Arithmetic Circuits and Polynomial Replacement Systems

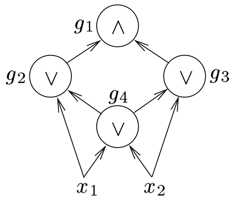
Pierre McKenzie[*] *Heribert Vollmer*[†] *Klaus W. Wagner*[‡]

Abstract

This paper addresses the problems of counting proof trees (as introduced by Venkateswaran and Tompa) and counting proof circuits, a related but seemingly more natural question. These problems lead to a common generalization of straight-line programs which we call polynomial replacement systems. We contribute a classification of these systems and we investigate their complexity. Diverse problems falling in the scope of this study include, for example, counting proof circuits, and evaluating $\{\cup, +\}$-circuits over the natural numbers. A number of complexity results are obtained, e.g., the former problem is shown #P-complete, while the latter is shown to be equivalent to a particular type of replacement systems.

1 Introduction

1.1 Motivation



When $+$ and $\times$ replace $\vee$ and $\wedge$ in the adjacent figure, the gate g_1 on input $x_1 = x_2 = 1$ evaluates to 9. Equivalently, the tree-like Boolean circuit T obtained from the circuit drawn has 9 *proof trees* [VT89], i.e. 9 different minimal subcircuits witnessing that T outputs 1 (gates replicated to form T are independent). This relationship between proof tree counting and monotone arithmetic circuits was used by Venkateswaran [Ven92] to characterize nondeterministic time classes, including #P [Val79], and by Vinay [Vin91a] to characterize the counting version of LOGCFL [Sud78]. The same relationship triggered the investigation of $\#\mathrm{NC}^1$ by Caussinus et al. [CMTV98], and that of $\#\mathrm{AC}^0$ by Allender et al. [AAD97]. See [AML98, LêT98, All98] for recent results and for motivation to study such "small" arithmetic classes.

A recent goal has been to capture small arithmetic classes by counting objects other than proof trees, notably paths in graphs. Allender et al. [AAB$^+$99] succeeded in identifying appropriate graphs for $\#\mathrm{AC}^0$. Given the growing importance of counting classes, our motivation for the present work was the desire to avoid unwinding circuits into trees before counting their "proofs". Define a proof *circuit* to be a minimal subcircuit witnessing that a circuit outputs 1. More precisely, for a Boolean circuit C and an input x, a proof circuit is an edge-induced connected subcircuit of C which evaluates to 1 on x. This subcircuit must contain the output gate of C, as well as exactly

[*]Informatique et recherche opérationnelle, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada. Research performed in part while on leave at the Universität Tübingen. Supported by the (German) DFG, the (Canadian) NSERC and the (Québec) FCAR. mckenzie@iro.umontreal.ca

[†]Theoretische Informatik, Universität Würzburg, Am Exerzierplatz 3, 97072 Würzburg, Germany. Research supported by Northern Telecom. vollmer@informatik.uni-wuerzburg.de

[‡]Theoretische Informatik, Universität Würzburg, Am Exerzierplatz 3, 97072 Würzburg, Germany. Research supported by DFG, grant Wa 847/1-2. wagner@informatik.uni-wuerzburg.de

one C-edge into each $\vee$-gate and all C-edges into each $\wedge$-gate. The reader should convince herself that the circuit depicted above, which had 9 proof trees on input $x_1 = x_2 = 1$, has only 7 proof circuits on that input.

What counting classes arise from counting proof circuits instead of trees? This question held in stock two surprises, the first of which is the following algorithm:

1. replace $\vee$ by $+$ and $\wedge$ by $\times$ in a negation-free Boolean circuit C,
2. view C as a straight-line program prescribing in the usual way a formal polynomial in the input variables $x_1, \ldots, x_n$,
3. compute the polynomial top-down, with an important proviso: at each step, knock any nontrivial exponent down to 1 in the intermediate sum-of-monomials representation.

We get the number of proof circuits of C on an input x by evaluating the final polynomial at x! For example, the circuit depicted above had 7 proof circuits on input $x_1 = x_2 = 1$ because

$$\begin{aligned} g_1 &\rightarrow g_2 g_3 && (1)\\ &\rightarrow (x_1 + g_4) g_3 && (2)\\ &\rightarrow (x_1 + g_4)(g_4 + x_2) \quad \rightarrow \quad x_1 g_4 + x_1 x_2 + g_4 + g_4 x_2 && (3)\\ &\rightarrow x_1(x_1 + x_2) + x_1 x_2 + (x_1 + x_2) + (x_1 + x_2) x_2, && (4) \end{aligned}$$

where g_4^2 became g_4 in the middle of step 3.

One's intuition might be that such a simple strategy could be massaged into an arithmetic circuit or at least into a sublinear parallel algorithm [VSBR83]. Our second surprise was that counting proof circuits, even for depth-4 semi-unbounded circuits, is #P-complete. Hence, not only is our strategy hard to parallelize, it likely genuinely requires exponential time!

Our "surprise" algorithm thus counts proof *trees* in the absence of the idempotent rules $y^2 \rightarrow y$, and it counts proof *circuits* in their presence. Moreover, whereas an arithmetic circuit computing the number of proof *trees* of a circuit is readily available, producing such a circuit to compute proof *circuits* seems intractable. What is special about the idempotent rules? What would the effect of multivariate rules be? Which nontrivial rules would nonetheless permit expressing the counting in the form of an arithmetic circuit? What is a general framework in which complexity questions such as these can be investigated?

1.2 Results

We view our results as forming three main contributions.

Our first contribution is to define and classify *polynomial replacement systems* (prs for short). Prs provide the answer to the framework question. A prs in its full generality is a start polynomial $q \in \mathbb{N}[x_1, \ldots, x_m]$ together with a set of replacement rules. A replacement rule is a pair of polynomials (p_1, p_2). Informally, (p_1, p_2) is applicable to a polynomial q if q can be written in a form in which p_1 appears. Applying (p_1, p_2) to q then consists of replacing p_1 by p_2 (see Sect. 3 for formal definitions).

A prs generally defines a *set* of polynomials, since the choice and sequencing of the rules, and the way in which the rules are applied, may generate different polynomials. Computational problems of interest include computing the polynomials themselves (POLY), evaluating the polynomials at specific points (EVAL), and testing membership in their ranges (RANGE). We identify four natural families of prs: *simple* if the rules only replace variables, *deterministic* if no two rules have the same left-hand side, *acyclic* if no nontrivial infinite sequence of rules is applicable, and *idempotent* if the rules (y^2, y) are present.

For general prs, we obtain canonical forms, we discuss representation, and we outline broad complexity issues. Our detailed complexity analysis involves *simple* prs. For instance, we exhibit simple and deterministic prs for which RANGE is NP-complete. When the prs is given as part of the input, POLY is P-hard and in coRP, while RANGE is NP-complete and EVAL is P-complete.

Our second contribution concerns the specific case of proof trees and proof circuits. We prove that, to any Boolean circuit C and input x, corresponds an easily computable idempotent, simple, deterministic and acyclic prs S having the property that the number of proof trees (resp. proof circuits) of C on x is the maximum (resp. minimum) value of the EVAL problem for S on x, and vice versa (see Lemma 5.13). This offers one viewpoint on the reason why our algorithm from Subsect. 1.1 counts proof circuits correctly. We also prove that computing the minimum of the EVAL problem for idempotent, simple, deterministic and acyclic prs is #P-complete, or equivalently, that counting proof circuits is #P-complete under Turing reductions (but not under many-one reductions unless P = NP). This provides a new characterization of #P which is to be contrasted with Venkateswaran's (poly-degree, poly-depth) characterization [Ven92] and with the retarded polynomials characterization of Babai and Fortnow [BF91]. We also prove that detecting whether a circuit has more proof trees than proof circuits is NP-complete.

Our third contribution concerns the specific case of simple and acyclic prs. We prove that the EVAL problem for such prs is the evaluation problem for $\{\cup, +, \times\}$-circuits. These circuits have been considered previously (under the name hierarchical descriptions) in [Wag84, Wag86]. They are obtained by generalizing, from trees to general circuits, the $\{\cup, +, \times\}$-expressions (a.k.a. integer expressions), whose evaluation problem was shown NP-complete 25 years ago by Stockmeyer and Meyer [SM73]. From a PSPACE-upper bound given in [Wag84] we conclude that evaluation of simple acyclic prs has a polynomial space algorithm, and from a PSPACE-hardness result given in [Yan00] we then conclude PSPACE-completeness of our problem.

