Instruction sequence based non-uniform complexity classes

Bergstra, J.; Middelburg, C.

DOI
10.7561/SACS.2014.1.47

Publication date
2014

Document Version
Final published version

Published in
Scientific Annals of Computer Science

Scientific Annals of Computer Science vol. 24 (1), 2014, pp. 47–89
doi: 10.7561/SACS.2014.1.47

Instruction Sequence Based Non-uniform Complexity Classes

Jan Bergstra[1], Cornelis Middelburg[1]

Abstract

We present an approach to non-uniform complexity in which single-pass instruction sequences play a key part, and answer various questions that arise from this approach. We introduce several kinds of non-uniform complexity classes. One kind includes a counterpart of the well-known non-uniform complexity class P/poly and another kind includes a counterpart of the well-known non-uniform complexity class NP/poly. Moreover, we introduce a general notion of completeness for the non-uniform complexity classes of the latter kind. We also formulate a counterpart of the well-known complexity theoretic conjecture that NP $\not\subseteq$ P/poly. We think that the presented approach opens up an additional way of investigating issues concerning non-uniform complexity.

Keywords: non-uniform complexity class, single-pass instruction sequence, projective Boolean function family

1 Introduction

The aim of this paper is to draw attention to an approach to non-uniform complexity which is based on the simple idea that each Boolean function can be computed by a single-pass instruction sequence that contains only instructions to read and write the contents of Boolean registers, forward jump instructions, and a termination instruction.

In the first place, we introduce a kind of non-uniform complexity classes which includes a counterpart of the classical non-uniform complexity class

[1]Informatics Institute, Faculty of Science, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, the Netherlands, Email: {J.A.Bergstra,C.A.Middelburg}@uva.nl.

P/poly and formulate a counterpart of the well-known complexity theoretic conjecture that NP $\not\subseteq$ P/poly. Some evidence for this conjecture is the Karp-Lipton theorem [17]. The counterpart of the conjecture formulated in this paper is called the non-uniform super-polynomial complexity conjecture. The counterpart of P/poly is denoted by $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$.

Over and above that, we introduce a kind of non-uniform complexity classes which includes a counterpart of the non-uniform complexity class NP/poly and introduce a general notion of completeness for the complexity classes of this kind. This general notion of completeness is defined using reducibility relations that can be regarded as non-uniform variants of the reducibility relation in terms of which NP-completeness is usually defined. The counterpart of NP/poly is denoted by $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$.

We show among other things that the complexity classes P/poly and NP/poly coincide with the complexity classes $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ and $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$, respectively, and that a problem closely related to 3SAT, and used to formulate the counterpart of the conjecture that NP $\not\subseteq$ P/poly, is NP-complete and $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$-complete.

In computer science, the meaning of programs usually plays a prominent part in the explanation of many issues concerning programs. Moreover, what is taken for the meaning of programs is mathematical by nature. Yet, it is customary that practitioners do not fall back on the mathematical meaning of programs in case explanation of issues concerning programs is needed. They phrase their explanations from an empirical perspective. An empirical perspective that we consider appealing is the perspective that a program is in essence an instruction sequence and an instruction sequence under execution produces a behaviour that is controlled by its execution environment in the sense that each step of the produced behaviour actuates the processing of an instruction by the execution environment and a reply returned at completion of the processing determines how the behaviour proceeds.

An attempt to approach the semantics of programming languages from the perspective mentioned above is made in [5]. The groundwork for the approach is an algebraic theory of single-pass instruction sequences, called program algebra, and an algebraic theory of mathematical objects that represent the behaviours produced by instruction sequences under execution, called basic thread algebra. The main advantages of the approach are that it does not require a lot of mathematical background and that it is more appealing to practitioners than the main approaches to programming language semantics.

As a continuation of the work on the above-mentioned approach to programming language semantics, the notion of an instruction sequence was subjected to systematic and precise analysis using the groundwork laid earlier. This led among other things to expressiveness results about the instruction sequences considered and variations of the instruction sequences considered (see e.g. [11, 12]). As another continuation of the work on the above-mentioned approach to programming language semantics, selected issues relating to well-known subjects from the theory of computation and the area of computer architecture were rigorously investigated thinking in terms of instruction sequences (see e.g. [8, 9]). The general aim of the work in both continuations mentioned is to bring instruction sequences as a theme in computer science better into the picture. The work presented in this paper forms a part of the last mentioned continuation.

The starting-point of program algebra is the perception of a program as a single-pass instruction sequence, i.e. a finite or infinite sequence of instructions of which each instruction is executed at most once and can be dropped after it has been executed or jumped over. This perception is simple, appealing, and links up with practice. The concepts underlying the primitives of program algebra are common in programming, but the particular form of the primitives is not common. The predominant concern in the design of program algebra has been to achieve simple syntax and semantics, while maintaining the expressive power of arbitrary finite control.

The objects considered in basic thread algebra represent in a direct way the behaviours produced by instruction sequences under execution: upon each action performed by such an object, a reply from an execution environment, which takes the action as an instruction to be processed, determines how it proceeds. The objects concerned are called threads. A thread may make use of services, i.e. components of the execution environment. Once introduced into threads and services, it is rather obvious that each Turing machine can be simulated by means of a thread that makes use of a service. The thread and service correspond to the finite control and tape of the Turing machine.

The approach to complexity followed in this paper is not suited to uniform complexity. This is not considered a great drawback. Non-uniform complexity is the relevant notion of complexity when studying what looks to be the major complexity issue in practice: the scale-dependence of what is an efficient solution for a computational problem.

This paper is organized as follows. First, we survey program algebra and basic thread algebra (Section 2). Next, we survey an extension of basic

thread algebra concerning the interaction of threads with services and give a description of Boolean register services (Sections 3 and 4). Then, we introduce the kind of complexity classes that includes $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ and formulate the non-uniform super-polynomial complexity conjecture (Sections 5, 6 and 7). After that, we introduce the kind of complexity classes that includes $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$ and the notion of completeness for the non-uniform complexity classes of this kind (Sections 8 and 9). We also introduce two additional kinds of complexity classes suggested by the counterpart of 3SAT used to formulate the non-uniform super-polynomial complexity conjecture (Section 10). Finally, we make some concluding remarks (Section 11).

Some familiarity with classical computational complexity is assumed. The relevant notions are explained in many textbooks, including [1, 3, 15]. Their precise definitions in different publications differ slightly. The definitions of classical notions on which some results in this paper are based are the ones from Chapters 1, 2 and 6 of [1].

This paper supersedes [7] and Section 5.2 of [10] in several respects. Generalization of the definitions of the complexity classes $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ and $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$ has put these complexity classes into a broader context, and a major technical change has made it possible to simplify the material that is concerned with the complexity class $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$. Moreover, two additional kinds of complexity classes are introduced, and various additional results are given.

2 Program Algebra and Basic Thread Algebra

In this section, we survey PGA (ProGram Algebra) and BTA (Basic Thread Algebra) and make precise in the setting of BTA which behaviours are produced on execution by the instruction sequences considered in PGA.

In PGA, it is assumed that there is a fixed but arbitrary set $\mathfrak{A}$ of *basic instructions*. The intuition is that the execution of a basic instruction may modify a state and produces a reply at its completion. The possible replies are T and F. The actual reply is generally state-dependent. Therefore, successive executions of the same basic instruction may produce different replies. The set $\mathfrak{A}$ is the basis for the set of instructions that may occur in the instruction sequences considered in PGA. The elements of the latter set are called *primitive instructions*.

PGA has the following primitive instructions:

- for each $a \in \mathfrak{A}$, a *plain basic instruction* a;

- for each $a \in \mathfrak{A}$, a *positive test instruction* $+a$;
- for each $a \in \mathfrak{A}$, a *negative test instruction* $-a$;
- for each $l \in \mathbb{N}$, a *forward jump instruction* $\#l$;
- a *termination instruction* $!$.

We write $\mathfrak{I}$ for the set of all primitive instructions.

On execution of an instruction sequence, these primitive instructions have the following effects:

- the effect of a positive test instruction $+a$ is that basic instruction a is executed and execution proceeds with the next primitive instruction if T is produced and otherwise the next primitive instruction is skipped and execution proceeds with the primitive instruction following the skipped one — if there is no primitive instruction to proceed with, inaction occurs;
- the effect of a negative test instruction $-a$ is the same as the effect of $+a$, but with the role of the value produced reversed;
- the effect of a plain basic instruction a is the same as the effect of $+a$, but execution always proceeds as if T is produced;
- the effect of a forward jump instruction $\#l$ is that execution proceeds with the lth next primitive instruction of the instruction sequence concerned — if l equals 0 or there is no primitive instruction to proceed with, inaction occurs;
- the effect of the termination instruction $!$ is that execution terminates.

PGA has one sort: the sort **IS** of *instruction sequences*. We make this sort explicit to anticipate the need for many-sortedness later on. To build terms of sort **IS**, PGA has the following constants and operators:

- for each $u \in \mathfrak{I}$, the *instruction* constant $u : \to \mathbf{IS}$;
- the binary *concatenation* operator $\_ ; \_ : \mathbf{IS} \times \mathbf{IS} \to \mathbf{IS}$;
- the unary *repetition* operator $\_^\omega : \mathbf{IS} \to \mathbf{IS}$.

Table 1: Axioms of PGA

$(X\,;Y)\,;Z = X\,;(Y\,;Z)$	PGA1
$(X^n)^\omega = X^\omega$	PGA2
$X^\omega\,;Y = X^\omega$	PGA3
$(X\,;Y)^\omega = X\,;(Y\,;X)^\omega$	PGA4

Terms of sort **IS** are built as usual. Throughout the paper, we assume that there are infinitely many variables of sort **IS**, including X, Y, Z. We use infix notation for concatenation and postfix notation for repetition.

A closed PGA term is considered to denote a non-empty, finite or eventually periodic infinite sequence of primitive instructions.[2] The instruction sequence denoted by a closed term of the form $P\,;Q$ is the instruction sequence denoted by P concatenated with the instruction sequence denoted by Q. The instruction sequence denoted by a closed term of the form P^ω is the instruction sequence denoted by P concatenated infinitely many times with itself.

Closed PGA terms are considered equal if they represent the same instruction sequence. The axioms for instruction sequence equivalence are given in Table 1. In this table, n stands for an arbitrary natural number greater than 0. For each $n > 0$, the term P^n, where P is a PGA term, is defined by induction on n as follows: $P^1 = P$ and $P^{n+1} = P\,;P^n$. The *unfolding* equation $X^\omega = X\,;X^\omega$ is derivable. Each closed PGA term is derivably equal to a term in *canonical form*, i.e. a term of the form P or $P;Q^\omega$, where P and Q are closed PGA terms in which the repetition operator does not occur.

A typical model of PGA is the model in which:

- the domain is the set of all finite and eventually periodic infinite sequences over the set $\mathfrak{I}$ of primitive instructions;
- the operation associated with $;$ is concatenation;
- the operation associated with $^\omega$ is the operation $^{\underline{\omega}}$ defined as follows:
  - if U is a finite sequence, then $U^{\underline{\omega}}$ is the unique eventually periodic

[2] An eventually periodic infinite sequence is an infinite sequence with only finitely many distinct suffixes.

infinite sequence U' such that U concatenated n times with itself is a proper prefix of U' for each $n \in \mathbb{N}$;

– if U is an eventually periodic infinite sequence, then $U^{\underline{\omega}}$ is U.

To simplify matters, we confine ourselves to this model of PGA, which is an initial model of PGA, for the interpretation of PGA terms. In the sequel, we use the term *instruction sequence* for the elements of the domain of this model, and we denote the interpretations of the constants and operators in this model by the constants and operators themselves.

In the remainder of this paper, we consider instruction sequences that can be denoted by closed PGA terms in which the repetition operator does not occur. Below, we will make precise which behaviours are produced by instruction sequences that can be denoted by closed PGA terms in which the repetition operator does not occur.

First, we survey BTA, an algebraic theory of mathematical objects which represent in a direct way the behaviours produced by instruction sequences under execution.

In BTA, it is assumed that a fixed but arbitrary set $\mathcal{A}$ of *basic actions*, with $\mathsf{tau} \notin \mathcal{A}$, has been given. Besides, tau is a special basic action. We write $\mathcal{A}_{\mathsf{tau}}$ for $\mathcal{A} \cup \{\mathsf{tau}\}$.

The objects considered in BTA are called threads. A thread represents a behaviour which consists of performing basic actions in a sequential fashion. Upon each basic action performed, a reply from an execution environment determines how the thread proceeds. The possible replies are the Boolean values T and F. Performing tau, which is considered performing an internal action, will always lead to the reply T.

BTA has one sort: the sort $\mathbf{T}$ of *threads*. We make this sort explicit to anticipate the need for many-sortedness later on. To build terms of sort $\mathbf{T}$, BTA has the following constants and operators:

- the *inaction* constant $\mathsf{D} : \to \mathbf{T}$;
- the *termination* constant $\mathsf{S} : \to \mathbf{T}$;
- for each $\alpha \in \mathcal{A}_{\mathsf{tau}}$, the binary *postconditional composition* operator $\_ \trianglelefteq \alpha \trianglerighteq \_ : \mathbf{T} \times \mathbf{T} \to \mathbf{T}$.

Terms of sort $\mathbf{T}$ are built as usual. Throughout the paper, we assume that there are infinitely many variables of sort $\mathbf{T}$, including x, y, z. We use infix notation for postconditional composition.

Table 2: Axiom of BTA

$x \trianglelefteq \mathsf{tau} \trianglerighteq y = x \trianglelefteq \mathsf{tau} \trianglerighteq x$	T1

We introduce *basic action prefixing* as an abbreviation: $\alpha \circ p$, where p is a BTA term, abbreviates $p \trianglelefteq \alpha \trianglerighteq p$. We identify expressions of the form $\alpha \circ p$ with the BTA term they stand for.

The thread denoted by a closed term of the form $p \trianglelefteq \alpha \trianglerighteq q$ will first perform α, and then proceed as the thread denoted by p if the reply from the execution environment is T and proceed as the thread denoted by q if the reply from the execution environment is F. The thread denoted by D will become inactive and the thread denoted by S will terminate.

BTA has only one axiom. This axiom is given in Table 2. Using the abbreviation introduced above, axiom T1 can be written as follows: $x \trianglelefteq \mathsf{tau} \trianglerighteq y = \mathsf{tau} \circ x$.

Each closed BTA term denotes a finite thread, i.e. a thread with a finite upper bound to the number of basic actions that it can perform. Infinite threads, i.e. threads without a finite upper bound to the number of basic actions that it can perform, can be defined by means of a set of recursion equations (see e.g. [6]). Regular threads, i.e. finite or infinite threads that can only be in a finite number of states, can be defined by means of a finite set of recursion equations.

The behaviours of the instruction sequences denoted by closed PGA terms are considered to be regular threads, with the basic instructions taken for basic actions. All regular threads in which tau does not occur represent behaviours of instruction sequences that can be denoted by closed PGA terms (see Proposition 2 in [21]). Closed PGA terms in which the repetition operator does not occur correspond to finite threads.

Henceforth, we will write $\mathrm{PGA}_{\mathrm{fin}}$ for PGA without the repetition operator and axioms PGA2–PGA4, and we will write $\mathrm{IS}_{\mathrm{fin}}$ for the set of all instruction sequences that can be denoted by closed $\mathrm{PGA}_{\mathrm{fin}}$ terms. Moreover, we will write $\mathit{length}(U)$, where $U \in \mathrm{IS}_{\mathrm{fin}}$, for the length of U.

We combine $\mathrm{PGA}_{\mathrm{fin}}$ with BTA and extend the combination with the *thread extraction* operator $|\_| : \mathbf{IS} \to \mathbf{T}$ and the axioms given in Table 3. In this table, a stands for an arbitrary basic instruction from $\mathfrak{A}$, u stands for an arbitrary primitive instruction from $\mathfrak{I}$, and l stands for an arbitrary natural number.

