The Reachability Problem for Petri Nets is Not Primitive Recursive*

Jérôme Leroux
CNRS & University of Bordeaux
jerome.leroux@labri.fr

Abstract

We provide an Ackermannian complexity lower bound for the reachability problem for checking programs, a model equivalent to Petri nets. Moreover in fixed dimension $2d + 4$, we show that the problem is $\mathbb{F}_d$-hard. As a direct corollary, the reachability problem in dimension 10 is not elementary.

1 Introduction

Checking programs [14], or equivalently vector addition systems with states [7], or vector addition systems [8], or Petri nets are one of the most popular formal methods for the representation and the analysis of parallel processes [5].

Those equivalent models are acting on counters ranging over the natural numbers thanks to increment, decrement, and test commands (only at the end of an execution). The central algorithmic problem for checking programs is reachability: given a checking program, decide whether there exists an execution from an initial configuration to a final one. Many important computational problems in logic and complexity reduce or are even equivalent to this problem [21, 6]. After an incomplete proof by Sacerdote and Tenney [20], decidability of the problem was established by Mayr [15, 16], whose proof was then simplified by Kosaraju [9]. Building on the further refinements made by Lambert in the 1990s [10], in 2015, a first complexity upper bound of the reachability problem was provided [12] more than thirty years after the presentation of the algorithm introduced by Mayr [9, 10]. The upper bound given in that paper is "cubic Ackermannian", i.e. in $\mathbb{F}_{\omega^3}$ (see [22]). This complexity bound is obtained by analyzing the Mayr algorithm. With a refined algorithm and a new ranking function for proving termination, an Ackermannian complexity upper bound was obtained [13] This means that the reachability problem can be solved in time bounded by $A(p(n))$ where p is a primitive recursive function and where A is the Ackermann function. This paper also showed that the reachability problem in fixed dimension d (the dimension is the number of used counters) is primitive recursive by bounding the length of executions by $O(F_{d+4}(n))$ where F_{d+4} is a primitive recursive function of the Grzegorczyk hierarchy (see Section 2 for the definition of those functions), and n is the size of the input.

Concerning the complexity lower bound, in 1976, the reachability problem was proved to be ExpSpace-hard [2]. This bound used to be the best one for more than forty years until 2019 when it was improved to a Tower complexity lower bound [3], i.e. a non elementary complexity.

Contributions. In this paper we provide an Ackermannian complexity lower bound for the reachability problem for checking programs closing the gap with the Ackermannian complexity

[*] This research has been supported by ANR programme BraVAS (ANR-17-CE40-0028).

upper bound solving a 45 years old open problem. Moreover, in fixed dimension $2d+4$ with $d \geq 3$, we prove that the reachability problem is hard for the complexity class $\mathbb{F}_d$ introduced in [22] associated with the function F_d. As a direct corollary, we derive that the reachability problem is not elementary in dimension 10.

This paper provides the last piece leading to the exact complexity of the reachability problem for checking programs. As previously mentioned, it follows a long series of results. This piece of work is not the most difficult one but it is an important one since it closes a long standing open problem. Technically, the most difficult piece is the notion of K-amplifiers introduced in the Tower complexity lower bound paper [3] that provides a way to postpone at the end of an execution the tests commands of a Minsky machine (a machine like a VASS but with commands that can test to zero some counters) with counters bounded by K.

In this paper, we provide several gadgets for proving an $\mathbb{F}_d$ complexity lower bound for the reachability problem in dimension $2d+4$. Recently and independently, other gadgets for implementing such a bound have been introduced in order to obtain the same complexity upper bound by Wojciech Czerwiński, and Łukasz Orlikowski [4] in dimension $6d$, by myself [14] in dimension $4d+5$, and by Sławomir Lasota [11] in dimension $3d+2$. We do not know if the lower bound provided in this paper, i.e. $\mathbb{F}_d$ in dimension $2d+4$ is optimal since the complexity upper bound is $\mathbb{F}_{d+4}$ in dimension d. It follows that the parameterized complexity (i.e. in fixed dimension) of the reachability problem is still open.

We think that several different solutions to the complexity lower bound for the reachability problem is useful not only for the confidence in the claimed result but also for future work. In fact, the reachability problem for many extensions is open for almost all natural extensions except for vector addition systems with hierarchical zero tests [1, 19]. Moreover, the best known complexity lower bounds for those models only come from checking programs. Finding gadgets that take benefits from the extra power given by the considered extensions is an open problem.

Outline. In Section 2, we recall some properties satisfied by the fast growing functions F_d and introduce a way to compute F_d be iterating a reduction rule evalF_d. In Section 3 we introduce the model of general programs, and the subclasses of test-free models that correspond to programs that cannot test counters to zero, and the checking programs that can only test counters at the end. Whereas the reachability problem for test-free program is equivalent to the so-called *coverability problem* for Petri nets (see [2, 18] for complexity results), the reachability problem for checking program is equivalent to the so-called reachability problem for Petri nets. In Section 4 we provide tools to postpone at the end of an execution test commands of a general program. Those tools are used in Section 5 in order to simulate the bounded semantics of general programs thanks to the so-called preamplifiers. In Section 6 we provide tools for iterating a test-free program a fixed number of times that depends on the valuation of some counters. Those tools are used in Section 7 to implements evalF_d thanks to a test-free program. By iterating this test-free program, we provide a way in Section 8 to implement an Ackermannian preamplifier. Finally, in Section 9 we collect intermediate results to provide complexity lower bounds for the reachability problem for checking programs.

2 Fast Growing Functions

We introduce the sequence $(F_d)_{d \in \mathbb{N}}$ of functions $F_d : \mathbb{N} \to \mathbb{N}$ defined by $F_0(n) = n+1$, and defined by induction on $d \geq 1$ by $F_d(n) = F_{d-1}^{n+1}(n)$. Rather than take the original definition of *Ackermann function*, we let $A(n) = F_\omega(n)$ defined by $F_\omega(n) = F_{n+1}(n)$, which behaves like the classical function for our complexity-theoretical purpose. We introduce the function $F_v : \mathbb{N} \to \mathbb{N}$ with $v \in \mathbb{N}^d$, defined as follows for every $n \in \mathbb{N}$:

$$F_v(n) = F_d^{v[d]} \circ \cdots \circ F_1^{v[1]}(n)$$

Remark 1. We have $F_1(n) = 2n+1$ and $F_2(n) = 2^{n+1}(n+1)-1$ for every $n \in \mathbb{N}$. The function F_3 behaves like a tower of n exponential and it is not elementary. Each function F_d is primitive recursive and F_ω is not primitive recursive.

Let us denote by 0_d the zero vector of $\mathbb{N}^d$, and by $1_{d,i}$ the ith unit vector of $\mathbb{N}^d$ defined for every $1 \leq j \leq d$ by $1_{d,i}[j] = 1$ if $j = i$ and $1_{d,i}[j] = 0$ otherwise. The lexicographic (strict) order $<_{\text{lex}}$ over $\mathbb{N}^d$ is defined by $w <_{\text{lex}} v$ if $w \neq v$ and the maximal $p \in \{1, \dots, d\}$ such that $w[p] \neq v[p]$ satisfies $w[p] < v[p]$.

Let $\text{evalF}_d : \mathbb{N}^d \times \mathbb{N} \to \mathbb{N}^d \times \mathbb{N}$ be the function partially defined over the pairs $(v, n) \in \mathbb{N}^d \times \mathbb{N}$ such that $v \neq 0_d$ as follows where p is the minimal index in $\{1, \dots, d\}$ such that $v[p] > 0$:

$$\text{evalF}_d(v,n) = \begin{cases} (v - 1_{d,p}, 2n+1) & \text{if } p = 1 \\ (v - 1_{d,p} + (n+1)1_{d,p-1}, n) & \text{if } p > 1 \end{cases}$$

Since $F_p(n) = F_{p-1}^{n+1}(n)$, one can easily prove (see [23] for more details) that if $v \neq 0_d$, then the pair (w, m) defined as $\text{evalF}_d(v, n)$ satisfies $F_w(m) = F_v(n)$. It follows that the function that maps (v, n) onto $F_v(n)$ is an invariant of evalF_d.

Moreover, as $w <_{\text{lex}} v$, it follows that we can iterate the function evalF_d on a pair (v, n) only a finite number of times (we use the well-foundedness of the lexicographic order). Let us introduce the function $\text{evalF}_d^{\max} : \mathbb{N}^d \times \mathbb{N} \to \mathbb{N}^d \times \mathbb{N}$ defined by $\text{evalF}_d^{\max}(v, n) = \text{evalF}_d^k(v, n)$ where k is the maximal number of times (it can be zero) we can apply the function evalF_d on (v, n). Notice that $\text{evalF}_d^{\max}(v, n)$ is a pair of the form (w, m) for some pair $(w, m) \in \mathbb{N}^d \times \mathbb{N}$. Moreover, by maximality of k, it follows that $w = 0_d$. Since the function $(v, n) \mapsto F_v(n)$ is an invariant of evalF_d, we deduce that $F_{0_d}(m) = F_v(n)$. It follows that the following equality holds (see [23] for more details):

$$\text{evalF}_d^{\max}(v, n) = (0_d, F_v(n))$$

As a direct corollary, we deduce the following lemma.

Lemma 2. *We have* $\text{evalF}_d^{\max}((n+1)1_{d,d}, n) = (0_d, F_{d+1}(n))$ *for every* $n \geq 0$ *and* $d \geq 1$.

The following lemma will be useful in the sequel. In that lemma, $|v|$ is $v[1] + \cdots + v[d]$ for any vector $v \in \mathbb{N}^d$.

Lemma 3. *We have* $F_v(n) \geq 2^{|v|}n + |v|$ *for every* $v \in \mathbb{N}^d$ *and for every* $n \in \mathbb{N}$.

Proof. Let $v \in \mathbb{N}^d$ and $n \in \mathbb{N}$. Assume that for every $w \in \mathbb{N}^d$ such that $w <_{\text{lex}} v$ we have $F_w(m) \geq 2^{|w|}m + |w|$ for every $m \in \mathbb{N}$. And let us prove that in that case $F_v(n) \geq 2n^{|v|} + |v|$ for every $n \in \mathbb{N}$. In fact, since the lexicographic order is well-founded, by induction the proof of the lemma reduces to that statement.

Notice that if $v = 0_d$ then $F_v(n) = n$ and we are done. So, we can assume that $v \neq 0_d$. In that case, let us introduce $(w, m) = \text{evalF}_d(v, n)$. Since $w <_{\text{lex}} v$, it follows that $F_w(m) \geq 2^{|w|}m + |w|$. Since $F_w(m) = F_v(n)$, we get $F_v(n) \geq 2^{|w|}m + |w|$. Let $p \in \{1, \dots, d\}$ be the minimal index such that $v[p] > 0$. Observe that if $p = 1$ then $|w| = |v| - 1$ and $m = 2n+1$. In particular $2^{|w|}m + |w| \geq 2^{|v|}n + |v|$. If $p > 1$ then $|w| = |v| + n$ and $m = n$. Hence $2^{|w|}m + |w| \geq 2^{|v|}n + |v|$. We have proved that $F_v(n) \geq 2^{|v|}n + |v|$ in any case. □

3 General Programs

We introduce in the section the formal model of programs acting on counters ranging over the natural numbers.

