Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

# A finite set of functions with an EXPTIME-complete composition problem

Marcin Kozik[*]

*Algorithmics Research Group, Jagiellonian University, Gronostajowa 3, 30-387 Kraków, Poland*
*Eduard Čech Center, Charles University, Sokolovska 83, 186 75 Praha 8, Czech Republic*

A R T I C L E  I N F O

*Article history:*
Received 27 June 2007
Received in revised form 11 June 2008
Accepted 27 June 2008
Communicated by J. Diaz

*Keywords:*
Computational complexity
Composition of functions
Clone membership problem

A B S T R A C T

We exhibit a finite family of functions over a finite set (i.e. a finite algebra), such that the problem whether a given function can be obtained as a composition of the members of this family (i.e. is a member of the clone generated by the algebra) is EXPTIME-complete.

© 2008 Elsevier B.V. All rights reserved.

## 1. Introduction

The class of EXPTIME-complete problems is well known for containing problems of finding the optimal strategies for chess [7], checkers [15], as well as some versions of go (comp. [12,4]). Another set of EXPTIME-complete problems consists of problems derived from questions of smaller complexity, by means of succinct circuits which allow one to present an input to the algorithms in exponentially smaller space (comp. [1]). Yet another EXPTIME-complete problem was found by H. Friedman about 1985 and published by Bergman, Juedes and Slutzki in [2]: it is an EXPTIME-complete problem to decide whether a function can be obtained as a composition of given functions.

The problem of composition of functions was studied for a number of reasons. Finding a constant function as a composition of a given set of functions over a finite set is connected to Černý's Conjecture [5] and was proved to be solvable in a polynomial time in [8]. On the other hand, Dexter Kozen had proved in [11] that for a finite family of unary functions (over a finite set) with one distinguished member, it is a PSPACE-complete problem to decide whether the distinguished function can be obtained as a composition of the others. The mentioned above result of H. Friedman shows that allowing non-unary functions causes the problem to become EXPTIME-complete.

The problem of composition of functions has a very natural algebraic description. The result of H. Friedman describes the computational complexity of the following problem.

| | |
|---|---|
| INPUT | a finite algebra $\mathbf{A}$ and a function $f : A^k \to A$ |
| PROBLEM | decide if $f \in \mathrm{Clo}(\mathbf{A})$. |

This problem splits into subproblems — for a fixed, finite algebra $\mathbf{A}$ we define

| | |
|---|---|
| INPUT | a function $f : A^k \to A$ |
| PROBLEM | decide if $f \in \mathrm{Clo}(\mathbf{A})$. |

[*] Corresponding address: Algorithmics Research Group, Jagiellonian University, Gronostajowa 3, 30-387 Krakow, Poland. Tel.: +48 12 664 69 35.
*E-mail address:* Marcin.Kozik@tcs.uj.edu.pl.

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.06.057

None of the such defined subproblems is more complex then the original problem. The natural question is how much easier are they. Does the complexity of the first problem rely on the fact that the algebra can vary? Can we find a fixed structure such that the connected subproblem remains complex? How complex is the problem of a membership in a finitely generated clone? This question follows the line of research devoted to describing the computational complexity of natural algebraic problems generated by fixed, finite structures. Such problems are studied extensively, and a number of results is already known (comp. [16,9,10]).

The main theorem of our paper answers this question. We construct a finite family of binary functions (over a finite set), such that the problem whether a given function can be obtained as a composition of functions from this fixed family is EXPTIME-complete. This proves that the membership problem for a finitely generated clone can be EXPTIME-complete i.e. as complex as the general problem. Moreover our construction allows us also to restate the result of [2], considering the computational complexity of the TERM-EQ problem in a more restrictive way.

In Section 4, we build a finite family of binary functions such that the question whether a given function can be obtained as a composition of these binary functions is EXPTIME-complete. The construction is motivated by the work of McKenzie [13]. In Section 5 we briefly address the consequences of our result to the TERM-EQ problem studied in [2].

## 2. Definitions

The following definitions allow us to state the problems formally. We define a set of functions obtained as compositions of unary functions in the following way.

**Definition 1.** For a given set $\mathcal{U}$ and functions $f_0, \ldots, f_n : \mathcal{U} \to \mathcal{U}$ we define the set of functions generated by $f_0, \ldots, f_n$ to be

$$\{\mathrm{id}_{\mathcal{U}}\} \cup \{s : \mathcal{U} \to \mathcal{U} \mid s = f_{a_0} \circ \cdots \circ f_{a_k} \text{ where } a_i \leq n \text{ for all } i \leq k\}.$$

and denote it by $\mathrm{Clo}_1^{\mathcal{U}}(f_0, \ldots, f_n)$.

In such a case the problem of of composition of unary functions follows.

**Composition of unary functions.** *Given a finite set $\mathcal{U}$ and a number of functions $s, f_0, \ldots, f_n : \mathcal{U} \to \mathcal{U}$ decide whether $s \in \mathrm{Clo}_1^{\mathcal{U}}(f_0, \ldots, f_n)$.*

In the case of composition of functions of higher arity we follow a standard notation which can be found in [3]. We define a composition of functions in the following way:

**Definition 2.** For a set of functions $f_0, \ldots, f_n$ such that $f_i : \mathcal{U}^{k_i} \to \mathcal{U}$ the set of functions generated by $f_0, \ldots, f_n$ is the minimal set $X$ such that

(1) for any number $m$ and any $i < m$ the function $f : \mathcal{U}^m \to \mathcal{U}$ defined to be $f(a_0, \ldots, a_{m-1}) = a_i$ is a member of $X$, and
(2) for any $i \leq n$ and any $g_0, \ldots, g_{k_i-1}$ members of $X$ the function

$$f_i(g_0, \ldots, g_{k_i-1}) \text{ is a member of } X.$$

We denote this set by $\mathrm{Clo}^{\mathcal{U}}(f_0, \ldots, f_n)$.

The general problem of composition of functions is defined in the following way.

**Composition of functions.** *Given a finite set $\mathcal{U}$ together with a number of functions $s, f_0, \ldots, f_n$ such that $f_i : \mathcal{U}^{k_i} \to \mathcal{U}$ and $s : \mathcal{U}^k \to \mathcal{U}$ decide whether $s \in \mathrm{Clo}^{\mathcal{U}}(f_0, \ldots, f_n)$.*

Note that, according to Definitions 1 and 2, projections are always members of the generated set. This assumption is introduced for clarity of presentation, and does not influence the computational complexity of problems considered in this paper.

## 3. Notation and preliminaries

We identify natural numbers with their binary representations of a fixed length. That is, for a fixed $n$, a number between 0 and $2^n - 1$ is a word of length $n$ over the alphabet $\{0, 1\}$, and the set of all such words is denoted by $\{0, 1\}^n$. Thus addition and substraction are partial functions on such words. We occasionally use regular expressions to denote a certain family of numbers i.e. 1*0 is the set of all even numbers between 0 and $2^n - 1$.

For two words $u$ and $v$ we denote by $u \cdot v$ or simply $uv$ a catenation of these two words. For a given word $w$ we denote by $w(k)$ the $k$-th letter of $w$, where $k$ is taken from the interval between 0 and $|w|-1$ and $|w|$ denotes the number of letters in $w$.

All of the constructions in this paper are based on Turing machines. We usually denote a Turing machine by **T**, its alphabet by $\mathcal{A}$ and the set of internal states by $\mathcal{S}$. We will denote the states of the machine by lowercase Greek letters with the starting state denoted by $\alpha$, and the accepting state denoted by $\omega$. The instructions of the machine are five-tuples of the form $\beta a b D \gamma$, for $\beta, \gamma \in \mathcal{S}$, $a, b \in \mathcal{A}$ and $D = R$ or $D = L$ meaning: "while in state $\beta$ reading $a$, write $b$ move in the direction $D$ and change the state to $\gamma$".