For each closed $\mathrm{PGA}_{\mathrm{fin}}$ term P, $|P|$ denotes the behaviour produced

Table 3: Axioms for the thread extraction operator

$\lvert a\rvert = a \circ \mathsf{D}$	$\lvert \#l\rvert = \mathsf{D}$
$\lvert a\,;X\rvert = a \circ \lvert X\rvert$	$\lvert \#0\,;X\rvert = \mathsf{D}$
$\lvert +a\rvert = a \circ \mathsf{D}$	$\lvert \#1\,;X\rvert = \lvert X\rvert$
$\lvert +a\,;X\rvert = \lvert X\rvert \trianglelefteq a \trianglerighteq \lvert \#2\,;X\rvert$	$\lvert \#l+2\,;u\rvert = \mathsf{D}$
$\lvert -a\rvert = a \circ \mathsf{D}$	$\lvert \#l+2\,;u\,;X\rvert = \lvert \#l+1\,;X\rvert$
$\lvert -a\,;X\rvert = \lvert \#2\,;X\rvert \trianglelefteq a \trianglerighteq \lvert X\rvert$	$\lvert !\rvert = \mathsf{S}$
	$\lvert !\,;X\rvert = \mathsf{S}$

by the instruction sequence denoted by P under execution. The use of a closed $\mathrm{PGA}_{\mathrm{fin}}$ term is sometimes preferable to the use of the corresponding closed BTA term because thread extraction can give rise to a combinatorial explosion. For instance, suppose that p is a closed BTA term such that

$$p = |\overbrace{+a\,;+b\,;\ldots;+a\,;+b}^{k\times}\,;c\,;!|\,.$$

Then the size of p is greater than $2^{k/2}$.

3 Interaction of Threads with Services

A thread may perform a basic action for the purpose of requesting a named service provided by an execution environment to process a method and to return a reply to the thread at completion of the processing of the method. In this section, we survey the extension of BTA with services and operators that are concerned with this kind of interaction between threads and services.

It is assumed that a fixed but arbitrary set $\mathcal{F}$ of *foci* has been given. Foci play the role of names of the services provided by an execution environment. It is also assumed that a fixed but arbitrary set $\mathcal{M}$ of *methods* has been given. For the set $\mathcal{A}$ of basic actions, we take the set $\{f.m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$. Performing a basic action $f.m$ is taken as making a request to the service named f to process method m.

A service is able to process certain methods. The processing of a method may involve a change of the service. The reply value produced by the service at completion of the processing of a method is either T, F or B. The special

reply B, standing for blocked, is used to deal with the situation that a service is requested to process a method that it is not able to process.

The following is assumed with respect to services:

- a many-sorted signature $\Sigma_{\mathcal{S}}$ has been given that includes the following sorts:
  - the sort $\mathbf{S}$ of *services*;
  - the sort $\mathbf{R}$ of *replies*;

  and the following constants and operators:
  - the *empty service* constant $\delta : \to \mathbf{S}$;
  - the *reply* constants $\mathsf{T}, \mathsf{F}, \mathsf{B} : \to \mathbf{R}$;
  - for each $m \in \mathcal{M}$, the *derived service* operator $\frac{\partial}{\partial m} : \mathbf{S} \to \mathbf{S}$;
  - for each $m \in \mathcal{M}$, the *service reply* operator $\varrho_m : \mathbf{S} \to \mathbf{R}$;
- a minimal $\Sigma_{\mathcal{S}}$-algebra $\boldsymbol{\mathcal{S}}$ has been given in which T, F, and B are mutually different, and
  - $\bigwedge_{m \in \mathcal{M}} \frac{\partial}{\partial m}(z) = z \wedge \varrho_m(z) = \mathsf{B} \Rightarrow z = \delta$ holds;
  - for each $m \in \mathcal{M}$, $\frac{\partial}{\partial m}(z) = \delta \Leftrightarrow \varrho_m(z) = \mathsf{B}$ holds.

The intuition concerning $\frac{\partial}{\partial m}$ and ϱ_m is that on a request to service S to process method m:

- if $\varrho_m(S) \neq \mathsf{B}$, S processes m, produces the reply $\varrho_m(S)$, and then proceeds as $\frac{\partial}{\partial m}(S)$;
- if $\varrho_m(S) = \mathsf{B}$, S is not able to process method m and proceeds as δ.

The empty service δ itself is unable to process any method.

We introduce the following additional operators:

- for each $f \in \mathcal{F}$, the binary *use* operator $\_ \,/_f\, \_ : \mathbf{T} \times \mathbf{S} \to \mathbf{T}$;
- for each $f \in \mathcal{F}$, the binary *apply* operator $\_ \bullet_f \_ : \mathbf{T} \times \mathbf{S} \to \mathbf{S}$.

We use infix notation for the use and apply operators.

The thread denoted by a closed term of the form $p \,/_f\, S$ and the service denoted by a closed term of the form $p \bullet_f S$ are the thread and service, respectively, that result from processing the method of each basic action of the form $f.m$ that the thread denoted by p performs by the service denoted

Table 4: Axioms for the use operators

$\mathsf{S} /_f S = \mathsf{S}$		U1
$\mathsf{D} /_f S = \mathsf{D}$		U2
$(\mathsf{tau} \circ x) /_f S = \mathsf{tau} \circ (x /_f S)$		U3
$(x \trianglelefteq g.m \trianglerighteq y) /_f S = (x /_f S) \trianglelefteq g.m \trianglerighteq (y /_f S)$	if $f \neq g$	U4
$(x \trianglelefteq f.m \trianglerighteq y) /_f S = \mathsf{tau} \circ (x /_f \frac{\partial}{\partial m}(S))$	if $\varrho_m(S) = \mathsf{T}$	U5
$(x \trianglelefteq f.m \trianglerighteq y) /_f S = \mathsf{tau} \circ (y /_f \frac{\partial}{\partial m}(S))$	if $\varrho_m(S) = \mathsf{F}$	U6
$(x \trianglelefteq f.m \trianglerighteq y) /_f S = \mathsf{tau} \circ \mathsf{D}$	if $\varrho_m(S) = \mathsf{B}$	U7

Table 5: Axioms for the apply operators

$\mathsf{S} \bullet_f S = S$		A1
$\mathsf{D} \bullet_f S = \delta$		A2
$(\mathsf{tau} \circ x) \bullet_f S = x \bullet_f S$		A3
$(x \trianglelefteq g.m \trianglerighteq y) \bullet_f S = \delta$	if $f \neq g$	A4
$(x \trianglelefteq f.m \trianglerighteq y) \bullet_f S = x \bullet_f \frac{\partial}{\partial m} S$	if $\varrho_m(S) = \mathsf{T}$	A5
$(x \trianglelefteq f.m \trianglerighteq y) \bullet_f S = y \bullet_f \frac{\partial}{\partial m} S$	if $\varrho_m(S) = \mathsf{F}$	A6
$(x \trianglelefteq f.m \trianglerighteq y) \bullet_f S = \delta$	if $\varrho_m(S) = \mathsf{B}$	A7

by S. When the method of a basic action of the form $f.m$ performed by a thread is processed by a service, the service changes in accordance with the method concerned and affects the thread as follows: the basic action turns into the internal action tau and the two ways to proceed reduce to one on the basis of the reply value produced by the service.

The axioms for the use operators are given in Table 4 and the axioms for the apply operators are given in Table 5. In these tables, f and g stand for arbitrary foci from $\mathcal{F}$, m stands for an arbitrary method from $\mathcal{M}$, and S stands for an arbitrary term of sort $\mathbf{S}$. The axioms simply formalize the informal explanation given above and in addition stipulate what is the result of use and apply if inappropriate foci or methods are involved.

The extension of BTA described in this section is a simple version of the extension of BTA presented in [9]. We have chosen to use the former extension because it is adequate to the purpose of this paper and it allows a terser survey.

4 Instruction Sequences Acting on Boolean Registers

In our approach to computational complexity, instruction sequences that act on Boolean registers play a key part. Preceding the presentation of this approach, we describe in this section services that make up Boolean registers, introduce special foci that serve as names of Boolean registers, and describe the instruction sequences that matter to the kinds of complexity classes introduced in this paper.

First, we describe services that make up Boolean registers. The Boolean register services are able to process the following methods:

- the *set to true method* $\mathsf{set{:}T}$;
- the *set to false method* $\mathsf{set{:}F}$;
- the *get method* get.

We write $\mathcal{M}_{\mathrm{br}}$ for the set $\{\mathsf{set{:}T}, \mathsf{set{:}F}, \mathsf{get}\}$. It is assumed that $\mathcal{M}_{\mathrm{br}} \subseteq \mathcal{M}$.

The methods that Boolean register services are able to process can be explained as follows:

- $\mathsf{set{:}T}$: the contents of the Boolean register becomes T and the reply is T;
- $\mathsf{set{:}F}$: the contents of the Boolean register becomes F and the reply is F;
- get: nothing changes and the reply is the contents of the Boolean register.

For $\Sigma_{\mathcal{S}}$, we take the signature that consists of the sorts, constants and operators that are mentioned in the assumptions with respect to services made in Section 3 and a constant BR_b of sort $\mathbf{S}$ for each $b \in \mathbb{B}$.

For $\boldsymbol{\mathcal{S}}$, we take a minimal $\Sigma_{\mathcal{S}}$-algebra that satisfies the conditions that are mentioned in the assumptions with respect to services made in Section 3 and the following conditions for each $b \in \mathbb{B}$:

$$\frac{\partial}{\partial \mathsf{set{:}T}}(BR_b) = BR_{\mathsf{T}} \;, \qquad \frac{\partial}{\partial \mathsf{get}}(BR_b) = BR_b \;,$$
$$\frac{\partial}{\partial \mathsf{set{:}F}}(BR_b) = BR_{\mathsf{F}} \;, \qquad \frac{\partial}{\partial m}(BR_b) = \delta \qquad \text{if } m \notin \{\mathsf{set{:}T}, \mathsf{set{:}F}, \mathsf{get}\} \;,$$

$$\varrho_{\mathsf{set{:}T}}(BR_b) = \mathsf{T} \;, \qquad \varrho_{\mathsf{get}}(BR_b) = b \;,$$
$$\varrho_{\mathsf{set{:}F}}(BR_b) = \mathsf{F} \;, \qquad \varrho_m(BR_b) = \mathsf{B} \qquad \text{if } m \notin \{\mathsf{set{:}T}, \mathsf{set{:}F}, \mathsf{get}\} \;.$$

In the instruction sequences which concern us in the remainder of this paper, a number of Boolean registers is used as input registers, a number of Boolean registers is used as auxiliary registers, and one Boolean register is used as output register.

It is assumed that $\mathsf{in}{:}1, \mathsf{in}{:}2, \ldots \in \mathcal{F}$, $\mathsf{aux}{:}1, \mathsf{aux}{:}2, \ldots \in \mathcal{F}$, and $\mathsf{out} \in \mathcal{F}$. These foci play special roles:

- for each $i \in \mathbb{N}^+$, $\mathsf{in}{:}i$ serves as the name of the Boolean register that is used as ith input register in instruction sequences;
- for each $i \in \mathbb{N}^+$, $\mathsf{aux}{:}i$ serves as the name of the Boolean register that is used as ith auxiliary register in instruction sequences;
- out serves as the name of the Boolean register that is used as output register in instruction sequences.

Henceforth, we will write $\mathcal{F}_{\mathsf{in}}$ for $\{\mathsf{in}{:}i \mid i \in \mathbb{N}^+\}$ and $\mathcal{F}_{\mathsf{aux}}$ for $\{\mathsf{aux}{:}i \mid i \in \mathbb{N}^+\}$.

$\mathrm{IS}_{\mathrm{br}}$ is the set of all instruction sequences from $\mathrm{IS}_{\mathrm{fin}}$ in which all plain basic instructions, positive test instructions and negative test instructions contain only basic instructions from the set

$$\{f.\mathsf{get} \mid f \in \mathcal{F}_{\mathsf{in}} \cup \mathcal{F}_{\mathsf{aux}}\} \cup \{f.\mathsf{set}{:}b \mid f \in \mathcal{F}_{\mathsf{aux}} \cup \{\mathsf{out}\} \wedge b \in \mathbb{B}\} \ ;$$

$\mathrm{IS}_{\mathrm{br}}$ is the set of all instruction sequences from $\mathrm{IS}_{\mathrm{fin}}$ that matter to the kinds of complexity classes which will be introduced in this paper.

For each $k, l \in \mathbb{N}$, we will write $\mathrm{IS}_{\mathrm{br}}^{k,l}$ for the set of all $X \in \mathrm{IS}_{\mathrm{br}}$ that satisfy:

- primitive instructions of the forms $\mathsf{aux}{:}i.m$, $+\mathsf{aux}{:}i.m$ and $-\mathsf{aux}{:}i.m$ with $i > k$ do not occur in X;
- primitive instructions of the form $\#l'$ with $l' > l$ do not occur in X.

Moreover, for each $k \in \mathbb{N}$, we will write $\mathrm{IS}_{\mathrm{br}}^{k}$ for the set $\bigcup_{l \in \mathbb{N}} \mathrm{IS}_{\mathrm{br}}^{k,l}$. Hence, $\mathrm{IS}_{\mathrm{br}}^{0}$ is the set of all instruction sequences from $\mathrm{IS}_{\mathrm{br}}$ in which no auxiliary registers are used, and $\mathrm{IS}_{\mathrm{br}}^{0,0}$ is the set of all instruction sequences from $\mathrm{IS}_{\mathrm{br}}^{0}$ in which jump instructions do not occur.

5 The Complexity Classes $IS\backslash F$

In this section, we introduce a kind of non-uniform complexity classes which includes a counterpart of the complexity class P/poly in the setting of single-pass instruction sequences.

The counterpart of P/poly defined in this section is denoted by $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$. Because it is isomorphic to the complexity class P/poly, we could have decided to loosely denote this complexity class by P/poly as well. The reason why we decided not to denote it by P/poly finds its origin in what we want to achieve with this paper: illustrating an approach to non-uniform complexity in which single-pass instruction sequences play a key part. We reserve the use of the name P/poly to where results obtained in the setting of Turing machines or the setting of Boolean circuits are involved.

In the field of computational complexity, it is quite common to study the complexity of computing functions on finite strings over a binary alphabet. Since strings over an alphabet of any fixed size can be efficiently encoded as strings over a binary alphabet, it is sufficient to consider only a binary alphabet. We adopt the set $\mathbb{B}$ as preferred binary alphabet.

An important special case of functions on finite strings over a binary alphabet is the case where the value of functions is restricted to strings of length 1. Such a function is often identified with the set of strings of which it is the characteristic function. The set in question is usually called a language or a decision problem. The identification mentioned above allows of looking at the problem of computing a function $f : \mathbb{B}^* \to \mathbb{B}$ as the problem of deciding membership of the set $\{w \in \mathbb{B}^* \mid f(w) = \mathsf{T}\}$.

With each function $f : \mathbb{B}^* \to \mathbb{B}$, we can associate an infinite sequence $\langle f_n \rangle_{n \in \mathbb{N}}$ of functions, with $f_n : \mathbb{B}^n \to \mathbb{B}$ for every $n \in \mathbb{N}$, such that f_n is the restriction of f to $\mathbb{B}^n$ for each $n \in \mathbb{N}$. The complexity of computing such sequences of functions, which we call Boolean function families, by instruction sequences is our concern in the remainder of this paper. One of the classes of Boolean function families with which we concern us is $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$, the class of all Boolean function families that can be computed by polynomial-length instruction sequences from $\mathrm{IS}_{\mathrm{br}}$.

An *n-ary Boolean function* is a function $f : \mathbb{B}^n \to \mathbb{B}$, and a *Boolean function family* is an infinite sequence $\langle f_n \rangle_{n \in \mathbb{N}}$ of functions, where f_n is an n-ary Boolean function for each $n \in \mathbb{N}$.

A Boolean function family $\langle f_n \rangle_{n \in \mathbb{N}}$ can be identified with the unique function $f : \mathbb{B}^* \to \mathbb{B}$ such that for each $n \in \mathbb{N}$, for each $w \in \mathbb{B}^n$, $f(w) = f_n(w)$. Considering sets of Boolean function families as complexity classes looks

to be most natural when studying non-uniform complexity. We will make the identification mentioned above only where connections with classical complexity classes such as P/poly are made.