1.3 Paper organization

Section 2 defines proof trees and circuits, and proves complexity results about them, in a self-contained way independent from prs. The formal definition of a prs, as well as their canonical forms, are found in Sect. 3. Section 4 briefly discusses the representation of polynomials and the complexity of equivalence testing. Section 5 describes the four natural families of prs, and relates these to straight-line programs, $\{\cup, +, \times\}$-circuits, and the proof trees vs. proof circuits problem above. Section 6 investigates the complexity theoretic properties of simple prs in depth.

2 Counting Circuits vs. Counting Trees

2.1 Definition of Problems

By a circuit C, in this paper, we will mean a circuit over the basis $\{\wedge, \vee\}$ in the usual sense, with $2n$ inputs labelled $x_1, \dots, x_n, \neg x_1, \dots, \neg x_n$.

Fix an input x to C. Unwind C into a tree C' by (repeatedly) duplicating gates with fan-out greater than 1. Define an *accepting subtree* as a subgraph H of C' whose gates evaluate to 1 and which additionally fulfills the following properties: Subtree H must contain the output gate of C.

For every $\wedge$ gate v in H, all the input wires of v must be in H, and for every $\vee$ gate v in H, exactly one input wire of v must be in H. Only wires and nodes obtained in this way belong to H. By $\#C(x)$ we denote the number of accepting subtrees of C.

Define an *accepting subcircuit* as a subcircuit H of C with the same properties as above. (I.e., the only difference is that now we do *not* start by unwinding C into a tree.) Given an input x, let $\#_c C(x)$ denote the number of accepting subcircuits of C on x.

Since an accepting subtree or subcircuit of a circuit C is some form of proof that C evaluates to 1 on the given input, we will also refer to these as *proof trees* and *proof circuits*, resp. We will consider the following problems:

Problem:	PT	*Problem:*	PC
Input:	circuit C over $\{\wedge, \vee\}$, an input $x \in \{0,1\}^*$, a number k in unary	*Input:*	circuit C over $\{\wedge, \vee\}$, an input $x \in \{0,1\}^*$
Output:	$\#C(x) \bmod 2^k$	*Output:*	$\#_c C(x)$

Observe that if we unwind a circuit into a tree there may be an exponential blowup in size, which has the consequence that the number of accepting subtrees may be doubly-exponential in the size of the original circuit. This is not possible for the problem PC; the values of this function can be at most exponential in the input length. In order to achieve a fair comparison of the complexity of the problems, we therefore count proof trees only modulo an exponential number.

2.2 Some Initial Reductions

Lemma 2.1. PT *is complete for* FP *under* $\leq_m^{\log}$.

Proof. By the well-known connection between counting accepting subtrees and the evaluation of arithmetic circuits [Vin91b, Ven92, Jia92], PT $\in$ FP since it is sufficient to evaluate such a circuit modulo an exponential number.

Let now $f \in$ FP and $x \in \{0,1\}^*$. Let f for inputs of length $|x|$ be computed by the polynomial size circuit C. Let $g_1, \ldots, g_m$ be the output gates of C, and let $D_1, \ldots, D_m$ be those subcircuits of C whose output gates are the gates $g_1, \ldots, g_m$. Say that C is unambiguous if for every circuit D_i there is at most one accepting subtree. We may suppose without loss of generality that C is unambiguous [Lan93]. Now let C_i be the trivial circuit with 2^i accepting subtrees. Then $f(x)$ is equal to the number of accepting subtrees of the circuit $\bigvee_{i=1}^{m}(D_i \wedge C_i)$. ❑

Lemma 2.2.
1. *The following problem is* NP*-complete under* $\leq_m^{\log}$*: Given a circuit C, is there an input x such that $\#C(x) \neq \#_c C(x)$?*
2. *The following problem is* P*-complete under* $\leq_m^{\log}$*: Given a circuit C and an input $x \in \{0,1\}^*$, is $\#C(x) \neq \#_c C(x)$?*

Proof. 1. Containment in NP is obtained by the following algorithm: On input C, guess an input x and evaluate every gate in C. Check if there is an $\vee$ gate g such that both its inputs evaluate to 1, and there are at least 2 different paths from g to the output of C which are simultaneously contained in some accepting subtree. This latter problem is essentially the GAP problem, since we only have to check that there are two paths from g to C's output gate which join at some $\wedge$ gate in C, such that all gates on these paths evaluate to 1.

To prove completeness, we give a reduction from 3-SAT: Given a 3-CNF formula F, we construct an $\{\wedge, \vee\}$ circuit C_F with the same structure (that is, C_F actually is a formula). Let C' be some fixed circuit with differing numbers of accepting subtrees and subcircuits. Then the circuit $C =_{\text{def}} C_F \wedge C'$ will have $\#C(x) \neq \#_c C(x)$ for some input x if and only if F is satisfiable.

2. Observe that the algorithm given in the proof of case 1 is deterministic, once the input x is guessed. Hence the problem examined here is in P. Completeness is also shown along the lines above, this time using the P-complete circuit value problem. ❑

Remark 2.3. If we restrict our attention to circuits of depth d, then the problem in case 2 of the above lemma is in DSPACE(d); this is witnessed by the well-known depth-first search circuit evaluation algorithm which is used to show (uniform-)DEPTH(s) $\subseteq$ DSPACE(s).

2.3 Counting Proof-Circuits is #P-Complete

For functions f, h, we say that $f \leq_{\text{1-T}}^{\log} h$ if there are functions g_1, g_2 computable in logarithmic space such that, for all x, $f(x) = g_1\big(x, h(g_2(x))\big)$.

Theorem 2.4. PC *is complete for* #P *under* $\leq_{\text{1-T}}^{\log}$*, but not under* $\leq_m^{\mathrm{p}}$ *unless* P = NP.

Proof. For a conjunctive normal form H with 3 literals per clause, define SA(H) to be the number of satisfying assignments to the variables of H. It is known (see [Val79]) that SA is $\leq_m^{\mathrm{p}}$-complete for #P. We describe a $\leq_{\text{1-T}}^{\log}$-reduction from SA to PC. Let PC(C, x) denote the number of accepting subcircuits of circuit C with input x.

Let $H(x_1, \ldots, x_n) = \bigwedge_{i=1}^m C_i$ be a conjunctive normal form with 3 literals per clause We define a $\{\vee, \wedge\}$-circuit C_H, all of whose inputs are set to 1, which has 6 levels, and is stratified, i. e., edges are only between gates of adjacent levels. The levels are as follows:

1. Level 1 consists of the input-gates v_{ikl} for $i = 1, \ldots, m$, $k = 0, \ldots, n$ and $l = 0, 1$. In the application below, all these will be set to 1.
2. Level 2 consists of the $\vee$-gates v_{ik} for $i = 1, \ldots, m$ and $k = 0, \ldots, n$. Gate v_{ik} has incoming edges from v_{ik0} and v_{ik1}.
3. Level 3 consists of the $\wedge$-gates v_i for $i = 1, \ldots, m$. Gate v_i has incoming edges from $v_{i0}, v_{i1}, \ldots, v_{in}$.
4. Level 4 consists of the $\wedge$-gates $w_1, \ldots, w_n, w'_1, \ldots, w'_n$,. Gate w_j has incoming edges from all gates v_i such that x_j appears in the clause C_i. Gate w'_j has incoming edges from all gates v_i such that $\neg x_j$ appears in the clause C_i. Here, we stipulate that an $\wedge$-gate with no input wires computes the constant 1.
5. Level 5 consists of the $\vee$-gates $u_1, \ldots, u_n$. Gate u_j has incoming edges from the gates w_j and w'_j.
6. Level 6 consists of the output gate which is a $\wedge$-gate. It has incoming edges from the gates $u_1, \ldots, u_n$.

Now let $a_1, \ldots, a_n \in \{0, 1\}$. We define the circuit $C_H^{[a_1 \cdots a_n]}$ to be that subcircuit of C_H which results from cutting the edge between w'_j and u_j if $a_j = 1$ and between w_j and u_j if $a_j = 0$ ($j = 1, \ldots, n$). The following facts are easy to see:

Fact 1: $\mathrm{PC}\big(C_H, (1 \cdots 1)\big) = \sum_{a_1, \ldots, a_n \in \{0,1\}} \mathrm{PC}\big(C_H^{[a_1 \cdots a_n]}, (1 \cdots 1)\big)$.