All the computations of the machine **T** will take place on a bounded tape, and we use two special symbols $\lozenge$ and $\blacklozenge$, taken from $\mathcal{A}$, to denote the leftmost and the rightmost, respectively, position on a tape. A configuration of the machine **T** is a triple $\langle \mathbf{w}, i, \beta \rangle$ where $\mathbf{w}$ is the word of fixed length denoting the tape of the machine **T**, $i$ is the number between 0 and $|\mathbf{w}| - 1$ describing the position of the head on a tape and $\beta$ is an internal state of the machine **T**.

We will use a particular type of a Turing machine in our reductions — an *alternating* Turing machines. Computations of an alternating Turing machine are identical with the computations of the usual Turing machine, but the set of accepting words can be different. The set of states of an alternating Turing machine splits into two parts: *universal* states and *existential* states. The machine accepts an input if and only if the starting configuration is *accepting* and the definition of an accepting configuration is recursive:

- any configuration in an accepting state $\omega$ is accepting;
- the configuration in an existential state is accepting if there *exists* an operation of the Turing machine which can be executed in this configuration and produces an accepting configuration and
- the configuration in an universal state is accepting if *all* the operations of the Turing machine which can be executed in this configuration produce accepting configurations.

Alternating Turing machines compute "more efficiently" (comp. [6]) than the usual Turing machines. This efficiency allows us to provide a tight bound on the composition problem.

We find it useful to work with the formal expressions defining functions, and we follow [3] in defining them.

**Definition 3.** For a set of functional symbols $f_0, \ldots, f_n$ of arities $k_0, \ldots, k_n$, and a set of variables $\{x, y, \ldots\}$ we define *a term* in a recursive way

(1) any variable is a term, and
(2) for any $i \leq n$ and any terms $g_0, \ldots, g_{k_i-1}$ the formal expression

$$f_i(g_0, \ldots, g_{k_i-1}) \text{ is a term as well, and}$$

(3) all the terms can be obtained in such a way.

Throughout the paper we often abuse the notation by identifying the functional symbols denoting the operations and the functions themselves. The distinction is always clear from the context.

It is a trivial observation that, having a fixed set of generating functions and a corresponding set of terms, each term defines a function, and each function that can be obtained as a composition is defined by a term. The notion of a subterm is intuitive — if we consider a term to be a labelled tree, then each term being a labelled subtree of our term is its subterm. For a more involved study of algebraic concepts we refer the reader to [3,14].

## 4. A finite set of functions with an EXPTIME-complete composition problem

In this section we exhibit a finite family of functions, such that the membership in the set of functions generated by this family is EXPTIME-complete. The family of functions is constructed, based on an alternating Turing machine **T** (working in polynomial space) satisfying the following conditions:

(1) the language accepted by **T** is EXPTIME-complete;
(2) every accepted word is of length $2^n - 2$ for some $n$ ,
(3) for a given input word $w$, the computations of the machine **T** never leave $\lozenge w \blacklozenge$, and at least one instruction is executed on each position of such a tape for each computation branch,
(4) no instruction of **T** can be executed in the state $\omega$,
(5) for each universal state of the machine **T** there are at most two instructions that can be executed (for a head reading some symbol on a tape).

We briefly argue an existence of such a machine. Note that we require that the machine accepts an EXPTIME-complete language (condition (1)) and works on a tape restricted to the size of the input (condition (3)). By the result of [6] there exists an alternating Turing machine recognizing an EXPTIME-complete language, and working in a polynomial space. Further, by a reduction computed in a polynomial time, one can change this machine and the language it accepts to comply with condition (3) — it suffices to extend each input by polynomially many blank symbols and enclose it by $\lozenge$ and $\blacklozenge$. The second part of the condition is easily obtained, by modifying the machine to pre-read all the symbols between $\lozenge$ and $\blacklozenge$ before starting its computations. Thus we obtain an alternating Turing machine satisfying conditions (1) and (3). The condition (2) can be easily obtained by enlarging accepted input words to the smallest length of form $2^n - 2$, and is done in a polynomial time. Finally, the condition (4) is pure technicality and the condition (5) can be achieved by introducing auxiliary internal states of the machine. Thus an alternating Turing machine satisfying conditions (1) to (5) exists.

The remaining part of this section is devoted to a construction of a set of functions that would model (by their compositions) a computation of the machine **T** satisfying the conditions above. More precisely, for a starting configuration of the machine, and for any configuration that can be reached via a computation of the machine **T**, we will introduce terms defining a function that "realizes" this computation. Further terms will define functions working, in a similar way, only for accepting configurations. Thus a starting configuration will be accepting if and only if there exists a composition witnessing this fact. This will show that the language recognized by **T** can be encoded into the composition problem for the functions we define, and prove the main result of the paper formulated in Theorem 4.7.1.

For every configuration, we will produce a term witnessing the computations starting from it. We achieve it by composing terms "realizing" single steps of the machine. The term's arguments will include a binary representation of a position on a tape of the machine and the information about the symbol at this position in the initial configuration — the function will compute the symbol at the same position after the computation which the term realizes. Roughly speaking, for an input of size $2^n$ and some computation of the machine, we will construct a $(n+1)$-ary function such that:

$$f(\text{“position on tape”}, \text{“tape symbol”}) = \text{“tape symbol after the computation”},$$

where the "position on tape" is represented as $n$ arguments forming a binary representation of a number. An internal state of the machine can be represented easily, and the current position of the head on a tape is encoded, together with the appropriate tape symbol. Thus the whole configuration of the Turing machine can be stored as a set of tuples, which are arguments of $(n+1)$-ary functions and the computations on them can be realized by terms.

We introduce the set (denoted by $\mathcal{U}$) and the functions on this set gradually while explaining their purpose. These functions will model the computations of the machine **T** and define an EXPTIME-complete problem for a restricted compositions of functions (comp. Proposition 4.6.1). Adding a little twist in a Section 4.7 will allow us to obtain the full result stated in Theorem 4.7.1.

The functions split into three disjoin families $\mathcal{F} \cup \mathcal{G} \cup \mathcal{H}$. All the functions in sets $\mathcal{F}$ and $\mathcal{H}$ are binary, while the functions in $\mathcal{G}$ are unary.

## *4.1. The garbage collector element and the position markers*

The set $\mathcal{U}$ contains an *absorbing* element $\perp$:

$$\perp \in \mathcal{U} \tag{1}$$

and for any $f \in \mathcal{F}$, $g \in \mathcal{G}$ and $h \in \mathcal{H}$

$$f(\perp, a) = f(a, \perp) = g(\perp) = h(\perp, a) = h(a, \perp) = \perp \quad \text{for all } a \in \mathcal{U}.$$

This element plays a role of a "garbage collector". Whenever a composition of functions produces, for an important evaluation, a result equal to $\perp$ we will discard such a composition.

The set $\mathcal{U}$ contains also

$$\mathcal{M} = \{0, 1\} \subset \mathcal{U} \tag{2}$$

and, for any $f \in \mathcal{F}$, $g \in \mathcal{G}$ and $h \in \mathcal{H}$, the following implications are true

$$\begin{aligned} f(a, b) \neq \perp &\implies a \in \mathcal{M} \wedge b \notin \mathcal{M}, \\ a \in \mathcal{M} &\implies g(a) = h(a, b) = h(b, a) = \perp \quad \text{for all } b \in \mathcal{U}, \end{aligned}$$

and the elements of the set $\mathcal{M}$ are outside of the range of all the functions in $\mathcal{F} \cup \mathcal{G} \cup \mathcal{H}$. These elements are used as a binary representation of a number which is a position on a tape of a Turing machine. The definition implies the following corollary.

**Corollary 4.1.1.** *If a function $s : \mathcal{U}^n \to \mathcal{U}$ is not constantly equal to $\perp$ and $s \in \mathrm{Clo}^{\mathcal{U}}(\mathcal{F} \cup \mathcal{G} \cup \mathcal{H})$, then whenever $f \in \mathcal{F}$ appears in a term defining $s$ then its appearance is of the form $f(x_i, t(\bar{x}))$ for some variable $x_i$ and term $t(\bar{x})$.*

Which, in turn, leads to another definition.