Let $n \in \mathbb{N}$, let $f : \mathbb{B}^n \to \mathbb{B}$, and let $X \in \mathrm{IS}_{\mathrm{br}}$. Then X *computes* f if there exists an $l \in \mathbb{N}$ such that for all $b_1, \ldots, b_n \in \mathbb{B}$:

$$(\ldots((\ldots(|X| \,/_{\mathsf{aux}:1}\, BR_{\mathsf{F}}) \ldots /_{\mathsf{aux}:l}\, BR_{\mathsf{F}}) \,/_{\mathsf{in}:1}\, BR_{b_1}) \ldots /_{\mathsf{in}:n}\, BR_{b_n}) \bullet_{\mathsf{out}} BR_{\mathsf{F}} = BR_{f(b_1,\ldots,b_n)}\,.^{3}$$

Moreover, let $IS \subseteq \mathrm{IS}_{\mathrm{br}}$ and $F \subseteq \{h \mid h : \mathbb{N} \to \mathbb{N}\}$. Then $IS\backslash F$ is the class of all Boolean function families $\langle f_n \rangle_{n\in\mathbb{N}}$ that satisfy:

> there exists an $h \in F$ such that for all $n \in \mathbb{N}$ there exists an $X \in IS$ such that X computes f_n and $\mathit{length}(X) \leq h(n)$.

Henceforth, we will write poly for $\{h \mid h : \mathbb{N} \to \mathbb{N} \wedge h \,\text{is polynomial}\}$. We are primarily interested in the complexity class $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$,[4] but we will also pay attention to other instantiations of the general definition just given.

The question arises whether all n-ary Boolean functions can be computed by an instruction sequence from $\mathrm{IS}_{\mathrm{br}}$. This question can answered in the affirmative. They can even be computed, without using auxiliary Boolean registers, by an instruction sequence that contains no other jump instructions than #2.

Theorem 1 *For each $n \in \mathbb{N}$, for each n-ary Boolean function $f : \mathbb{B}^n \to \mathbb{B}$, there exists an $X \in \mathrm{IS}^0_{\mathrm{br}}$ in which no other jump instruction than #2 occurs such that X computes f and $\mathit{length}(X) = O(2^n)$.*[5]

Proof: Let inseq_n be the function from the set of all n-ary Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ to $\mathrm{IS}^0_{\mathrm{br}}$ defined by induction on n as follows:

$$\begin{aligned} \mathit{inseq}_0(f) &= \begin{cases} -\mathsf{out.set{:}T}\,;\#2\,;!\,; & \text{if } f() = \mathsf{T} \\ +\mathsf{out.set{:}F}\,;\#2\,;!\,; & \text{if } f() = \mathsf{F}\,, \end{cases} \\ \mathit{inseq}_{n+1}(f) &= -\mathsf{in}{:}n{+}1.\mathsf{get}\,;\#2\,;\mathit{inseq}_n(f_{\mathsf{T}})\,;\mathit{inseq}_n(f_{\mathsf{F}})\,, \end{aligned}$$

[3] In the extension of BTA presented in [9], which has a sort of (named) service families and a service family composition operator ($\oplus$), the left-hand side of this equation can be written as follows: $(|X| \,/\, ((\bigoplus_{i=1}^{n} \mathsf{in}{:}i.BR_{b_i}) \oplus (\bigoplus_{j=1}^{l} \mathsf{aux}{:}j.BR_{\mathsf{F}}))) \bullet \mathsf{out}.BR_{\mathsf{F}}$.

[4] In precursors of this paper, the temporary name P^* is used for the complexity class $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ (see e.g. [7]).

[5] Theorem 1 sharpens the result found in precursors of this paper (see e.g. [7]). We owe the sharpened result to Inge Bethke from the University of Amsterdam.

where for each $f : \mathbb{B}^{n+1} \to \mathbb{B}$ and $b \in \mathbb{B}$, $f_b : \mathbb{B}^n \to \mathbb{B}$ is defined as follows:

$$f_b(b_1, \ldots, b_n) = f(b_1, \ldots, b_n, b) \; .$$

It is easy to prove by induction on n that $|\#2 \,;\, \mathit{inseq}_n(f_{\mathsf{T}}) \,;\, X| = |X|$. Using this fact, it is easy to prove by induction on n that $\mathit{inseq}_n(f)$ computes f. Moreover, it is easy to see that $\mathit{length}(\mathit{inseq}_n(f)) = O(2^n)$. □

Henceforth, we will use the notation $IS \backslash O(f(n))$ for the complexity class $IS \backslash \{h \mid h : \mathbb{N} \to \mathbb{N} \wedge h(n) = O(f(n))\}$. This notation is among other things used in the following corollary of Theorem 1.

Corollary 1 *All Boolean function families belong to* $\mathrm{IS}^0_{\mathrm{br}} \backslash O(2^n)$.

In the proof of Theorem 1, the instruction sequences yielded by the function inseq_n contain the jump instruction #2. Each occurrence of #2 belongs to a jump chain ending in the instruction sequence $-\mathsf{out.set{:}T} \,;\, \#2 \,;\, !$ or the instruction sequence $+\mathsf{out.set{:}F} \,;\, \#2 \,;\, !$. Therefore, each occurrence of #2 can safely be replaced by the instruction $+\mathsf{out.set{:}F}$, which like #2 skips the next instruction. This leads to the following corollary.

Corollary 2 $\mathrm{IS}^{0,0}_{\mathrm{br}} \backslash O(2^n) = \mathrm{IS}^0_{\mathrm{br}} \backslash O(2^n) = \mathrm{IS}_{\mathrm{br}} \backslash O(2^n)$.

We consider the proof of Theorem 1 once again. Because the content of the Boolean register concerned is initially F, the question arises whether $\mathsf{out.set{:}F}$ can be dispensed with in instruction sequences computing Boolean functions. This question can be answered in the affirmative if we permit the use of auxiliary Boolean registers.

Theorem 2 *Let* $n \in \mathbb{N}$, *let* $f : \mathbb{B}^n \to \mathbb{B}$, *and let* $X \in \mathrm{IS}_{\mathrm{br}}$ *be such that* X *computes* f. *Then there exists a* $Y \in \mathrm{IS}_{\mathrm{br}}$ *in which the basic instruction* $\mathsf{out.set{:}F}$ *does not occur such that* Y *computes* f *and* $\mathit{length}(Y)$ *is linear in* $\mathit{length}(X)$.

Proof: Let $o \in \mathbb{N}^+$ be such that the basic instructions $\mathsf{aux{:}}o\mathsf{.set{:}T}$, $\mathsf{aux{:}}o\mathsf{.set{:}F}$, and $\mathsf{aux{:}}o\mathsf{.get}$ do not occur in X. Let X' be obtained from X by replacing each occurrence of the focus out by $\mathsf{aux{:}}o$. Suppose that $X' = u_1 \,;\, \ldots \,;\, u_k$. Let Y be obtained from $u_1 \,;\, \ldots \,;\, u_k$ as follows:

1. stop if $u_1 \equiv !$;
2. stop if there exists no $j \in [2, k]$ such that $u_{j-1} \not\equiv \mathsf{out.set{:}T}$ and $u_j \equiv !$;

3. find the least $j \in [2,k]$ such that $u_{j-1} \not\equiv \mathsf{out.set{:}T}$ and $u_j \equiv !$;
4. replace u_j by $+\mathsf{aux{:}}o.\mathsf{get}\,;\mathsf{out.set{:}T}\,;!$;
5. for each $i \in [1,k]$, replace u_i by $\#l{+}2$ if $u_i \equiv \#l$ and $i < j < i + l$;
6. repeat the preceding steps for the resulting instruction sequence.

It is easy to prove by induction on k that the Boolean function computed by X and the Boolean function computed by Y are the same. Moreover, it is easy to see that $\mathit{length}(Y) < 3 \cdot \mathit{length}(X)$. Hence, $\mathit{length}(Y)$ is linear in $\mathit{length}(X)$. □

The following proposition gives an upper bound for the number of instruction sequences from $\mathrm{IS}_{\mathrm{br}}^{k-1,k-1}$ of length k that compute an n-ary Boolean function. From each instruction sequence from $\mathrm{IS}_{\mathrm{br}}$ of length k that computes an n-ary Boolean function, we can obtain an instruction sequence from $\mathrm{IS}_{\mathrm{br}}^{k-1,k-1}$ of length k that computes the same n-ary Boolean function by replacement of the primitive instructions that are not permitted in $\mathrm{IS}_{\mathrm{br}}^{k-1,k-1}$. Moreover, each n-ary Boolean function that can be computed by an instruction sequence from $\mathrm{IS}_{\mathrm{br}}$ of length less than k, can also be computed by an instruction sequence from $\mathrm{IS}_{\mathrm{br}}$ of length k.

Proposition 1 *For each $k \in \mathbb{N}^+$ and $n \in \mathbb{N}$, the number of instruction sequences from $\mathrm{IS}_{\mathrm{br}}^{k-1,k-1}$ of length k that compute an n-ary Boolean function is not greater than $(3n + 10k - 2)^k$.*

Proof: The set of basic instructions from which the plain basic instructions, positive test instructions and negative test instructions occurring in the instruction sequences concerned are built consists of n basic instructions of the form $\mathsf{in{:}}i.\mathsf{get}$, $k-1$ basic instructions of each of the forms $\mathsf{aux{:}}i.\mathsf{set{:}T}$, $\mathsf{aux{:}}i.\mathsf{set{:}F}$ and $\mathsf{aux{:}}i.\mathsf{get}$, and the two basic instructions $\mathsf{out.set{:}T}$ and $\mathsf{out.set{:}F}$. Moreover, there are k different jump instructions that may occur and one termination instruction. This means that there are $3(n+3(k-1)+2)+k+1 = 3n+10k-2$ different primitive instructions that may occur in these instruction sequences. Hence, the number of instruction sequences concerned is not greater than $(3n + 10k - 2)^k$. □

Theorem 1 states that all n-ary Boolean functions can be computed by an instruction sequence from $\mathrm{IS}_{\mathrm{br}}$ whose length is exponential in n. The following theorem shows that, for large enough n, not all n-ary Boolean functions can be computed by an instruction sequence from $\mathrm{IS}_{\mathrm{br}}$ whose length is polynomial in n.

Theorem 3 *For each $n \in \mathbb{N}$ with $n > 11$, there exists a n-ary Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ such that, for each $X \in \mathrm{IS}_{\mathrm{br}}$ that computes f, $\mathit{length}(X) > \lfloor 2^n/n \rfloor$.*

Proof: Let $n \in \mathbb{N}$ be such that $n > 11$. By Proposition 1 and the remarks immediately preceding Proposition 1, the number of n-ary Boolean functions that can be computed by instruction sequences from $\mathrm{IS}_{\mathrm{br}}$ of length less than or equal to k is not greater than $(3n + 10k - 2)^k$. For $k = \lfloor 2^n/n \rfloor$, this number is not greater than $(3n + 10\lfloor 2^n/n \rfloor - 2)^{\lfloor 2^n/n \rfloor}$. We have that $(3n + 10\lfloor 2^n/n \rfloor - 2)^{\lfloor 2^n/n \rfloor} \leq (3n + 10(2^n/n) - 2)^{2^n/n} < (11(2^n/n))^{2^n/n} = (11/n)^{2^n/n} \cdot (2^n)^{2^n/n} = (11/n)^{2^n/n} \cdot 2^{(2^n)} < 2^{(2^n)}$. Here, we have used the given that $n > 11$ in the second step and the last step. So there exist less than $2^{(2^n)}$ n-ary Boolean functions that can be computed by instruction sequences from $\mathrm{IS}_{\mathrm{br}}$ of length less than or equal to $\lfloor 2^n/n \rfloor$, whereas there exist $2^{(2^n)}$ n-ary Boolean functions. Hence, there exists an n-ary Boolean function that cannot be computed by an instruction sequence from $\mathrm{IS}_{\mathrm{br}}$ of length less than or equal to $\lfloor 2^n/n \rfloor$. □

Theorem 3 gives rise to the following corollary concerning $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$.

Corollary 3 $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly} \subset \mathrm{IS}_{\mathrm{br}}\backslash O(2^n)$.

Theorem 3 will be used in the proof of the following hierarchy theorem for $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$.

Theorem 4 *For each $k \in \mathbb{N}$, $\mathrm{IS}_{\mathrm{br}}\backslash O(n^k) \subset \mathrm{IS}_{\mathrm{br}}\backslash O(n^{k+1})$.*

Proof: Let $\langle f_n \rangle_{n \in \mathbb{N}}$ be a Boolean function family such that, for each $n > 11$, for each $X \in \mathrm{IS}_{\mathrm{br}}$ that computes f_n, $\mathit{length}(X) > \lfloor 2^n/n \rfloor$. Such a Boolean function family exists by Theorem 3. Let $k \in \mathbb{N}$, and let $\langle g_n \rangle_{n \in \mathbb{N}}$ be the Boolean function family such that, for each $n \in \mathbb{N}$, $g_n(b_1, \ldots, b_n) = f_n(b_1, \ldots, b_n)$ if $n < 2^{k+3}$ and $g_n(b_1, \ldots, b_n) = f_{\lceil \log((k+2)n^{k+1}) \rceil}(b_1, \ldots, b_{\lceil \log((k+2)n^{k+1}) \rceil})$ if $n \geq 2^{k+3}$. Then $\langle g_n \rangle_{n \in \mathbb{N}} \in \mathrm{IS}_{\mathrm{br}}\backslash O(n^{k+1})$ by Theorem 1. Moreover, for each $n \geq 2^{k+3}$, for each $Y \in \mathrm{IS}_{\mathrm{br}}$ that computes g_n, $\mathit{length}(Y) > \lfloor 2^{\lceil \log((k+2)n^{k+1}) \rceil} / \lceil \log((k+2)n^{k+1}) \rceil \rfloor \geq \lfloor 2^{\log((k+2)n^{k+1})} / (\log((k+2)n^{k+1}) + 1) \rfloor \geq \lfloor (k+2)n^{k+1} / (k+2)\log(n) \rfloor = \lfloor n^{k+1} / \log(n) \rfloor$. Here, we have used the given that $n \geq 2^{k+3}$ in the last step but one. From the fact that, for all $m \in \mathbb{N}$, there exists an $n \in \mathbb{N}$ such that $n > m \log(n)$, it follows that not $\lfloor n^{k+1} / \log(n) \rfloor = O(n^k)$. Hence, $\langle g_n \rangle_{n \in \mathbb{N}} \notin \mathrm{IS}_{\mathrm{br}}\backslash O(n^k)$. □

As a corollary of the general definition of the non-uniform complexity

classes $IS\backslash F$, the fact that $\text{poly} = \bigcup_{k\in\mathbb{N}} \{h \mid h : \mathbb{N} \to \mathbb{N} \wedge h(n) = O(n^k)\}$, and Theorem 4, we have the following result.

Corollary 4 *For each $k \in \mathbb{N}$, $\text{IS}_{\text{br}}\backslash\text{poly} \not\subseteq \text{IS}_{\text{br}}\backslash O(n^k)$.*

6 Instruction Sequences, Boolean Formulas and Circuits

In this section, we investigate connections of single-pass instruction sequences with Boolean formulas and Boolean circuits which are relevant to non-uniform complexity and show that $\text{IS}_{\text{br}}\backslash\text{poly}$ coincides with P/poly. The definitions of Boolean circuits, P/poly and related notions on which some results in this section and the coming ones are based are the definitions from Chapter 6 of [1].

First, we dwell on obtaining instruction sequences that compute the Boolean functions induced by Boolean formulas from the Boolean formulas concerned.

Hereafter, we will write $\phi(b_1, \ldots, b_n)$, where ϕ is a Boolean formula containing the variables $v_1, \ldots, v_n$ and $b_1, \ldots, b_n \in \mathbb{B}$, to indicate that ϕ is satisfied by the assignment σ to the variables $v_1, \ldots, v_n$ defined by $\sigma(v_1) = b_1$, $\ldots$, $\sigma(v_n) = b_n$.

Let ϕ be a Boolean formula containing the variables $v_1, \ldots, v_n$. Then the Boolean function *induced* by ϕ is the n-ary Boolean function f defined by $f(b_1, \ldots, b_n) = \mathsf{T}$ iff $\phi(b_1, \ldots, b_n)$.

The Boolean function induced by a CNF-formula can be computed, without using auxiliary Boolean registers, by an instruction sequence that contains no other jump instructions than #2 and whose length is linear in the size of the CNF-formula.