Fact 2: $(a_1, \ldots, a_n)$ satisfies exactly k clauses of $H \Leftrightarrow \mathrm{PC}\big(C_H^{[a_1 \cdots a_n]}, (1 \cdots 1)\big) = 2^{k \cdot (n+1)}$.

Fact 3: If $(a_1, \ldots, a_n)$ satisfies H then $\mathrm{PC}\big(C_H^{[a_1 \cdots a_n]}, (1 \cdots 1)\big) = 2^{m \cdot (n+1)}$.

If $(a_1, \ldots, a_n)$ does not satisfy H then $\mathrm{PC}\big(C_H^{[a_1 \cdots a_n]}, (1 \cdots 1)\big) \leq 2^{(m-1) \cdot (n+1)}$.

Fact 4: $\mathrm{SA}(H) \cdot 2^{m\cdot(n+1)} \leq \mathrm{PC}\big(C_H, (1\cdots 1)\big) < \big(\mathrm{SA}(H)+1\big) \cdot 2^{m\cdot(n+1)}$.

Hence $\mathrm{SA}(H)$ can easily be computed from $\mathrm{PC}(C_H, (1\cdots 1))$, i.e., $\mathrm{SA} \leq^{\log}_{\text{1-T}} \mathrm{PC}$.

Now, assume PC is complete for #P under $\leq^{\mathrm{p}}_m$ reductions. Let $A \in \mathrm{NP}$, then there is a function $f \in \#\mathrm{P}$ such that $x \in A \iff f(x) > 0$ for all x. Under our assumption, there is a many-one reduction g from f to PC, hence $x \in A \iff \mathrm{PC}\big(g(x)\big) > 0$. This latter condition however can be checked in polynomial time since the underlying Boolean circuit has a positive number of proof circuits if and only if it evaluates to 1 (over the Boolean semi-ring). ❑

The just given proof shows, that the problem PC remains #P-complete even restricted to monotone circuits.

3 How to Generate Polynomials

A straight line program P over variables $x_1, \ldots, x_m$ is a set of instructions of one of the following types: $x_i \leftarrow x_j + x_k$, $x_i \leftarrow x_j \cdot x_k$, $x_i \leftarrow 0$, $x_i \leftarrow 1$, where $j, k < i$. Every variable appears at most once on the left hand side of the $\leftarrow$. Those variables that never appear on the left hand side of the $\leftarrow$ are the *input variables.* The variable x_m is the *output variable.* Given values for the input variables, the values of all other variables are computed in the obvious way. The value computed by P is the value of the output variable. Let p_P be the number-theoretic function computed in this way by P.

A straight line program hence is just another way of looking at an arithmetic circuit. By the above mentioned connection between the problem of counting proof trees and the evaluation of arithmetic circuits, we see that an obvious algorithm to determine the number of proof trees of a circuit consists of producing the appropriate straight line program and evaluating it in the order of its variables.

In Sect. 1.1 we sketched a related proof circuit counting algorithm. Next we give a more precise description of this algorithm: Given circuit C with input gates $x_1, \ldots, x_n$ and inputs $a_1, \ldots, a_n \in \{0,1\}$, let $g_1, \ldots, g_s$ be the non-input gates of C in topological order (i. e., a total order on the gates of C which is consistent with the partial order given by the wires, hence $u < v$ whenever there is a wire from u to v).

1. Transform C into a straight-line program P as above.
2. In the following, we manipulate a polynomial p, which has variables $g_1, \ldots, g_s, x_1, \ldots, x_n$. Initially we set p to g_s.
3. **for** $i = s, s-1, \ldots, 1$ **do**
   - 3.1. Replace variable g_i in p by the right hand side of that instruction in P, whose left hand side is g_i.
   - 3.2. Transform p into the "sum-of-monomials" form.
   - 3.3. Reduce all powers of variables in p to 1, i. e., replace all z^2 by z, where z is a variable. At the end of this step, p will be a multilinear polynomial.
4. Substitute the values of $a_1, \ldots, a_n$ for $x_1, \ldots, x_n$ and evaluate the resulting expression.

To see why this algorithm correctly counts proof circuits, let $M(C)$ be the following nondeterministic Turing machine. On input x, $M(C)$ first evaluates all the gates of C on input x and prunes away any zero gate. Unless the output gate g_s of C was pruned away, $M(C)$ starts at g_s in the pruned circuit and implements a recursive procedure *evaluate.* At an $\wedge$ gate g, *evaluate* triggers consecutive recursive calls, one for each predecessor of g. At an $\vee$ gate, *evaluate* triggers a single recursive call, for a predecessor of g chosen nondeterministically. At a gate with no predecessor, *evaluate* simply accepts. It is well known that $M(C)$ has $\#C(x)$ accepting paths (see, e.g. [CMTV98]). But how can $M(C)$ be made to have $\#_c C(x)$ accepting paths instead? Simply by making sure that, whenever *evaluate* encounters a gate g a second time, g is treated as the constant gate 1: this will reflect the fact that the choice of an accepting subcircuit rooted at g was already made, on the first encounter! Let $M_c(C)$ be the machine $M(C)$ modified in this way. The algorithm of Sect. 1.1, formalized above, precisely computes the number of accepting paths of $M_c(C)$. Encountering a gate g twice means that different paths out of g meet at an $\wedge$ gate (recall the proof of Lemma 2.2). This is equivalent, in the straight-line arithmetic program point of view, to obtaining a monomial in which g^2 appears. Replacing g^2 by g thus precisely models counting the number of accepting paths of $M_c(C)$, and thus the number of proof circuits of C on input x.

To summarize, the algorithm to compute proof circuits produces a polynomial (computing for argument $(x_1, \ldots, x_n)$ the number of proof trees of circuit C on input $x_1 \cdots x_n$), which is obtained by evaluating the straight-line program given by C, but whenever possible during the computation we replace a power x^2 by x. This is only a particular type of *polynomial replacement systems* we define next. Polynomial replacement systems will produce sets of polynomials from a given start polynomial, using rules replacing certain polynomials by other polynomials. This will be very similar to the way formal grammars produce sets of words from a start symbol, applying production rules.

In this paper we almost exclusively consider polynomials with nonnegative integer coefficients. This is motivated by the application to proof trees and proof circuits discussed above. We write $p(z_1, \ldots, z_s)$ to denote that p is such a polynomial in variables $z_1, \ldots, z_s$.

Below, the variable vector $\overline{x}$ will always be defined to consist of $\overline{x} = (x_1, \ldots, x_m)$. Let us say that the variable x_i is *fictive* (or, *inessential*) in the polynomial $p(\overline{x})$ if for all $a_1, \ldots, a_m, a'_i \in \mathbb{N}$ we have $p(a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_m) = p(a_1, \ldots, a_{i-1}, a'_i, a_{i+1}, \ldots, a_m)$. This means that x_i is fictive in p if and only if p can be written as a term in which x_i does not appear.

Definition 3.1. A *polynomial replacement system* (for short: *prs*) is defined as a quadruple $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ where

- $\{x_1, \ldots, x_n\}$ is the set of *terminal variables*,
- $\{x_{n+1}, \ldots, x_m\}$ is the set of *nonterminal variables*,
- q is a polynomial in the variables $x_1, \ldots, x_m$, the *start polynomial*, and
- R is a finite set of *replacement rules*, i. e., a finite set of pairs of polynomials in the variables $x_1, \ldots, x_m$.

How does such a system generate polynomials?

Definition 3.2. Let $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ be a prs, let p_1, p_2 be polynomials in the variables $\overline{x}$.

$$p_1 \underset{S}{\Longrightarrow} p_2 \iff_{\text{def}} \begin{array}{l}\text{there exist } (p_3, p_4) \in R \text{ and a polynomial } p_5(\overline{x}, y) \text{ such that} \\ p_1(\overline{x}) = p_5(\overline{x}, p_3(\overline{x})) \text{ and } p_2(\overline{x}) = p_5(\overline{x}, p_4(\overline{x})).\end{array}$$

Let $\overset{*}{\underset{S}{\Longrightarrow}}$ be the reflexive and transitive closure of $\underset{S}{\Longrightarrow}$, i. e., $p_1 \overset{*}{\underset{S}{\Longrightarrow}} p_2$ if and only if there exist $t \geq 0$ and polynomials $q_0(\overline{x}), q_1, (\overline{x}) \ldots, q_t(\overline{x})$ such that $p_1 = q_0 \underset{S}{\Longrightarrow} q_1 \underset{S}{\Longrightarrow} \cdots \underset{S}{\Longrightarrow} q_t = p_2$.