**Definition 4.** A term $f^{(k)}(x_{i_k}, \ldots f^{(0)}(x_{i_0}, y))$ (for $f^{(j)} \in \mathcal{F}$ for all $j$) is an $\mathcal{F}$-factor of a term $t(\bar{x})$ if and only if $f^{(k)}(x_{i_k}, \ldots f^{(0)}(x_{i_0}, t'(\bar{x})))$ is a subterm of $t(\bar{x})$, for some term $t'(\bar{x})$, and the operation applied to it in $t(\bar{x})$ as well as the out-most operation of $t'(\bar{x})$ are not in $\mathcal{F}$.

A $\mathcal{F}$-factor of a term, under further conditions, will be responsible for "decoding" a sequence of elements of $\mathcal{M}$ into a position on a tape. Moreover a $\mathcal{F}$-factor followed by the operation of the set $\mathcal{G} \cup \mathcal{H}$ will be responsible for "realizing" a single computing (or validating) step of the Turing machine **T**.

## *4.2. Decoding a postion*

In order to model a computation, we need a uniform way of "decoding" a sequence of elements of $\mathcal{M}$ into a position on a tape. In other words, since $\mathcal{F}$-factors of a term are responsible for this "decoding", we need to make sure that all the $\mathcal{F}$-factors recognize the same variable storing the first bit of a binary word, the same variable storing the second bit and so on. We achieve it by introducing additional elements of $\mathcal{U}$:

$$\mathcal{P} = \{A, B, C, D\} \subset \mathcal{U}, \tag{3}$$

and defining the functions of $\mathcal{F} \cup \mathcal{G} \cup \mathcal{H}$ on them in the following way.

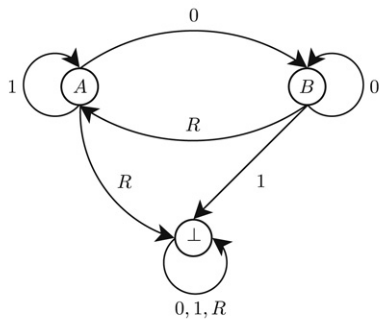

**Fig. 1.** Forcing the order of the variables.

For any $f \in \mathcal{F}$, $g \in \mathcal{G}$ and $h \in \mathcal{H}$ and any $a \in \mathcal{M}$, $b, d \in \mathcal{P}$ we set:

$$f(a, b) = d \text{ iff } (b \xrightarrow{a} d \text{ in Figs. 1 or 2})$$

$$g(b) = d \text{ iff } (b \xrightarrow{R} d \text{ in Figs. 1 or 2})$$

$$h(b, c) = \begin{cases} d & \text{if } b = c \wedge (b \xrightarrow{R} d \text{ in Figs. 1 or 2}) \\ \perp & \text{if } b \neq c. \end{cases}$$

Moreover the above definitions are the only possibilities to obtain an element of $\mathcal{P}$ as a result of an application of a function from $\mathcal{F} \cup \mathcal{G} \cup \mathcal{H}$. Note that the conditions on the functions in $\mathcal{F} \cup \mathcal{G} \cup \mathcal{H}$ fully define them on $\mathcal{M} \cup \mathcal{P} \cup \{\perp\}$.

We are ready to characterize terms defining certain functions on $\mathcal{U}$. All of the functions "realizing" the computations of the Turing machine will satisfy Conditions [CI ] and [CII ]. In the remaining part of the subsection we prove that such terms have identical sequence of variables in $\mathcal{F}$-factors. The first condition, for a function $s : \mathcal{U}^{n+1} \to \mathcal{U}$, states:

**CI**. For any $a \in 1^*0^+ \subset \{0, 1\}^n$ we have $s(a, A) = A$.
The condition [CI ] has an immediate corollary.

**Corollary 4.2.1.** *If $t(\bar{x}, y)$ is a term defining a function satisfying [CI ], then each $\mathcal{F}$-factor of $t(\bar{x}, y) = t(x_0, \ldots, x_{n-1}, y) - f^{(k)}(x_{i_k}, \ldots f^{(0)}(x_{i_0}, y))$ satisfies $i_0 \leq \cdots \leq i_k$.*

**Proof.** Let us fix $f^{(k)}(x_{i_k}, \ldots f^{(0)}(x_{i_0}, t'(\bar{x}, y)))$ provided by Definition 4 for an $\mathcal{F}$-factor $f^{(k)}(x_{i_k}, \ldots f^{(0)}(x_{i_0}, y))$ of a term $t(\bar{x}, y)$ defining a function satisfying [CI ]. Since $t(a, A) = A$ we immediately get $t'(a, A) \in \{A, B\}$ and, using a definition of $\mathcal{F}$-factors, we infer that the term $t'(\bar{x}, y)$ is either equal to $y$ or has the outmost operation coming from the set $\mathcal{G} \cup \mathcal{H}$. In both cases, by considering ranges of the operations, we immediately obtain $t'(a, A) = A$ for all $a$'s from [CI ]. Now suppose, for a contradiction, that $i_l < i_j$ for some $j < l$; then for $a = 1^{i_j - 1}0^{n - i_j + 1}$ we have $f^{(j)}(0, \ldots, f^{(0)}(a(i_0), A)) \in \{B, \perp\}$. Thus $f^{(l)}(1, \ldots, f^{(0)}(a(i_0), A)) = \perp$ which is a contradiction with [CI ]. □

We introduce the second condition.

**CII**. For any $a \in 1^*01^* \subset \{0, 1\}^n$ we have $s(a, C) = C$.
The following corollary is proved in exactly the same way as Corollary 4.2.1 and we omit the proof of it.

**Corollary 4.2.2.** *If $t(\bar{x}, y)$ is a term defining a function satisfying [CI ] and [CII ], then all $\mathcal{F}$-factors of $t(\bar{x}, y)$ are of the form $f^{(n-1)}(x_{n-1}, \ldots, f^{(0)}(x_0, y))$.*

A straightforward analysis of domains and ranges of the operations implies the following corollary.

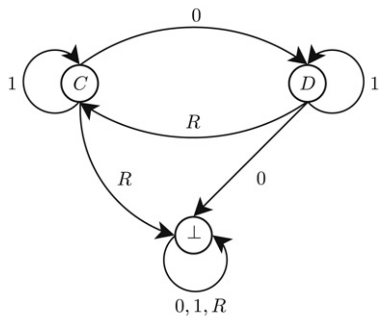

**Fig. 2.** Forcing the arity of the variables.

**Corollary 4.2.3.** *If $t(\bar{x}, y)$ is a term defining a function satisfying [CI ] and [CII ], then the operations of $\mathcal{G} \cup \mathcal{H}$ are never applied directly after each other in $t(\bar{x}, y)$. Moreover the outmost operation of $t(\bar{x}, y)$ belongs to $\mathcal{G} \cup \mathcal{H}$.*

Thus all the $\mathcal{F}$-factors of interesting terms have the same order of variables — this provides a uniform way of encoding a position on a tape.

### *4.3. Modeling the tape of the machine*

We move to define a next part of the set $\mathcal{U}$. Elements of this set store the symbols on a tape of the Turing machine (the fourth coordinate), a relation to the position of the head of the machine (Left or Right or Head) and two auxiliary True/False fields. We put

$$\mathcal{T} = \{T, F\}^2 \times \{L, H, R\} \times \mathcal{A} \subset \mathcal{U}. \tag{4}$$

The first two coordinates are auxiliary symbols that are used by $\mathcal{F}$-factors of the terms to identify the current and the future position of the head of the machine in a computation.

The set $\mathcal{F}$ consists of the functions $f_0^0(x, y), f_1^1(x, y), f_1^0(x, y), f_0^1(x, y)$ and the $\mathcal{F}$-factor appearing after a term computing a configuration with the head at position $i$ will, under certain conditions, have a sequence of superscripts of its elements equal to $i$ (comp. Proposition 4.3.1). The sequence of subscripts of such a $\mathcal{F}$-factor will be a position of the head after the next operation of the machine.