Formally, an implicit infinite countable set of elements called *counters* is given. A *configuration* is a function ρ that maps every counter on a natural number in such a way that the set of counters c such that $\rho(\mathsf{c}) \neq 0$ is finite. We denote by 0 the configuration ρ such that $\rho(\mathsf{c}) = 0$ for every counter c. The *increment, decrement and test commands* of a counter c are respectively denoted as $\mathbf{inc}(\mathsf{c})$, $\mathbf{dec}(\mathsf{c})$, and $\mathbf{test}(\mathsf{c})$. We associate with such a command **cmd**, the binary relation $\xrightarrow{\mathbf{cmd}}$ over the configurations defined by $\alpha \xrightarrow{\mathbf{cmd}} \beta$ if $\alpha(\mathsf{x}) = \beta(\mathsf{x})$ for every counter $\mathsf{x} \neq \mathsf{c}$, and satisfying additionally:

$$\begin{cases} \beta(\mathsf{c}) = \alpha(\mathsf{c}) + 1 & \text{if } \mathbf{cmd} \text{ is } \mathbf{inc}(\mathsf{c}) \\ \beta(\mathsf{c}) = \alpha(\mathsf{c}) - 1 & \text{if } \mathbf{cmd} \text{ is } \mathbf{dec}(\mathsf{c}) \\ \beta(\mathsf{c}) = 0 = \alpha(\mathsf{c}) & \text{if } \mathbf{cmd} \text{ is } \mathbf{test}(\mathsf{c}) \end{cases}$$

A *(general) program* M is defined as an increment/decrement/test command, or inductively as a loop program $\mathbf{loop}\ M_0$, a series composition $M_1; M_2$, or a non-deterministic choice M_1 **or** M_2 where M_0, M_1, M_2 are programs. The *size* of a program M is the number $\text{size}(M)$ defined inductively as 1 if M is a command, $1 + \text{size}(M_0)$ if $M = \mathbf{loop}\ M_0$, and $1 + \text{size}(M_1) + \text{size}(M_2)$ if $M = M_1; M_2$ or $M = M_1$ **or** M_2. The *dimension* of M is the cardinal of the set of counters used by M, and formally defined as expected. We associate with every program M the binary relation $\xrightarrow{M}$ over the configurations defined as follows:

$$\xrightarrow{M} \quad = \quad \begin{cases} \xrightarrow{\mathbf{cmd}} & \text{if } M = \mathbf{cmd} \text{ is a command} \\ (\xrightarrow{M_0})^* & \text{if } M = \mathbf{loop}\ M_0 \\ \xrightarrow{M_1};\xrightarrow{M_2} & \text{if } M = M_1; M_2 \\ \xrightarrow{M_1} \cup \xrightarrow{M_2} & \text{if } M = M_1 \textbf{ or } M_2 \end{cases}$$

Where $(\rightarrow_0)^*$ is the reflexive and transitive closure of $\rightarrow_0$, and $\rightarrow_1; \rightarrow_2$ is defined by $\alpha \rightarrow_1; \rightarrow_2 \beta$ if there exists a configuration ρ such that $\alpha \rightarrow_1 \rho$ and $\rho \rightarrow_2 \beta$. Series compositions and non-deterministic choices are clearly associative with respect to the relation $\rightarrow$. In particular given a sequence $M_1, \ldots, M_n$ of Minsky programs, the relations $\xrightarrow{M_1;\ldots;M_n}$ and $\xrightarrow{M_1 \textbf{ or } \ldots \textbf{ or } M_n}$ are well defined. We denote by $M^{(n)}$ the series composition of M by itself n times. We also denote by $\mathbf{inc}(\mathsf{c}_1, \ldots, \mathsf{c}_n)$ the program $\mathbf{inc}(\mathsf{c}_1); \ldots; \mathbf{inc}(\mathsf{c}_n)$, by $\mathbf{dec}(\mathsf{c}_1, \ldots, \mathsf{c}_n)$ the program $\mathbf{dec}(\mathsf{c}_1); \ldots; \mathbf{dec}(\mathsf{c}_n)$, and similarly $\mathbf{test}(\mathsf{c}_1, \ldots, \mathsf{c}_n)$ the program $\mathbf{test}(\mathsf{c}_1); \ldots; \mathbf{test}(\mathsf{c}_n)$.

We say that a program is *test-free* if it does not use any test command. A *checking program* is a program of the form $M; \mathbf{test}(\mathsf{c}_1, \ldots, \mathsf{c}_n)$ where M is a test-free program and $\mathsf{c}_1, \ldots, \mathsf{c}_n$ are some counters.

The *reachability problem* for general programs asks, given a general program M, whether there exists a configuration β such that $0 \xrightarrow{M} \beta$. This problem is undecidable [17] even in dimension 2 but provides a way to define complexity classes beyond Elementary as recalled in Section 9. The reachability problem for test-free programs is equivalent (i.e. inter-reducible) to the so-called coverability problem for Petri nets (see [2, 18] for the complexity of the problem), and the reachability problem for checking programs is equivalent (i.e. inter-reducible) to the so-called reachability problem for Petri nets.

In this paper, we use standard notions from model theory by considering counters as variables, and configurations as valuations of the variables. It means that a configuration is used to replace in an expression e over the counters, each occurrence of a counter c by $\rho(\mathsf{c})$. We denote by $\rho(e)$ the expression we obtain this way. When ϕ is a constraint over the counters, we also denote by $\rho(\phi)$ the constraint we obtain by considering ϕ as an expression over the counters. We say that ρ satisfies ϕ if the constraint $\rho(\phi)$ is true. For instance we say that ρ satisfies $\mathsf{y} = 2^{\mathsf{c}}\mathsf{x}$ where y, c and x are counters if $\rho(\mathsf{y}) = 2^{\rho(\mathsf{c})}\rho(\mathsf{x})$.

4 Simulating Test Commands

We provide in this section the central idea for simulating test commands. Assume that B denotes a finite set of counters. The simulation of a test command $\mathbf{test}(\mathsf{c})$ for some counter $\mathsf{c} \in \mathsf{B}$ is using two distinct auxiliary counters x and y not in B. During the simulation, we consider configurations ρ satisfying $\mathsf{y} \geq K\mathsf{x}$ where $K = \rho(\sum_{\mathsf{b}\in\mathsf{B}} \mathsf{b})$. Intuitively, when the configuration satisfies $\mathsf{y} = K\mathsf{x}$ it means that all the previous simulation of test commands were correct and when $\mathsf{y} > K\mathsf{x}$ is means that at least one simulation was incorrect. Those simulations are obtained by introducing the following test-free program. This program is using any implicit enumeration $\mathsf{b}_0, \ldots, \mathsf{b}_d$ of the counters in B satisfying $\mathsf{b}_0 = \mathsf{c}$. Such an enumeration naturally depends on the counter c.

$$\mathbf{simtest}_{\mathsf{y},\mathsf{B},\mathsf{x}}(\mathsf{c}) \quad = \quad \left|\begin{array}{l} \mathbf{loop}\\ \quad \mathbf{dec}(\mathsf{b}_1);\mathbf{inc}(\mathsf{b}_0);\mathbf{dec}(\mathsf{y})\\ \vdots\\ \mathbf{loop}\\ \quad \mathbf{dec}(\mathsf{b}_d);\mathbf{inc}(\mathsf{b}_{d-1});\mathbf{dec}(\mathsf{y})\\ \mathbf{loop}\\ \quad \mathbf{dec}(\mathsf{b}_{d-1});\mathbf{inc}(\mathsf{b}_d);\mathbf{dec}(\mathsf{y})\\ \vdots\\ \mathbf{loop}\\ \quad \mathbf{dec}(\mathsf{b}_0);\mathbf{inc}(\mathsf{b}_1);\mathbf{dec}(\mathsf{y})\\ \mathbf{dec}(\mathsf{x})^{(2)} \end{array}\right.$$

Intuitively this program is transferring the content of the counter b_i into b_{i-1} thanks to the ith loop with $i \in \{1, \ldots, d\}$. Then the program is transferring back the contents of the counters thanks to the last d loops. During each step of the transfer, the counter y is decremented. Since $K = \sum_{\mathsf{b}\in\mathsf{B}} \mathsf{b}$ is an invariant, the counter y can be decremented by at most K in total with the first d loops, and by K as well in total with the last d loops. It follows that y can be decremented by at most $2K$. Moreover, the only way to decrement y by exactly $2K$ is when initially $\mathsf{c} = 0$ and when all loops are executed the maximal number of times, meaning that the transfers are total, and in particular that the initial and the final configurations coincide on the counters in B. Since x is decremented by 2, it follows that if $\mathsf{y} \geq K\mathsf{x}$ before the execution of the program, then the same constraint holds at the end. Moreover, if additionally $\mathsf{y} = K\mathsf{x}$ at the end, then necessarily the same constraint holds at the beginning and it means that y has been decremented by $2K$, and in particular initially $\mathsf{c} = 0$ and all the transfers were performed maximally. The following two lemmas formally proved those intuitions.

Lemma 4. *Let α, β be two configurations such that $\alpha \xrightarrow{\textit{test}(\mathsf{c});\textit{dec}(\mathsf{y})^{(2K)}\textit{dec}(\mathsf{x})^{(2)}} \beta$ with $\mathsf{c} \in \mathsf{B}$ and such that $K = \alpha(\sum_{\mathsf{b}\in\mathsf{B}} \mathsf{b})$. Then we have:*

$$\alpha \xrightarrow{\textit{simtest}_{\mathsf{y},\mathsf{B},\mathsf{y}}(\mathsf{c})} \beta$$

Proof. We are going to prove that there exists an execution in such a way the loops are executed $m_1, \ldots, m_d, n_d, \ldots, n_1$ times (in that order), where $m_i = n_i = \alpha(\mathsf{b}_i)$ for every $1 \leq i \leq d$. Notice that the decrement commands $\mathbf{dec}(\mathsf{y})$ cannot prevent such an execution since $\alpha(\mathsf{y}) \geq 2K = \sum_{i=1}^{d}(n_i + m_i)$. Moreover, the decrement commands $\mathbf{dec}(\mathsf{x})^{(2)}$ are also executable at the end of the program since $\alpha(\mathsf{x}) \geq 2$. So, for proving the existence of an execution of the program with the loops executed the right number of times, we can forget the decrement commands on the counters y and x.

For the first d loops, such an execution is possible by observing that the ith loop with $i \in \{1, \ldots, d\}$ only decrements the counter b_i (recall that we forget the counter y for the proof) that is untouched by the previous loops. After executing those d first loops, we get a configuration ρ satisfying for every counter c the following equalities:

$$\rho(\mathsf{c}) = \begin{cases} \alpha(\mathsf{b}_{i+1}) & \text{if } \mathsf{c} = \mathsf{b}_i \text{ with } 0 \leq i < d \\ 0 & \text{if } \mathsf{c} = \mathsf{b}_d \\ \alpha(\mathsf{y}) - K & \text{if } \mathsf{c} = \mathsf{y} \\ \alpha(\mathsf{c}) & \text{otherwise} \end{cases}$$

Starting form ρ, the last d loops are symmetrical to the first d loops. In particular, the same argument as previously mentioned shows that there exists an execution from ρ that executes the last d loops the right number of times. Now, just observe that with such an execution, we have proved the lemma. □

Lemma 5. *Let α, β be two configurations such that $\alpha \xrightarrow{\textit{simtest}_{\mathsf{y},\mathsf{B},\mathsf{x}}(\mathsf{c})} \beta$ and let $K = \alpha(\sum_{\mathsf{b} \in \mathsf{B}} \mathsf{b})$. Then $\beta(\sum_{\mathsf{b} \in \mathsf{B}} \mathsf{b}) = K$ and if α satisfies $\mathsf{y} \geq K\mathsf{x}$ then β satisfies the same constraint. Moreover if additionally β satisfies $\mathsf{y} = K\mathsf{x}$ then α satisfies the same constraint and:*

$$\alpha \xrightarrow{\textit{test}(\mathsf{c});\textit{dec}(\mathsf{y})^{(2K)}\,\textit{dec}(\mathsf{x})^{(2)}} \beta$$

Proof. Let us denote by $m_1, \ldots, m_d, n_d, \ldots, n_1$ the number of times loops of the program are executed (in that order). Observe that the ith loop with $i \in \{1, \ldots, d\}$ decrements the counter b_i, and that counter is untouched by the previous loops. It follows that there exists a sequence $r_1, \ldots, r_d$ of natural numbers such that $m_i = \alpha(\mathsf{b}_i) - r_i$ for every $i \in \{1, \ldots, d\}$. We derive the following equality by observing that $\sum_{i=0}^{d} \alpha(\mathsf{b}_i) = K$:

$$\sum_{i=1}^{d} m_i = K - \alpha(\mathsf{b}_0) - \sum_{i=1}^{d} r_i$$

Let us denote by ρ the configuration we obtain just after executing those d first loops. Since $\sum_{i=0}^{d} \mathsf{b}_i$ is an invariant of every line of the program, we get $\beta(\sum_{i=0}^{d} \mathsf{b}_i) = K$ and $\rho(\sum_{i=0}^{d} \mathsf{b}_i) = K$. Now, observe that the argument used in the previous paragraph holds for the last d loops. It follows that there exists a sequence $s_1, \ldots, s_d$ of natural numbers such that $n_i = \rho(\mathsf{b}_{i-1}) - s_i$ for every $1 \leq i \leq d$, and we derive the following equality:

$$\sum_{i=1}^{d} n_i = K - \rho(\mathsf{b}_d) - \sum_{i=1}^{d} s_i$$

From $\beta(\mathsf{y}) = \alpha(\mathsf{y}) - \sum_{i=1}^{d}(m_i + n_i)$ and $\beta(\mathsf{x}) = \alpha(\mathsf{x}) - 2$, we deduce the following equalities:

$$\begin{aligned} \beta(\mathsf{y} - K\mathsf{x}) &= \alpha(\mathsf{y} - K\mathsf{x}) + 2K - \sum_{i=1}^{n}(m_i + n_i) \\ &= \alpha(\mathsf{y} - K\mathsf{x}) + \alpha(\mathsf{b}_0) + \rho(\mathsf{b}_d) + \sum_{i=1}^{d}(r_i + s_i) \end{aligned}$$