Proposition 2 *For each* CNF*-formula ϕ, there exists an $X \in \text{IS}^0_{\text{br}}$ in which no other jump instruction than #2 occurs such that X computes the Boolean function induced by ϕ and length(X) is linear in the size of ϕ.*

Proof: Let $inseq_{\text{cnf}}$ be the function from the set of all CNF-formulas containing the variables $v_1, \ldots, v_n$ to IS^0_{br} defined as follows:

$$
\begin{aligned}
&\mathit{inseq}_{\mathrm{cnf}}(\bigwedge_{i\in[1,m]}\bigvee_{j\in[1,n_i]}\xi_{ij}) =\\
&\quad \mathit{inseq}'_{\mathrm{cnf}}(\xi_{11})\,;\ldots;\,\mathit{inseq}'_{\mathrm{cnf}}(\xi_{1n_1})\,;+\mathsf{out.set{:}F}\,;\#2\,;!;\\
&\qquad\vdots\\
&\quad \mathit{inseq}'_{\mathrm{cnf}}(\xi_{m1})\,;\ldots;\,\mathit{inseq}'_{\mathrm{cnf}}(\xi_{mn_m})\,;+\mathsf{out.set{:}F}\,;\#2\,;!\,;+\mathsf{out.set{:}T}\,;!\,,
\end{aligned}
$$

where

$$
\begin{aligned}
\mathit{inseq}'_{\mathrm{cnf}}(v_k) &= +\mathsf{in}{:}k.\mathsf{get}\,;\#2\,,\\
\mathit{inseq}'_{\mathrm{cnf}}(\neg\, v_k) &= -\mathsf{in}{:}k.\mathsf{get}\,;\#2\,.
\end{aligned}
$$

It is easy to see that no other jump instruction than #2 occurs in $\mathit{inseq}_{\mathrm{cnf}}(\phi)$. Recall that a disjunction is satisfied if one of its disjuncts is satisfied and a conjunction is satisfied if each of its conjuncts is satisfied. Using these facts, it is easy to prove by induction on the number of clauses in a CNF-formula, and in the basis step by induction on the number of literals in a clause, that $\mathit{inseq}_{\mathrm{cnf}}(\phi)$ computes the Boolean function induced by ϕ. Moreover, it is easy to see that $\mathit{length}(\mathit{inseq}_{\mathrm{cnf}}(\phi))$ is linear in the size of ϕ. □

In the proof of Proposition 2, it is shown that the Boolean function induced by a CNF-formula can be computed, without using auxiliary Boolean registers, by an instruction sequence that contains no other jump instructions than #2. However, the instruction sequence concerned contains the termination instruction more than once and both $\mathsf{out.set{:}T}$ and $\mathsf{out.set{:}F}$. This raises the question whether further restrictions are possible. We have a negative result.

Proposition 3 *Let ϕ be the Boolean formula $v_1 \wedge v_2 \wedge v_3$. Then there does not exist an $X \in \mathrm{IS}^{0,0}_{\mathrm{br}}$ in which the termination instruction does not occur more than once and the basic instruction* $\mathsf{out.set{:}F}$ *does not occur such that X computes the Boolean function induced by ϕ.*

Proof: Suppose that $X = u_1\,;\ldots;u_k$ is an instruction sequence from $\mathrm{IS}^{0,0}_{\mathrm{br}}$ satisfying the restrictions and computing the Boolean function induced by ϕ. Consider the smallest $l \in [1,k]$ such that u_l is either $\mathsf{out.set{:}T}$, $+\mathsf{out.set{:}T}$ or $-\mathsf{out.set{:}T}$ (there must be such an l). Because ϕ is not satisfied by all assignments to the variables v_1, v_2, v_3, it cannot be the case that $l = 1$. In the case where $l > 1$, for each $i \in [1, l-1]$, u_i is either $\mathsf{in}{:}j.\mathsf{get}$, $+\mathsf{in}{:}j.\mathsf{get}$ or $-\mathsf{in}{:}j.\mathsf{get}$ for some $j \in \{1,2,3\}$. This implies that, for each $i \in [0, l-1]$, there exists a basic Boolean formula ψ_i over the variables v_1, v_2, v_3 that is unique up to logical equivalence such that, for each $b_1, b_2, b_3 \in \mathbb{B}$, if the initial states of

the Boolean registers named $\mathsf{in}{:}1$, $\mathsf{in}{:}2$ and $\mathsf{in}{:}3$ are b_1, b_2 and b_3, respectively, then u_{i+1} will be executed iff $\psi_i(b_1, b_2, b_3)$. We have that $\psi_0 \Leftrightarrow \mathsf{T}$ and, for each $i \in [1, l-1]$, $\psi_i \Leftrightarrow (\psi_{i-1} \Rightarrow \mathsf{T})$ if $u_i \equiv \mathsf{in}{:}j.\mathsf{get}$, $\psi_i \Leftrightarrow (\psi_{i-1} \Rightarrow v_j)$ if $u_i \equiv +\mathsf{in}{:}j.\mathsf{get}$, and $\psi_i \Leftrightarrow (\psi_{i-1} \Rightarrow \neg\, v_j)$ if $u_i \equiv -\mathsf{in}{:}j.\mathsf{get}$. Hence, for each $i \in [0, l-1]$, $\psi_i \Rightarrow \phi$ implies $\mathsf{T} \Rightarrow \phi$ or $v_j \Rightarrow \phi$ or $\neg\, v_j \Rightarrow \phi$ for some $j \in \{1, 2, 3\}$. Because the latter three Boolean formulas are no tautologies, $\psi_i \Rightarrow \phi$ is no tautology either. This means that, for each $i \in [1, l-1]$, $\psi_i \Rightarrow \phi$ is not satisfied by all assignments to the variables v_1, v_2, v_3. Hence, X cannot exist. □

According to Proposition 2, the Boolean function induced by a CNF-formula can be computed, without using auxiliary Boolean registers, by an instruction sequence that contains no other jump instructions than #2 and whose length is linear in the size of the formula. If we permit arbitrary jump instructions, this result generalizes from CNF-formulas to arbitrary basic Boolean formulas, i.e. Boolean formulas in which no other connectives than $\neg$, $\vee$ and $\wedge$ occur.

Proposition 4 *For each basic Boolean formula ϕ, there exists an $X \in \mathrm{IS}^0_{\mathrm{br}}$ in which the basic instruction* $\mathsf{out.set{:}F}$ *does not occur such that X computes the Boolean function induced by ϕ and $\mathit{length}(X)$ is linear in the size of ϕ.*

Proof: Let $\mathit{inseq}_{\mathrm{bf}}$ be the function from the set of all basic Boolean formulas containing the variables $v_1, \ldots, v_n$ to $\mathrm{IS}^0_{\mathrm{br}}$ defined as follows:

$$\mathit{inseq}_{\mathrm{bf}}(\phi) = \mathit{inseq}'_{\mathrm{bf}}(\phi)\,;+\mathsf{out.set{:}T}\,;!\,,$$

where

$$\begin{aligned}
\mathit{inseq}'_{\mathrm{bf}}(v_k) &= +\mathsf{in}{:}k.\mathsf{get}\,,\\
\mathit{inseq}'_{\mathrm{bf}}(\neg\,\phi) &= \mathit{inseq}'_{\mathrm{bf}}(\phi)\,;\#2\,,\\
\mathit{inseq}'_{\mathrm{bf}}(\phi \vee \psi) &= \mathit{inseq}'_{\mathrm{bf}}(\phi)\,;\#\mathit{length}(\mathit{inseq}'_{\mathrm{bf}}(\psi))+1\,;\mathit{inseq}'_{\mathrm{bf}}(\psi)\,,\\
\mathit{inseq}'_{\mathrm{bf}}(\phi \wedge \psi) &= \mathit{inseq}'_{\mathrm{bf}}(\phi)\,;\#2\,;\#\mathit{length}(\mathit{inseq}'_{\mathrm{bf}}(\psi))+2\,;\mathit{inseq}'_{\mathrm{bf}}(\psi)\,.
\end{aligned}$$

Using the same facts about disjunctions and conjunctions as in the proof of Proposition 2, it is easy to prove by induction on the structure of ϕ that $\mathit{inseq}_{\mathrm{bf}}(\phi)$ computes the Boolean function induced by ϕ. Moreover, it is easy to see that $\mathit{length}(\mathit{inseq}_{\mathrm{bf}}(\phi))$ is linear in the size of ϕ. □

In the next proposition, we consider Boolean circuits instead of Boolean formulas.

Let C be a Boolean circuit with n input nodes and a single output node. Then the Boolean function *induced* by C is the n-ary Boolean function f defined by $f(b_1,\ldots,b_n) = C(b_1,\ldots,b_n)$, where $C(b_1,\ldots,b_n)$ denotes the output of C on input $(b_1,\ldots,b_n)$.

Because Boolean formulas can be looked upon as Boolean circuits with a single output node in which all gates have out-degree 1, the question arises whether Proposition 4 generalizes from Boolean formulas to Boolean circuits with a single output node. This question can be answered in the affirmative if we permit the use of auxiliary Boolean registers.

Proposition 5 *For each Boolean circuit C with a single output node that contains no other gates than $\neg$-gates, $\vee$-gates and $\wedge$-gates, there exists an $X \in \mathrm{IS}_{\mathrm{br}}$ in which the basic instruction* $\mathsf{out.set{:}F}$ *does not occur such that X computes the Boolean function induced by C and length(X) is linear in the size of C.*

Proof: Let $\mathit{inseq}_{\mathrm{bc}}$ be the function from the set of all Boolean circuits with input nodes $\mathit{in}_1,\ldots,\mathit{in}_n$, gates $g_1,\ldots,g_m$ and a single output node *out* to $\mathrm{IS}^0_{\mathrm{br}}$ defined as follows:

$$\mathit{inseq}_{\mathrm{bc}}(C) = \mathit{inseq}'_{\mathrm{bc}}(g_1)\,;\ldots;\,\mathit{inseq}'_{\mathrm{bc}}(g_m)\,;+\mathsf{aux{:}}m\mathsf{.get}\,;+\mathsf{out.set{:}T}\,;!\,,$$

where

$$\begin{aligned}
&\mathit{inseq}'_{\mathrm{bc}}(g_k) =\\
&\quad \mathit{inseq}''_{\mathrm{bc}}(p)\,;\#2\,;+\mathsf{aux{:}}k\mathsf{.set{:}T}\\
&\quad \text{if } g_k \text{ is a } \neg\text{-gate with direct preceding node } p\,,\\
&\mathit{inseq}'_{\mathrm{bc}}(g_k) =\\
&\quad \mathit{inseq}''_{\mathrm{bc}}(p)\,;\#2\,;\,\mathit{inseq}''_{\mathrm{bc}}(p')\,;+\mathsf{aux{:}}k\mathsf{.set{:}T}\\
&\quad \text{if } g_k \text{ is a } \vee\text{-gate with direct preceding nodes } p \text{ and } p'\,,\\
&\mathit{inseq}'_{\mathrm{bc}}(g_k) =\\
&\quad \mathit{inseq}''_{\mathrm{bc}}(p)\,;\#2\,;\#3\,;\,\mathit{inseq}''_{\mathrm{bc}}(p')\,;+\mathsf{aux{:}}k\mathsf{.set{:}T}\\
&\quad \text{if } g_k \text{ is a } \wedge\text{-gate with direct preceding nodes } p \text{ and } p'\,,
\end{aligned}$$

and

$$\begin{aligned}
\mathit{inseq}''_{\mathrm{bc}}(\mathit{in}_k) &= +\mathsf{in{:}}k\mathsf{.get}\,,\\
\mathit{inseq}''_{\mathrm{bc}}(g_k) &= +\mathsf{aux{:}}k\mathsf{.get}\,.
\end{aligned}$$

Using the same facts about disjunctions and conjunctions as in the proofs of Propositions 2 and 4, it is easy to prove by induction on the depth of C

that $inseq_{\mathrm{bc}}(C)$ computes the Boolean function induced by C if $g_1, \ldots, g_m$ is a topological sorting of the gates of C. Moreover, it is easy to see that $length(inseq_{\mathrm{bc}}(C))$ is linear in the size of C. □

$\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ includes Boolean function families that correspond to uncomputable functions from $\mathbb{B}^*$ to $\mathbb{B}$. Take an undecidable set $N \subseteq \mathbb{N}$ and consider the Boolean function family $\langle f_n \rangle_{n\in\mathbb{N}}$ with, for each $n \in \mathbb{N}$, $f_n : \mathbb{B}^n \to \mathbb{B}$ defined by

$$f_n(b_1, \ldots, b_n) = \mathsf{T} \text{ if } n \in N \ ,$$
$$f_n(b_1, \ldots, b_n) = \mathsf{F} \text{ if } n \notin N \ .$$

For each $n \in N$, f_n is computed by the instruction sequence $\mathsf{out.set{:}T}$; !. For each $n \notin N$, f_n is computed by the instruction sequence $\mathsf{out.set{:}F}$; !. The length of these instruction sequences is constant in n. Hence, $\langle f_n \rangle_{n\in\mathbb{N}}$ is in $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$. However, the corresponding function $f : \mathbb{B}^* \to \mathbb{B}$ is clearly uncomputable. This reminds of the fact that P/poly includes uncomputable functions from $\mathbb{B}^*$ to $\mathbb{B}$.

It happens that $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ and P/poly coincide, provided that we identify each Boolean function family $\langle f_n \rangle_{n\in\mathbb{N}}$ with the unique function $f : \mathbb{B}^* \to \mathbb{B}$ such that for each $n \in \mathbb{N}$, for each $w \in \mathbb{B}^n$, $f(w) = f_n(w)$.

Theorem 5 $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly} = \mathrm{P/poly}$.

Proof: We will prove the inclusion $\mathrm{P/poly} \subseteq \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ using the definition of P/poly in terms of Boolean circuits and we will prove the inclusion $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly} \subseteq \mathrm{P/poly}$ using the characterization of P/poly in terms of Turing machines that take advice (see e.g. Chapter 6 of [1]).

$\mathrm{P/poly} \subseteq \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$: Suppose that $\langle f_n \rangle_{n\in\mathbb{N}}$ in P/poly. Then, for all $n \in \mathbb{N}$, there exists a Boolean circuit C such that C computes f_n and the size of C is polynomial in n. For each $n \in \mathbb{N}$, let C_n be such a C. From Proposition 5 and the fact that linear in the size of C_n implies polynomial in n, it follows that each Boolean function family in P/poly is also in $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$.

$\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly} \subseteq \mathrm{P/poly}$: Suppose that $\langle f_n \rangle_{n\in\mathbb{N}}$ in $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$. Then, for all $n \in \mathbb{N}$, there exists an $X \in \mathrm{IS}_{\mathrm{br}}$ such that X computes f_n and $length(X)$ is polynomial in n. For each $n \in \mathbb{N}$, let X_n be such an X. Then f can be computed by a Turing machine that, on an input of size n, takes a binary description of X_n as advice and then just simulates the execution of X_n. It is easy to see that under the assumption that $X_n \in \mathrm{IS}_{\mathrm{br}}^{k-1,k-1}$, where $k = length(X_n)$, the size of the description of X_n and the number of steps

that it takes to simulate the execution of X_n are both polynomial in n. We can make this assumption without loss of generality (see the remarks immediately preceding Proposition 1). Hence, each Boolean function family in $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ is also in P/poly. □

It is unknown to us whether $\mathrm{IS}^k_{\mathrm{br}}\backslash\mathrm{poly}$ is different from $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ for all $k \in \mathbb{N}$ (see also Section 11).

7 The Non-uniform Super-polynomial Complexity Conjecture

In this section, we formulate a complexity conjecture which is a counterpart of the well-known complexity theoretic conjecture that NP $\not\subseteq$ P/poly in the current setting. The definitions of NP, NP-hardness, NP-completeness and related notions on which some results in this section and the coming ones are based are the ones from Chapter 2 of [1].

The counterpart of the conjecture that NP $\not\subseteq$ P/poly formulated in this section corresponds to the conjecture that 3SAT $\notin$ P/poly. By the NP-completeness of 3SAT, 3SAT $\notin$ P/poly is equivalent to NP $\not\subseteq$ P/poly. If the conjecture that NP $\not\subseteq$ P/poly is right, then the conjecture that NP $\neq$ P is right as well.