We introduce the functions of the set $\mathcal{F}$. Their definitions are tailored with Proposition 4.3.1 in mind. For any $b \in \mathcal{M}$ and any $(I, J, D, a) \in \mathcal{T}$ we put the function $f_0^0(x, y)$ to be

| $I$ | $J$ | $D$ | $a$ | $b$ | $f_0^0(b, (I, J, D, a))$ |
|---|---|---|---|---|---|
| $F$ | $F$ | any | any | any | $(F, F, D, a)$ |
| $T$ | $T$ | any | any | 0 | $(T, T, D, a)$ |
| $T$ | $T$ | any | any | 1 | $(F, F, D, a)$ |
| else | else | else | else | else | $\perp$ |

dually we define $f_1^1(x, y)$

| $I$ | $J$ | $D$ | $a$ | $b$ | $f_1^1(b, (I, J, D, a))$ |
|---|---|---|---|---|---|
| $F$ | $F$ | any | any | any | $(F, F, D, a)$ |
| $T$ | $T$ | any | any | 0 | $(F, F, D, a)$ |
| $T$ | $T$ | any | any | 1 | $(T, T, D, a)$ |
| else | else | else | else | else | $\perp$ |

and two more complicated functions $f_1^0(x, y)$

| $I$ | $J$ | $D$ | $a$ | $b$ | $f_1^0(b, (I, J, D, a))$ |
|---|---|---|---|---|---|
| $F$ | $F$ | any | any | any | $(F, F, D, a)$ |
| $T$ | $T$ | $H$ or $L$ | any | 0 | $(T, F, D, a)$ |
| $T$ | $T$ | $R$ | any | 1 | $(F, T, D, a)$ |
| $F$ | $T$ | $L$ | any | 0 | $(F, F, D, a)$ |
| $F$ | $T$ | $L$ | any | 1 | $(F, T, D, a)$ |
| $T$ | $F$ | $R$ or $H$ | any | 0 | $(T, F, D, a)$ |
| $T$ | $F$ | $R$ or $H$ | any | 1 | $(F, F, D, a)$ |
| else | else | else | else | else | $\perp$ |

and $f_0^1(x, y)$

| $I$ | $J$ | $D$ | $a$ | $b$ | $f_0^1(b, (I, J, D, a))$ |
|---|---|---|---|---|---|
| $F$ | $F$ | any | any | any | $(F, F, D, a)$ |
| $T$ | $T$ | $L$ | any | 0 | $(F, T, D, a)$ |
| $T$ | $T$ | $R$ or $H$ | any | 1 | $(T, F, D, a)$ |
| $F$ | $T$ | $R$ | any | 0 | $(F, T, D, a)$ |
| $F$ | $T$ | $R$ | any | 1 | $(F, F, D, a)$ |
| $T$ | $F$ | $L$ or $H$ | any | 0 | $(F, F, D, a)$ |
| $T$ | $F$ | $L$ or $H$ | any | 1 | $(T, F, D, a)$ |
| else | else | else | else | else | $\perp$ |

The following notation allows for an easier analysis of the $\mathcal{F}$-factors of certain terms. Please note that the variables in these terms are ordered in a fixed manner (i.e. changing subscripts of the variables appearing in a term will cause a term not to comply with this definition).

**Definition 5.** For any words $u$, $v$ of equal length over the alphabet $\{0, 1\}$ we define the following terms:

$$f_{\varepsilon}^{\varepsilon}(\bar{x}, y) = y,$$

where $\varepsilon$ denotes the empty word, and

$$f_{vb}^{ua}(\bar{x}, y) = f_b^a(x_{|ua|-1}, f_v^u(\bar{x}, y))$$

recursively, where $a, b \in \{0, 1\}$ and $f_b^a(x, y)$ is one of the four functions in $\mathcal{F}$.

According to Corollaries 4.2.1 and 4.2.2, each $\mathcal{F}$-factor of the term defining function satisfying Condition [CI ] and [CII ] is of the form $f_v^u(\bar{x}, y)$ for some words $u$, $v$.

The following set consist of all possible tuples which can be associated with a configuration with a head at the position $i$ and, at the same time, the exact set on which the $\mathcal{F}$-factor recognizes the position of the head and the positions next to it. Let $P_i \subset \{0, 1\}^n \times \mathcal{T}$ be such that $(b, (I, J, D, a)) \in P_i$ if and only if $I = J = T$ and

$$(b < i \wedge D = L) \vee (b = i \wedge D = H) \vee (b > i \wedge D = R).$$

The following proposition shows that the only $\mathcal{F}$-factors that can be applied to $P_i$ without $\perp$ among the results are either trivial ($u = v$ in a proposition below), or recognize the positions $i-1$ and $i+1$. This fact is a key ingredient of our proof. Once these positions are identified, a unary function will compute (comp. Corollary 4.5.1) the next configuration of the machine.

**Proposition 4.3.1.** *For any $u$, $v$, words of length $n$ over the alphabet $\{0, 1\}$, the following conditions are equivalent:*

1. $f_v^u(b, (I, J, D, a)) \neq \perp$ *for all* $(b, (I, J, D, a)) \in P_i$,
2. $u = v$ *or* $i = u = v - 1$ *or* $i = u = v + 1$.

**Proof.** We begin with a proof of the implication from (2) to (1). The case when $u = v$ is a trivial analysis of the definitions of functions $f_0^0(x, y)$ and $f_1^1(x, y)$. Assume now, that $i = u = v + 1$ and that $w$ is the longest common prefix of $u$ and $v$. Then $u = w10^{|w|-1}$ and $v = w01^{|w|-1}$. Lets fix an arbitrary $(b, (T, T, D, a)) \in P_i$. If $w$ is not a prefix of $b$ then $f_w^w(b, (T, T, D, a)) = (F, F, D, a)$ and thus $f_v^u(b, (T, T, D, a)) \neq \perp$ as required. Assume now that $w$ is a prefix of $b$. The word $w0$ is a prefix of $v$, and $w1$ is a prefix of $u$ - basing on this fact we consider two cases:

- if $b < i = u$, then $w0$ is a prefix of $b$ and by the definition of $P_i$ we obtain $D = L$ and thus
$$\begin{aligned} f_{w0}^{w1}(b, (T, T, L, a)) &= f_0^1(0, f_w^w(b, (T, T, L, a))) \\ &= f_0^1(0, (T, T, L, a)) = (F, T, L, a). \end{aligned}$$
Since the function $f_1^0(x, y)$ applied to any element of the form $(F, T, L, a)$ produces either $(F, T, L, a)$ or $(F, F, L, a)$ and (since $f_1^0(c, (F, F, L, a)) = (F, F, L, a)$ for any $c \in \{0, 1\}$) we get $f_v^u(b, (T, T, L, a)) \neq \perp$ as required.
- if $b \geq i = u$ then $w1$ is a prefix of $b$ and by the definition of $P_i$ we infer that $D \in \{R, H\}$. In such a case
$$\begin{aligned} f_{w0}^{w1}(b, (T, T, D, a)) &= f_0^1(1, f_w^w(b, (T, T, D, a))) \\ &= f_0^1(1, (T, T, D, a)) = (T, F, D, a), \end{aligned}$$
and exactly the same reasoning shows that $f_v^u(b, (T, T, L, a)) \neq \perp$ as required.

This finishes the case of $i = u = v + 1$ and the remaining case of the implication, $i = u = v - 1$, is an alphabetical variant of the same proof. The implication from (2) to (1) is proved.