Now, assume that α satisfies $\mathsf{y} \geq K\mathsf{x}$. From the previous equality, we deduce that $\beta(\mathsf{y} - K\mathsf{x})$ is a sum of natural numbers. In particular β satisfies $\mathsf{y} \geq K\mathsf{x}$. If additionally β satisfies $\mathsf{y} = K\mathsf{x}$ the previous equality shows that $\alpha(\mathsf{y} - K\mathsf{x}) = 0$, $\alpha(\mathsf{b}_0) = 0$, $\rho(\mathsf{b}_d) = 0$, and $r_i = s_i = 0$ for every $1 \leq i \leq d$. From $r_i = 0$ for every $1 \leq i \leq d$, it follows that $m_i = \alpha(\mathsf{c}_i)$ for every $1 \leq i \leq d$. We deduce that $\rho(\mathsf{b}_i) = \alpha(\mathsf{b}_{i+1})$ for every $0 \leq i < d$. Now, from $s_i = 0$ for every $1 \leq i \leq d$, we deduce that $n_i = \alpha(\mathsf{b}_{i+1})$ for every $1 \leq i \leq d$. In particular $\beta(\mathsf{b}_i) = \alpha(\mathsf{b}_i)$ for every $0 \leq i \leq d$. Finally, just observe that from $\sum_{i=1}^{d} m_i = K$ and $\sum_{i=1}^{d} n_i = K$, we deduce that $\beta(\mathsf{y}) = \alpha(\mathsf{y}) - 2K$. We have proved the lemma. □

5 Bounded Semantics and Preamplifiers

We introduce in this section the K-bounded semantics of general programs, where K is a natural number. Intuitively this semantics is obtained by bounding the sum of the counters by K. We also introduce the notion of K-preamplifiers that provides a way to simulate the K-bounded semantics of general programs thanks to checking programs equipped with the classical (unbounded) semantics.

More formally, we say that a configuration ρ is K-bounded for some $K \in \mathbb{N}$ if $\sum_{\mathsf{c}} \rho(\mathsf{c}) \leq K$. Denoting by $\mathrm{Conf}_{\leq K}$ the set of K-bounded configurations, we define the binary relation $\xrightarrow{M}_{\leq K}$ over $\mathrm{Conf}_{\leq K}$ where M is a general program inductively as follows:

$$\xrightarrow{M}_{\leq K} \quad = \quad \begin{cases} (\xrightarrow{\mathbf{cmd}}) \cap (\mathrm{Conf}_{\leq K} \times \mathrm{Conf}_{\leq K}) & \text{if } M = \mathbf{cmd} \text{ is a command} \\ (\xrightarrow{M_0}_{\leq K})^* \cap (\mathrm{Conf}_{\leq K} \times \mathrm{Conf}_{\leq K}) & \text{if } M = \mathbf{loop}\ M_0 \\ \xrightarrow{M_1}_{\leq K} ; \xrightarrow{M_2}_{\leq K} & \text{if } M = M_1; M_2 \\ \xrightarrow{M_1}_{\leq K} \cup \xrightarrow{M_2}_{\leq K} & \text{if } M = M_1 \ \mathbf{or}\ M_2 \end{cases}$$

Intuitively $\alpha \xrightarrow{M}_{\leq K} \beta$ for two configurations α, β if, and only if, there exists an execution of M such that every visited configuration including α and β, during the computation is K-bounded. The relation $\xrightarrow{M}_{\leq K}$ is called the *K-bounded semantics* of M.

The K-bounded semantics can be simulated by checking programs thanks to the so-called K-preamplifiers. A K-preamplifier [3] for a triple of counters $(\mathsf{x}, \mathsf{y}, \mathsf{b})$ is a checking program A such that:

- For every configuration β such that $0 \xrightarrow{A} \beta$ we have $\mathsf{y} \geq \mathsf{bx}$ and $\beta(\mathsf{c}) = 0$ for every counter $\mathsf{c} \not\in \{\mathsf{x}, \mathsf{y}, \mathsf{b}\}$. Moreover, if β satisfies $\mathsf{y} = \mathsf{bx}$ then $\beta(\mathsf{b}) = K$.
- For every $\ell \geq 1$ there exists a configuration β satisfying $0 \xrightarrow{A} \beta$, $\mathsf{y} = \mathsf{bx}$, and $\mathsf{x} = \ell$.

Remark 6. A K-amplifier [3] for a triple of counters $(\mathsf{x}, \mathsf{y}, \mathsf{b})$ is a checking program A such that for any configuration β, we have $0 \xrightarrow{A} \beta$ if, and only if, there exists $\ell > 0$ such that $\beta(\mathsf{x}, \mathsf{y}, \mathsf{b}) = (\ell, K\ell, K)$ and such that $\beta(\mathsf{c}) = 0$ for any counter $\mathsf{c} \not\in \{\mathsf{x}, \mathsf{y}, \mathsf{b}\}$. From a K-preamplifier, one can compute a K-amplifier by introducing some additional counters. We do not introduce K-amplifiers in this paper in order to reduce the number of counters used by the simulation.

Given a K-preamplifier A and a general program M of dimension d, we can compute in time $\mathrm{size}(A) + O(d\,\mathrm{size}(M))$ a checking program $A \rhd M$ such that for any configuration β, we have $0 \xrightarrow{M}_{\leq K} \beta$, if, and only if, $0 \xrightarrow{A \rhd M} \beta$. It follows that K-preamplifiers provide a way to postpone at the end of an execution test commands of general programs equipped with the K-bounded semantics. The size of $A \rhd M$ is $\mathrm{size}(A) + O(d\,\mathrm{size}(M))$. Concerning the dimension of $A \rhd M$, let us first classify the counters used by A. We say that a counter c used by the preamplifier A is *unsafe* (for the simulation), if it belongs to $\{\mathsf{x}, \mathsf{y}, \mathsf{b}\}$ or if it occurs in a test command at the end of A, and let us say it is *safe* if it is not unsafe. Then denoting by u and s respectively the number of unsafe and safe counters of A, the dimension of $A \rhd M$ is equal to $u + \max(s, d)$.

The checking program $A \rhd M$ is obtained as follows. By renaming the counters of A, we can assume without loss of generality that the unsafe counters of A are disjoint from the counters used by M. Moreover, with such a renaming we can additionally assume that the cardinal of the safe counters of A union the counters used by M is $\max(s, d)$. We assume that A is a checking program of the form $A'; \mathbf{test}(\mathsf{c}_1, \ldots, \mathsf{c}_n)$ where A' is a test-free program. We can assume that b is not in $\{\mathsf{c}_1, \ldots, \mathsf{c}_n\}$ since otherwise $K = 0$ and in this case A can by replaced by the 0-preamplifier **loop inc**(x). We denote by B the counters used by M union $\{\mathsf{b}\}$. By adding a test command **test**(c) for some counter c used by M, we can assume that M starts with a test command (notice that if M is not using any counter, then M is the empty program and the construction $A \rhd M$ can be defined as M).

We introduce the test-free program M' obtained from M by replacing each increment command $\textbf{inc}(\mathsf{c})$ by $\textbf{inc}(\mathsf{c});\textbf{dec}(\mathsf{b})$, each decrement command $\textbf{dec}(\mathsf{c})$ by $\textbf{dec}(\mathsf{c});\textbf{inc}(\mathsf{b})$, and each test command $\textbf{test}(\mathsf{c})$ by $\textbf{simtest}_{\mathsf{x},\mathsf{B},\mathsf{y}}(\mathsf{c})$. The checking program $A \rhd M$ is then defined as follows:

$$A \rhd M \quad = \quad \left| \begin{array}{l} \text{A'} \\ \text{M'} \\ \textbf{loop} \\ \quad \textbf{dec}(\mathsf{b}) \\ \textbf{test}(\mathsf{x},\mathsf{y},\mathsf{b},\mathsf{c}_1,\ldots,\mathsf{c}_n) \end{array} \right.$$

For every configuration β, we have $0 \xrightarrow{M}_{\leq K} \beta$ if, and only if, $0 \xrightarrow{A \rhd M} \beta$. A formal proof for a variant construction of $A \rhd M$ (that cannot reuse safe counters of A, that introduces several additional test commands, and that uses additional counters) is given in [3]. In the next two paragraphs, we just recall briefly the key ingredients used for the formal proof.

For one direction, assume that $0 \xrightarrow{M}_{\leq K} \beta$ for some configuration β and let us consider an execution of M from 0 to β such that every visited configuration including β is K-bounded. Denoting by m the number of times this execution is using a test command, we introduce $\ell = 2m$. Since M starts with a test command, it follows that $\ell > 0$. We consider an execution of A' that leads to a configuration ρ such that $\rho(\mathsf{x},\mathsf{y},\mathsf{b}) = (\ell, K\ell, K)$ and $\rho(\mathsf{c}) = 0$ for every counter $\mathsf{c} \not\in \{\mathsf{x},\mathsf{y},\mathsf{b}\}$. Observe that ρ satisfies $K = \rho(\sum_{\mathsf{c}\in\mathsf{B}} \mathsf{c})$. From the execution of M, we derive an execution of M' from ρ to a configuration δ satisfying $\delta(\mathsf{x}) = 0$, $\delta(\mathsf{y}) = 0$, and $\delta(\mathsf{c}) = \beta(\mathsf{c})$ for every counter c used by M. In fact, every time a test command $\textbf{test}(\mathsf{c})$ is executed, it is simulated by $\textbf{simtest}_{\mathsf{x},\mathsf{B},\mathsf{y}}(\mathsf{c})$ using Lemma 4 that decrements x by 2, and y by $2K$. Finally, from δ we iterate the last loop exactly $\delta(\mathsf{b})$ times in such a way we get the configuration β. This configuration can then execute the last test commands of $A \rhd M$.

For the other direction, assume that $0 \xrightarrow{A \rhd M} \beta$ for some configuration β and let us consider an execution of $A \rhd M$ witnessing that property. In $A \rhd M$ the test-free program A' is executed first and the test commands $\textbf{test}(\mathsf{c}_1,\ldots,\mathsf{c}_n)$ of A are postponed at the end of $A \rhd M$. Since those tested counters are no longer used in between, the execution of A' can only produce from the zero configuration a configuration ρ satisfying $\mathsf{y} \geq \mathsf{b}\mathsf{x}$ and $\rho(\mathsf{c}) = 0$ for every counter $\mathsf{c} \not\in \{\mathsf{x},\mathsf{y},\mathsf{b}\}$. We denote by δ the configuration obtained from ρ after executing M'. Let $K' = \rho(\mathsf{b})$. Assume by contradiction that ρ satisfies the strict constraint $\mathsf{y} > K'\mathsf{x}$. Lemma 5 shows that δ satisfies the same strict inequality. In particular $\beta(\mathsf{y}) = \delta(\mathsf{y}) > 0$ and the last test command $\textbf{test}(\mathsf{y})$ fails on β. We get a contradiction. It follows that δ satisfies the equality $\mathsf{y} = \mathsf{b}\mathsf{x}$ and since A is a K-preamplifier, we get $K' = K$ in that case. Lemma 5 shows that δ satisfies $\mathsf{y} \geq K\mathsf{x}$. Moreover, since $\beta(\mathsf{y}) = \beta(\mathsf{x}) = 0$ we deduce that δ satisfies the equality $\mathsf{y} = K\mathsf{x}$. From Lemma 5 we deduce that every execution of $\textbf{simtest}_{\mathsf{x},\mathsf{B},\mathsf{y}}(\mathsf{c})$ in M' has the same effect as the execution of $\textbf{test}(\mathsf{c});\textbf{dec}(\mathsf{y})^{(2K)};\textbf{dec}(\mathsf{x})^{(2)}$. From the execution of M' we deduce that $\alpha \xrightarrow{M}_{\leq K} \beta$.