To formulate the conjecture, we need a Boolean function family $\langle 3\mathrm{SAT}'_n\rangle_{n\in\mathbb{N}}$ that corresponds to 3SAT. We obtain this Boolean function family by encoding 3CNF-formulas as sequences of Boolean values.

We write $H(k)$ for $\binom{2k}{1} + \binom{2k}{2} + \binom{2k}{3}$.[6] $H(k)$ is the number of combinations of at most 3 elements from a set with $2k$ elements. Notice that $H(k) = (4k^3 + 5k)/3$.

It is assumed that a countably infinite set $\{v_1, v_2, \ldots\}$ of propositional variables has been given. Moreover, it is assumed that a family of bijections

$$\langle \alpha_k : [1, H(k)] \to \{L \subseteq \{v_1, \neg\, v_1, \ldots, v_k, \neg\, v_k\} \mid 1 \leq \mathrm{card}(L) \leq 3\}\rangle_{k\in\mathbb{N}}$$

has been given that satisfies the following two conditions:

$$\forall i \in \mathbb{N} \bullet \forall j \in [1, H(i)] \bullet {\alpha_i}^{-1}(\alpha_{i+1}(j)) = j \ ,$$
$$\alpha \text{ is polynomial-time computable} \ ,$$

[6] As usual, we write $\binom{k}{l}$ for the number of l-element subsets of a k-element set.

where $\alpha : \mathbb{N}^+ \to \{L \subseteq \{v_1, \neg\, v_1, v_2, \neg\, v_2, \ldots\} \mid 1 \leq \mathrm{card}(L) \leq 3\}$ is defined by

$$\alpha(i) = \alpha_{\min\{j|i\in[1,H(j)]\}}(i) \ .$$

The function α is well-defined owing to the first condition on $\langle\alpha_k\rangle_{k\in\mathbb{N}}$. The second condition is satisfiable, but it is not satisfied by all $\langle\alpha_k\rangle_{k\in\mathbb{N}}$ satisfying the first condition.

The basic idea underlying the encoding of 3CNF-formulas as sequences of Boolean values is as follows:

- if $n = H(k)$ for some $k \in \mathbb{N}$, then the input of $3\mathrm{SAT}'_n$ consists of one Boolean value for each disjunction of at most three literals from the set $\{v_1, \neg\, v_1, \ldots, v_k, \neg\, v_k\}$;
- each Boolean value indicates whether the corresponding disjunction occurs in the encoded 3CNF-formula;
- if $H(k) < n < H(k+1)$ for some $k \in \mathbb{N}$, then only the first $H(k)$ Boolean values form part of the encoding.

For each $n \in \mathbb{N}$, $3\mathrm{SAT}'_n : \mathbb{B}^n \to \mathbb{B}$ is defined as follows:

- if $n = H(k)$ for some $k \in \mathbb{N}$:

$$3\mathrm{SAT}'_n(b_1, \ldots, b_n) = \mathsf{T} \quad \text{iff} \quad \bigwedge_{i\in[1,n] \text{ s.t. } b_i=\mathsf{T}} \bigvee \alpha_k(i) \ \text{ is satisfiable} \ ,$$

  where k is such that $n = H(k)$;

- if $H(k) < n < H(k+1)$ for some $k \in \mathbb{N}$:

$$3\mathrm{SAT}'_n(b_1, \ldots, b_n) = 3\mathrm{SAT}'_{H(k)}(b_1, \ldots, b_{H(k)}) \ ,$$

  where k is such that $H(k) < n < H(k+1)$.

Because $\langle\alpha_k\rangle_{k\in\mathbb{N}}$ satisfies the condition that ${\alpha_i}^{-1}(\alpha_{i+1}(j)) = j$ for all $i \in \mathbb{N}$ and $j \in [1, H(i)]$, we have for each $n \in \mathbb{N}$, for all $b_1, \ldots, b_n \in \mathbb{B}$:

$$3\mathrm{SAT}'_n(b_1, \ldots, b_n) = 3\mathrm{SAT}'_{n+1}(b_1, \ldots, b_n, \mathsf{F}) \ .$$

In other words, for each $n \in \mathbb{N}$, $3\mathrm{SAT}'_{n+1}$ can in essence handle all inputs that $3\mathrm{SAT}'_n$ can handle. We will come back to this phenomenon in Section 10.

$3\mathrm{SAT}'$ is meant to correspond to 3SAT. Therefore, the following theorem does not come as a surprise. We identify in this theorem the Boolean function family $3\mathrm{SAT}' = \langle 3\mathrm{SAT}'_n\rangle_{n\in\mathbb{N}}$ with the unique function $3\mathrm{SAT}' : \mathbb{B}^* \to \mathbb{B}$ such that for each $n \in \mathbb{N}$, for each $w \in \mathbb{B}^n$, $3\mathrm{SAT}'(w) = 3\mathrm{SAT}'_n(w)$.

Theorem 6 3SAT′ *is* NP*-complete.*

Proof: 3SAT′ is NP-complete iff 3SAT′ is in NP and 3SAT′ is NP-hard. Because 3SAT is NP-complete, it is sufficient to prove that 3SAT′ is polynomial-time Karp reducible to 3SAT and 3SAT is polynomial-time Karp reducible to 3SAT′, respectively. In the rest of the proof, α is defined as above.

3SAT′ is polynomial-time Karp reducible to 3SAT: Take the function f from $\mathbb{B}^*$ to the set of all 3CNF-formulas containing the variables $v_1, \ldots, v_k$ for some $k \in \mathbb{N}$ that is defined by $f(b_1, \ldots, b_n) = \bigwedge_{i \in [1, \max\{H(k) \mid H(k) \leq n\}] \text{ s.t. } b_i = \mathsf{T}} \bigvee \alpha(i)$. We have that $3\mathrm{SAT}'(b_1, \ldots, b_n) = 3\mathrm{SAT}(f(b_1, \ldots, b_n))$. It remains to show that f is polynomial-time computable. To compute $f(b_1, \ldots, b_n)$, α has to be computed for a number of times that is not greater than n and α is computable in time polynomial in n. Hence, f is polynomial-time computable.

3SAT is polynomial-time Karp reducible to 3SAT′: Take the unique function g from the set of all 3CNF-formulas containing the variables $v_1, \ldots, v_k$ for some $k \in \mathbb{N}$ to $\mathbb{B}^*$ such that for all 3CNF-formulas ϕ containing the variables $v_1, \ldots, v_k$ for some $k \in \mathbb{N}$, $f(g(\phi)) = \phi$ and there exists no $w \in \mathbb{B}^*$ shorter than $g(\phi)$ such that $f(w) = \phi$. We have that $3\mathrm{SAT}(\phi) = 3\mathrm{SAT}'(g(\phi))$. It remains to show that g is polynomial-time computable. Let l be the size of ϕ. To compute $g(\phi)$, α has to be computed for each clause a number of times that is not greater than $H(l)$ and α is computable in time polynomial in $H(l)$. Moreover, ϕ contains at most l clauses. Hence, g is polynomial-time computable. □

Before we turn to the non-uniform super-polynomial complexity conjecture, we touch lightly on the choice of the family of bijections in the definition of 3SAT′. It is easy to see that the choice is not essential. Let 3SAT″ be the same as 3SAT′, but based on another family of bijections, say $\langle \alpha'_n \rangle_{n \in \mathbb{N}}$, and let, for each $i \in \mathbb{N}$, for each $j \in [1, H(i)]$, $b'_j = b_{\alpha_i^{-1}(\alpha'_i(j))}$. Then:

- if $n = H(k)$ for some $k \in \mathbb{N}$:

$$3\mathrm{SAT}'_n(b_1, \ldots, b_n) = 3\mathrm{SAT}''_n(b'_1, \ldots, b'_n) \;;$$

- if $H(k) < n < H(k+1)$ for some $k \in \mathbb{N}$:

$$3\mathrm{SAT}'_n(b_1, \ldots, b_n) = 3\mathrm{SAT}''_n(b'_1, \ldots, b'_{H(k)}, b_{H(k)+1}, \ldots, b_n) \;,$$

  where k is such that $H(k) < n < H(k+1)$.

This means that the only effect of another family of bijections is another order of the relevant arguments.

The *non-uniform super-polynomial complexity conjecture* is the following conjecture:

Conjecture 1 $3\mathrm{SAT}' \notin \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$.

$3\mathrm{SAT}' \notin \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ expresses in short that there does not exist a polynomial function $h : \mathbb{N} \to \mathbb{N}$ such that for all $n \in \mathbb{N}$ there exists an $X \in \mathrm{IS}_{\mathrm{br}}$ such that X computes $3\mathrm{SAT}'_n$ and $\mathit{length}(X) \leq h(n)$. This corresponds with the following informal formulation of the non-uniform super-polynomial complexity conjecture:

> the lengths of the shortest instruction sequences that compute the Boolean functions $3\mathrm{SAT}'_n$ are not bounded by a polynomial in n.

The statement that Conjecture 1 is a counterpart of the conjecture that $3\mathrm{SAT} \notin \mathrm{P/poly}$ is made rigorous in the following theorem.

Theorem 7 $3\mathrm{SAT}' \notin \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ *iff* $3\mathrm{SAT} \notin \mathrm{P/poly}$.

Proof: This follows immediately from Theorems 5 and 6 and the fact that 3SAT is NP-complete. □

8 The Complexity Classes $IS\backslash\backslash F$

In this section, we introduce a kind of non-uniform complexity classes which includes a counterpart of the complexity class NP/poly in the setting of single-pass instruction sequences and show that this counterpart coincides with NP/poly. Some results in this section are based on the definition of NP in terms of P, which can for example be found in [1] (and which uses the idea of checking certificates), and the general definition of non-uniform complexity classes $\mathcal{C}/F$, which can for example be found in [3] (and which uses the idea of taking advice).

Let $IS \subseteq \mathrm{IS}_{\mathrm{br}}$ and let $F \subseteq \{h \mid h : \mathbb{N} \to \mathbb{N}\}$. Then $IS\backslash\backslash F$ is the class of all Boolean function families $\langle f_n \rangle_{n\in\mathbb{N}}$ that satisfy:

> there exist a monotonic $h \in F$ and a Boolean function family $\langle g_n \rangle_{n\in\mathbb{N}} \in IS\backslash F$ such that, for all $n \in \mathbb{N}$, for all $w \in \mathbb{B}^n$:
>
> $$f_n(w) = \mathsf{T} \Leftrightarrow \exists c \in \mathbb{B}^{h(n)} \bullet g_{n+h(n)}(w\,c) = \mathsf{T}\ .$$

If a $c \in \mathbb{B}^*$ and a $w \in \mathbb{B}^n$ for which $f_n(w) = \mathsf{T}$ satisfy $g_{n+h(n)}(w\,c) = \mathsf{T}$, then we call c a *certificate* for w.

In the sequel, the monotonicity requirement in the definition given above is only used to show that $\mathrm{IS}_{\mathrm{br}}\!\setminus\!\!\setminus\mathrm{poly}$ coincides with NP/poly (see Theorems 11 and 12).

For each $IS \subseteq \mathrm{IS}_{\mathrm{br}}$ and $F \subseteq \{h \mid h : \mathbb{N} \to \mathbb{N}\}$, the connection between the complexity classes $IS\backslash F$ and $IS\!\setminus\!\!\setminus F$ is like the connection between the complexity classes P and NP in the sense that it concerns the difference in complexity between finding a valid solution and checking whether a given solution is valid.

We are primarily interested in the complexity class $\mathrm{IS}_{\mathrm{br}}\!\setminus\!\!\setminus\mathrm{poly}$,[7] but we will also pay attention to other instantiations of the general definition just given.

We have that $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ is included in $\mathrm{IS}_{\mathrm{br}}\!\setminus\!\!\setminus\mathrm{poly}$.

Theorem 8 $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly} \subseteq \mathrm{IS}_{\mathrm{br}}\!\setminus\!\!\setminus\mathrm{poly}$.

Proof: Suppose that $\langle f_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$. Then, for all $n \in \mathbb{N}$, for all $w \in \mathbb{B}^n$:

$$f_n(w) = \mathsf{T} \Leftrightarrow \exists c \in \mathbb{B}^{h(n)} \bullet f_{n+h(n)}(w\,c) = \mathsf{T}$$

for the monotonic $h \in \mathrm{poly}$ defined by $h(n) = 0$ for all $n \in \mathbb{N}$, because the empty sequence can be taken as certificate for all w. □

Henceforth, we will use the notation $IS\!\setminus\!\!\setminus O(f(n))$ for the complexity class $IS\!\setminus\!\!\setminus \{h \mid h : \mathbb{N} \to \mathbb{N} \wedge h(n) = O(f(n))\}$. This notation is among other things used in the following corollary of the proof of Theorem 8.

Corollary 5 *For each* $k \in \mathbb{N}$, $\mathrm{IS}_{\mathrm{br}}\backslash O(n^k) \subseteq \mathrm{IS}_{\mathrm{br}}\!\setminus\!\!\setminus O(n^k)$.

Henceforth, we will use the notation $IS\!\setminus\!\!\setminus B(f(n))$ for the complexity class $IS\!\setminus\!\!\setminus \{h \mid h : \mathbb{N} \to \mathbb{N} \wedge \forall n \in \mathbb{N}^+ \bullet h(n) \leq f(n)\}$. This notation is among other things used in the following theorem about the complexity class $\mathrm{IS}_{\mathrm{br}}\!\setminus\!\!\setminus O(n^k)$.

Theorem 9 *For each* $k \in \mathbb{N}$, $\mathrm{IS}_{\mathrm{br}}\!\setminus\!\!\setminus O(n^k) \subseteq \bigcup_{a\in\mathbb{N}^+} \mathrm{IS}_{\mathrm{br}}\!\setminus\!\!\setminus B(a\,n^k)$.

Proof: Let $k \in \mathbb{N}$. It is a direct consequence of the definition of $\mathrm{IS}_{\mathrm{br}}\!\setminus\!\!\setminus O(n^k)$ that, for all Boolean function families $\langle f_n\rangle_{n\in\mathbb{N}}$, $\langle f_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS}_{\mathrm{br}}\!\setminus\!\!\setminus O(n^k)$ implies

[7] In precursors of this paper, the temporary name P^{**} is used for the complexity class $\mathrm{IS}_{\mathrm{br}}\!\setminus\!\!\setminus\mathrm{poly}$ (see e.g. [7]).

that there exists an $a \in \mathbb{N}^+$ such that $\langle f_n \rangle_{n\in\mathbb{N}} \in \mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus B(a\,n^k)$. Hence $\mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus O(n^k) \subseteq \bigcup_{a\in\mathbb{N}^+} \mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus B(a\,n^k)$. □

In the proof of the following hierarchy theorem for $\mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus\mathrm{poly}$, we will use the notation $IS\!\!\setminus\!\!\setminus_n F$ for $\{f : \mathbb{B}^n \to \mathbb{B} \mid \exists \langle g_n \rangle_{n\in\mathbb{N}} \in IS\!\!\setminus\!\!\setminus F \bullet f = g_n\}$.

Theorem 10 *For each $k \in \mathbb{N}$, $\mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus O(n^k) \subset \mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus O(n^{k^2+3})$.*

Proof: We will prove this theorem by defining a Boolean function family that by definition does not belong to $\mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus O(n^k)$ and showing that it does belong to $\mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus O(n^{k^2+3})$. The definition concerned makes use of a natural number m_a and an m_a-ary Boolean function g_a for each positive natural number a. These auxiliaries are defined first, using additional auxiliaries. Let $k \in \mathbb{N}$.