It turns out that the above form for derivations can be simplified:

Definition 3.3. Let S, p_1, p_2 be as above.

$$p_1 \underset{S}{\rightarrow} p_2 \iff_{\text{def}} \begin{array}{l}\text{there exist } (p_3, p_4) \in R \text{ and polynomials } p_5(\overline{x}), p_6(\overline{x}) \text{ such that} \\ p_1(\overline{x}) = p_5(\overline{x}) \cdot p_3(\overline{x}) + p_6(\overline{x}) \text{ and } p_2(\overline{x}) = p_5(\overline{x}) \cdot p_4(\overline{x}) + p_6(\overline{x}).\end{array}$$

Let $\overset{*}{\underset{S}{\rightarrow}}$ be the reflexive and transitive closure of $\underset{S}{\rightarrow}$.

Lemma 3.4 (Normal Form of Replacement).
For any prs $S = (\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R)$ and any polynomials $p_1(\overline{x}), p_2(\overline{x})$, we have:

$$p_1 \overset{*}{\underset{S}{\Longrightarrow}} p_2 \quad \textit{iff} \quad p_1 \overset{*}{\underset{S}{\rightarrow}} p_2.$$

Proof. Clearly, $p_1 \underset{S}{\rightarrow} p_2$ implies $p_1 \underset{S}{\Longrightarrow} p_2$. We prove that $p_1 \underset{S}{\Longrightarrow} p_2$ implies $p_1 \overset{*}{\underset{S}{\rightarrow}} p_2$. Let $(p_3, p_4) \in R$, and let p_5 be a polynomial such that $p_1(\overline{x}) = p_5(\overline{x}, p_3(\overline{x}))$ and $p_2(\overline{x}) = p_5(\overline{x}, p_4(\overline{x}))$. We can represent p_5 as $p_5(\overline{x}, y) = \sum_{i=0}^{k} q_i(\overline{x}) \cdot y^i$ with suitable $k \geq 0$ and polynomials $q_0, q_1, \ldots, q_k$. Hence $p_1(\overline{x}) = \sum_{i=0}^{k} q_i(\overline{x}) \cdot p_3(\overline{x})^i$ and $p_2(\overline{x}) = \sum_{i=0}^{k} q_i(\overline{x}) \cdot p_4(\overline{x})^i$. Defining the polynomials

$$r_m(\overline{x}, y) =_{\text{def}} \sum_{i=0}^{m-1} q_i(\overline{x}) \cdot p_4(\overline{x})^i + \sum_{i=m}^{k} q_i(\overline{x}) \cdot p_4(\overline{x})^{m-1} \cdot p_3(\overline{x})^{i-m} \cdot y$$

for $m = 1, \ldots, k$, and

$$t_m(\overline{x}) =_{\text{def}} \sum_{i=0}^{m-1} q_i(\overline{x}) \cdot p_4(\overline{x})^i + \sum_{i=m}^{k} q_i(\overline{x}) \cdot p_4(\overline{x})^{m-1} \cdot p_3(\overline{x})^{i-m+1}$$

for $m = 1, \ldots, k+1$, we get $r_m(\overline{x}, p_3(\overline{x})) = t_m(\overline{x})$ and $r_m(\overline{x}, p_4(\overline{x})) = t_{m+1}(\overline{x})$ for $m = 1, \ldots, k$, and moreover, $t_1(\overline{x}) = p_1(\overline{x})$ and $t_{k+1}(\overline{x}) = p_2(\overline{x})$.

Consequently, $p_1 = t_1 \underset{S}{\rightarrow} t_2 \underset{S}{\rightarrow} \cdots \underset{S}{\rightarrow} t_k \underset{S}{\rightarrow} t_{k+1} = p_2$ and thus $p_1 \overset{*}{\underset{S}{\rightarrow}} p_2$. ❑

A prs thus generates a set of polynomials; hence we define:

Definition 3.5. For a prs $S = (\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R)$, let

$$\text{POLY}(S) = \{p(x_1, \ldots, x_n) \mid \text{there exists } p'(\overline{x}) \text{ such that } q \overset{*}{\underset{S}{\Longrightarrow}} p' \text{ and } p(x_1, \ldots, x_n) = p'(x_1, \ldots, x_n, a_{n+1}, \ldots, a_m) \text{ for all } a_{n+1}, \ldots, a_m \in \mathbb{N}\}.$$

From the set $\text{POLY}(S)$ of polynomials we derive several sets of natural numbers, whose complexities we will determine in the upcoming sections.

Definition 3.6. Let $S = (\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R)$ be a prs. Define

- $\text{RANGE}(S) =_{\text{def}} \{ p(a) \mid p \in \text{POLY}(S) \wedge a \in \mathbb{N}^n \}$;
- $\text{EVAL}(S) =_{\text{def}} \{ (a, p(a)) \mid p \in \text{POLY}(S) \text{ and } a \in \mathbb{N}^n \}$.

Observe that if we also allow negative numbers as coefficients for our polynomials, then there are prs S such that $\text{RANGE}(S)$ is not decidable. This is seen as follows. By the Robinson-Matiasjevič result (see [Mat93]), every recursively enumerable set can be represented in the form $\{ p(a) \mid a \in \mathbb{N}^n \}$ where p is a suitable n-ary polynomial with integer coefficients. Now let p be such an n-ary polynomial such that $\{ p(a) \mid a \in \mathbb{N}^n \}$ is not decidable. Defining the prs $S_p =_{\text{def}} (\{x_1, \ldots, x_n\}, \emptyset, p, \emptyset)$ we obtain $\text{POLY}(S_p) = \{p\}$ and $\text{RANGE}(S_p) = \{ p(a) \mid a \in \mathbb{N}^n \}$.

Besides the membership problems $\text{POLY}(S)$, $\text{RANGE}(S)$, and $\text{EVAL}(S)$, we also consider the corresponding *variable membership problems.*

Definition 3.7. – $\text{POLY}(\cdot) =_{\text{def}} \left\{ (S,p) \;\middle|\; S \text{ prs and } p \in \text{POLY}(S) \right\}$;

– $\text{RANGE}(\cdot) =_{\text{def}} \left\{ (S,a) \;\middle|\; S \text{ prs and } a \in \text{RANGE}(S) \right\}$;

– $\text{EVAL}(\cdot) =_{\text{def}} \left\{ (S,a,p(a)) \;\middle|\; S \text{ prs}, p \in \text{POLY}(S), \text{ and } a \in \mathbb{N}^* \right\}$.

4 Representations of Polynomials and Equivalence Test

When we want to examine the complexity of the above defined sets we have to talk about representations of polynomials. We distinguish different kinds of representations.

Definition 4.1. Let $p(\overline{x})$ be a polynomial.

(1) The *full representation* of p is its sum-of-monomials form. That is, we describe p as a sequence of vectors $(c, e_1, \ldots, e_m)$, each consisting of $m+1$ nonnegative integers, where $c, e_1, \ldots, e_m$ are given in unary. Such a vector stands for the monomial $c \cdot \prod_{i=1}^{m} x_i^{e_i}$.

(2) In the *extended full representation* (*ef representation*, for short), we describe p as a sequence of vectors as above, but this time c is given in binary while all numbers $e_1, \ldots, e_m$ are given in unary.

(3) In the *formula representation*, p is described by a formula involving the variables $\overline{x}$. More precisely, the set of formulas in variables $\overline{x}$ is defined inductively by the following rules:

   (a) $x_1, \ldots, x_m, 0, 1$ are formulas.

   (b) If F, G are formulas, then so are $(F+G)$ and $(F \times G)$.

(4) In the *straight-line program representation* (*slp representation*, for short), we describe p by a straight-line program with input variables $\overline{x}$ that computes p.

If Φ is a representation of a polynomial of one of the above types, then we denote the polynomial represented by Φ by p_Φ.

It is easy to see that the above definition introduces a chain of representations with increasing succinctness. For every representation of type (i) of a polynomial, equivalent representations of types (j) for $j > i$ can be computed in logarithmic space. Moreover, if p has a full or ef or formula representation of size n, then the degree of p is bounded by n, and if p has an slp representation of size n, then the degree of p is at most exponential in n.