It follows that the K-bounded semantics of general programs can be simulated by checking programs of small size as soon as there exists small size K-preamplifiers.

6 Loop at Most

In this section we present a way to iterate test-free programs a numbers of times that depends on the valuation of some counters. Given two distinct counters c and c', and a test-free program M, we introduce the following test-free program:

```
Loop at most c + c' times M  =  | loop
                                |     dec(c);inc(c')
                                | loop
                                |     dec(c');inc(c);M
```

Let us denote by C the counter expression $\mathsf{c}+\mathsf{c}'$. In the following lemmas, we assume that M is any test-free program that does not use the counters c and c'.

Lemma 7. *For every configuration α, β, we have:*

$$\alpha \xrightarrow{\textbf{\textit{Loop at most}}\ \mathsf{c}+\mathsf{c}'\ \textbf{\textit{times}}\ M} \beta$$

if, and only if, there exists $n, \ell \in \mathbb{N}$ such that $n \leq \alpha(\mathsf{c})$, $\ell \leq n+\alpha(\mathsf{c}')$, and such that:

$$\alpha \xrightarrow{(\textbf{\textit{dec}}(\mathsf{c});\textbf{\textit{inc}}(\mathsf{c}'))^{(n)};(\textbf{\textit{dec}}(\mathsf{c}');\textbf{\textit{inc}}(\mathsf{c});M)^{(\ell)}} \beta$$

In particular $\ell \leq \alpha(\mathsf{C})$. If this inequality is an equality then $\beta(\mathsf{c}) = \alpha(\mathsf{C})$ and $\beta(\mathsf{c}') = 0$.

Proof. We denote by N and $N_{n,\ell}$ the following test-free programs where $n, \ell \in \mathbb{N}$:

```
N  =  | 1: loop
      | 2:     dec(c);inc(c')
      | 3: loop
      | 4:     dec(c');inc(c);M
```

$$N_{n,\ell} \quad = \quad \left|\ \begin{array}{l} (\mathbf{dec}(\mathsf{c});\mathbf{inc}(\mathsf{c}'))^{(n)} \\ (\mathbf{dec}(\mathsf{c}');\mathbf{inc}(\mathsf{c});M)^{(\ell)} \end{array}\right.$$

Let α, β be two configurations such that $\alpha \xrightarrow{N} \beta$. We fix some execution witnessing that property. Let n be the number of times line 2 is executed and ℓ be the number of times line 4 is executed. Since each execution of line 2 decrements c, we deduce that after executing the first loop we get a configuration ρ satisfying $\rho(\mathsf{c}) = \alpha(\mathsf{c}) - n$ and $\rho(\mathsf{c}') = \alpha(\mathsf{c}') + n$. In particular $n \leq \alpha(\mathsf{c})$. Symmetrically, since each execution of line 4 decrements c', we deduce that $\beta(\mathsf{c}') = \rho(\mathsf{c}') - \ell = \alpha(\mathsf{c}') + n - \ell$ and $\beta(\mathsf{c}) = \rho(\mathsf{c}) + \ell = \alpha(\mathsf{c}) - n + \ell$. In particular $\ell \leq n + \alpha(\mathsf{c}')$. Observe that $\alpha \xrightarrow{N_{n,\ell}} \beta$. Moreover, we have $\ell \leq \alpha(\mathsf{c})$. Observe that if $\ell = \alpha(\mathsf{C})$, then from $\beta(\mathsf{c}') = \alpha(\mathsf{c}') + n - \ell$ we get $\beta(\mathsf{c}') = n - \alpha(\mathsf{c})$. In particular $n \geq \alpha(\mathsf{c})$ and with $n \leq \alpha(\mathsf{c})$ we get $n = \alpha(\mathsf{c})$. We deduce that $\beta(\mathsf{c}) = \alpha(\mathsf{C})$ and $\beta(\mathsf{c}') = 0$.

Conversely, let α, β be two configurations such that $\alpha \xrightarrow{N_{n,\ell}} \beta$ for two natural numbers $n, \ell \in \mathbb{N}$ such that $n \leq \alpha(\mathsf{c})$ and $\ell \leq n + \alpha(\mathsf{c}')$. Just observe that from the execution witnessing $\alpha \xrightarrow{N_{n,\ell}} \beta$, we deduce an execution witnessing $\alpha \xrightarrow{N} \beta$ by executing the first loop n times and the second loop ℓ times. □

Lemma 8. *For every configuration α, β, we have:*

$$\alpha \xrightarrow{\textbf{\textit{Loop at most}}\ \mathsf{c}+\mathsf{c}'\ \textbf{\textit{times}}\ (\textbf{\textit{inc}}(\mathsf{c});M)} \beta$$

if, and only if, there exists $n, \ell \in \mathbb{N}$ such that $n \leq \alpha(\mathsf{c})$, $\ell \leq n+\alpha(\mathsf{c}')$, and such that:

$$\alpha \xrightarrow{(\textbf{\textit{dec}}(\mathsf{c});\textbf{\textit{inc}}(\mathsf{c}'))^{(n)};(\textbf{\textit{dec}}(\mathsf{c}');\textbf{\textit{inc}}(\mathsf{c})^{(2)};M)^{(\ell)}} \beta$$

In particular $\ell \leq \alpha(\mathsf{C})$ and $\beta(\mathsf{C}) \leq 2\alpha(\mathsf{C})$. If one of those two inequalities is an equality then $\ell = \alpha(\mathsf{C})$, $\beta(\mathsf{c}) = 2\alpha(\mathsf{C})$, and $\beta(\mathsf{c}') = 0$.

Proof. The proof is very similar to the proof of Lemma 7. □

Lemma 9. *For every configuration α, β, we have:*

$$\alpha \xrightarrow{\textbf{\textit{Loop at most}}\ \mathsf{c}+\mathsf{c}'\ \textbf{\textit{times}}\ (\textbf{\textit{dec}}(\mathsf{c}');M)} \beta$$

if, and only if, there exists $n, \ell \in \mathbb{N}$ such that $n \leq \alpha(\mathsf{c})$, $\ell \leq \frac{n+\alpha(\mathsf{c}')}{2}$, and such that:

$$\alpha \xrightarrow{(\textbf{\textit{dec}}(\mathsf{c});\textbf{\textit{inc}}(\mathsf{c}'))^{(n)};(\textbf{\textit{dec}}(\mathsf{c}');\textbf{\textit{inc}}(\mathsf{c});\textbf{\textit{dec}}(\mathsf{c}');M)^{(\ell)}} \beta$$

In particular $\ell \leq \frac{\alpha(\mathsf{C})}{2}$. If this inequality is an equality then $\beta(\mathsf{c}) = \frac{\alpha(\mathsf{C})}{2}$, and $\beta(\mathsf{c}') = 0$.

Proof. We denote by N and $N_{n,\ell}$ the following test-free programs where $n, \ell \in \mathbb{N}$:

```
N  =  1: loop
      2:     dec(c);inc(c′)
      3: loop
      4:     dec(c′);inc(c);dec(c′);M
```

```
N_{n,ℓ}  =  (dec(c); inc(c′))^(n)
            (dec(c′); inc(c); dec(c′); M)^(ℓ)
```

Let α, β be two configurations such that $\alpha \xrightarrow{N} \beta$. We fix some execution witnessing that property. Let n be the number of times line 2 is executed and ℓ be the number of times line 4 is executed. Since each execution of line 2 decrements c, we deduce that after executing the first loop we get a configuration ρ satisfying $\rho(\mathsf{c}) = \alpha(\mathsf{c}) - n$ and $\rho(\mathsf{c}') = \alpha(\mathsf{c}') + n$. In particular $n \leq \alpha(\mathsf{c})$. Symmetrically, since each execution of line 4 decrements c' two times, we deduce that $\beta(\mathsf{c}') = \rho(\mathsf{c}') - 2\ell = \alpha(\mathsf{c}') + n - 2\ell$ and $\beta(\mathsf{c}) = \rho(\mathsf{c}) + \ell = \alpha(\mathsf{c}) - n + \ell$. In particular $\ell \leq \frac{n+\alpha(\mathsf{c}')}{2}$. Observe that $\alpha \xrightarrow{N_{n,\ell}} \beta$. Moreover, we have $\ell \leq \frac{\alpha(\mathsf{C})}{2}$. Observe that if $\ell = \frac{\alpha(\mathsf{C})}{2}$, then from $\beta(\mathsf{c}') = \alpha(\mathsf{c}') + n - 2\ell$ we get $\beta(\mathsf{c}') = n - \alpha(\mathsf{c})$. In particular $n \geq \alpha(\mathsf{c})$ and with $n \leq \alpha(\mathsf{c})$ we get $n = \alpha(\mathsf{c})$. We deduce that $\beta(\mathsf{c}) = \frac{\alpha(\mathsf{C})}{2}$ and $\beta(\mathsf{c}') = 0$.

Conversely, let α, β be two configurations such that $\alpha \xrightarrow{N_{n,\ell}} \beta$ for two natural numbers $n, \ell \in \mathbb{N}$ such that $n \leq \alpha(\mathsf{c})$ and $\ell \leq \frac{n+\alpha(\mathsf{c}')}{2}$. Just observe that from the execution witnessing $\alpha \xrightarrow{N_{n,\ell}} \beta$, we deduce an execution witnessing $\alpha \xrightarrow{N} \beta$ by executing the first loop n times and the second loop ℓ times. □

Lemma 10. *For every configuration α, β, we have:*

$$\alpha \xrightarrow{\textbf{\textit{Loop at most}}\ \mathsf{c}+\mathsf{c}'\ \textbf{\textit{times}}\ (\textbf{\textit{dec}}(\mathsf{c}');\textbf{\textit{inc}}(\mathsf{c});M)} \beta$$

if, and only if, there exists $n, \ell \in \mathbb{N}$ such that $n \leq \alpha(\mathsf{c})$, $\ell \leq \frac{n+\alpha(\mathsf{c}')}{2}$, and such that:

$$\alpha \xrightarrow{(\textbf{\textit{dec}}(\mathsf{c});\textbf{\textit{inc}}(\mathsf{c}'))^{(n)};((\textbf{\textit{dec}}(\mathsf{c}');\textbf{\textit{inc}}(\mathsf{c}))^{(2)};M)^{(\ell)}} \beta$$

In particular $\ell \leq \frac{\alpha(\mathsf{C})}{2}$. If this inequality is an equality then $\beta(\mathsf{c}) = \alpha(\mathsf{C})$, and $\beta(\mathsf{c}') = 0$.

Proof. The proof is very similar to the proof of Lemma 9. □

7 Implementing evalF_d

In this section, we introduce a test-free program $\mathbf{evalF}_d$ that implements the function evalF_d. This program is using counters $\mathsf{x}, \mathsf{x}', \mathsf{x}_1, \ldots, \mathsf{x}_{d+1}, \mathsf{y}, \mathsf{b}, \mathsf{b}', \mathsf{c}_0, \mathsf{c}_0', \mathsf{c}_1, \ldots, \mathsf{c}_d$. We say that a configuration ρ is *good* if it satisfies $\mathsf{x}' = \mathsf{y} = \mathsf{b}' = \mathsf{c}_0' = 0$, $\mathsf{x} > 0$, $\mathsf{b} = 2^{\mathsf{c}_0}$, $\mathsf{x}_1 = 2^{\mathsf{c}_0}\mathsf{x}$ and $\mathsf{x}_i = 2^{\mathsf{c}_{i-1}}\mathsf{x}_{i-1}$ for every $i \in \{2, \ldots, d+1\}$. We say that a good configuration ρ encodes a pair $(v, n) \in \mathbb{N}^d \times \mathbb{N}$ if $\rho(\mathsf{c}_1, \ldots, \mathsf{c}_d) = v$ and $\rho(\mathsf{c}_0) = n$.