For each $a \in \mathbb{N}^+$, let the function $H_a : \mathbb{N} \to \mathbb{N}$ be defined by $H_a(n) = (3n+10a(n+a\,n^k)^k-2)^{a(n+a\,n^k)^k}$. By Proposition 1, the remarks immediately preceding Proposition 1 and the definition of $\mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus B(a\,n^k)$, we have that, for each $a \in \mathbb{N}^+$, the number of n-ary Boolean functions that belong to $\mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus_n B(a\,n^k)$ is not greater than $H_a(n)$ if $n > 0$. So $|\mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus_n B(a\,n^k)| \leq H_a(n)$ if $n > 0$. By simple arithmetical calculations we find that $H_a(n) < (12(a+1)^{k+1}n^{(k^2)})^{(a+1)^{k+1}n^{(k^2)}} < 2^{4(a+1)^{2(k+1)}n^{k^2+1}}$ if $n > 0$. Hence $H_a(n) < 2^{n^{k^2+2}}$ if $n \geq 4(a+1)^{2(k+1)}$. By simple arithmetical calculations we also find that $(4(a+1)^{2(k+1)})^{k^2+2} = 2^{(k^2+2)(2(k+1)\log(a+1)+\log(4))}$ and that $(k^2+2)(2(k+1)\log(a+1)+\log(4)) < 2(k^2+2)(k+1)(a+1) < 4(a+1)^{2(k+1)}$. Hence $n^{k^2+2} < 2^n$ if $n \geq 4(a+1)^{2(k+1)}$. For each $a \in \mathbb{N}^+$, let $m_a = 4(a+1)^{2(k+1)}$. We immediately have that, for all $a \in \mathbb{N}^+$, $m_a < m_{a+1}$ and, for all $n \geq m_a$, $H_a(n) < 2^{n^{k^2+2}}$ and $n^{k^2+2}+1 \leq 2^n$.

For each $a \in \mathbb{N}^+$, let $S_a \in (\mathbb{B}^{m_a})^{{m_a}^{k^2+2}+1}$ be such that the elements of S_a are mutually different, and let $\widehat{S_a}$ be the set of all elements of S_a. Because ${m_a}^{k^2+2}+1 \leq 2^{m_a}$, we have that $\widehat{S_a} \subseteq \mathbb{B}^{m_a}$. The number of functions from $\widehat{S_a}$ to $\mathbb{B}$ is $2^{{m_a}^{k^2+2}+1}$, and we have that $2^{{m_a}^{k^2+2}+1} > H_a(m_a) \geq |\mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus_{m_a} B(a\,{m_a}^k)|$. Because each function $f' : \widehat{S_a} \to \mathbb{B}$ can be extended to a function $f : \mathbb{B}^{m_a} \to \mathbb{B}$ by defining $f(w) = f'(w)$ if $w \in \widehat{S_a}$ and $f(w) = \mathsf{F}$ otherwise, this means that there exists an m_a-ary Boolean function that does not belong to $\mathrm{IS}_{\mathrm{br}}\!\!\setminus\!\!\setminus_{m_a} B(a\,{m_a}^k)$. For each $a \in \mathbb{N}^+$, let g_a be such an m_a-ary Boolean function.

Let $\langle g_n \rangle_{n\in\mathbb{N}}$ be the Boolean function family such that, for each $n \in \mathbb{N}$, g_n is defined as follows:

- if $n = m_a$ for some $a \in \mathbb{N}^+$: $g_n = g_a$, where a is such that $n = m_a$;
- if $n \neq m_a$ for all $a \in \mathbb{N}^+$: $g_n(w) = \mathsf{F}$ for all $w \in \mathbb{B}^n$.

For each $a \in \mathbb{N}^+$, there exists an $n \in \mathbb{N}^+$ such that g_n does not belong to $\mathrm{IS}_{\mathrm{br}} \sslash_n B(a\,n^k)$. Hence, by Theorem 9, $\langle g_n \rangle_{n \in \mathbb{N}} \notin \mathrm{IS}_{\mathrm{br}} \sslash O(n^k)$.

For each $n \in \mathbb{N}$, we can construct an instruction sequence X that computes g_n as follows:

- if $n = m_a$ for some $a \in \mathbb{N}^+$: $X = X^{w_1,g(w_1)};\ldots;X^{w_{n^{k^2+2}+1},g(w_{n^{k^2+2}+1})};!$, where for each $i \in [1, n^{k^2+2}+1]$, w_i is the ith element of S_a (where a is such that $n = m_a$), and $X^{w,b}$ is an instruction sequence that sets the output register to b if all input registers together contain w and jumps to the next instruction sequence otherwise;
- if $n \neq m_a$ for all $a \in \mathbb{N}^+$: $X = !$.

In the case where $n = m_a$ for some $a \in \mathbb{N}^+$, $\mathit{length}(X) \leq (n^{k^2+2}+1) \cdot (2n+1) + 1$. Otherwise, $\mathit{length}(X) = 1$. Hence, $\langle g_n \rangle_{n \in \mathbb{N}} \in \mathrm{IS}_{\mathrm{br}} \backslash O(n^{k^2+3})$. From this and Corollary 5, it follows that $\langle g_n \rangle_{n \in \mathbb{N}} \in \mathrm{IS}_{\mathrm{br}} \sslash O(n^{k^2+3})$. □

The approach followed in the proof of the hierarchy theorem for $\mathrm{IS}_{\mathrm{br}} \backslash \mathrm{poly}$ does not seem to work for the proof of the hierarchy theorem for $\mathrm{IS}_{\mathrm{br}} \sslash \mathrm{poly}$.

In the general definition of the complexity classes $IS \sslash F$, a pair of an input and a certificate is turned into a single sequence by simply concatenating the input and the certificate. In the usual definition of NP in terms of P, on the other hand, a pair of an input and a certificate is uniquely encoded by a single sequence from which the input and certificate are recoverable. A function that does so is commonly called a pairing function. In the case of $\mathrm{IS}_{\mathrm{br}} \sslash \mathrm{poly}$, a definition in which a pair of an input and a certificate is turned into a single sequence in the latter way could have been given as well.

Consider the pairing function from $\mathbb{B}^* \times \mathbb{B}^*$ to $\mathbb{B}^*$ that converts each two sequences $(b_1, \ldots, b_n)$ and $(b'_1, \ldots, b'_{n'})$ into $(b_1, b_1, \ldots, b_n, b_n, \mathsf{T}, \mathsf{F}, b'_1, \ldots, b'_{n'})$. Henceforth, we will write $w \cdot w'$, where $w, w' \in \mathbb{B}^*$, to denote the result of applying this pairing function to w and w'. Take, for each $m \in \mathbb{N}$, the function from $\bigcup_{i \geq m} \mathbb{B}^i$ to $\mathbb{B}^*$ that converts each sequence $(b_1, \ldots, b_m, b_{m+1}, \ldots, b_n)$ into $(b_1, \ldots, b_m) \cdot (b_{m+1}, \ldots, b_n)$, and the two projection functions from the range of the above-mentioned pairing function to $\mathbb{B}^*$ that extract from each sequence $(b_1, \ldots, b_m) \cdot (b_{m+1}, \ldots, b_n)$ the sequences $(b_1, \ldots, b_m)$ and $(b_{m+1}, \ldots, b_n)$. Then, for all $n \in \mathbb{N}$, the restrictions of these functions to

the sequences that belong to $\mathbb{B}^n$ are computable by an instruction sequence $X \in \mathrm{IS}^0_{\mathrm{br}}$ with $\mathit{length}(X) = O(n)$.

The following theorem gives an alternative characterization of $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$.

Theorem 11 *Let $\langle f_n\rangle_{n\in\mathbb{N}}$ be a Boolean function family. Then we have that $\langle f_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$ iff*

> *there exist an $h \in \mathrm{poly}$ and a Boolean function family $\langle g_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ such that, for all $n \in \mathbb{N}$, for all $w \in \mathbb{B}^n$:*
>
> $$f_n(w) = \mathsf{T} \Leftrightarrow \exists c \in \mathbb{B}^* \bullet (|c| \leq h(n) \wedge g_{|w\cdot c|}(w \cdot c) = \mathsf{T}) \,.$$

Proof: The implication from left to right follows directly from the definition of $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$ and the remark made above about the projection functions associated with the pairing function used here.

The implication from right to left is proved as follows. Let $\langle g_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ be such that there exists an $h \in \mathrm{poly}$ such that, for all $n \in \mathbb{N}$, for all $w \in \mathbb{B}^n$, $f_n(w) = \mathsf{T} \Leftrightarrow \exists c \in \mathbb{B}^* \bullet (|c| \leq h(n) \wedge g_{|w\cdot c|}(w \cdot c) = \mathsf{T})$. For each $w \in \mathbb{B}^*$, let c_w be a certificate for w. Suppose that $\langle \gamma_n\rangle_{n\in\mathbb{N}}$, with $\gamma_n : \mathbb{B}^n \to \mathbb{B}^*$ for every $n \in \mathbb{N}$, is an infinite sequence of injective functions satisfying: (i) there exists a monotonic $h \in \mathrm{poly}$ such that, for all $n \in \mathbb{N}$, for all $w \in \mathbb{B}^n$, $|\gamma_n(c_w)| = h(n)$, (ii) for all $n \in \mathbb{N}$, ${\gamma_n}^{-1}$ is computable by an instruction sequence $X \in \mathrm{IS}^0_{\mathrm{br}}$ with $\mathit{length}(X) = O(h(n))$. Then there exists a monotonic $h \in \mathrm{poly}$ such that, for all $n \in \mathbb{N}$, for all $w \in \mathbb{B}^n$, $\gamma_n(c_w) \in \mathbb{B}^{h(n)}$ and there exists a $\langle g'_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ such that, for all $n \in \mathbb{N}$, for all $w \in \mathbb{B}^n$, $g_{|w\cdot c|}(w \cdot c_w) = \mathsf{T} \Leftrightarrow g'_{n+h(n)}(w\,\gamma_n(c_w)) = \mathsf{T}$. The existence of a suitable $\langle g'_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ is not guaranteed for an $h \in \mathrm{poly}$ with the property that there exist $n, n' \in \mathbb{N}$ such that $n + h(n) = n' + h(n')$ and $n \neq n'$, but this property is excluded by the required monotonicity of h. It remains to show that the functions γ_n supposed above exist. For γ_n, we can pick the function that converts each sequence $(b_1, \ldots, b_n)$ into $(b_1, b'_1, \ldots, b_n, b'_n)$, where $b'_i = \mathsf{T}$ if $i \neq n$ and $b'_n = \mathsf{F}$, and adds at the end of the converted sequence sufficiently many F's to obtain results of the required length. □

It happens that $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$ and NP/poly coincide, provided that we identify each Boolean function family $\langle f_n\rangle_{n\in\mathbb{N}}$ with the unique function $f : \mathbb{B}^* \to \mathbb{B}$ such that for each $n \in \mathbb{N}$, for each $w \in \mathbb{B}^n$, $f(w) = f_n(w)$.

Theorem 12 $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly} = \mathrm{NP/poly}$.

Proof: It follows by elementary reasoning from the general definition of non-uniform complexity classes $\mathcal{C}/F$ and the definition of NP in terms of P that $f \in \mathrm{NP/poly}$ iff there exist a polynomial function $h : \mathbb{N} \to \mathbb{N}$ and a $g \in \mathrm{P/poly}$ such that, for all $w \in \mathbb{B}^*$:

$$f(w) = \mathsf{T} \Leftrightarrow \exists c \in \mathbb{B}^* \bullet (|c| \leq h(|w|) \wedge g(w \cdot c) = \mathsf{T})$$

(cf. Fact 2 in [25]). From this characterization of NP/poly and the characterization of $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$ given in Theorem 11, it follows easily that $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly} = \mathrm{NP/poly}$. □

In Section 7, we have conjectured that $3\mathrm{SAT}' \notin \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$. The question arises whether $3\mathrm{SAT}' \in \mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$. This question can be answered in the affirmative.

Theorem 13 $3\mathrm{SAT}' \in \mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$.

Proof: $3\mathrm{SAT}' \in \mathrm{NP}$ by Theorem 6, $\mathrm{NP} \subseteq \mathrm{NP/poly}$ by the general definition of non-uniform complexity classes $\mathcal{C}/F$ (see e.g. [3]), and $\mathrm{NP/poly} = \mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$ by Theorem 12. Hence, $3\mathrm{SAT}' \in \mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$. □

9 Completeness for the Complexity Classes $IS\backslash\backslash F$

In this section, we introduce the notion of $IS\backslash\backslash F$-completeness, a general notion of completeness for complexity classes $IS\backslash\backslash F$ where F is closed under function composition. Like NP-completeness, $IS\backslash\backslash F$-completeness will be defined in terms of a reducibility relation.

Let $IS \subseteq \mathrm{IS}_{\mathrm{br}}$, let $l, m, n \in \mathbb{N}$, and let $f{:}\mathbb{B}^n \to \mathbb{B}$ and $g{:}\mathbb{B}^m \to \mathbb{B}$. Then f is *l-length IS-reducible* to g, written $f \leq_l^{IS} g$, if there exist $h_1, \ldots, h_m{:}\mathbb{B}^n \to \mathbb{B}$ such that:

- there exist $X_1, \ldots, X_m \in IS$ such that $X_1, \ldots, X_m$ compute $h_1, \ldots, h_m$ and $\mathit{length}(X_1), \ldots, \mathit{length}(X_m) \leq l$;
- for all $b_1, \ldots, b_n \in \mathbb{B}$, $f(b_1, \ldots, b_n) = g(h_1(b_1, \ldots, b_n), \ldots, h_m(b_1, \ldots, b_n))$.

Let $IS \subseteq \mathrm{IS}_{\mathrm{br}}$, let $F \subseteq \{h \mid h : \mathbb{N} \to \mathbb{N}\}$ be such that F is closed under function composition, and let $\langle f_n \rangle_{n\in\mathbb{N}}$ and $\langle g_n \rangle_{n\in\mathbb{N}}$ be Boolean function families. Then $\langle f_n \rangle_{n\in\mathbb{N}}$ is *non-uniform F-length IS-reducible* to $\langle g_n \rangle_{n\in\mathbb{N}}$, written $\langle f_n \rangle_{n\in\mathbb{N}} \leq_F^{IS} \langle g_n \rangle_{n\in\mathbb{N}}$, if there exists an $h \in F$ such that:

- for all $n \in \mathbb{N}$, there exist $l, m \in \mathbb{N}$ with $l, m \leq h(n)$ such that $f_n \leq_l^{IS} g_m$.

Let $IS \subseteq \mathrm{IS}_{\mathrm{br}}$, let F be as above, and let $\langle f_n \rangle_{n\in\mathbb{N}}$ be a Boolean function family. Then $\langle f_n \rangle_{n\in\mathbb{N}}$ is *$IS\!\setminus\!\!\setminus F$-complete* if:

- $\langle f_n \rangle_{n\in\mathbb{N}} \in IS\!\setminus\!\!\setminus F$;
- for all $\langle g_n \rangle_{n\in\mathbb{N}} \in IS\!\setminus\!\!\setminus F$, $\langle g_n \rangle_{n\in\mathbb{N}} \leq_F^{IS} \langle f_n \rangle_{n\in\mathbb{N}}$.

The most important properties of non-uniform F-length IS-reducibility and $IS\!\setminus\!\!\setminus F$-completeness as defined above are stated in the following two propositions.