The idea to use slp's as a data structure for polynomials was introduced and promoted by Erich Kaltofen (see, e.g., [Kal88]). Though the succinctness of this way of representing polynomials compared with full or ef representation (or other so called sparse representations considered in the literature) seems to be non-negligible, one of the results of this paper will be that the complexity of the sets defined in Sect. 3 does not differ when we go from one form of representation to another.

A central problem in the upcoming sections will be the complexity of the problem to determine if two representations stand for the same polynomial, i. e., the complexity of the *equivalence problem*

$$\left\{ \langle \Phi_1, \Phi_2 \rangle \;\middle|\; \Phi_1 \text{ and } \Phi_2 \text{ are representations of polynomials such that } p_{\Phi_1} = p_{\Phi_2} \right\}$$

for the various types of representations. A similar result, as we give in our next theorem, was obtained in [IM83].

Theorem 4.2. *The equivalence problem is*

1. *in* P *for full and ef representation;*
2. *in* coRP *for formula and slp representation.*

The proof will make instrumental use of the following well-known result [Sch80, Zip79] (see also [MR95, p. 165f]):

Proposition 4.3 (Schwartz-Zippel). *Let $p(x_1, \ldots, x_n)$ be a polynomial of degree d over some field $\mathbb{F}$, that is not the zero polynomial. Let $\mathbb{S} \subseteq \mathbb{F}$ be a finite set, and let $a_1, \ldots, a_n$ be chosen independently and uniformly at random from $\mathbb{S}$. Then*

$$\operatorname{prob}\left[p(a_1, \ldots, a_n) = 0\right] \leq \frac{d}{|\mathbb{S}|}.$$

Proof. (of Theorem 4.2)

Two polynomials in (extended) full representation are equivalent if and only if they have the same monomials. This can be checked in polynomial time.

Given now two formulas F_1 and F_2, we know that the degree d of the two represented polynomials is at most linear in the length of F_1 and F_2. Hence by Proposition 4.3 it suffices to evaluate F_1 and F_2 for a random input drawn from an exponential number of possible input vectors. The actual evaluation of the formulas is in P.

Finally we are given two slp's P_1 and P_2. Compared with the above, we now have to deal with the fact the degree of the represented polynomial can be exponential, hence the numbers we operate with can be double exponential in $n =_{\text{def}} |P_1| + |P_2|$. Therefore we cannot carry out the necessary arithmetic operations in polynomial time. Instead we proceed as follows (see also [MR95, p. 169]):

Suppose the two numbers $a, b < 2^{2^n}$ are different, then $|a-b|$ can have at most 2^n different prime divisors. Let $\pi(m)$ denote the number of primes smaller than or equal to m. Hence for a random prime smaller than $k2^n \log(k2^n)$, the probability that $a \equiv b \pmod{p}$ is at most $\frac{2^n}{\pi\left(k2^n \log(k2^n)\right)} \leq O\left(\frac{1}{k}\right)$. Thus if two numbers are different, then they are different modulo most small primes.

Hence P_1 and P_2 are not equivalent iff for most inputs out of an exponential range we get inequality iff for most inputs out of an exponential range we get inequivalence modulo most small primes. Since the set of prime numbers is in ZPP, this algorithm is an RP algorithm. ❑

5 Different Types of Replacement Systems

The definition of polynomial replacement systems we presented above is very general. Here, we introduce a number of natural restrictions. Our approach is similar to the way different restrictions of grammar types were introduced, e.g., in the definition of the classes of the Chomsky hierarchy. We will later view the problems of counting proof trees and proof circuits as two instances of a problem about these restricted prs types.

5.1 Simple Polynomial Replacement Systems

Definition 5.1. A prs $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ is *simple* (or *context-free*), if the polynomials in the left-hand sides of the rules of R are variables from $\{x_{n+1}, \ldots, x_m\}$.

All definitions made in the preceding section for general prs carry over to the special cases of simple systems. However, for simple prs we additionally define a particular type of replacement, where the application of a rule (z, q) results in the replacement of *all occurrences* of z with q. This latter form is denoted by $\underset{S}{\models\!\!\!\Rightarrow}$, in contrast to the notation $\underset{S}{\Longrightarrow}$ for the derivations defined so far. Formally:

Definition 5.2. Let $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ be a simple prs.

$$p_1 \underset{S}{\models\!\!\Rightarrow} p_2 \iff_{\text{def}} \begin{array}{l}\text{there exist } (x_i, p_3) \in R \text{ such that}\\ p_2(\overline{x}) = p_1(x_1, \ldots, x_{i-1}, p_3(\overline{x}), x_{i+1}, \ldots, x_m).\end{array}$$

Let $\overset{*}{\underset{S}{\models\!\!\Rightarrow}}$ be the reflexive and transitive closure of $\underset{S}{\models\!\!\Rightarrow}$.

For the sets of polynomials and numbers derived from simple systems using our new derivation type, we use the same names as before but now use square brackets $[\cdots]$ instead of parentheses $(\cdots)$; formally:

Definition 5.3. For a simple prs $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$, let

$$\textsc{poly}[S] = \Big\{p(x_1, \ldots, x_n) \;\Big|\; \text{there exists } p'(\overline{x}) \text{ such that } q \overset{*}{\underset{S}{\models\!\!\Rightarrow}} p' \text{ and } p(x_1, \ldots, x_n) = p'(x_1, \ldots, x_n, a_{n+1}, \ldots, a_m) \text{ for all } a_{n+1}, \ldots, a_m \in \mathbb{N}\Big\}.$$

As in the case of $\textsc{poly}(S)$ we define

Definition 5.4. Let $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ be a simple prs. Define
- $\textsc{range}[S] =_{\text{def}} \{\, p(a) \mid p \in \textsc{poly}[S] \wedge a \in \mathbb{N}^n \,\}$;
- $\textsc{eval}[S] =_{\text{def}} \{\, (a, p(a)) \mid p \in \textsc{poly}[S] \text{ and } a \in \mathbb{N}^n \,\}$.

We also define the following variable membership problems:
- $\textsc{poly}[\cdot] =_{\text{def}} \{\, (S, p) \mid S \text{ simple prs and } p \in \textsc{poly}[S] \,\}$;
- $\textsc{range}[\cdot] =_{\text{def}} \{\, (S, a) \mid S \text{ simple prs and } a \in \textsc{range}[S] \,\}$;
- $\textsc{eval}[\cdot] =_{\text{def}} \{\, (S, a, p(a)) \mid S \text{ simple prs}, p \in \textsc{poly}[S] \text{ and } a \in \mathbb{N}^* \,\}$.

It is clear that for any simple prs S, we have $\textsc{poly}[S] \subseteq \textsc{poly}(S)$; hence also $\textsc{range}[S] \subseteq \textsc{range}(S)$, $\textsc{eval}[S] \subseteq \textsc{eval}(S)$, $\textsc{poly}[\cdot] \subseteq \textsc{poly}(\cdot)$, $\textsc{range}[\cdot] \subseteq \textsc{range}(\cdot)$, and $\textsc{eval}[\cdot] \subseteq \textsc{eval}(\cdot)$.

5.2 Simple Deterministic or Acyclic Polynomial Replacement Systems

Definition 5.5. A prs $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ is said to be *deterministic*, if no two different rules in R have the same left-hand side.

Definition 5.6. Let $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ be a prs. The *dependency graph* G_S of S is the directed graph $G_S = \big(\{1, \ldots, m\}, E_S\big)$, where E_S consists of all edges (j, i) for which there exists a rule $(p_1, p_2) \in R$ such that x_i is essential in p_1 and x_j is essential in p_2. The prs S is said to be *acyclic*, if its dependency graph G_S is acyclic.