Remark 11. The counters x and x' should have been denoted as x_0 and x_0' to simplify a little bit the definition of good configurations. However, since those two counters appear many times in the sequel, we prefer the notation without indexes. Moreover, this notation match the one used for preamplifiers. In fact, in the next section we introduce an Ackermannian preamplifier for the triple of counters $(\mathsf{x}, \mathsf{y}, \mathsf{b})$ without any variable renaming.

We introduce the counter expressions C_0, X and B defined as $\mathsf{c}_0 + \mathsf{c}_0'$, $\mathsf{x} + \mathsf{x}'$, and $\mathsf{b} + \mathsf{b}'$ respectively.

Intuitively, when the computation of $\mathbf{evalF}_d$ is correct from a good configuration that encodes a pair (v, n) with $v \neq 0_d$ then the computation terminates on a good configuration that encodes $\mathrm{evalF}_d(v, n)$. When the computation is incorrect, we obtain a so-called bad configuration. Moreover, from a bad configuration any computation of $\mathbf{evalF}_d$ leads to a bad configuration. Formally, we say that a configuration is *i-bad* for some $i \in \{2, \ldots, d+1\}$ if it satisfies $\mathsf{x} + \mathsf{x}' > 0$, $\mathsf{x}_i + \mathsf{y} > 2^{\mathsf{c}_{i-1}}(\mathsf{x}_{i-1} + \mathsf{y})$, and $\mathsf{x}_j + \mathsf{y} = 2^{\mathsf{c}_{j-1}}(\mathsf{x}_{j-1} + \mathsf{y})$ for every $j \in \{i+1, \ldots, d+1\}$. A configuration is *1-bad* if is satisfies $\mathsf{x}' = \mathsf{y} = 0$, $\mathsf{x} > 0$, $\mathsf{b} + \mathsf{b}' < 2^{\mathsf{c}_0 + \mathsf{c}_0'}$, $\mathsf{x}_1 = 2^{\mathsf{c}_0 + \mathsf{c}_0'}\mathsf{x}$, and $\mathsf{x}_j = 2^{\mathsf{c}_{j-1}}\mathsf{x}_{j-1}$ for every $j \in \{2, \ldots, d+1\}$. We say that a configuration α is *bad* if it is i-bad for some i. A configuration that is *good* or *bad* is said to be *conform*. We associate to any conform configuration α its index of badness $\mathrm{index}(\alpha)$ defined by $\mathrm{index}(\alpha) = i$ if α is i-bad, and by $\mathrm{index}(\alpha) = 0$ if α is good. Intuitively, from any conform configuration α, the computation of $\mathbf{evalF}_d$ can only produce conform configurations β such that $\mathrm{index}(\beta) \geq \mathrm{index}(\alpha)$. In particular, if β is good, then α is good as well.

The program $\mathbf{evalF}_d$ is defined as a non-deterministic choice of programs $\mathbf{evalF}_{d,p}$ with $p \in \{1, \ldots, d\}$. Intuitively from a good configuration α that encodes a pair (v, n) with $v \neq 0_d$, the computation of $\mathbf{evalF}_{d,p}$ can lead to a good configuration only if p is the minimal index such that $v[p] > 0$. In that case the good configuration produced at the end of the computation encodes $\mathrm{evalF}_d(v, n)$.

We introduce the following test-free program for $p \in \{1, \ldots, d\}$. The first loop at line 2 intuitively transfer $\min \mathsf{x}_1, \ldots, \mathsf{x}_d$ into y. Thanks to Lemma 7, the second loop at line 6 transfer back from y the value of X. Notice that x_p is incremented twice during each iteration of the loop. The loop at line 6 is interpreted thanks to Lemma 7 if $p > 1$ and Lemma 8 if $p = 1$ as a loop that is iterated C_0 times, the loop at line 8 is interpreted thanks to Lemma 9 as a loop that is iterated X times and that divides X by 2, and finally the loop at line 10 is interpreted thanks to Lemma 7 as a loop that is iterated B times.

```
$\mathbf{evalF}_{d,p}$ =
 1: dec($\mathsf{c}_p$);inc($\mathsf{c}_{p-1}$)
 2: loop
 3:     dec($\mathsf{x}_1, \ldots, \mathsf{x}_{d+1}$);inc($\mathsf{y}$)
 4: loop at most $\mathsf{x} + \mathsf{x}'$ times
 5:     dec($\mathsf{y}$);inc($\mathsf{x}_p$);inc($\mathsf{x}_1, \ldots, \mathsf{x}_{d+1}$)
 6: loop at most $\mathsf{c}_0 + \mathsf{c}'_0$ times
 7:     inc($\mathsf{c}_{p-1}$)
 8:     loop at most $\mathsf{x} + \mathsf{x}'$ times
 9:         dec($\mathsf{x}'$)
10:         loop at most $\mathsf{b} + \mathsf{b}'$ times
11:             dec($\mathsf{y}$);inc($\mathsf{x}_p$);inc($\mathsf{x}_p, \ldots, \mathsf{x}_{d+1}$)
12: $\mathbf{updateB}_{d,p}$
```

Where $\mathbf{updateB}_{d,p}$ is the empty program if $p > 1$ and the following one if $p = 1$. Notice that the loop at line 2 can be interpreted thanks to Lemma 10 as a loop that is iterated $\frac{\mathsf{C}_0}{2}$ times, and the loop at line 4 is interpreted thanks to Lemma 8 as a loop that multiply by 2 the value of B.

```
$\mathbf{updateB}_{d,1}$ =
1: inc($\mathsf{c}_0$)
2: loop at most $\mathsf{c}_0 + \mathsf{c}'_0$ times
3:     dec($\mathsf{c}'_0$);inc($\mathsf{c}_0$)
4:     loop at most $\mathsf{b} + \mathsf{b}'$ times
5:         inc($\mathsf{b}$)
6: dec($\mathsf{c}_0$)
```

We first provide two lemmas describing the behaviour of $\mathbf{updateB}_{d,1}$.

Lemma 12. *Assume that $\alpha \xrightarrow{\textit{updateB}_{d,1}} \beta$ for any two configurations α, β. Then $\beta(\mathsf{C}_0) = \alpha(\mathsf{C}_0)$ and we have:*

$$\beta(\mathsf{B}) \leq \alpha(2^{\frac{\mathsf{C}_0+1}{2}}\mathsf{B}) \tag{1}$$

If the previous inequality is an equality and $\alpha(\mathsf{B}) > 0$ then β satisfies $\mathsf{b}' = 0$ and $\mathsf{c}'_0 = 0$.

Proof. Let s be the number of times line 3 is executed. Lemma 10 shows that $s \leq \frac{1+\alpha(\mathsf{C}_0)}{2}$ (the 1 in the expression comes form the increment command at line 6). Let ρ_{j-1} for $j \in \{1, \ldots, s\}$ be the configuration just before executing that line the jth time. We also denote by ρ_s the configuration just before the execution of line 6. Lemma 8 shows that $\rho_j(\mathsf{B}) \leq \rho_{j-1}(\mathsf{B})$ for every $j \in \{1, \ldots, s\}$. We deduce that the inequality (1) holds. If this inequality is an equality and $\alpha(\mathsf{B}) > 0$, then $s = \frac{1+\alpha(\mathsf{C}_0)}{2}$. Lemma 10 shows that $\beta(\mathsf{c}'_0) = 0$. Notice that since the inequality (1) is an equality, then the inequality $\rho_j(\mathsf{B}) \leq 2\rho_{j-1}(\mathsf{B})$ is also an equality for every $j \in \{1, \ldots, k\}$. In particular $\rho_k(\mathsf{b}') = 0$ from Lemma 8. It follows that $\beta(\mathsf{b}') = 0$. □

Lemma 13. *Let α be any configuration satisfying $\alpha(\mathsf{C}_0)$ is odd. Then $\alpha \xrightarrow{\textit{updateB}_{d,1}} \beta$ where β is the configuration defined for every counter c as follows:*

$$\beta(\mathsf{c}) = \begin{cases} \alpha(\mathsf{C}_0) & \textit{if } \mathsf{c} = \mathsf{c}_0 \\ 0 & \textit{if } \mathsf{c} = \mathsf{c}'_0 \\ \alpha(2^{\frac{\mathsf{C}_0+1}{2}}\mathsf{B}) & \textit{if } \mathsf{c} = \mathsf{b} \\ 0 & \textit{if } \mathsf{c} = \mathsf{b}' \\ \alpha(\mathsf{c}) & \textit{otherwise} \end{cases}$$

Proof. Just observe that we can apply Lemma 10 and Lemma 8. □

Lemma 14. *Let α be a conform configuration and β be any configuration such that $\alpha \xrightarrow{\textbf{\textit{evalF}}_{d,p}} \beta$ with $p \in \{1,\ldots,d\}$. Then β is conform and* $\text{index}(\beta) \geq \text{index}(\alpha)$. *In particular, if β is good then α is good. In that case, α satisfies* b *divides* x, *denoting by (v,n) the pair encoded by α then $v \neq 0_d$, p is the minimal index such that $v[p] > 0$,* $\text{evalF}_d(v,n)$ *is encoded by β, and* $\beta(\mathsf{x}_{d+1}) = \alpha(\mathsf{x}_{d+1})$.

Proof. Let r, s, k, m and n be the number of times lines 3, 5, 7, 9, and 11 are executed respectively. Observe that $r \leq \min\{\alpha(\mathsf{x}_1),\ldots,\alpha(\mathsf{x}_{d+1})\}$ since every time line 3 is executed, the counters $\mathsf{x}_1,\ldots,\mathsf{x}_{d+1}$ are decremented, $s \leq \alpha(\mathsf{X})$ thanks to Lemma 7, $k \leq \alpha(\mathsf{C}_0)$, thanks to Lemma 7 if $p > 1$ and Lemma 8 if $p = 1$ (in fact in that case $\mathsf{c}_{p-1} = \mathsf{c}_0$), and $n \leq \alpha(\mathsf{B})m$ thanks to Lemma 7.

Let us prove that $\beta(\mathsf{X}) > 0$. Observe that $\alpha(\mathsf{X}) > 0$ since α is conform. Moreover, line 9 is the unique line that may prevent $\beta(\mathsf{X}) > 0$ holding. However, each time this line is executed, by definition of **loop at most**, notice that x is incremented just before. It follows that $\beta(\mathsf{X}) > 0$.

For each $j \in \{1,\ldots,k\}$, let m_j be the number of times line 9 is executed during the jth execution of the loop at line 6. We denote by α_{j-1} the configuration just before the jth execution of line 9, and by α_k the configuration just before executing $\mathbf{updateB}_{d,p}$. Notice that $\alpha_0(\mathsf{X}) = \alpha(\mathsf{X})$, $2m_j \leq \alpha_{j-1}(\mathsf{X})$, and $\alpha_j(\mathsf{X}) = \alpha_{j-1}(\mathsf{X}) - m_j$ for every $j \in \{1,\ldots,k\}$. Let us denote by h_j the natural number such that $\alpha_{j-1}(\mathsf{X}) = 2m_j + h_j$. By induction on $i \in \{0,\ldots,k\}$, we get the following equality:

$$\sum_{j=1}^{i} m_j = \alpha(\mathsf{X})[1 - 2^{-i}] - \sum_{j=1}^{i} 2^{j-i-1} h_j$$

In particular with $i = k$, since $m = \sum_{j=1}^{k} m_j$, we get the following equality by introducing the non negative rational number $h = \sum_{j=1}^{k} 2^{j-k-1} h_j$:

$$2^{\alpha(\mathsf{C}_0)} m = -2^{\alpha(\mathsf{C}_0)} h - \frac{1}{2^k} \alpha(\mathsf{X})(2^{\alpha(\mathsf{C}_0)} - 2^k) + \alpha((2^{\mathsf{C}_0} - 1)\mathsf{X}) \tag{2}$$

Assume first that α is i-bad for some $i \in \{p+2,\ldots,d+1\}$. In that case α satisfies $\mathsf{x}_i + \mathsf{y} > 2^{\mathsf{c}_{i-1}}(\mathsf{x}_{i-1} + \mathsf{y})$, and $\mathsf{x}_j + \mathsf{y} = 2^{\mathsf{c}_{j-1}}(\mathsf{x}_{j-1} + \mathsf{y})$ for every $j \in \{i+1,\ldots,d+1\}$. Notice that β also satisfies the same constraints since $\mathsf{x}_j + \mathsf{y}$ for $j \in \{i,\ldots,d+1\}$, and c_j for $j \in \{i,\ldots,d\}$ are invariant. Hence β is i-bad in that case.