To start with the reverse implication, assume that $u \neq v$, and let $w$ denote the maximal common prefix of $u$ and $v$. First, we assume that $u > v$, that is $w1$ is a prefix of $u$ and $w0$ is a prefix of $v$. Thus, for each $(b, (T, T, D, a)) \in P_i$ such that $w$ is a prefix of $b$, we have

$$f_{w0}^{w1}(b, (T, T, D, a)) = f_0^1(b_{|w1|-1}, f_w^w(b, (T, T, D, a))) = f_0^1(b_{|w1|-1}, (T, T, D, a)) \neq \perp,$$

which, by the definition of $f_0^1(x, y)$, implies that $D = L$ whenever $b_{|w1|-1} = 0$ and $D \in \{R, H\}$ whenever $b_{|w1|-1} = 1$. The construction of $P_i$ immediately implies that $i = w10^{n-|w1|}$. Let us fix an element of $P_i$ of the form $(w01^{n-|w1|}, (T, T, L, a))$, then

$$f_{w0}^{w1}(w01^{n-|w1|}, (T, T, L, a)) = f_0^1(0, (T, T, L, a)) = (F, T, L, a),$$

and, since $f_1^0(x, y)$ is the only operation that does not produce $\perp$ on $(F, T, L, a)$, we imply that $u = w10^{n-|w1|}$ and $v = w01^{n-|w1|}$ as required. The case of $u < v$ is an alphabetical variant of the same proof, and the proposition is proved. □

The following corollary is an easy consequence of the proposition.

**Corollary 4.3.2.** *Let $u, v$ be members of $\{0,1\}^n$ such that $f^u_v(b,(T,T,D,a)) \neq \perp$ for all $(b,(T,T,D,a)) \in P_i$. Then, for any element $(b,(T,T,D,a)) \in P_i$,*

$$[f^u_v(b,(T,T,D,a)) = (T,J,D,a) \text{ for some } J \in \{T,F\}] \text{ if and only if } b = u;$$

*and*

$$[f^u_v(b,(T,T,D,a)) = (I,T,D,a) \text{ for some } I \in \{T,F\}] \text{ if and only if } b = v.$$

**Proof.** For any function $f^i_j(x,y) \in \mathcal{F}$, and any $k \in \{0,1\}$ a straightforward analysis of the definitions of functions shows that if $f^i_j(k,(I,J,D,a)) \neq \perp$ then

$$f^i_j(k,(I,J,D,a)) = (T,J,D,a) \quad \text{if and only if} \quad (I = T \text{ and } i = k).$$

This immediately implies the first equivalence of the corollary. The second equivalence is proved by the same reasoning for the the second coordinate of the element of $\mathcal{T}$, and the pair $j, k$ (instead of $i, k$). □

The operations of $\mathcal{F}$ are defined on the set $\mathcal{T}$ in such a way that the last coordinate of the element of $\mathcal{T}$ "carries over" to the result (unless the result is $\perp$), and has no influence on the result being equal to $\perp$. This implies the following corollary.

**Corollary 4.3.3.** *For any $u, v \in \{0,1\}^n$ and any $P \subset P_i$ such that for any $b \in \{0,1\}^n$ there exists $(T,T,D,a) \in \mathcal{T}$ such that $(b,(T,T,D,a)) \in P$, the following conditions are equivalent:*

- *for any $(b,(T,T,D,a)) \in P$ we have $f^u_v(b,(T,T,D,a)) \neq \perp$;*
- *for any $(b,(T,T,D,a)) \in P_i$ we have $f^u_v(b,(T,T,D,a)) \neq \perp$.*

We move on to define the operations of the set $\mathcal{G} \cup \mathcal{H}$ on $\mathcal{T}$. The operations of the set $\mathcal{G}$ compute the steps of the Turing machine, while the operations from $\mathcal{H}$ "reverse" the computation checking whether a configuration is accepting. Note that backtracking a computation of a Turing machine is very similar to computing it, and thus operations of $\mathcal{G}$ and $\mathcal{H}$ have a lot in common.

For a $\mathcal{F}$-factor acting on a set $P_i$ (using Proposition 4.3.1) we need to consider three cases (given by item (2) of the proposition). The instruction of the set $\mathcal{G}$, is constructed in such a way that only the $\mathcal{F}$-factor identifying a new position for a head of the machine is "compatible" with it (will not produce $\perp$ on any element of $P_i$). The first coordinate of the result of an application of the compatible $\mathcal{F}$-factor to an element of $P_i$ is $T$, only for the position $i$ (the current position of the head), and the second is $T$ on a coordinate with the future position of the head. The third coordinate of the element of $\mathcal{T}$ allows us to identify the direction into which the head of the Turing machine is moving. Therefore all the other $\mathcal{F}$-factors will produce $\perp$ on some element of $P_i$ when composed with the operation of the set $\mathcal{G}$.

We set $\bar{R} = L$ and $\bar{L} = R$, which allows us to present the definitions in a more compact way. We define the set $\mathcal{G}$ – the instructions from this set generate the configurations of **T** following the computations of the machine. For each instruction of **T**, denoted by $\beta abD\gamma$, we put $g^{\beta abD\gamma}$ to be, for any $c \in \mathcal{T}$,

$$g^{\beta abD\gamma}(c) = \begin{cases} (T,T,\bar{D},b) & \text{if } c = (T,F,H,a) \\ (T,T,H,d) & \text{if } c = (F,T,D,d) \\ (T,T,G,d) & \text{if } c = (F,F,G,d) \text{ for some } G \neq H \\ \perp & \text{else.} \end{cases}$$

We move on to define the instructions of the set $\mathcal{H}$ on $\mathcal{T}$. They allow us to check whether an alternating Turing machine accepts a given configuration. Each of the instructions will provide an "accepted configuration" backtracking the computations of **T**. Thus, for any pair of the instructions of **T** of the form $\beta abD\gamma$, $\beta acE\delta$ such that:

- if $\beta$ is a universal state then $\beta abD\gamma$ and $\beta acE\delta$ are the only two instructions that can be executed in a state $\beta$ with the head reading $a$;
- if $\beta$ is an existential state then $\beta abD\gamma = \beta acE\delta$;

we define $h^{\beta abD\gamma}_{\beta acE\delta}$ on $d, e \in \mathcal{T}$ to be

$$h^{\beta abD\gamma}_{\beta acE\delta}(d,e) = \begin{cases} (T,T,H,a) & \text{if } d = (F,T,\bar{D},b) \text{ and } c = (F,T,\bar{E},c) \\ (T,T,D,o) & \text{if } D = E \text{ and } d = (T,F,H,o), e = (T,F,H,o) \\ (T,T,D,o) & \text{if } d = (T,F,H,o) \text{ and } e = (F,F,D,o) \\ (T,T,E,o) & \text{if } d = (F,F,E,o) \text{ and } e = (T,F,H,o) \\ (T,T,G,o) & \text{if } d = e = (F,F,G,o) \text{ for some } G \neq H \\ \perp & \text{else.} \end{cases}$$

Note that the following corollary is a simple consequence of the definitions:

**Corollary 4.3.4.** *For each $h^{\beta abD\gamma}_{\beta acE\delta} \in \mathcal{H}$ and for any $(I, J, G, d), (I', J', G'', d') \in \mathcal{T}$ and any $D'' \in \{L, H, R\}$, $d'' \in \mathcal{A}$ the following conditions are equivalent*

(1) $h^{\beta abD\gamma}_{\beta acE\delta}((I, J, D, d), (I', J', D', d')) = (T, T, D'', d'')$,
(2) $g^{\beta abD\gamma}((J, I, D'', d'')) = (T, T, D, d)$ *and*
$g^{\beta acE\delta}((J', I', D'', d'')) = (T, T, D', d')$.

Thus the operations of the set $\mathcal{H}$ act as a "reverse" of the operations from $\mathcal{G}$, recreating the tape from before a step of a Turing machine.