Lemma 5.7. *For every simple and deterministic prs S, there exists a simple, deterministic, and acyclic prs S' such that $\textsc{poly}(S) = \textsc{poly}(S')$ and $\textsc{poly}[S] = \textsc{poly}[S']$. The system S' can be obtained from S in polynomial time.*

Proof. Let $S = \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_m\}, q, R\big)$ be a simple and deterministic prs. If S is not acyclic then there exist $k \geq 1$, $i_1, \ldots, i_k \in \{n+1, \ldots, m\}$ and polynomials $p_1(\overline{x}), \ldots, p_k(\overline{x})$ such that $(x_{i_1}, p_1), (x_{i_2}, p_2), \ldots, (x_{i_k}, p_k) \in R$ and $x_{i_{r+1}}$ is essential in p_r for $r = 1, \ldots, k$ (where we set $i_{k+1} = i_1$). Since $x_{i_1}, \ldots, x_{i_k}$ are nonterminal variables they have to be fictive in every polynomial p such that $q \overset{*}{\underset{S}{\Longrightarrow}} p \overset{*}{\underset{S}{\Longrightarrow}} p'$ and $p' \in \textsc{poly}(S)$. However, S is deterministic, and hence such a replacement would necessarily run along the cycle described by the rules $(x_{i_1}, p_1), (x_{i_2}, p_2), \dots, (x_{i_k}, p_k)$. Hence, if p has a non-fictive variable from $\{x_{i_1}, \dots, x_{i_k}\}$ then every p' such that $p \overset{*}{\underset{S}{\Longrightarrow}} p'$ will have a non-fictive variable from $\{x_{i_1}, \dots, x_{i_k}\}$. Thus we have $\text{POLY}(S) = \text{POLY}(S')$ and $\text{POLY}[S] = \text{POLY}[S']$ where

$$\begin{aligned} S' &=_{\text{def}} \big(\{x_1, \dots, x_n\}, \{x_{n+1}, \dots, x_m\} \setminus \{x_{i_1}, \dots, x_{i_k}\}, q, R'\big) \quad \text{and} \\ R' &=_{\text{def}} R \setminus \big\{\, (x_i, p) \bigm| x_i \text{ or an essential variable in } p \text{ is from } \{x_{i_1}, \dots, x_{i_k}\} \,\big\} \end{aligned}$$

Now we repeat this cycle removement step as long as the prs still has cycles. Since in every step at least one nonterminal variable is removed, we get after a polynomial number of steps a prs which is simple, deterministic and acyclic. ❑

We also obtain the following easy properties:

Lemma 5.8. *1. If S is a simple and deterministic prs then $\text{POLY}(S) = \text{POLY}[S]$, and this set consists of at most one polynomial.*

2. If S is a simple and acyclic prs then $\text{POLY}(S)$ and $\text{POLY}[S]$ are finite.

Proof. If S is simple, deterministic, and without loss of generality moreover acyclic, then the rules replace every variable by a unique polynomial. Since S is simple we always have to replace all occurrences of each nonterminal variable sooner or later. Hence the result of the whole derivation process is unique.

If S is acyclic then every nonterminal variable can only be replaced in a constant number of steps, hence we can only obtain finitely many polynomials. ❑

Note that there are simple and acyclic prs S such that $\text{POLY}[S] \subsetneq \text{POLY}(S)$. For example take $S = (\{x\}, \{z\}, 2z, \{(z, x), (z, 2x)\})$ where $\text{POLY}[S] = \{2x, 4x\}$ and $\text{POLY}(S) = \{2x, 3x, 4x\}$. Thus, the requirement that S is deterministic is necessary in Lemma 5.8.1.

In the remainder of this subsection, we relate simple deterministic and simple acyclic prs to different forms of circuits operating over the natural numbers.

First, it is intuitively clear that there is some connection between simple, deterministic, and acyclic systems and straight-line programs. This is made precise in the following lemma.

Lemma 5.9. *1. If S is a simple, deterministic, and acyclic prs such that $\text{POLY}(S) \neq \emptyset$ then there exists a slp P such that $\text{POLY}(S) = \{p_P\}$.*

2. If P is a slp then there exists a simple, deterministic, and acyclic prs S such that $\{p_P\} = \text{POLY}(S)$.

3. The transformations from a simple, deterministic, and acyclic prs to the corresponding slp and vice-versa can be computed in logarithmic space.

Proof. The program P is obtained from S by transforming every single replacement rule into a sequence of slp instructions and then ordering these according to a topological order of G_S. Statement 2 is proved similarly. Statement 3 is obvious. ❑

Next we show that acyclic systems are strongly related to a new type of arithmetic circuit we now define. These circuits are immediate generalizations of *integer expressions*, introduced by Stockmeyer and Meyer [SM73]. Therefore we call our circuits *integer circuits* (not to be confused with ordinary arithmetic circuits), or, referring to the operations allowed, $(\cup, +, \times)$-circuits.

An integer circuit with n inputs is a circuit C where the inner nodes compute one of the operations $\cup, +, \times$. Such a circuit C has a specified output gate g_s. It computes a function $f_C \colon \mathbb{N}^n \to 2^{\mathbb{N}}$ as follows: We first define for every gate $g \in C$ the function f_g computed by g.

1. If g is an input gate x_i, then $f_g(a_1, \dots, a_n) =_{\text{def}} \{a_i\}$ for all $a_1, \dots, a_n \in \mathbb{N}$.
2. If g is $+$ gate with predecessors g_l, g_r, then $f_g(a_1, \dots, a_n) =_{\text{def}} \{\, k+m \mid k \in f_{g_l}(a_1, \dots, a_n),\ m \in f_{g_r}(a_1, \dots, a_n) \,\}$. The function computed by a $\times$ gate is defined analogously.
3. If g is a $\cup$ gate with predecessors g_l, g_r, then $f_g(a_1, \dots, a_n) =_{\text{def}} f_{g_l}(a_1, \dots, a_n) \cup f_{g_r}(a_1, \dots, a_n)$.

Finally, the function computed by C is $f_C =_{\text{def}} f_{g_s}$.

The following relation between simple, acyclic replacement systems and integer circuits is obtained by an easy induction:

Lemma 5.10. *1. For every simple, acyclic prs $S = (\{x_1, \dots, x_n\}, \{x_{n+1}, \dots, x_m\}, q, R)$, there is an integer circuit C with n inputs such that $f_C(a) = \{\, b \mid (a, b) \in \text{EVAL}(S) \,\}$ for all $a \in \mathbb{N}^n$.*

2. For every integer circuit C with n inputs, there is a simple, acyclic prs S such that $\{\, b \mid (a, b) \in \text{EVAL}(S) \,\} = f_C(a)$ for all $a \in \mathbb{N}^n$.

3. The transformations from a simple, acyclic prs to the corresponding integer circuit and vice-versa can be computed in logarithmic space.

We consider the following problems:

$$\begin{aligned}
\mathbb{N}\text{-MEMBER}(\cup, +, \times) &=_{\text{def}} \{\, (C, a_1, \dots, a_n, b) \mid C \text{ is an integer circuit with } n \text{ inputs}, \\
&\qquad a_1, \dots, a_n, b \in \mathbb{N} \text{ and } b \in f_C(a_1, \dots, a_n) \,\} \\
\mathbb{N}\text{-RANGE}(\cup, +, \times) &=_{\text{def}} \{\, (C, b) \mid C \text{ is an integer circuit with } n \text{ inputs}, \\
&\qquad b \in \mathbb{N} \text{ and } (\exists a_1, \dots, a_n) b \in f_C(a_1, \dots, a_n) \,\}
\end{aligned}$$

Analogous notations will be used when we restrict the gate types allowed.

The following lemma is immediate from Lemma 5.10:

Lemma 5.11. *For all representations, the following holds:*

1. $\mathbb{N}\text{-MEMBER}(\cup, +, \times) \equiv_m^{\log} \text{EVAL}(\cdot)$.

2. $\mathbb{N}\text{-RANGE}(\cup, +, \times) \equiv_m^{\log} \text{RANGE}(\cdot)$.

5.3 Idempotent Polynomial Replacement Systems

Definition 5.12. For a prs $S = (\{x_1, \dots, x_n\}, \{x_{n+1}, \dots, x_m\}, q, R)$, let $S_{\text{idem}} =_{\text{def}} (\{x_1, \dots, x_n\}, \{x_{n+1}, \dots, x_m\}, q, R \cup \{\, (x_i^2, x_i) \mid 1 \le i \le m \,\})$ be the *idempotent prs* derived from S.

In the case that S is simple (deterministic, acyclic, resp.), we will say that S_{idem} is an idempotent simple (deterministic, acyclic, resp.) prs.

For a prs $S = (\{x_1, \dots, x_n\}, \{x_{n+1}, \dots, x_m\}, q, R)$ and $a \in \mathbb{N}^n$, we write $\min \text{EVAL}(S, a)$ as a shorthand for $\min\{\, p(a) \mid p \in \text{POLY}(S) \,\}$ (analogously, we use $\max \text{EVAL}(S, a)$).