So we can assume that α is not i-bad for every $i \in \{p+2,\ldots,d+1\}$. Since α is conform, we deduce that α satisfies $\mathsf{x}_{p+1} + \mathsf{y} \geq 2^{\mathsf{c}_p}(\mathsf{x}_p + \mathsf{y})$, and $\mathsf{x}_j + \mathsf{y} = 2^{\mathsf{c}_{j-1}}(\mathsf{x}_{j-1} + \mathsf{y})$ for every $j \in \{p+2,\ldots,d+1\}$. Since $\mathsf{x}_j + \mathsf{y}$ for $j \in \{p+1,\ldots,d+1\}$, and c_j for $j \in \{p+1,\ldots,d\}$ are invariant, we deduce that β satisfies $\mathsf{x}_j + \mathsf{y} = 2^{\mathsf{c}_{j-1}}(\mathsf{x}_{j-1} + \mathsf{y})$ for every $j \in \{p+2,\ldots,d+1\}$.

Observe that we have the following equalities:

$$\begin{aligned}
\beta(\mathsf{x}_{p+1} + \mathsf{y}) &= \alpha(\mathsf{x}_{p+1} + \mathsf{y}) \\
\beta(\mathsf{c}_p) &= \alpha(\mathsf{c}_p) - 1 \\
\beta(\mathsf{x}_p + \mathsf{y}) &= \alpha(\mathsf{x}_p + \mathsf{y}) + s + n \\
\beta(\mathsf{y}) &= \alpha(\mathsf{y}) + r - (s + n)
\end{aligned}$$

From those equalities, we derive:

$$\begin{aligned}
\beta(\mathsf{x}_{p+1} + \mathsf{y} - 2^{\mathsf{c}_p}(\mathsf{x}_p + \mathsf{y})) &= \alpha(\mathsf{x}_{p+1} + \mathsf{y} - 2^{\mathsf{c}_p}(\mathsf{x}_p + \mathsf{y})) \\
&\quad + 2^{\alpha(\mathsf{c}_p)-1}(\alpha(\mathsf{x}_p) - r) \\
&\quad + 2^{\alpha(\mathsf{c}_p)-1}\beta(\mathsf{y})
\end{aligned}$$

Notice that the right hand-side of the last equality is a sum of three natural numbers. It means that if one of those numbers is strictly positive, then β is $(p+1)$-bad. So, we can assume that those three numbers are zero. It means that α and β satisfies $\mathsf{x}_{p+1}+\mathsf{y}=2^{\mathsf{c}_p}(\mathsf{x}_p+\mathsf{y})$, $r=\alpha(\mathsf{x}_p)$, and $\beta(\mathsf{y})=0$. In particular α is not $(p+1)$-bad.

Assume by contradiction that α is i-bad for some $i\in\{2,\dots,p\}$. It means that α satisfies $\mathsf{x}_i+\mathsf{y}>2^{\mathsf{c}_{i-1}}(\mathsf{x}_{i-1}+\mathsf{y})$ and $\mathsf{x}_j+\mathsf{y}=2^{\mathsf{c}_{j-1}}(\mathsf{x}_{j-1}+\mathsf{y})$ for every $j\in\{i+1,\dots,d+1\}$. In particular α satisfies $\mathsf{x}_i>\mathsf{x}_{i-1}$ and $\mathsf{x}_j\geq\mathsf{x}_{j-1}$ for every $j\in\{i+1,\dots,d\}$. We deduce that α satisfies $\mathsf{x}_i<\mathsf{x}_p$. It follows that $r\leq\min\{\alpha(\mathsf{x}_1),\dots,\alpha(\mathsf{x}_{d+1})\}<\alpha(\mathsf{x}_p)=r$ and we get a contradiction.

Since α is not i-bad for every $i\in\{2,\dots,d+1\}$, we deduce that α is 1-bad or good. In both cases, α satisfies $\mathsf{x}'=\mathsf{y}=0$, $\mathsf{B}\leq 2^{\mathsf{C}_0}$, $\mathsf{x}_1=2^{\mathsf{C}_0}\mathsf{x}$, and $\mathsf{x}_i=2^{\mathsf{c}_{i-1}}\mathsf{x}_{i-1}$ for every $i\in\{2,\dots,d+1\}$. In particular $\alpha(\mathsf{x}_{d+1})\geq\cdots\geq\alpha(\mathsf{x})>0$. Notice that from $\alpha(\mathsf{y})=0$ and $\beta(\mathsf{y})=0$, we deduce from $\beta(\mathsf{y})=\alpha(\mathsf{y})+r-(s+n)$ that $r=s+n$.

Assume by contradiction that $\alpha(\mathsf{c}_{i-1})>0$ for some $i\in\{2,\dots,p\}$. Since α satisfies $\mathsf{x}_i=2^{\mathsf{c}_{i-1}}\mathsf{x}_{i-1}$, $\mathsf{c}_{i-1}>0$, and $\mathsf{x}_{i-1}>0$, we deduce that $\alpha(\mathsf{x}_i)>\alpha(\mathsf{x}_{i-1})$. In particular, $\alpha(\mathsf{x}_p)\geq\alpha(\mathsf{x}_i)>\alpha(\mathsf{x}_{i-1})$. Like in the previous paragraph we get a contradiction with $r\leq\min\{\alpha(\mathsf{x}_1),\dots,\alpha(\mathsf{x}_{d+1})\}<\alpha(\mathsf{x}_p)=r$. Therefore $\alpha(\mathsf{c}_{i-1})=0$ for every $i\in\{2,\dots,p\}$. We deduce that $\alpha(\mathsf{x}_1)=\cdots=\alpha(\mathsf{x}_p)$. It follows from $r=\alpha(\mathsf{x}_p)$ and $r=s+n$ that $\alpha(\mathsf{x}_1)=s+n$.

Observe that we have (by developing the equalities and replacing $2^{\alpha(\mathsf{C}_0)}m$ by using the equality 2):

$$\begin{aligned}
n &= -(\alpha(\mathsf{B})m-n)-\alpha(2^{\mathsf{B}_0}-\mathsf{B})m-2^{\alpha(\mathsf{C}_0)}h-\frac{1}{2^k}\alpha(\mathsf{X})(2^{\alpha(\mathsf{C}_0)}-2^k)+\alpha((2^{\mathsf{C}_0}-1)\mathsf{X}))\\
s &= -(\alpha(\mathsf{X})-s)+\alpha(\mathsf{X})
\end{aligned}$$

It follows that we have (we use the fact that $\alpha(\mathsf{x}_1-2^{\mathsf{C}_0}\mathsf{X})=0$):

$$\begin{aligned}
0=\alpha(\mathsf{x}_1)-(s+n)=&(\alpha(\mathsf{B})m-n)\\
&+(2^{\alpha(\mathsf{C}_0)}-\alpha(\mathsf{B}))m\\
&+(\alpha(\mathsf{X})-s)\\
&+2^{\alpha(\mathsf{C}_0)}h\\
&+\frac{1}{2^k}\alpha(\mathsf{X})(2^{\alpha(\mathsf{C}_0)}-2^k)
\end{aligned}$$

Since each term of the right hand side of the previous equality are non negative, we deduce that all the terms are zero. It means that $n=\alpha(\mathsf{B})m$, $(2^{\alpha(\mathsf{C}_0)}-\alpha(\mathsf{B}))m=0$, $s=\alpha(\mathsf{X})$, $h=0$, $2^{\alpha(\mathsf{C}_0)}=2^k$ (for the last equality, we use $\alpha(\mathsf{X})>0$). Since $k\leq\alpha(\mathsf{C}_0)$ we derive from $2^{\alpha(\mathsf{C}_0)}=2^k$ that $k=\alpha(\mathsf{C}_0)$. By replacing h by 0, and k by $\alpha(\mathsf{C}_0)$ in equation 2, we derive $m=\alpha((1-2^{-\mathsf{C}_0})\mathsf{X})$.

Assume that α is 1-bad. In that case $\alpha(\mathsf{B})<2^{\alpha(\mathsf{C}_0)}$. It follows from $(2^{\alpha(\mathsf{C}_0)}-\alpha(\mathsf{B}))m=0$ that $m=0$. From $m=\alpha((1-2^{-\mathsf{C}_0})\mathsf{X})$ and $\alpha(\mathsf{X})>0$ we get $\alpha(\mathsf{C}_0)=0$. So, from $\alpha(\mathsf{B})<2^{\alpha(\mathsf{C}_0)}$ we get $\alpha(\mathsf{B})=0$. Observe that in that case β is 1-bad (the case $p=1$ is obtained thanks to Lemma 12). So, we can now assume that α is not 1-bad.

Since α is conform and not bad, we deduce that α is good. Hence α satisfies $\mathsf{x}'=\mathsf{c}_0'=\mathsf{b}'=\mathsf{y}=0$ and $\mathsf{b}=2^{\mathsf{c}_0}$. It follows from $n=\alpha(\mathsf{B})m$ that $n=\alpha((2^{\mathsf{C}_0}-1)\mathsf{X})$. Let ρ be the configuration we obtain just before executing $\mathbf{updateB}_{d,p}$. Since $h=0$, then $h_k=0$. It follows that $\rho(\mathsf{x}')=0$. As $s=\alpha(\mathsf{X})$, Lemma 7 shows that $\rho(\mathsf{x}')=0$. Moreover, as $n=\alpha(\mathsf{B})m$, Lemma 7 shows that $\rho(\mathsf{b}')=0$. It follows that $\rho(\mathsf{x})=\alpha(\mathsf{x})-m=\alpha(2^{-\mathsf{c}_0}\mathsf{x})$. Notice that $\rho(\mathsf{x}_i)$ with $i\in\{1,\dots,p-1\}$ is equal to $\alpha(\mathsf{x}_i)-r+s=s=\alpha(\mathsf{x})$. We also have $\rho(\mathsf{x}_p)=\alpha(\mathsf{x}_p)-r+2s+2n=2\alpha(\mathsf{x}_p)$. Notice that $\rho(\mathsf{x}_i)$ with $i\in\{p+1,\dots,d+1\}$ is equal to $\alpha(\mathsf{x}_i)$ since $\mathsf{x}_i+\mathsf{y}$ is an invariant and $\alpha(\mathsf{y})=\rho(\mathsf{y})=0$.

It follows that for every counter c, we have:

$$\rho(\mathsf{c}) = \begin{cases} 0 & \text{if } \mathsf{c} \in \{\mathsf{x}', \mathsf{c}_0', \mathsf{b}', \mathsf{y}\} \\ \alpha(2^{-\mathsf{c}_0}\mathsf{x}) & \text{if } \mathsf{c} = \mathsf{x} \\ \alpha(\mathsf{x}) & \text{if } \mathsf{c} = \mathsf{x}_i \text{ with } i \in \{1, \dots, p-1\} \\ 2\alpha(\mathsf{x}_p) & \text{if } \mathsf{c} = \mathsf{x}_p \\ \alpha(\mathsf{c}_p) - 1 & \text{if } \mathsf{c} = \mathsf{c}_p \\ \alpha(1 + \mathsf{c}_{p-1} + \mathsf{c}_0) & \text{if } \mathsf{c} = \mathsf{c}_{p-1} \\ \alpha(\mathsf{c}) & \text{otherwise} \end{cases}$$

Notice that if $p > 1$ then $\alpha(\mathsf{c}_{p-1}) = 0$ and $\beta = \rho$. In that case β is good and satisfies the lemma.