Proposition 6 *Let $IS \subseteq \mathrm{IS}_{\mathrm{br}}$, and let F be as above. Then:*

1. *if $\langle f_n \rangle_{n\in\mathbb{N}} \leq_F^{IS} \langle g_n \rangle_{n\in\mathbb{N}}$ and $\langle g_n \rangle_{n\in\mathbb{N}} \in IS\backslash F$, then $\langle f_n \rangle_{n\in\mathbb{N}} \in IS\backslash F$;*
2. *$\leq_F^{IS}$ is reflexive and transitive.*

Proof: Both properties follow immediately from the definition of $\leq_F^{IS}$. □

Proposition 7 *Let $IS \subseteq \mathrm{IS}_{\mathrm{br}}$, and let F be as above. Then:*

1. *if $\langle f_n \rangle_{n\in\mathbb{N}}$ is $IS\!\setminus\!\!\setminus F$-complete and $\langle f_n \rangle_{n\in\mathbb{N}} \in IS\backslash F$, then $IS\!\setminus\!\!\setminus F = IS\backslash F$;*
2. *if $\langle f_n \rangle_{n\in\mathbb{N}}$ is $IS\!\setminus\!\!\setminus F$-complete, $\langle g_n \rangle_{n\in\mathbb{N}} \in IS\!\setminus\!\!\setminus F$ and $\langle f_n \rangle_{n\in\mathbb{N}} \leq_F^{IS} \langle g_n \rangle_{n\in\mathbb{N}}$, then $\langle g_n \rangle_{n\in\mathbb{N}}$ is $IS\!\setminus\!\!\setminus F$-complete.*

Proof: The first property follows immediately from the definition of $IS\!\setminus\!\!\setminus F$-completeness, and the second property follows immediately from the definition of $IS\!\setminus\!\!\setminus F$-completeness and the transitivity of $\leq_F^{IS}$. □

The properties stated in Proposition 7 make $IS\!\setminus\!\!\setminus F$-completeness as defined above adequate for our purposes. In the following proposition, non-uniform polynomial-length $\mathrm{IS}_{\mathrm{br}}$-reducibility ($\leq_{\mathrm{poly}}^{\mathrm{IS}_{\mathrm{br}}}$) is related to polynomial-time Karp reducibility ($\leq_{\mathrm{poly}}^{\mathrm{Karp}}$).[8]

Proposition 8 *Let $\langle f_n \rangle_{n\in\mathbb{N}}$ and $\langle g_n \rangle_{n\in\mathbb{N}}$ be Boolean function families, and let f and g be the unique functions $f, g : \mathbb{B}^* \to \mathbb{B}$ such that for each $n \in \mathbb{N}$, for each $w \in \mathbb{B}^n$, $f(w) = f_n(w)$ and $g(w) = g_n(w)$. Then $f \leq_{\mathrm{poly}}^{\mathrm{Karp}} g$ only if $\langle f_n \rangle_{n\in\mathbb{N}} \leq_{\mathrm{poly}}^{\mathrm{IS}_{\mathrm{br}}} \langle g_n \rangle_{n\in\mathbb{N}}$.*

[8] For a definition of polynomial-time Karp reducibility, see e.g. Chapter 2 of [1].

Proof: This property follows immediately from the definitions of $\leq_{\mathrm{poly}}^{\mathrm{Karp}}$ and $\leq_{\mathrm{poly}}^{\mathrm{IS_{br}}}$, the fact that $\mathrm{P} \subseteq \mathrm{P/poly}$ (which follows directly from the general definition of non-uniform complexity classes $\mathcal{C}/F$), and Theorem 5. $\square$

The property stated in Proposition 8 allows for results concerning polynomial-time Karp reducibility to be reused when dealing with non-uniform polynomial-length $\mathrm{IS_{br}}$-reducibility.

We would like to call $\mathrm{IS_{br}}\backslash\backslash\mathrm{poly}$-completeness the counterpart of NP/poly-completeness in the current setting, but the notion of NP/poly-completeness looks to be absent in the literature on complexity theory. The closest to NP/poly-completeness that we could find is p-completeness for pD, a notion introduced in [22].

Because $3\mathrm{SAT}'$ is closely related to 3SAT and $3\mathrm{SAT}' \in \mathrm{IS_{br}}\backslash\backslash\mathrm{poly}$, we expect $3\mathrm{SAT}'$ to be $\mathrm{IS_{br}}\backslash\backslash\mathrm{poly}$-complete.

Theorem 14 $3\mathrm{SAT}'$ *is* $\mathrm{IS_{br}}\backslash\backslash\mathrm{poly}$*-complete.*

Proof: By Theorem 13, we have that $3\mathrm{SAT}' \in \mathrm{IS_{br}}\backslash\backslash\mathrm{poly}$. It remains to prove that for all $\langle f_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS_{br}}\backslash\backslash\mathrm{poly}$, $\langle f_n\rangle_{n\in\mathbb{N}} \leq_{\mathrm{poly}}^{\mathrm{IS_{br}}} 3\mathrm{SAT}'$.

Suppose that $\langle f_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS_{br}}\backslash\backslash\mathrm{poly}$. Let $\langle g_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS_{br}}\backslash\mathrm{poly}$ be such that there exists a monotonic $h \in \mathrm{poly}$ such that, for each $n \in \mathbb{N}$, for each $w \in \mathbb{B}^n$, $f_n(w) = \mathsf{T} \Leftrightarrow \exists c \in \mathbb{B}^{h(n)} \bullet g_{n+h(n)}(w\,c) = \mathsf{T}$. Such a $\langle g_n\rangle_{n\in\mathbb{N}}$ exists by the definition of $\mathrm{IS_{br}}\backslash\backslash\mathrm{poly}$. Let $h \in \mathrm{poly}$ be such that, for each $n \in \mathbb{N}$, for each $w \in \mathbb{B}^n$, $f_n(w) = \mathsf{T} \Leftrightarrow \exists c \in \mathbb{B}^{h(n)} \bullet g_{n+h(n)}(w\,c) = \mathsf{T}$. Let $n \in \mathbb{N}$, and let $m = h(n)$. Let $X \in \mathrm{IS_{br}}$ be such that X computes g_{n+m} and $\mathit{length}(X)$ is polynomial in $n + m$.

Assume that $\mathsf{out.set{:}T}$ occurs only once in X, that $\#l$ does not occur in X at positions where there is no lth next primitive instruction, and that test instructions do not occur in X at the last but one or last position. These assumptions can be made without loss of generality: by Theorem 2 we may assume without loss of generality that $\mathsf{out.set{:}F}$ does not occur in X and therefore multiple occurrences of $\mathsf{out.set{:}T}$ can always be eliminated by replacing them with the exception of the last one by jump instructions, occurrences of $\#l$ at positions where there is no lth next primitive instruction can always be eliminated by replacing them by $\#0$, and occurrences of test instructions at the last but one or last position can be eliminated by adding once or twice $\#0$ at the end. Suppose that $X = u_1\,;\ldots;u_k$, and let $l \in [1,k]$ be such that u_l is either $\mathsf{out.set{:}T}$, $+\mathsf{out.set{:}T}$ or $-\mathsf{out.set{:}T}$.

First of all, we look for a transformation that gives, for each $b_1,\ldots,b_n \in \mathbb{B}$, a Boolean formula $\phi_{b_1,\ldots,b_n}$ such that $f_n(b_1,\ldots,b_n) = \mathsf{T}$ iff $\phi_{b_1,\ldots,b_n}$ is

satisfiable. We have that $f_n(b_1,\ldots,b_n) = \mathsf{T}$ iff there exist initial states of the Boolean registers named $\mathsf{in}{:}n{+}1,\ldots,\mathsf{in}{:}n{+}m$ for which there exists an execution path through X that reaches u_l in case the initial states of the Boolean registers named $\mathsf{in}{:}1,\ldots,\mathsf{in}{:}n$ are $b_1,\ldots,b_n$, respectively. This brings up the formula $\phi_{b_1,\ldots,b_n}$ given below. In this formula, propositional variables $r_1,\ldots,r_{n+m}$ and $v_1,\ldots,v_k$ are used. The truth value assigned to r_i ($i \in [1,n+m]$) is intended to indicate whether the content of the input register named $\mathsf{in}{:}i$ is T and the truth value assigned to v_j ($j \in [1,k]$) is intended to indicate whether the primitive instruction u_i is executed.

For each $b_1,\ldots,b_n \in \mathbb{B}$, let $\phi_{b_1,\ldots,b_n}$ be $\bigwedge_{i\in[1,n]} \chi_i \wedge v_1 \wedge v_l \wedge \bigwedge_{j\in[1,k]} \psi_j$, where: for each $i \in [1,n]$, χ_i is

- r_i if $b_i = \mathsf{T}$;
- $\neg\, r_i$ if $b_i = \mathsf{F}$;

for each $j \in [1,k]$, ψ_j is

- $v_j \;\Rightarrow\; v_{j+1}$ if $u_j \equiv a$;
- $v_j \;\Rightarrow\; v_{j+1}$ if $u_j \equiv +f.\mathsf{set}{:}\mathsf{T}$ or $u_j \equiv -f.\mathsf{set}{:}\mathsf{F}$;
- $v_j \;\Rightarrow\; \neg\, v_{j+1} \wedge v_{j+2}$ if $u_j \equiv +f.\mathsf{set}{:}\mathsf{F}$ or $u_j \equiv -f.\mathsf{set}{:}\mathsf{T}$;
- $(v_j \wedge r_i \;\Rightarrow\; v_{j+1}) \wedge (v_j \wedge \neg\, r_i \;\Rightarrow\; \neg\, v_{j+1} \wedge v_{j+2})$ if $u_j \equiv +\mathsf{in}{:}i.\mathsf{get}$;
- $(v_j \wedge \neg\, r_i \;\Rightarrow\; v_{j+1}) \wedge (v_j \wedge r_i \;\Rightarrow\; \neg\, v_{j+1} \wedge v_{j+2})$ if $u_j \equiv -\mathsf{in}{:}i.\mathsf{get}$;
- $(v_j \wedge \bigvee_{j'\in B^{i,\mathsf{T}}_{1,j-1}} (v_{j'} \wedge \bigwedge_{j''\in B^{i,\mathsf{F}}_{j'+1,j-1}} \neg\, v_{j''}) \;\Rightarrow\; v_{j+1}) \wedge$
  $(v_j \wedge \bigvee_{j'\in B^{i,\mathsf{F}}_{1,j-1}} (v_{j'} \wedge \bigwedge_{j''\in B^{i,\mathsf{T}}_{j'+1,j-1}} \neg\, v_{j''}) \;\Rightarrow\; \neg\, v_{j+1} \wedge v_{j+2})$
  if $u_j \equiv +\mathsf{aux}{:}i.\mathsf{get}$;
- $(v_j \wedge \bigvee_{j'\in B^{i,\mathsf{F}}_{1,j-1}} (v_{j'} \wedge \bigwedge_{j''\in B^{i,\mathsf{T}}_{j'+1,j-1}} \neg\, v_{j''}) \;\Rightarrow\; v_{j+1}) \wedge$
  $(v_j \wedge \bigvee_{j'\in B^{i,\mathsf{T}}_{1,j-1}} (v_{j'} \wedge \bigwedge_{j''\in B^{i,\mathsf{F}}_{j'+1,j-1}} \neg\, v_{j''}) \;\Rightarrow\; \neg\, v_{j+1} \wedge v_{j+2})$
  if $u_j \equiv -\mathsf{aux}{:}i.\mathsf{get}$;
- $\neg\, v_j$ if $u_j \equiv \#0$;
- $v_j \;\Rightarrow\; \bigwedge_{j'\in[j+1,j+l-1]} \neg\, v_{j'} \wedge v_{j+l}$ if $u_j \equiv \#l$ and $1 \leq l \leq k-j$;
- v_j if $u_j \equiv !$;

where $B_{j,j'}^{i,b}$ is the set of all $j'' \in [j,j']$ for which $u_{j''}$ is either $\mathsf{aux}{:}i.\mathsf{set}{:}b$, $+\mathsf{aux}{:}i.\mathsf{set}{:}b$ or $-\mathsf{aux}{:}i.\mathsf{set}{:}b$.

If there exist initial states of the Boolean registers named $\mathsf{in}{:}n{+}1$, $\ldots$, $\mathsf{in}{:}n{+}m$ for which there exists an execution path through X that reaches u_l in case the initial states of the Boolean registers named $\mathsf{in}{:}1, \ldots, \mathsf{in}{:}n$ are $b_1, \ldots, b_n$, respectively, then $\phi_{b_1,\ldots,b_n}$ is satisfiable by assigning truth values to the variables according to the intention mentioned above. On the other hand, if $\phi_{b_1,\ldots,b_n}$ is satisfiable, then a satisfying assignment indicates for which initial states of the Boolean registers named $\mathsf{in}{:}n{+}1, \ldots, \mathsf{in}{:}n{+}m$ there exists an execution path through X that reaches u_l and which instructions are on this execution path. Thus $f_n(b_1, \ldots, b_n) = \mathsf{T}$ iff $\phi_{b_1,\ldots,b_n}$ is satisfiable.

For some $l \in \mathbb{N}$, $\phi_{b_1,\ldots,b_n}$ still has to be transformed into a $w_{b_1,\ldots,b_n} \in \mathbb{B}^l$ such that $\phi_{b_1,\ldots,b_n}$ is satisfiable iff $\mathrm{3SAT}'_l(w_{b_1,\ldots,b_n}) = \mathsf{T}$. We look upon this transformation as a composition of two transformations: first $\phi_{b_1,\ldots,b_n}$ is transformed into a 3CNF-formula $\psi_{b_1,\ldots,b_n}$ such that $\phi_{b_1,\ldots,b_n}$ is satisfiable iff $\psi_{b_1,\ldots,b_n}$ is satisfiable, and next, for some $l \in \mathbb{N}$, $\psi_{b_1,\ldots,b_n}$ is transformed into a $w_{b_1,\ldots,b_n} \in \mathbb{B}^l$ such that $\psi_{b_1,\ldots,b_n}$ is satisfiable iff $\mathrm{3SAT}'_l(w_{b_1,\ldots,b_n}) = \mathsf{T}$.

It is easy to see that the size of $\phi_{b_1,\ldots,b_n}$ is polynomial in n and that $(b_1, \ldots, b_n)$ can be transformed into $\phi_{b_1,\ldots,b_n}$ in time polynomial in n. It is well-known that each Boolean formula ψ can be transformed in time polynomial in the size of ψ into a 3CNF-formula ψ', with size and number of variables linear in the size of ψ, such that ψ is satisfiable iff ψ' is satisfiable (see e.g. Theorem 3.7 in [3]). Moreover, it is known from the proof of Theorem 6 that every 3CNF-formula ϕ can be transformed in time polynomial in the size of ϕ into a $w \in \mathbb{B}^{H(k')}$, where k' is the number of variables in ϕ, such that $\mathrm{3SAT}(\phi) = \mathrm{3SAT}'(w)$. From these facts, and Proposition 8, it follows easily that $\langle f_n \rangle_{n \in \mathbb{N}}$ is non-uniform polynomial-length $\mathrm{IS}_{\mathrm{br}}$-reducible to $\mathrm{3SAT}'$. $\square$

The proof of Theorem 14 has been partly inspired by the proof of the NP-completeness of SAT in [14].

A known result about classical complexity classes turns out to be a corollary of Theorems 5, 6, 12 and 14.

Corollary 6 NP $\not\subseteq$ P/poly *iff* NP/poly $\not\subseteq$ P/poly.

10 Projective Boolean Function Families

In Section 7, we have noticed that, for each $n \in \mathbb{N}$, $3\mathrm{SAT}'_{n+1}$ can in essence handle all inputs that $3\mathrm{SAT}'_n$ can handle because we have $3\mathrm{SAT}'_n(b_1,\ldots,b_n) = 3\mathrm{SAT}'_{n+1}(b_1,\ldots,b_n,\mathsf{F})$. In this section, we come back to this phenomenon.

For each $m,n \in \mathbb{N}$ such that $m \geq n$, we define a *projection* function $\pi^m_n : (\mathbb{B}^m \to \mathbb{B}) \to (\mathbb{B}^n \to \mathbb{B})$ as follows:

$$\pi^m_n(f)(b_1,\ldots,b_n) = f(b_1,\ldots,b_n,\overbrace{\mathsf{F},\ldots,\mathsf{F}}^{m-n\ \times})$$

for all $f : \mathbb{B}^m \to \mathbb{B}$ and $b_1,\ldots,b_n \in \mathbb{B}$.

A *projective Boolean function family* is a Boolean function family $\langle f_n\rangle_{n\in\mathbb{N}}$ such that $f_n = \pi^{n+1}_n(f_{n+1})$ for all $n \in \mathbb{N}$.

This means that in the case where a Boolean function family $\langle f_n\rangle_{n\in\mathbb{N}}$ is projective, for each $m,n \in \mathbb{N}$ with $m > n$, f_m can in essence handle all inputs that f_n can handle. For that reason, complexity classes that are restricted to projective Boolean function families are potentially interesting.

Let $IS \subseteq \mathrm{IS}_{\mathrm{br}}$ and $F \subseteq \{h \mid h : \mathbb{N} \to \mathbb{N}\}$. Then $IS\backslash_\pi F$ is the class of all projective Boolean function families $\langle f_n\rangle_{n\in\mathbb{N}}$ that satisfy:

> there exists an $h \in F$ such that for all $n \in \mathbb{N}$ there exists an $X \in IS$ such that X computes f_n and $\mathit{length}(X) \leq h(n)$.