Lemma 5.13. *1. For every Boolean circuit C, input x, and $k \in \mathbb{N}$, there exists a simple, deterministic and acyclic polynomial replacement system S such that $\min \text{EVAL}(S_{\text{idem}}, (1, \dots, 1)) = \#_c C(x)$, and $\max \text{EVAL}(S_{\text{idem}}, (1, \dots, 1)) = \#C(x)$.*

2. For every simple, deterministic, and acyclic prs S_{idem} there exists a Boolean circuit C such that $\min \text{EVAL}(S_{\text{idem}}, (1, \dots, 1)) = \#_c C(x)$, and $\max \text{EVAL}(S_{\text{idem}}, (1, \dots, 1)) = \#C(x)$.

3. The transformations from an idempotent simple, deterministic, and acyclic prs to the corresponding circuit and vice-versa can be computed in logarithmic space.

Proof. Arithmetic circuits and straight-line programs are only two different views of the same concept. Hence we go from circuit C to system S and vice versa exactly as in Lemma 5.9. Clearly the simple, deterministic, and acyclic system S computes $\#C(x)$ as explained in the discussion in the beginning of Sect. 3.

Now convert S into the idempotent system S_{idem}. That the number of proof circuits coincides with the minimal element in $\text{EVAL}\big(S_{\text{idem}}, (1, \dots, 1)\big)$ is a consequence of the algorithm given at the beginning of Sect. 3. The minimal element is obtained if and only if during the derivation of a polynomial, all rules (x_i^2, x_i) are applied whenever possible. The maximal element is obtained if and only if we never pick such a rule, i.e., if we use only rules from S. ❑

6 Complexity Results for Simple Replacement Systems

6.1 Deterministic Systems

In this section, we consider the complexity of the above defined sets for simple replacement systems. Let us start with the complexity of fixed membership problems.

Theorem 6.1. *Let S be simple and deterministic.*

1. $\text{POLY}(S), \text{POLY}[S]$ *are* P*-complete for all representations.*
2. $\text{RANGE}(S), \text{RANGE}[S] \in \text{NP}$*. In fact, there are systems S such that the problems* $\text{RANGE}(S)$ *and* $\text{RANGE}[S]$ *are* NP*-complete.*
3. $\text{EVAL}(S), \text{EVAL}[S] \in \text{TC}^0$.

Proof. First we recall that $\text{POLY}(S)$ and $\text{POLY}[S]$ consist of at most one polynomial p. In full and ef representation, p has only finitely many representations, hence the complexity of $\text{POLY}(S), \text{POLY}[S]$ is trivial. However, in formula and slp representation, p may have infinitely many representations. That is, we face the following problem: Given an slp P (the case of formulas is even easier), does $p_P = p$ (where p is fixed)?

Say that P is reduced if the following holds: P has no instructions of the form $x_j \leftarrow \cdots$, where the x_j never appears on the right hand side of any other instructions (unless x_j is the output variable). Additionally, if $x_j \leftarrow x_i + x_k$ is an instruction in P, then the polynomials computed by x_i and x_k are both not the constant zero polynomial (otherwise, the instruction is useless, for, e.g., if $x_i \equiv 0$, then $x_j \equiv x_k$, hence we do not compute anything new; we might delete the above instruction and in further instructions we use x_k instead of x_j). Similarly, for $x_j \leftarrow x_i \cdot x_k$ we require that x_i and x_k are not constant one.

Now the following holds: Every polynomial p has only finitely many representations by reduced slp's (except for an isomorphic renumbering of the variables). Moreover, a program P can be transformed into reduced form as follows: Inductively, determine the sets of variables that compute the constant zero or constant one polynomial. Then remove those instructions that do not compute a new polynomial, and change the variables in the other instructions as described above. This shows that $\text{POLY}(S), \text{POLY}[S] \in \text{P}$.

Hardness follows from a reduction from the circuit value problem as follows: Let C be a Boolean circuit. Let p be the polynomial that gives the number of proof trees of C. Let $La(S) = \text{POLY}[S] = \{p'\}$; then $p + p' \in \text{POLY}(S) = \text{POLY}[S]$ iff C does not evaluate to 1 iff C is not in CVP.

For 2 we have to decide the range of a multivariate polynomial with nonnegative coefficients. This is clearly in NP by the obvious guessing algorithm. For the hardness proof, we give a reduction from the quadratic Diophantine equations problem (problem AN8 in [GJ79, p. 250]) as follows: This problem consists of all triples (a,b,c) such that the quadratic equation $ax^2+by=c$ has a solution in positive integers. Define $\langle x,y,z\rangle =_{\rm def} z+(y+z)^2+(x+y+z)^3$. Observe that $\langle\cdots\rangle$ is one-one. Now, define the 4-variable polynomial $p(u,v,x,y) =_{\rm def} ux^2+vy$. Then for all $a,b,c\in\mathbb{N}$, the equation $ax^2+by=c$ has a solution if and only if there are x,y such that $p(a,b,x,y)=c$ if and only if there are x,y such that $\big\langle a,b,p(a,b,x,y)\big\rangle = \langle a,b,c\rangle$ if and only if there are u,v,x,y such that $\big\langle u,v,p(u,v,x,y)\big\rangle = \langle a,b,c\rangle$. Hence we see that the quadratic Diophantine equations problem reduces to the range of the polynomial $q(u,v,x,z) =_{\rm def} \big\langle u,v,p(u,v,x,y)\big\rangle$.

For 3 observe we have to evaluate a fixed polynomial, hence the result follows since addition and multiplication are in TC^0. ❑

Concerning variable membership problems of simple, deterministic systems, we obtain:

Theorem 6.2. *For simple and deterministic prs and for all representations,*

1. POLY$(\cdot)$ *and* POLY$[\cdot]$ *are in* coRP *and* P*-hard under* $\le_m^{\log}$*,*
2. RANGE$(\cdot)$ *and* RANGE$[\cdot]$ *are* NP*-complete under* $\le_m^{\log}$*,*
3. EVAL$(\cdot)$ *and* EVAL$[\cdot]$ *are* P*-complete under* $\le_m^{\log}$*.*

Proof. Containment: In all cases, given the simple and deterministic prs S, first construct in polynomial time the slp representation of the unique polynomial p (if any) in POLY(S) = POLY$[S]$ by Lemma 5.7 and Lemma 5.8. Statement 1 then follows by equivalence test (Theorem 4.2). For statement 2, guess suitable input values and evaluate p, for statement 3 just evaluate p (until finished or the bounds are exceeded, as above).

Hardness: Statement 3: We give a reduction from the circuit value problem CVP as follows: Given a circuit C and an input a to C, produce the replacement system S which computes the number of proof trees of C on input a. S is simple and deterministic (see Sect. 3), and if we "hardwire" the input a of C directly into S, the resulting polynomial has only fictive variables, i.e., is essentially a natural number. Now C accepts a iff this number is greater than 0, hence $(C,a)\in$ CVP iff $(S,a,0)\notin$ RANGE$[\cdot]$ = RANGE$(\cdot)$. Statement 1: Proceeding as above, we obtain $(C,a)\in$ CVP iff the constant 0 polynomial is not in POLY(S) = POLY$[S]$. Statement 2: Follows from Theorem 6.1.2. ❑

6.2 Acyclic Systems

Let us next deal with simple, acyclic, but not necessarily deterministic systems. For such systems S the sets POLY(S) and POLY$[S]$ are finite (by Lemma 5.8), hence we obtain analogously to Theorem 6.1:

Theorem 6.3. *Let S be simple and acyclic.*

1. POLY(S), POLY$[S]\in$ P *for all representations.*
2. RANGE(S), RANGE$[S]\in$ NP*, and there are systems S such that the problems* RANGE(S) *and* RANGE$[S]$ *are* NP*-complete.*
3. EVAL(S), EVAL$[S]\in \mathrm{TC}^0$*.*

Again, interesting questions arise when we examine variable membership problems.

Theorem 6.4. *For simple and acyclic prs and for all representations,*

1. POLY$[\cdot]$ *is contained in* MA *and is* NP*-hard under* $\le_m^{\log}$*,*
2. RANGE$[\cdot]$ *and* EVAL$[\cdot]$ *are* NP*-complete under* $\le_m^{\log}$*.*

Proof. The containment proofs are similar to that of Theorem 6.2, but before applying Lemma 5.7 we select nondeterministically a deterministic prs by deleting some of the rules of the originally given prs. The claim follows since MA $= \exists \cdot$ coRP [Gol99].