If $p = 1$, notice that $\rho(\mathsf{c}_0) = 1 + 2\alpha(\mathsf{c}_0)$. From Lemma 12 we deduce that β is either 1-bad or good depending depending if the inequality of that lemma is strict or an equality. Notice that if the inequality is an equality, the configuration β is good and it encodes $\mathrm{evalF}_d(v, n)$. □

Lemma 15. *Let α be a good configuration satisfying b divides x and that encodes a pair (v, n) with $v \neq 0_d$, and let p is the minimal index such that $v[p] > 0$. There exists a good configuration β that encodes $\mathrm{evalF}_d(v, n)$ satisfying $\beta(\mathsf{x}_{d+1}) = \alpha(\mathsf{x}_{d+1})$, and such that $\alpha \xrightarrow{\textbf{\textit{evalF}}_{d,p}} \beta$.*

Proof. We are going to prove that there exists an execution of $\mathbf{evalF}_{d,p}$ from α such that lines 3, 5, 7 are executed r, s, k times with:

$$\begin{aligned} r &= \alpha(\mathsf{x}_1) \\ s &= \alpha(\mathsf{x}) \\ k &= \alpha(\mathsf{c}_0) \end{aligned}$$

We also introduce the configurations that will appear along the execution of loop at line 6. To do so, we introduce the configurations $\alpha_0, \dots, \alpha_k$ defined as follows for every counter c and for every $j \in \{0, \dots, k\}$:

$$\alpha_j(\mathsf{c}) = \alpha\left(\begin{cases} \frac{\mathsf{x}}{2^j} & \text{if } \mathsf{c} = \mathsf{x} \\ \mathsf{x} & \text{if } \mathsf{c} = \mathsf{x}_i \text{ with } i \in \{1, \dots, p-1\} \\ 2\mathsf{b}\mathsf{x} + 2\mathsf{x}(1 - 2^{\mathsf{c}_0 - j}) & \text{if } \mathsf{c} = \mathsf{x}_p \\ \mathsf{x}_i + \mathsf{x}(1 - 2^{\mathsf{c}_0 - j}) & \text{if } \mathsf{c} = \mathsf{x}_i \text{ with } i \in \{p+1, \dots, d+1\} \\ \mathsf{x}(2^{\mathsf{c}_0 - j} - 1) & \text{if } \mathsf{c} = \mathsf{y} \\ \mathsf{c}_p - 1 & \text{if } \mathsf{c} = \mathsf{c}_p \\ \mathsf{c}_{p-1} + 1 + j & \text{if } \mathsf{c} = \mathsf{c}_{p-1} \text{ and } p > 1 \\ \mathsf{c}_0 - j & \text{if } \mathsf{c} = \mathsf{c}_0' \\ j & \text{if } \mathsf{c} = \mathsf{c}_0 \text{ and } p > 1 \\ 1 + 2j & \text{if } \mathsf{c} = \mathsf{c}_0 \text{ and } p = 1 \\ \mathsf{c} & \text{otherwise} \end{cases}\right)$$

Since α is good, it follows that $\alpha(\mathsf{x}_i) \geq r$ for every $i \in \{1, \dots, d+1\}$, and $r \geq \alpha(\mathsf{x})$. In particular line 3 and 5 can be effectively executed r and s times respectively. Notice that α_0 is the configuration we obtain after executing those two loops and the hidden first loop of the **loop at most** at line 6 exactly $\alpha(\mathsf{x})$ times.

We introduce the test-free program N corresponding to the subprogram between line 7 and line 11 (including those those lines) and prefixed by $\mathbf{dec}(\mathsf{c}_0'); \mathbf{inc}(\mathsf{c}_0)$. Let us prove by induction on $j \in \{1, \dots, k\}$ that $\alpha_{j-1} \xrightarrow{N} \alpha_j$. To do so, we are going to prove that there exists an execution of N such that lines 9, 11 are executed m_j, n_j times with:

$$m_j = \alpha_{j-1}(\frac{\mathsf{x}}{2})$$
$$n_j = \alpha_{j-1}(m_j \mathsf{b})$$

Notice that those three numbers are natural numbers since 2^k divides $\alpha(\mathsf{x})$. This execution is obtained thanks to Lemma 7 applied on line 11 in order to execute maximally the corresponding loop, and by applying Lemma 9 applied on line 9. We have proved that $\alpha_{j-1} \xrightarrow{M} \alpha_j$. In particular, by executing k times loop 6, we get the configuration α_k. Notice that this configuration satisfies for every counter c:

$$\alpha_k(\mathsf{c}) = \alpha \left(\begin{cases} \frac{\mathsf{x}}{\mathsf{b}} & \text{if } \mathsf{c} = \mathsf{x} \\ \mathsf{x} & \text{if } \mathsf{c} = \mathsf{x}_i \text{ with } i \in \{1, \dots, p-1\} \\ 2\mathsf{bx} & \text{if } \mathsf{c} = \mathsf{x}_p \\ \mathsf{x}_i & \text{if } \mathsf{c} = \mathsf{x}_i \text{ with } i \in \{p+1, \dots, d+1\} \\ 0 & \text{if } \mathsf{c} = \mathsf{y} \\ \mathsf{c}_p - 1 & \text{if } \mathsf{c} = \mathsf{c}_p \\ \mathsf{c}_{p-1} + 1 + \mathsf{c}_0 & \text{if } \mathsf{c} = \mathsf{c}_{p-1} \\ \mathsf{c} & \text{otherwise} \end{cases} \right)$$

In particular, by executing $\mathbf{updateB}_{d,1}$ following Lemma 13 (if $p = 1$), we get from α_k a configuration β satisfying the lemma. □

Since the test-free program $\mathbf{evalF}_d$ is defined as $\mathbf{evalF}_{d,1}$ **or** $\cdots$ **or** $\mathbf{evalF}_{d,d}$, we deduce from Lemma 14 and Lemma 15 the following two corollaries.

Corollary 16. *Let α be a conform configuration and β be any configuration such that $\alpha \xrightarrow{\mathbf{evalF}_d} \beta$. Then β is conform and* $\text{index}(\beta) \geq \text{index}(\alpha)$. *In particular, if β is good then α is good. In that case, α satisfies* b *divides* x, *denoting by (v, n) the pair encoded by α then $v \neq 0_d$,* $\text{evalF}_d(v, n)$ *is encoded by β, and $\beta(\mathsf{x}_{d+1}) = \alpha(\mathsf{x}_{d+1})$.*

Corollary 17. *Let α be a good configuration satisfying* b *divides* x *and that encodes a pair (v, n) with $v \neq 0_d$. There exists a good configuration β that encodes* $\text{evalF}_d(v, n)$ *satisfying $\beta(\mathsf{x}_{d+1}) = \alpha(\mathsf{x}_{d+1})$, and such that $\alpha \xrightarrow{\mathbf{evalF}_d} \beta$.*

8 Ackermannian Preamplifiers

In this section we prove that the following checking program is a K-preamplifier for $K = 2^{F_{d+1}(n)}$ of size $O(d4^n)$.

$\mathbf{Ack}_{d,n}$ =

```
$\mathbf{inc}(\mathsf{c}_0)^{(n)}; \mathbf{inc}(\mathsf{c}_d)^{(n+1)}$
$\mathbf{inc}(\mathsf{x}); \mathbf{inc}(\mathsf{x}_1, \dots, \mathsf{x}_d)^{(2^n)}; \mathbf{inc}(\mathsf{x}_{d+1})^{(2^{2n+1})}$
loop
    $\mathbf{inc}(\mathsf{x}); \mathbf{inc}(\mathsf{x}_1, \dots, \mathsf{x}_d)^{(2^n)}; \mathbf{inc}(\mathsf{x}_{d+1})^{(2^{2n+1})}$
loop
    $\mathbf{evalF}_d$
loop
    $\mathbf{inc}(\mathsf{y}); \mathbf{dec}(\mathsf{x}_1, \dots, \mathsf{x}_{d+1})$
loop
    $\mathbf{dec}(\mathsf{c}_0)$
$\mathbf{test}(\mathsf{x}_{d+1}, \mathsf{b}', \mathsf{c}'_0, \mathsf{c}_0, \dots, \mathsf{c}_d)$
```

Lemma 18. *The checking program* $\boldsymbol{Ack}_{d,n}$ *is a* K*-preamplifier for* $K = 2^{F_{d+1}(n)}$.

Proof. Let $\ell \geq 1$ and let β be the configuration satisfying $\beta(\mathsf{x}, \mathsf{y}, \mathsf{b}) = (\ell, K\ell, K)$ and $\beta(\mathsf{c}) = 0$ for every counter $\mathsf{c} \notin \{\mathsf{x}, \mathsf{y}, \mathsf{b}\}$. Let us prove that $0 \xrightarrow{\mathbf{Ack}_{d,n}} \beta$.

To do so, we consider an execution of $\mathbf{Ack}_{d,n}$ defined as follows. We execute the first loop $2^{F_{d+1}(n)-2n-1}\ell - 1$ times (recall that $F_{d+1}(n) \geq 2n + 1$). Just after this loop we get a good configuration ρ_0 that encodes $((0, \dots, 0, n), n + 1)$ and such that $\rho_0(\mathsf{x}_{d+1}) = K\ell$. Then we iterate $\mathbf{evalF}_d$ as many times as possible following Lemma 15. This way we get a sequence $\rho_0, \dots, \rho_k$ of good configurations such that $\rho_j(\mathsf{x}_{d+1}) = 2^{F_{d+1}(n)}\ell$ and such that ρ_j encodes the pair (v_j, n_j) defined as $(v_j, n_j) = \mathrm{evalF}_d^j((n+1)1_{d,d}, n)$. Since $F_{v_j}(n_j)$ does not depend on j, we deduce that $F_{v_j}(n_j) = F_{v_0}(n_0) = F_d^{n+1}(n) = F_{d+1}(n)$.

Since ρ_k is a good configuration, it follows that ρ_k satisfies $\mathsf{b} = 2^{\mathsf{c}_0}$ and $\mathsf{x}_{d+1} = 2^{\mathsf{c}_d+\dots+\mathsf{c}_0}\mathsf{x}$. As ρ_k encodes (v_j, n_j), we deduce that $\rho_k(\mathsf{b}) = 2^{n_j}$ and $\rho_k(\mathsf{x}) = 2^{F_{d+1}(n)-n_j-|v_j|}\ell$.

We have proved the following equality:

$$\rho_k(\frac{\mathsf{x}}{\mathsf{b}}) = 2^{F_{v_k}(n_k)-2n_k-|v_k|}$$

Lemma 3 shows that if $v_k \neq 0_d$ then ρ_k satisfies b divides x. In particular from the good configuration ρ_k we can execute $\mathbf{evalF}_d$ one more time following Lemma 15 and get a contradiction with the maximality of k. It follows that $v_k = 0_d$. From $F_{v_k}(n_k) = F_{d+1}(n)$, we deduce that $n_k = F_{d+1}(n)$. Hence $\rho_k(\mathsf{b}) = K$.

Since $v_k = 0_d$ and ρ_k is a good configuration, we deduce that ρ_k satisfies $\mathsf{x}_{d+1} = \dots = \mathsf{x}_1$. It follows that $\rho_k(\mathsf{x}_i) = K\ell$ for every $i \in \{1, \dots, d + 1\}$. Based on this observation, from ρ_k we can execute the third loop $K\ell$ times. Then we execute the last loop $\rho_k(\mathsf{c}_0)$ times. This way, we get the configuration β.

Finally, let us consider a configuration β such that $0 \xrightarrow{\mathrm{Ack}_{d,n}} \beta$ and let us prove that β satisfies $\mathsf{y} \geq \mathsf{bx}$ and $\beta(\mathsf{c}) = 0$ for every counter $\mathsf{c} \notin \{\mathsf{x}, \mathsf{y}, \mathsf{b}\}$. Moreover, if the inequality is an equality then let us prove that $\beta(\mathsf{b}) = K$.

We denote by ρ_0 the configuration we obtain after executing the first loop. Let k be the number of times the second loop is executed and let us denote by ρ_j the configuration we obtain after the jth execution of the second loop for $j \in \{1, \dots, k\}$. Lemma 14 shows that ρ_j is conform.

Assume by contradiction that ρ_k is i-bad for $i \in \{2, \dots, d + 1\}$. In that case ρ_k satisfies $\mathsf{x}_i + \mathsf{y} > 2^{\mathsf{c}_{i-1}}(\mathsf{x}_{i-1} + \mathsf{y})$ and $\mathsf{x}_j + \mathsf{y} = 2^{\mathsf{c}_j}(\mathsf{x}_{j-1} + \mathsf{y})$ for every $j \in \{i + 1, \dots, d + 1\}$. In particular $\rho_k(\mathsf{x}_{d+1}) > \rho_k(\mathsf{x}_{i-1})$. It follows that whatever the number of time the third loop is executed, the configuration β satisfy $\beta(\mathsf{x}_{d+1}) > \beta(\mathsf{x}_{i-1})$ and the last test command $\mathbf{test}(\mathsf{x}_{d+1})$ fails. We get a contradiction. It follows that ρ_k is 1-bad or good.