### *4.4. The states of the machine*

We introduce the final part of the set $\mathcal{U}$. It is responsible for representing the internal state of the Turing machine and includes two copies of the set of the states (defined below). One copy is used for the computation (the functions of the $\mathcal{G}$ set) of the Turing machine **T**, and the other for backtracking and checking the accepting states (operations from $\mathcal{H}$). Among these sets one element is common — the accepting configuration $\omega$ is the only possibility for switching from computing to backtracking. More formally, we put

$$\mathcal{R} = \mathcal{S} \cup \mathcal{S}' \subset \mathcal{U} \tag{5}$$

where the set $\mathcal{S}'$ consists of copies of elements from $\mathcal{S}$ with the elements $\omega$ and $\omega'$ identified. There is a natural bijection from $\mathcal{S}$ onto $\mathcal{S}'$ mapping $\beta \mapsto \beta'$ for each state $\beta$. For any $f(x, y) \in \mathcal{F}$ and any $a \in \mathcal{M}$ and $\beta \in \mathcal{R}$ we put

$$f(a, \beta) = \beta.$$

For any $g^{\beta abD\gamma}(x) \in \mathcal{G}$ and any $\delta \in \mathcal{R}$ we put

$$g^{\beta abD\gamma}(\delta) = \begin{cases} \gamma & \text{if } \beta = \delta \\ \bot & \text{else.} \end{cases}$$

For any $h^{\beta abD\gamma}_{\beta acE\delta} \in \mathcal{H}$ and any $\pi, \tau \in \mathcal{R}$ we put

$$h^{\beta abD\gamma}_{\beta acE\delta}(\pi, \tau) = \begin{cases} \beta' & \text{if } \pi = \gamma' \text{ and } \tau = \delta' \\ \bot & \text{else.} \end{cases}$$

All the applications of the operations which were not defined explicitly are equal to $\bot$.

### *4.5. Computation and checking functions*

We are now in position to introduce a correspondence between the configurations of a Turing machine **T**, and tuples of the elements of the set $\mathcal{U}$. These tuples will be the arguments of functions modeling computations. The focus of this subsection is to establish this correspondence, and prove that the computations of the Turing machine **T** are equivalent to the applications of certain terms to such tuples. To prove an equivalence, we show that any application of functions which is not producing $\bot$ on a set of such tuples, corresponds to a computation step of the Turing machine. This will put the computations on configurations in a direct correspondence with terms acting on these sets.

For any configuration of the machine **T**, denoted by $\langle \mathbf{w}, i, \beta\rangle$, where $\mathbf{w} \in \mathcal{A}^{2^n}$, $i < 2^n$ and $\beta \in \mathcal{S}$, we define the corresponding set $P_{\langle \mathbf{w}, i, \beta\rangle}$. The first part of it is a subset of $P_i$, and is responsible for maintaining the information about the tape of the machine:

$$(b, (I, J, D, a)) \in P_{\langle \mathbf{w}, i, \beta\rangle} \cap P_i \quad \text{if and only if} \quad \mathbf{w}(b) = a.$$

The second part of $P_{\langle \mathbf{w}, i, \beta\rangle}$, which is outside of $P_i$, is equal to $\{0, 1\}^n \times \{\beta\}$ and keeps track of the internal state of the machine. Together they describe, in a natural way, a configuration of the machine. The next definition allows us to follow the computations of the machine using these sets. For any configuration $\langle \mathbf{w}, i, \beta\rangle$ and an operation of the Turing machine of the form $\gamma abD\delta$, we put

$$P_{\langle \mathbf{w}, i, \beta\rangle : \gamma abD\delta} = \{(c, d) \mid (c, e) \in P_{\langle \mathbf{w}, i, \beta\rangle} \text{ and } g^{\gamma abD\delta}(f^i_v(c, e)) = d\}$$

where $v = i + 1$ if $D = R$ and $v = i - 1$ if $D = L$.

The following corollary is the main part of the construction. It states that not having $\bot$ in a range of a term on a set associated with a configuration, is equivalent to the fact that the term models one step of the computation of the Turing machine, and produces output modelling the resulting configuration.

**Corollary 4.5.1.** *For any configuration $\langle \mathbf{w}, i, \beta\rangle$, any instruction of the machine **T** $\gamma abD\delta$ and any pair of words $u, v \in \{0, 1\}^n$ the following conditions are equivalent.*

(1) $\bot \notin g^{\gamma abD\delta}(f^u_v(P_{\langle \mathbf{w}, i, \beta\rangle}))$,

(2) *the set*

$$P_{\langle \mathbf{w},i,\beta\rangle:\gamma abD\delta} = P_{\langle \mathbf{w}',j,\pi\rangle}$$

*for some configuration* $\langle \mathbf{w}', j, \pi\rangle$, *and* $u = i$ *and* $v = i + 1$ *if* $D = R$ *and* $v = i - 1$ *if* $D = L$,

(3) *the instruction* $\gamma abD\delta$ *can be executed in a configuration* $\langle \mathbf{w}, i, \beta\rangle$ *and* $u = i$ *and* $v = i + 1$ *if* $D = R$ *and* $v = i - 1$ *if* $D = L$.

*Moreover, if these conditions are satisfied, the configuration obtained from* $\langle \mathbf{w}, i, \beta\rangle$ *after executing* $\gamma abD\delta$ *is equal to* $\langle \mathbf{w}', j, \pi\rangle$.

**Proof.** The implication from (2) to (1) is trivial. For the implication (1) to (3) we use Proposition 4.3.1 and Corollary 4.3.3 to obtain $u = v$ or $i = u$ and $|v - u| \leq 1$. Moreover Corollary 4.3.2 together with the definition of $g^{\gamma abD\delta}(x)$, implies that $u \neq v$ (since $g^{\gamma abD\delta}((T, T, E, c)) =\perp$ for any choice of $E$ and $c$). Similarly, since $g^{\gamma abD\delta}((F, T, E, c)) \neq\perp$ implies $E = D$, we obtain $v = i + 1$ if $D = R$ and $v = i - 1$ if $D = L$. Finally, since $g^{\gamma abD\delta}((T, F, H, c)) \neq\perp$ implies $c = a$ we infer that $w(i) = a$, and similarly since $g^{\gamma abD\delta}(f^u_v(0^n, \beta)) \neq\perp$, we imply that $\beta = \gamma$ and (3) is proved. It remains to show the implication from (3) to (2). The arguments above applied to $g^{\gamma abD\delta}(f^i_{i+1}(\bar{x}, y))$ if $D = R$ and to $g^{\gamma abD\delta}(f^i_{i-1}(\bar{x}, y))$ if $D = L$ together with Proposition 4.3.1, Corollary 4.3.2 and the definition of functions in $\mathcal{G}$ proves the implication. A proof of this fact provides also an argument that the configuration obtained from $\langle \mathbf{w}, i, \beta\rangle$ after execution of $\gamma abD\delta$ is equal to $\langle \mathbf{w}', j, \pi\rangle$. □

Using the previous corollary together with Corollaries 4.2.1 and 4.2.2, we put the full correspondence between terms not producing $\perp$ on sets associated with configurations and the possible computations of the Turing machine.

