs. By the nature of the ZAP operator, we are not restricted to monotone circuits any more which is different compared to the # case. The second result is that in the case of $\mathcal{C} \in \{\mathbf{NC}^i, \mathbf{AC}^i | i \geq 0\}$, the ZAP-$\mathcal{C}$ functions can be written as differences of $\#\mathcal{C}$ functions with the restriction that the result must not be 0. This uses the fact that each circuit can be transformed in a way that each input has exactly either one negative or one positive proof tree. Those two results place ZAP-$\mathcal{C}$ right between $\#\mathcal{C}$ and GAP-$\mathcal{C}$. So the ZAP version of classes might give us new possibilities to examine the differences between the # and the GAP versions. As always we are most interested in the $\mathbf{NC}^1$ case.

Contributions on branching programs We extend the ZAP idea to BPs. To do so, we need an analogon to the negative proof trees known from circuits. We found a natural analogue for negative proofs in the case of BPs, which is the notion of *cuts*. A cut in our sense is a partition of the BP's nodes in two, such that source and target are separated and no undesired edge goes between the two parts. Under this definition, it is clear that, given a BP and some input, there is a path, if and only if, there is no cut. We show, how BPs are related to ZAP-$\mathbf{NC}^1$. Also we show a simulation of ZAP-$\mathbf{NC}^1$ functions with BPs. The BPs generated that way are planar but not bounded.

Contributions on Turing machines In the end, we will complete the picture by considering Turing machines and propose a way to define negative proofs for them. The class **NP** will be used as a running example. Like many other classes, **NP** can be characterized by skew circuits. Such a circuit-based characterization is in fact very close to the actual Turing machine and it is reasonable to use the negative proofs from the circuits for Turing machines. We then can link prominent open problems into our framework.

Organization of the paper In Section 2, we cover the basic definitions. In Section 3, we introduce negative proofs to circuits and

- show an arithmetization procedure (Theorem 1).
- show how ZAP-$\mathcal{C}$ functions can be composed similarly to GAP-$\mathcal{C}$ (Theorem 2).

Section 4 extends the idea of negative proofs to branching programs and we show lower and upper bounds for ZAP-$\mathbf{NC}^1$ in terms of branching programs (Theorem 3). Section 5 proposes negative proofs for Turing machines and Section 6 is the discussion.

We thank the anonymous referees for their helpful comments which benefited the paper greatly.

2. Preliminaries

In this paper words are always composed from letters of the alphabet $\Sigma = \{0, 1\}$. There are simulation methods for different alphabets. By $\mathbb{Z}$ we denote integers and by $\mathbb{N}$ the nonnegative integers. A language L is a subset of Σ^*.

[1] The naming is motivated by the set of integers $\mathbb{Z}$ and GAP.