Now, assume that ρ_k is 1-bad. In that case ρ_k satisfies $\mathsf{x}' = \mathsf{y} = 0$, $\mathsf{x} > 0$, $\mathsf{b} + \mathsf{b}' < 2^{\mathsf{c}_0+\mathsf{c}_0'}$, $\mathsf{x}_1 = 2^{\mathsf{c}_0+\mathsf{c}_0'}\mathsf{x}$ and $\mathsf{x}_i = 2^{\mathsf{c}_{i-1}}\mathsf{x}_{i-1}$ for every $i \in \{2, \dots, d+1\}$. Since the execution from ρ_k to β only modify $\mathsf{y}, \mathsf{x}_1, \dots, \mathsf{x}_{d+1}, \mathsf{c}_0$, then β and ρ_k coincides on the other counters. Since β successfully execute the last test commands, we deduce that ρ_k satisfies $\mathsf{b}' = \mathsf{c}_0' = \mathsf{c}_1 = \dots = \mathsf{c}_d = 0$. It follows that ρ_k satisfies $\mathsf{x}_{d+1} = \dots = \mathsf{x}_1$. Since $\beta(\mathsf{x}_{d+1}) = 0$, it means that third loop was executed $\rho_k(\mathsf{x}_{d+1})$ times. We deduce that $\beta(\mathsf{x}_i) = 0$ for every $i \in \{1, \dots, d\}$. It follows that β satisfies $\mathsf{y} > \mathsf{bx}$ and $\beta(\mathsf{c}) = 0$ for every counter $\mathsf{c} \notin \{\mathsf{x}, \mathsf{y}, \mathsf{b}\}$.

Finally, assume that ρ_k is good. From Lemma 14 we deduce that ρ_j is good for every $j \in \{1, \dots, k\}$ and denoting by (v_j, n_j) the pair encoded by ρ_j we have $(v_j, n_j) = \mathrm{evalF}_d^j(v_0, n_0)$. In particular $F_{v_j}(n_j) = F_{d+1}(n)$. Since ρ_k is good it satisfies $\mathsf{x}' = \mathsf{y} = \mathsf{b}' = \mathsf{c}_0' = 0$, $\mathsf{x} > 0$, $\mathsf{b} = 2^{\mathsf{c}_0}$, $\mathsf{x}_1 = 2^{\mathsf{c}_0}\mathsf{x}$ and $\mathsf{x}_i = 2^{\mathsf{c}_{i-1}}\mathsf{x}_{i-1}$ for every $i \in \{2, \dots, d + 1\}$. Since the execution from ρ_k to β only modify $\mathsf{y}, \mathsf{x}_1, \dots, \mathsf{x}_{d+1}, \mathsf{c}_0$, then β and ρ_k coincides on the other counters. Since β successfully execute the last test commands, we deduce that ρ_k satisfies $\mathsf{c}_1 = \dots = \mathsf{c}_d = 0$. It follows that $v_k = 0_d$ and from $F_{v_k}(n_k) = F_{d+1}(n)$ we get $n_k = F_{d+1}(n)$. Since ρ_k satisfies

$\mathsf{b} = 2^{\mathsf{c}_0}$ we deduce that $\rho_k(\mathsf{b}) = K$. Since ρ_k satisfies $\mathsf{x}_{d+1} = \cdots = \mathsf{x}_1$ and $\beta(\mathsf{x}_{d+1}) = 0$, it means that third loop was executed $\rho_k(\mathsf{x}_{d+1})$ times. We deduce that $\beta(\mathsf{x}_i) = 0$ for every $i \in \{1, \ldots, d\}$. We have proved that β satisfies $\mathsf{y} = \mathsf{b}\mathsf{x}$, $\rho_k(\mathsf{b}) = K$, and $\rho_k(\mathsf{c}) = 0$ for every counter $\mathsf{c} \notin \{\mathsf{x}, \mathsf{y}, \mathsf{b}\}$.

Therefore $\mathrm{Ack}_{d,n}$ is a K-preamplifier. □

Corollary 19. *We can compute in time $O(nd4^n)$ a K-preamplifier of size $O(nd4^n)$ with $K = 2^{F_{d+1}(n)}$ using $2d + 6$ counters such that d of them are safe.*

Proof. The checking program $\mathrm{Ack}_{d,n}$ is a K-preamplifier of size $O(d4^n)$ computable in time $O(d4^d)$ using $2d + 8$ counters such that $\mathsf{x}', \mathsf{x}_1, \ldots, \mathsf{x}_d$ are safe counters. Notice that the counter c_d is first initialized to n and then during an execution it is only decremented. It follows that its value can be encoded in the control structure of the program, i.e. by unfolding the program n times. Moreover, notice that during the execution, the value of $\mathsf{x}_d + \mathsf{y}$ can only increase. It follows that the counter x_d can be removed as well by observing that its value (for good configuration) is in fact equals to $\mathsf{x}_{d+1}2^{-\mathsf{c}_d}$ and the value of c_d is hard coded in the control structure. This way, we get a K-preamplifier satisfying the lemma. □

9 Complexity Classes Beyond Elementary

The reachability problem for general programs equipped with the bounded semantics provides a way to define complexity classes beyond Elementary. In fact, following [22], the problem that asks, given a 2-dimensional general program M of size n, whether there exists β such that $0 \xrightarrow{M}_{\leq K} \beta$ with $K = 2^{F_d(n)}$ is $\mathbb{F}_d$-complete for every $d \geq 3$. We deduce as a direct corollary the following theorem.

Theorem 20. *The reachability problem for $(2d+4)$-dimensional checking programs is $\mathbb{F}_d$-hard for any $d \geq 3$. In particular the reachability problem for 10-dimensional checking programs is not Elementary.*

Proof. Given a 2-dimensional program M, let $n = \mathrm{size}(M)$. Corollary 19 shows that we can compute in time $O(nd4^n)$ a K-preamplifier A of size $O(nd4^n)$ with $K = 2^{F_d(n)}$ using $2(d-1)+6$ counters such that $d-1$ of them are safe. Now recall that the checking program N defined as $A \rhd M$ is computable in time $\mathrm{size}(A) + O(\mathrm{size}(M))$. It follows that N is computable in time $O(nd4^n)$. Moreover, as the number of safe counters of A is larger than or equal to 2, we deduce that the dimension of N is bounded by $2d+4$. Finally, just observe that for every configuration β we have $0 \xrightarrow{M}_{\leq K} \beta$ if, and only if, $0 \xrightarrow{N} \beta$. □

Theorem 21 is nearly optimal since in [13] it is proved that the reachability problem for d-dimensional checking programs is in $\mathbb{F}_{d+4}$.

Finally, let us recall [22] that the problem that asks, given a 2-dimensional general program M of size n, whether there exists a configuration β such that $0 \xrightarrow{M}_{\leq K} \beta$ with $K = 2^{F_\omega(n)}$ is $\mathbb{F}_\omega$-complete. We deduce as a direct corollary the following theorem.

Theorem 21. *The reachability problem for checking programs is $\mathbb{F}_\omega$-hard.*

Proof. Given a 2-dimensional program M, let $n = \mathrm{size}(M)$. Corollary 19 shows that we can compute in time $O(n^2 4^n)$ a K-preamplifier A of size $O(n^2 4^n)$ with $K = 2^{F_{n+1}(n)} = 2^{F_\omega(n)}$. Now recall that the checking program N defined as $A \rhd M$ is computable in time $\mathrm{size}(A) + O(\mathrm{size}(M))$. It follows that N is computable in time $O(n^2 4^n)$. Finally, just observe that for every configuration β we have $0 \xrightarrow{M}_{\leq K} \beta$ if, and only if, $0 \xrightarrow{N} \beta$. □

Theorem 21 is optimal since in [13] it is proved that the reachability problem for checking programs is in $\mathbb{F}_\omega$.

10 Conclusion

This paper proves that the reachability problem for checking programs is Ackermannian-complete. It also reduces the gap for the parameterized complexity of the reachability problem in fixed dimension. In order to close this gap, we see two possible research directions:

- Either we find a primitive recursive algorithm computing from $d, n \in \mathbb{N}$ a K-preamplifier for $K = F_d(n)$ with a dimension $d + O(1)$,
- Or we find a new algorithm for deciding the reachability problem with a complexity upper bound in $\mathbb{F}_{\frac{d}{2}+O(1)}$.

Acknowledgements

I gratefully thank Philippe Schnoebelen for helpful comments and encouraging feedback.

References

[1] Rémi Bonnet. The reachability problem for vector addition system with one zero-test. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 145–157. Springer, 2011. doi: 10.1007/978-3-642-22993-0_16. URL `https://doi.org/10.1007/978-3-642-22993-0_16`.

[2] E. Cardoza, R. J. Lipton, and A. R. Meyer. Exponential space complete problems for petri nets and commutative semigroups: Preliminary report. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA*, pages 50–54. ACM, 1976. doi: 10.1145/800113.803630.

[3] W. Czerwiński, S. Lasota, R. Lazić, J. Leroux, and F. Mazowiecki. The reachability problem for petri nets is not elementary. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 24–33. ACM, 2019. doi: 10.1145/3313276.3316369.

[4] Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is ackermann-complete. In *62st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, Colorado, USA, February 7-10 2022*. IEEE, 2022. To appear.

[5] J. Esparza and M. Nielsen. Decidability issues for petri nets - a survey. *Bulletin of the European Association for Theoretical Computer Science*, 52:245–262, 1994.

[6] M. H. T. Hack. *Decidability questions for Petri nets*. PhD thesis, MIT, 1975. URL `http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-161.pdf`.

[7] J. E. Hopcroft and J.-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoritical Computer Science*, 8:135–159, 1979.

[8] R. M. Karp and R. E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2): 147–195, 1969. doi: 10.1016/S0022-0000(69)80011-5.

[9] S. R. Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *STOC*, pages 267–281. ACM, 1982. doi: 10.1145/800070.802201.

[10] J.-L. Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99 (1):79–104, 1992. doi: 10.1016/0304-3975(92)90173-D.

[11] Sławomir Lasota. Improved ackermannian lower bound for the vass reachability problem, 2021.

[12] J. Leroux and S. Schmitz. Demystifying reachability in vector addition systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 56–67. IEEE Computer Society, 2015. doi: 10.1109/LICS.2015.16.

[13] J. Leroux and S. Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi: 10.1109/LICS.2019.8785796.

[14] Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. In *62st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, Colorado, USA, February 7-10 2022*. IEEE, 2022. To appear.

[15] E. W. Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 238–246. ACM, 1981. doi: 10.1145/800076.802477.

[16] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984. doi: 10.1137/0213029.

[17] Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967. URL `https://dl.acm.org/citation.cfm?id=1095587`.

[18] C. Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi: 10.1016/0304-3975(78)90036-1.

[19] Klaus Reinhardt. Reachability in petri nets with inhibitor arcs. *Electronic Notes in Theoretical Computer Science*, 223:239–264, 2008. ISSN 1571-0661. doi: https://doi.org/10.1016/j.entcs.2008.12.042. URL `https://www.sciencedirect.com/science/article/pii/S1571066108005057`. Proceedings of the Second Workshop on Reachability Problems in Computational Models (RP 2008).

[20] G. S. Sacerdote and R. L. Tenney. The decidability of the reachability problem for vector addition systems (preliminary version). In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 61–76. ACM, 1977. doi: 10.1145/800105.803396.

[21] S. Schmitz. The complexity of reachability in vector addition systems. *SIGLOG News*, 3 (1):4–21, 2016. URL `https://dl.acm.org/citation.cfm?id=2893585`.

[22] Sylvain Schmitz. Complexity hierarchies beyond elementary. *TOCT*, 8(1):3:1–3:36, 2016. URL `http://doi.acm.org/10.1145/2858784`.

[23] Philippe Schnoebelen. Revisiting ackermann-hardness for lossy counter machines and reset petri nets. In Petr Hlinený and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 616–628. Springer, 2010. doi: 10.1007/978-3-642-15155-2_54. URL `https://doi.org/10.1007/978-3-642-15155-2_54`.