**Corollary 4.5.2.** *Let* $t(\bar{x}, y)$ *be a term using functional symbols from* $\mathcal{F} \cup \mathcal{G}$. *If* $t(\bar{x}, y)$ *defines a function satisfying Conditions [CI] and [CII ] and such that* $\perp\notin t(P_{\langle \mathbf{w},i,\beta\rangle})$ *for some configuration* $\langle \mathbf{w}, i, \beta\rangle$, *then the set*

$$\{(c, d) \mid (c, e) \in P \text{ and } t(c, e) = d\} \text{ equals } P_{\langle \mathbf{w}',j,\gamma\rangle}$$

*for some configuration* $\langle \mathbf{w}', j, \gamma\rangle$ *which can be obtained by a Turing machine* **T**, *starting its computations on* $\langle \mathbf{w}, i, \beta\rangle$. *Moreover for each such a configuration* $\langle \mathbf{w}', j, \gamma\rangle$ *there exists a term producing an appropriate set.*

Thus, so far, the computations of the Turing machine are fully modelled by the terms with operations coming from the set $\mathcal{F} \cup \mathcal{G}$. Unfortunately this is not sufficient for modelling the computation of an alternating Turing machine. In order to decide whether an initial configuration of the Turing machine was accepting, we need to backtrack the computations propagating accepting configurations up the computation tree. This is done using the functions of the set $\mathcal{H}$ in a way very similar to modelling the computation using the elements of $\mathcal{G}$. To exhibit an analogue of Corollary 4.5.1 for the functions from the set $\mathcal{H}$ we introduce, for each configuration, a new set $P'_{\langle \mathbf{w},i,\beta\rangle}$. The $P_i$ part of the set is the same as of $P'_{\langle \mathbf{w},i,\beta\rangle}$ i.e.

$$(b, (I, J, D, a)) \in P'_{\langle \mathbf{w},i,\beta\rangle} \cap P_i \quad \text{if and only if} \quad \mathbf{w}(b) = a$$

and the second part of $P_{\langle \mathbf{w},i,\beta\rangle}$, from outside of $P_i$, is equal to $\{0, 1\}^n \times \{\beta'\}$. These sets allow us to backtrack the computations of the machine in the same way the previous sets were used to model them. Note that $P'_{\langle \mathbf{w},i,\beta\rangle} \cap P_i = P_{\langle \mathbf{w},i,\beta\rangle} \cap P_i$ and $P'_{\langle \mathbf{w},i,\omega\rangle} = P_{\langle \mathbf{w},i,\omega\rangle}$ for any **w**, $i$ and $\beta$.

We define $P^{\delta abD\pi:\langle \mathbf{w},i,\beta\rangle}_{\delta acE\tau:\langle \mathbf{w}',i',\gamma\rangle}$ to consist of the pairs $(c, d)$ such that

$$\exists(c, e) \in P'_{\langle \mathbf{w},i,\beta\rangle}, \exists(c, e') \in P'_{\langle \mathbf{w}',i',\gamma\rangle} \text{ such that } h^{\delta abD\pi}_{\delta acE\tau}(f^i_v(c, e), f^{i'}_{v'}(c, e')) = d$$

where $v = i-1$ if $D = R$ and $v = i+1$ if $D = L$ and $v' = i'-1$ if $E = R$ and $v' = i'+1$ if $E = L$. An analogue of Corollary 4.5.1 states

**Corollary 4.5.3.** *For any configurations* $\langle \mathbf{w}, i, \beta\rangle$, $\langle \mathbf{w}', i', \gamma\rangle$ *and any instructions of the machine* **T**: $\delta abD\pi$, $\delta acE\tau$ *and any four words* $u, v, u', v' \in \{0, 1\}^n$ *the following conditions are equivalent.*

(1) $h^{\delta abD\pi}_{\delta acE\tau}(f^u_v(c, e), f^{u'}_{v'}(c, e')) \neq\perp$, *for any* $(c, e) \in P'_{\langle \mathbf{w},i,\beta\rangle}$ *and any* $(c, e') \in P'_{\langle \mathbf{w}',i',\gamma\rangle}$,

(2) *the set*

$$P^{\delta abD\pi:\langle \mathbf{w},i,\beta\rangle}_{\delta acE\tau:\langle \mathbf{w}',i',\gamma\rangle} = P'_{\langle \mathbf{w}'',i'',\sigma\rangle}$$

*for some configuration* $\langle \mathbf{w}'', i'', \sigma\rangle$, *and* $u = i$, $u' = i'$ *and* $v = v'$ *and* $v = u - 1$ *if* $D = R$ *and* $v = u + 1$ *if* $D = L$ *and* $v' = u' - 1$ *if* $E = R$ *and* $v' = u' + 1$ *if* $E = L$

(3) *there exists a configuration* $\langle \mathbf{w}'', i'', \sigma\rangle$ *such that the instructions* $\delta abD\pi$ *and* $\delta acE\tau$ *can be executed in it producing* $\langle \mathbf{w}, i, \beta\rangle$ *and* $\langle \mathbf{w}', i', \gamma\rangle$ *respectively. Moreover if* $\delta$ *is a universal state, then* $\delta abD\pi$ *and* $\delta acE\tau$ *are the only two instructions that can be executed in a state* $\delta$ *with the head reading a, and if* $\delta$ *is an existential state then* $\delta abD\pi = \delta acE\tau$. *Finally,* $u = i$, $u' = i'$ *and* $v = v'$ *and* $v = u - 1$ *if* $D = R$ *and* $v = u + 1$ *if* $D = L$ *and* $v' = u' - 1$ *if* $E = R$ *and* $v' = u' + 1$ *if* $E = L$

*and the configurations denoted by* $\langle \mathbf{w}'', i'', \sigma\rangle$ *in 2 and 3 are equal.*

Proof of this corollary is a carbon copy of the proof of Corollary 4.5.1 using Corollary 4.3.4. Finally, using the corollary above together Corollary 4.5.2, we establish the final result of the construction. This result puts into a correspondence accepting computations of the Turing machine, and terms constructed from our functions.

**Corollary 4.5.4.** *Let $t(\bar{x}, y)$ be a term such that the outmost functional symbol is from $\mathcal{H}$. If $t(\bar{x}, y)$ defines a function satisfying Conditions [CI ] and [CII ] and such that $\perp \notin t(P_{\langle \mathbf{w}, i, \beta\rangle})$ for some configuration $\langle \mathbf{w}, i, \beta\rangle$, then the set*

$$\{(c, d) \mid (c, e) \in P \text{ and } t(c, e) = d\} \text{ equals } P'_{\langle \mathbf{w}', i', \gamma\rangle}$$

*for some configuration $\langle \mathbf{w}', i', \gamma\rangle$ which is an accepting configuration of a Turing machine* **T**. *Moreover for each accepting configuration which can be obtained working backwards from the accepting configurations obtained from $\langle \mathbf{w}, i, \beta\rangle$, there exists a term producing the appropriate set.*

## 4.6. The function for restricted problem

It remains to define, for each given input word $w$ of length $2^n - 2$, a function which is going to be expressed by a term if and only if, the starting configuration on the word $w$ is accepted. We accomplish this goal in two steps. First we introduce a *partial function* such that the $w$ is accepted if and only if, there exists a term which, restricted to given set of tuples, defines this function. Next we show that a problem of finding such a partial function is computationally equivalent to finding a full composition for a slightly different set of functions.

The function $s : \mathcal{U}^n \times \mathcal{V} \to \mathcal{U}$ is $(n+1)$-ary and as such the size of its description (approximately $C \cdot |\mathcal{U}|^{n+1}$) is polynomial with respect to the length on the input word which was equal to $2^n - 2$. The definition of the function implies immediately that such a function can be constructed in a polynomial time, and thus the problem of accepting a word will reduce, in a polynomial time, to the problem of composing functions.

Thus, for an input word $w$, we put $\langle \Diamond w \blacklozenge, 0^n, \alpha\rangle$ to be the starting configuration of the machine **T** and define $s : \mathcal{U}^n \times \mathcal{V} \to \mathcal{U}$ to be

$$s(b, a) = \begin{cases} (b, a) & \text{if } (b, a) \in P_{\langle \Diamond w \blacklozenge, 0^n, \alpha\rangle} \cap P_{0^n} \\ \alpha' & \text{if } a = \alpha \\ A & \text{if } b \in 1^*0^+ \text{ and } a = A \\ C & \text{if } b \in 1^*01^* \text{ and } a = C \\ \perp & \text{else,} \end{cases}$$

for $\mathcal{V} = \{(I, J, D, a) \in \mathcal{T} \mid I = J = T\} \cup \{\alpha\} \cup \{A, C\}$. The Corollary 4.5.4 immediately implies that if the function $s$ can be found as a restriction of the element of $\mathrm{Clo}^{\mathcal{U}}(\mathcal{F} \cup \mathcal{G} \cup \mathcal{H})$, then the word $w$ is accepted by the machine **T**. If, on the other hand, the word is accepted by **T** the same corollary provides us with an existence of a function, say $s'$, satisfying Conditions [CI ] and [CII ] and coinciding with $s$ on $P_{\langle \Diamond w \blacklozenge, 0^n, \alpha\rangle}$. It remains to show that

- $s'(b, A) = \perp$ whenever $b \notin 1^*0^+$,
- $s'(b, C) = \perp$ whenever $b \notin 1^*01^*$,
- $s'(b, (T, T, D, a)) = \perp$ whenever $(b, (T, T, D, a)) \notin P_{\langle \Diamond w \blacklozenge, 0^n, \alpha\rangle}$.