Let $IS \subseteq \mathrm{IS}_{\mathrm{br}}$ and let $F \subseteq \{h \mid h : \mathbb{N} \to \mathbb{N}\}$. Then $IS\backslash\!\backslash_\pi F$ is the class of all projective Boolean function families $\langle f_n\rangle_{n\in\mathbb{N}}$ that satisfy:

> there exist a monotonic $h \in F$ and a Boolean function family $\langle g_n\rangle_{n\in\mathbb{N}} \in IS\backslash_\pi F$ such that, for all $n \in \mathbb{N}$, for all $w \in \mathbb{B}^n$:
>
> $$f_n(w) = \mathsf{T} \Leftrightarrow \exists c \in \mathbb{B}^{h(n)} \bullet g_{n+h(n)}(w\,c) = \mathsf{T}\ .$$

It follows immediately from the definitions concerned that $IS\backslash_\pi F$ and $IS\backslash\!\backslash_\pi F$ are subsets of $IS\backslash F$ and $IS\backslash\!\backslash F$, respectively. $\mathrm{IS}_{\mathrm{br}}\backslash_\pi\mathrm{poly}$ is a proper subset of $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$. This is easy to see. In Section 5, we gave an example of a Boolean function family corresponding to an uncomputable function from $\mathbb{B}^*$ to $\mathbb{B}$ that belongs to $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$. The Boolean function family concerned is not a projective Boolean function family, and consequently does not belong to $\mathrm{IS}_{\mathrm{br}}\backslash_\pi\mathrm{poly}$.

The question arises whether the restriction to projective Boolean function families is a severe restriction. We have that every Boolean function

family is non-uniform linear-length $\mathrm{IS}_{\mathrm{br}}$-reducible to a projective Boolean function family.

Below, we will write lin for the set $\{h \mid h : \mathbb{N} \to \mathbb{N} \wedge h \,\mathrm{is\,linear}\}$.

Theorem 15 *Let $\langle f_n\rangle_{n\in\mathbb{N}}$ be a Boolean function family. Then there exists a projective Boolean function family $\langle g_n\rangle_{n\in\mathbb{N}}$ such that $\langle f_n\rangle_{n\in\mathbb{N}} \leq^{\mathrm{IS}_{\mathrm{br}}}_{\mathrm{lin}} \langle g_n\rangle_{n\in\mathbb{N}}$.*

Proof: The idea is to convert inputs $(b_1,\ldots,b_n)$ into $(b_1,b'_1,\ldots,b_n,b'_n)$, where $b'_i = \mathsf{T}$ if $i \neq n$ and $b'_n = \mathsf{F}$, because the converted inputs can be recovered after additions at the end. This conversion has been used before to prove Theorem 11.

For each $n \in \mathbb{N}$, for each $i \in [1,n]$, let $h^n_{2i-1} : \mathbb{B}^n \to \mathbb{B}$ be defined by $h^n_{2i-1}(b_1,\ldots,b_n) = b_i$ and let $h^n_{2i} : \mathbb{B}^n \to \mathbb{B}$ be defined by $h^n_{2i}(b_1,\ldots,b_n) = \mathsf{T}$ if $i \neq n$ and $h^n_{2i}(b_1,\ldots,b_n) = \mathsf{F}$ if $i = n$. Clearly, these functions can be computed by instruction sequences from $\mathrm{IS}_{\mathrm{br}}$ whose lengths are linear in n. Therefore, we are done with the proof if we show that there exists a projective Boolean function family $\langle g_n\rangle_{n\in\mathbb{N}}$ such that for all $n \in \mathbb{N}$:

$$f_n(b_1,\ldots,b_n) = g_{2n}(h^n_1(b_1,\ldots,b_n),\ldots,h^n_{2n}(b_1,\ldots,b_n)) \;.$$

A witness is the projective Boolean function family $\langle g_n\rangle_{n\in\mathbb{N}}$ with, for each $n \in \mathbb{N}$, $g_{2n} : \mathbb{B}^{2n} \to \mathbb{B}$ defined by $g_{2n}(b_1,b'_1,\ldots,b_n,b'_n) = f_m(b_1,\ldots,b_m)$, where m is the unique $j \in [1,n]$ such that $b'_i = \mathsf{T}$ for all $i \in [1,j-1]$ and $b'_j = \mathsf{F}$ if such an m exists and $g_{2n}(b_1,b'_1,\ldots,b_n,b'_n) = \mathsf{F}$ otherwise; and $g_{2n+1} : \mathbb{B}^{2n+1} \to \mathbb{B}$ defined by $g_{2n+1}(b_1,b'_1,\ldots,b_n,b'_n,b) = g_{2n}(b_1,b'_1,\ldots,b_n,b'_n)$. □

The following result is a corollary of Theorem 15 and the definitions of $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ and $\mathrm{IS}_{\mathrm{br}}\backslash_\pi\mathrm{poly}$.

Corollary 7 *Let $\langle f_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$. Then there exists a $\langle g_n\rangle_{n\in\mathbb{N}} \in \mathrm{IS}_{\mathrm{br}}\backslash_\pi\mathrm{poly}$ such that $\langle f_n\rangle_{n\in\mathbb{N}} \leq^{\mathrm{IS}_{\mathrm{br}}}_{\mathrm{lin}} \langle g_n\rangle_{n\in\mathbb{N}}$.*

11 Concluding Remarks

We have presented an approach to non-uniform complexity which is based on the simple idea that each Boolean function can be computed by a single-pass instruction sequence that contains only instructions to read and write the contents of Boolean registers, forward jump instructions, and a termination instruction.

We have answered various questions that arise from this approach, but many open questions remain. We mention:

- We do not know whether Theorem 10 can be sharpened. In particular, it is an open question whether, for each $k \in \mathbb{N}$, $\mathrm{IS}_{\mathrm{br}} \mathbin{\backslash\!\backslash} O(n^k) \subset \mathrm{IS}_{\mathrm{br}} \mathbin{\backslash\!\backslash} O(n^{k+1})$.
- We know little about complexity classes $IS \backslash F$ where $IS \subset \mathrm{IS}_{\mathrm{br}}$. In particular, it is an open question whether:
  - $\mathrm{IS}^0_{\mathrm{br}} \backslash \mathrm{poly} \subset \mathrm{IS}_{\mathrm{br}} \backslash \mathrm{poly}$;
  - for each $l \in \mathbb{N}$, $\mathrm{IS}^{0,l}_{\mathrm{br}} \backslash \mathrm{poly} \subset \mathrm{IS}^0_{\mathrm{br}} \backslash \mathrm{poly}$;
  - for each $k \in \mathbb{N}$, $\mathrm{IS}^0_{\mathrm{br}} \backslash O(n^k) \subset \mathrm{IS}_{\mathrm{br}} \backslash O(n^k)$;
  - for each $k, l \in \mathbb{N}$, $\mathrm{IS}^{0,l}_{\mathrm{br}} \backslash O(n^k) \subset \mathrm{IS}^0_{\mathrm{br}} \backslash O(n^k)$.
- Likewise, we know little about complexity classes $IS \mathbin{\backslash\!\backslash} F$ where $IS \subset \mathrm{IS}_{\mathrm{br}}$. It is also an open question whether:
  - $\mathrm{IS}^0_{\mathrm{br}} \mathbin{\backslash\!\backslash} \mathrm{poly} \subset \mathrm{IS}_{\mathrm{br}} \mathbin{\backslash\!\backslash} \mathrm{poly}$;
  - for each $l \in \mathbb{N}$, $\mathrm{IS}^{0,l}_{\mathrm{br}} \mathbin{\backslash\!\backslash} \mathrm{poly} \subset \mathrm{IS}^0_{\mathrm{br}} \mathbin{\backslash\!\backslash} \mathrm{poly}$;
  - for each $k \in \mathbb{N}$, $\mathrm{IS}^0_{\mathrm{br}} \mathbin{\backslash\!\backslash} O(n^k) \subset \mathrm{IS}_{\mathrm{br}} \mathbin{\backslash\!\backslash} O(n^k)$;
  - for each $k, l \in \mathbb{N}$, $\mathrm{IS}^{0,l}_{\mathrm{br}} \mathbin{\backslash\!\backslash} O(n^k) \subset \mathrm{IS}^0_{\mathrm{br}} \mathbin{\backslash\!\backslash} O(n^k)$.
- We also know little about the connections between complexity classes $IS \backslash F$ and $IS \mathbin{\backslash\!\backslash} F$ with $IS \subset \mathrm{IS}_{\mathrm{br}}$ and classical complexity classes. In particular, it is an open question whether there are classical complexity classes that coincide with the complexity classes $\mathrm{IS}^{0,l}_{\mathrm{br}} \backslash \mathrm{poly}$, $\mathrm{IS}^0_{\mathrm{br}} \mathbin{\backslash\!\backslash} \mathrm{poly}$, and $\mathrm{IS}^{0,l}_{\mathrm{br}} \mathbin{\backslash\!\backslash} \mathrm{poly}$.

There are not yet indications that the above-mentioned open questions concerning proper inclusions of complexity classes $IS \backslash F$ and $IS \mathbin{\backslash\!\backslash} F$ with $IS \subset \mathrm{IS}_{\mathrm{br}}$ are interdependent.

It is easy to see that $\mathrm{IS}^0_{\mathrm{br}} \backslash \mathrm{poly}$ coincides with the classical complexity class L/poly. It is well-known that, for all $f : \mathbb{B}^* \to \mathbb{B}$, $f \in \mathrm{L/poly}$ iff f has polynomial-size branching programs (see e.g. Theorem 4.53 in [23]).[9] Let $\langle f_n \rangle_{n \in \mathbb{N}} \in \mathrm{IS}^0_{\mathrm{br}} \backslash \mathrm{poly}$. Then, for all $n \in \mathbb{N}$, the thread produced by the instruction sequence that computes f_n is in essence a branching program and its size is polynomially bounded in n. As a consequence of this, $\mathrm{IS}^0_{\mathrm{br}} \backslash \mathrm{poly}$ coincides with L/poly.

[9] L is the class of all $f : \mathbb{B}^* \to \mathbb{B}$ that are logarithmic-space computable, see e.g. Chapter 4 of [1].

The approaches to computational complexity based on loop programs [19], straight-line programs [16], and branching programs [13] appear to be the closest related to the approach followed in this paper.

The notion of loop program is far from abstract or general: a loop program consists of assignment statements and possibly nested loop statements of a special kind. Loop programs are nevertheless closer to instruction sequences than Turing machines or Boolean circuits. After a long period of little interest, there is currently a revival of interest in the approach to issues relating to non-uniform computational complexity based on loop programs (see e.g. [4, 18, 20]). The notion of loop program used in recent work on computational complexity is usually more general than the one originally used.

The notion of straight-line program is relatively close to the notion of single-pass instruction sequence: a straight-line program is a sequence of steps, where in each step a language is generated by selecting an element from an alphabet or by taking the union, intersection or concatenation of languages generated in previous steps. In other words, straight-line programs can be looked upon as single-pass instruction sequences with special basic instructions, and without test and jump instructions. To our knowledge, the notion of straight-line program is only used in the work presented in [2, 16].

The notion of branching program is actually a generalization of the notion of decision tree from trees to graphs, so the term branching program seems rather far-fetched. However, branching programs are in essence threads, i.e. the objects that we use to represent the behaviours produced by instruction sequences under execution. Branching programs are related to non-uniform space complexity like Boolean circuits are related to non-uniform time complexity. Like the notion of Boolean circuit, the notion of branching program looks to be lasting in complexity theory (see e.g. [23, 24]).

The complexity class $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$ can alternatively be defined in the same style as $\mathrm{IS}_{\mathrm{br}}\backslash\mathrm{poly}$ in a setting that allows instruction sequence splitting. In [7], we introduce an extension of PGA that allows single-pass instruction sequence splitting and an extension of BTA with a behavioural counterpart of instruction sequence splitting that is reminiscent of thread forking, and define $\mathrm{IS}_{\mathrm{br}}\backslash\backslash\mathrm{poly}$ in this alternative way.

References

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge, 2009.

[2] J. L. Balcázar, J. Díaz, and J. Gabarró. Uniform characterizations of non-uniform complexity measures. *Information and Control*, 67(1):53–69, 1985. doi:10.1016/S0019-9958(85)80026-7.

[3] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1988.

[4] A. M. Ben-Amram, N. D. Jones, and L. Kristiansen. Linear, polynomial or exponential? Complexity inference in polynomial time. In A. Beckmann, C. Dimitracopoulos, and B. Löwe, editors, *CiE 2008*, volume 5028 of *Lecture Notes in Computer Science*, pages 67–76. Springer-Verlag, 2008. doi:10.1007/978-3-540-69407-6_7.

[5] J. A. Bergstra and M. E. Loots. Program algebra for sequential code. *Journal of Logic and Algebraic Programming*, 51(2):125–156, 2002. doi:10.1016/S1567-8326(02)00018-8.

[6] J. A. Bergstra and C. A. Middelburg. Distributed strategic interleaving with load balancing. *Future Generation Computer Systems*, 24(6):530–548, 2008. doi:10.1016/j.future.2007.08.001.

[7] J. A. Bergstra and C. A. Middelburg. Instruction sequences and non-uniform complexity theory. arXiv:0809.0352v3 [cs.CC], July 2010.

[8] J. A. Bergstra and C. A. Middelburg. Indirect jumps improve instruction sequence performance. *Scientific Annals of Computer Science*, 22(2):253–265, 2012. doi:10.7561/SACS.2012.2.253.

[9] J. A. Bergstra and C. A. Middelburg. Instruction sequence processing operators. *Acta Informatica*, 49(3):139–172, 2012. doi:10.1007/s00236-012-0154-2.

[10] J. A. Bergstra and C. A. Middelburg. *Instruction Sequences for Computer Science*, volume 2 of *Atlantis Studies in Computing*. Atlantis Press, Amsterdam, 2012.

[11] J. A. Bergstra and C. A. Middelburg. On the expressiveness of single-pass instruction sequences. *Theory of Computing Systems*, 50(2):313–328, 2012. doi:10.1007/s00224-010-9301-8.

[12] J. A. Bergstra and A. Ponse. An instruction sequence semigroup with involutive anti-automorphisms. *Scientific Annals of Computer Science*, 19:57–92, 2009.

[13] A. Borodin, D. Dolev, F. E. Fich, and W. Paul. Bounds for width two branching programs. *SIAM Journal of Computing*, 15(2):549–560, 1986. doi:10.1137/0215040.

[14] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of STOC '71*, pages 151–158. ACM Press, 1971. doi:10.1145/800157.805047.

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

[16] G. B. Goodrich, R. E. Ladner, and M. J. Fischer. Straight-line programs to compute finite languages. In *Conference on Theoretical Computer Science, Waterloo*, pages 221–229, 1977.

[17] R. M. Karp and R. J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of STOC '80*, pages 302–309. ACM Press, 1980. doi:10.1145/800141.804678.

[18] L. Kristiansen and K.-H. Niggl. On the computational complexity of imperative programming languages. *Theoretical Computer Science*, 318(1–2):139–161, 2004. doi:10.1016/j.tcs.2003.10.016.

[19] A. R. Meyer and D. M. Ritchie. The complexity of loop programs. In S. Rosenthal, editor, *22nd ACM National Conference*, pages 465–469. ACM Press, 1967. doi:10.1145/800196.806014.

[20] K.-H. Niggl and H. Wunderlich. Certifying polynomial time and linear/polynomial space for imperative programs. *SIAM Journal of Computing*, 35(5):1122–1147, 2006. doi:10.1137/S0097539704445597.

[21] A. Ponse and M. B. van der Zwaag. An introduction to program and thread algebra. In A. Beckmann et al., editors, *CiE 2006*, volume 3988 of *Lecture Notes in Computer Science*, pages 445–458. Springer-Verlag, 2006. doi:10.1007/11780342_46.

[22] S. Skyum and L. G. Valiant. A complexity theory based on boolean algebra. *Journal of the ACM*, 32(2):484–502, 1985. doi:10.1145/3149.3158.

[23] T. Thierauf. *The Computational Complexity of Equivalence and Isomorphism Problems*, volume 1852 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2000.

[24] I. Wegener. *Branching Programs and Binary Decision Diagrams – Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, 2000. doi:10.1137/1.9780898719789.

[25] C. K. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26(3):287–300, 1983. doi:10.1016/0304-3975(83)90020-8.