The first two points are obvious consequences of the definition of the functions on the set $\mathcal{P}$. For the last one we remark that for each $(b, (T, T, D, a)) \notin P_{\langle \Diamond w \blacklozenge, 0^n, \alpha\rangle}$ there exists $(b, (T, T, D', a')) \in P_{\langle \Diamond w \blacklozenge, 0^n, \alpha\rangle}$, and since the computation of **T** visits each square of the tape and the term does not evaluate to $\perp$ on $(b, (T, T, D', a'))$ it has to evaluate to $\perp$ on $(b, (T, T, D, a))$.

Thus we have proved the following, technical, proposition:

**Proposition 4.6.1.** *There exists a set $\mathcal{U}$ and a finite set of at most binary functions on it such that, for a fixed subset $\mathcal{V}$ of $\mathcal{U}$, it is EXPTIME-complete to decide whether a given function $s : \mathcal{U}^n \times \mathcal{V} \to \mathcal{U}$ can be obtained as a restriction of a composition of these functions.*

An example of such a set is the set $\mathcal{U}$ and the family of functions $\mathcal{F} \cup \mathcal{G} \cup \mathcal{H}$ which proves the hardness part of the proposition. The completeness is obtained by the algorithm proving completeness for the usual composition of functions and presented in e.g. [2]. It remains to show that a problem of finding a function, and not a restriction of the function, is as complex.

## 4.7. The general case

In this section we construct a set $\mathcal{W} = \mathcal{U} \cup \mathcal{V}' \cup \{\top\}$, which is a disjoint union of the three sets. Set $\mathcal{V}'$ consists of copies of the elements of $\mathcal{V}$ and there is a bijection $b \mapsto b'$ between $\mathcal{V}$ and $\mathcal{V}'$. We define the functions:

$$g_i(a_0, \ldots, a_{k_i - 1}) = \begin{cases} f_i(a_0, \ldots, a_{k_i - 1}) & \text{if } a_j \in \mathcal{U} \text{ for all } j < k_i \\ \top & \text{else} \end{cases}$$

for all $i \leq n$, and an additional function

$$g_{n+1}(a) = \begin{cases} b & \text{if } a = b' \in \mathcal{V}' \\ \top & \text{else.} \end{cases}$$

Finally we put

$$s''(a_0, \dots, a_{k-1}) = \begin{cases} s(a_0, \dots, a_{k-2}, a) & \text{if } a_0, \dots, a_{k-2} \in \mathcal{U} \text{ and } a_{k-1} = a' \in \mathcal{V}' \\ \top & \text{else.} \end{cases}$$

The construction immediately implies that $s'' \in \mathrm{Clo}^{\mathcal{U}}(g_0, \dots, g_n, g_{n+1})$ if and only if, the restricted composition problem for $s$ and $f_0, \dots, f_n$ had a positive solution. This proves the reduction of the restricted composition problem to the composition problem.

It is a trivial observation that all of the unary operations in our construction can be substituted by binary (putting $\top$, as a result of an application to non-identical arguments). This allows us to state the main theorem of the paper in a more uniform way.

**Theorem 4.7.1.** *There exists a finite set $\mathcal{W}$, and a family of functions $f_0, \dots, f_n : \mathcal{W}^2 \to \mathcal{W}$ such that the question whether a given function can be found as a composition of $f_0, \dots, f_n$ is EXPTIME-complete.*

## 5. Consequences

The authors in [2] tackle the problem denoted by TERM-EQ, that is

| | |
|---|---|
| INPUT | a pair of finite algebras (**A**, **B**) over the same set |
| PROBLEM | decide if Clo(**B**) = Clo(**A**). |

and prove that this problem is EXPTIME-complete. Our reasoning implies that the algebra **A** can be fixed (to be the algebra constructed in Section 4) and the computational complexity of the problem will not decrease (the reduction is obtained by taking **B**'s identical to A with added various functions $\mathcal{s}$). This proves that there is a finite algebra **A** such that the problem

| | |
|---|---|
| INPUT | a finite algebra **B** |
| PROBLEM | decide if Clo(**B**) = Clo(**A**). |

is EXPTIME-complete.

## Acknowledgement

The research was supported by Eduard Čech Center grant no. LC505.

## References

[1] José L. Balcázar, The complexity of searching implicit graphs, Artificial Intelligence 86 (1) (1996) 171–188, MR1410132 (98a:68088).
[2] Clifford Bergman, David Juedes, Giora Slutzki, Computational complexity of term-equivalence, Internat. J. Algebra Comput. 9 (1) (1999) 113–128, MR1695293 (2000b:68088).
[3] Stanley Burris, H. P. Sankappanavar, A course in universal algebra, in: Graduate Texts in Mathematics, vol. 78, Springer-Verlag, New York, 1981, MR648287 (83k:08001).
[4] Elwyn Berlekamp, David Wolfe, Mathematical Go, A K Peters Ltd., Wellesley, MA, 1994, Chilling gets the last point, With a foreword by James Davies. MR1274921 (95i:90131).
[5] Ján Černý, A remark on homogeneous experiments with finite automata, Mat.-Fyz. Časopis Sloven. Akad. Vied 14 (1964) 208–216, MR0168429 (29 #5692).
[6] Ashok K. Chandra, Dexter C. Kozen, Larry J. Stockmeyer, Alternation, J. Assoc. Comput. Mach. 28 (1) (1981) 114–133, MR603186 (83g:68059).
[7] Aviezri S. Fraenkel, David Lichtenstein, Computing a perfect strategy for $n \times n$ chess requires time exponential in $n$, J. Combin. Theory Ser. A 31 (2) (1981) 199–214, MR629595 (83b:68044).
[8] M.R. Garey, D.S. Johnson, Composing functions to minimize image size, SIAM J. Comput. 14 (2) (1985) 500–503, MR784752 (86d:68031).
[9] Marcel Jackson, Ralph McKenzie, Interpreting graph colorability in finite semigroups, Internat. J. Algebra Comput. 16 (1) (2006) 119–140, MR2217645 (2006m:20081).
[10] Marcin Kozik, Gábor Kun, The subdirectly irreducible algebras in the variety generated by graph algebras, Algebra Universalis 58 (2) (2008) 229–242, MR2386530 (08B26 (68Q17)).
[11] Dexter Kozen, Lower bounds for natural proof systems, in: 18th Annual Symposium on Foundations of Computer Science (Providence, R.I., 1977), IEEE Comput. Sci., Long Beach, Calif., 1977, pp. 254–266, MR0495200 (58 #13931).
[12] David Lichtenstein, Michael Sipser, GO is polynomial-space hard, J. Assoc. Comput. Mach. 27 (2) (1980) 393–401, MR567056 (81b:68052).
[13] Ralph McKenzie, The residual bound of a finite algebra is not computable, Internat. J. Algebra Comput. 6 (1) (1996) 29–48.
[14] Ralph N. McKenzie, George F. McNulty, Walter F. Taylor, Algebras, lattices, varieties, in: The Wadsworth & Brooks/Cole Mathematics Series, Vol. I, Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, CA, 1987, MR883644 (88e:08001).
[15] J. M. Robson, $N$ by $N$ checkers is Exptime complete, SIAM J. Comput. 13 (2) (1984) 252–267, MR739988 (86f:90168).
[16] Zoltán Székely, Computational complexity of the finite algebra membership problem for varieties, Internat. J. Algebra Comput. 12 (6) (2002) 811–823, MR1949698 (2003k:08009).