Hardness for $\text{POLY}[\cdot]$ is proven by giving a logspace many-one reduction from the sum-of-subset problem SOS (problem SP13 in [GJ79, p. 223]) to $\text{POLY}[\cdot]$ as follows: Let $m, b_1, \ldots, b_m, c \in \mathbb{N} \setminus \{0\}$. Define the prs S by

$$S =_{\text{def}} \big(\{x_1, \ldots, x_n\}, \{x_{n+1}, \ldots, x_{n+m}\}, x_{n+m}, R\big), \qquad \text{where}$$
$$R =_{\text{def}} \big\{(x_{n+1}, 0), (x_{n+1}, b_1)\big\} \cup \bigcup_{i=2}^{m} \big\{(x_{n+i}, x_{n+i-1}), (x_{n+i}, x_{n+i-1} + b_i)\big\}.$$

Note that S is acyclic and simple, and $\text{POLY}[S]$ consists only of polynomials without essential variables. For $c \in \mathbb{N}$, define $p_c(x_1, \ldots, x_n) =_{\text{def}} c$. Now we obtain

$$(b_1, \ldots, b_m, c) \in \text{SOS} \iff p_c \in \text{POLY}[S] \iff (S, p_c) \in \text{POLY}[\cdot].$$

Since a deterministic prs can (by Lemma 5.7) be assumed to be acyclic, hardness for $\text{RANGE}[\cdot]$ follows as in Theorem 6.2.

We next prove hardness for $\text{EVAL}[\cdot]$ by giving a logspace many-one reduction from the SOS problem: Given $m, b_1, \ldots, b_m, c > 0$, define S as in the proof of $\text{POLY}[\cdot]$ above. Then we obtain $(b_1, \ldots, b_m, c) \in \text{SOS} \iff (S, a, c) \in \text{EVAL}[\cdot]$ for any $a \in \mathbb{N}^n$. ❑

Next, we turn to different variable membership problems for simple, acyclic systems under "$\Longrightarrow$"- derivations.

Stockmeyer and Meyer considered integer expressions (in our terminology, these are integer circuits with fan-out of non-input gates at most 1) where the only operations allowed are $\cup$ and $+$. They proved that the membership problem in that case is NP-complete. It is easy to see that their result carries over to the case that we also allow multiplication, i. e., the problems $\mathbb{N}$-MEMBER$(\cup, +)$ and $\mathbb{N}$-MEMBER$(\cup, +, \times)$ for expressions are NP-complete. The corresponding problems for circuits were not considered in their paper, but in later papers by Wagner [Wag84, Wag86] (under the name *hierarchical descriptions*). Only PSPACE as upper bound is known from there, but recently it was shown by Ke Yang that both circuit problems are PSPACE-hard [Yan00].

Since (by Lemma 5.11), the member and range problems for these circuits are equivalent to the $\text{EVAL}(\cdot)$ and $\text{RANGE}(\cdot)$ problems for simple acyclic prs, we conclude:

Theorem 6.5. *For simple and acyclic prs and for all representations,*

1. $\text{POLY}(\cdot) \in$ EXPTIME,
2. $\text{RANGE}(\cdot), \text{EVAL}(\cdot)$ *are* PSPACE-*complete.*

6.3 Idempotent Systems

Again we note that also in this case $\text{POLY}(S)$ and $\text{POLY}[S]$ are finite, and we obtain results analogous to Theorem 6.1:

Theorem 6.6. *Let S be simple, deterministic, and acyclic.*

1. $\text{POLY}(S_{\text{idem}}) \in$ P *for all representations.*
2. $\text{RANGE}(S_{\text{idem}}) \in$ NP. *In fact, there are systems S such that the problem $\text{RANGE}(S_{\text{idem}})$ is* NP-*complete.*
3. $\text{EVAL}(S_{\text{idem}}) \in \text{TC}^0$.

For the variable membership problems the following can be said:

Theorem 6.7. *For idempotent, simple, deterministic, and acyclic prs and for all representations,* $\text{POLY}(\cdot), \text{RANGE}(\cdot), \text{EVAL}(\cdot) \in \text{EXPTIME}$.

Proof. Follows by the trivial evaluation of the system. ❑

Lemma 5.13 shows the importance of the minimization and maximization operations in the case of idempotent systems.

Theorem 6.8. *For idempotent, simple, deterministic, and acyclic replacement systems and for all representations,*

1. *the functions* $\min \text{EVAL}(\cdot)$ *and* $\min \text{EVAL}[\cdot]$ *are* $\#\text{P}$*-complete under* $\leq^{\log}_{1\text{-T}}$*-reductions,*
2. *the functions* $\max \text{EVAL}(\cdot)$ *and* $\max \text{EVAL}[\cdot]$ *are* FP*-complete under* $\leq^{\text{p}}_{m}$.

Proof. Immediate consequence of Lemma 2.1, Theorem 2.4 and Lemma 5.13. ❑

For completeness, we note:

Remark 6.9. For simple, deterministic and for simple, acyclic prs, the functions $\min \text{EVAL}(\cdot)$, $\min \text{EVAL}[\cdot], \max \text{EVAL}(\cdot), \max \text{EVAL}[\cdot]$ are FP-complete.

Proof. For simple, deterministic S and $a \in \mathbb{N}^*$, obtain from Lemma 5.7 and Lemma 6.1 the slp-representation of the unique polynomial p in $\text{POLY}(S)$ (if there is one). Then compute $p(a)$ which hence is the minimum and maximum. This element can be computed as in Theorem 6.2, showing that the functions are in FP. Hardness follows immediately from Lemma 2.1.

For simple, acyclic S, we evaluate S according to a topological order of G_S. The minimal element is obtained if for every variable x we only consider those rules with left hand-side x, whose right hand side is minimal. Hence we essentially deal with a deterministic system, and the upper bound follows from the above. The case of maximization is treated similarly. Hardness follows from hardness of the problem in the deterministic case. ❑

7 Conclusion

It might be of interest to consider the complexity of PC for circuits of restricted depth. The circuit constructed in the proof of Theorem 2.4 has depth 5. Simply melting the $\wedge$ gates on levels 3 and 4 makes it depth 4. Furthermore, it is easy to see that for depth $d+1$ circuits with an $\vee$ output-gate the problem is as hard as for depth d circuits with an $\wedge$ output-gate. Next observe that the problem is easy (i. e., in FP) for depth 2 circuits with an $\wedge$ output-gate (and hence also for depth 3 circuits with an $\vee$ output-gate).

This means that the complexity of PC is not known only in the case of depth 3 circuits with an $\wedge$ output-gate (or, equivalently, for depth 4 circuits with an $\vee$ output-gate). It is not hard to see that this reduces to the case of 4-level stratified circuits with

- level 1: 1-gates
- level 2: $\vee$ gates
- level 3: $\vee$ gates
- level 4: $\wedge$ gate (output gate)

This problem in its turn is equivalent to the following problem: Given: naturals $b_1, \ldots, b_m$ and the polynomial $f(x_1, \ldots, x_m) = \prod_{k=1,\ldots,n} \sum_{i=1,\ldots,m} a_{ki} \cdot x_i$ with $\mathrm{a}_{ki} \in \{0,1\}$. Exploit distributivity to reexpress this polynomial as a sum of monomials, and replace every x_i^r, $r > 1$, with x_i. Let $f^*(x_1, \ldots, x_m)$ be this new polynomial. Goal: Compute $f^*(b_1, \ldots, b_m)$. (The complexity is measured in the value – i.e. unary – of $n + b_1 + \cdots + b_m$.)

It is interesting to see that the problem of computing the permanent can be formulated in a similar way, only substituting "replace every x_i^r, $r > 1$, with x_i" by "replace every x_i^r, $r \geq 1$, with 0". Nevertheless, it is not clear how to use this idea to reduce the problem of computing the permanent to the above problem.

To determine the complexity of the sets $\text{RANGE}(S), \text{RANGE}[S]$ for fixed S is in most of the cases examined equivalent to determining the complexity of the range of a multivariate polynomial with nonnegative integer coefficients. While this is always an NP-problem, we showed that there is a 4-variable polynomial of degree 6 whose range is NP complete. Can this be improved?

From a complexity theoretic point of view, further obvious open questions are of course, if the gap between lower and upper bound in Theorems 6.5 and 6.7 can be closed.

A lot of interesting questions about prs remain open. To come back to some of the problems posed in Subsect. 1.1, we did not look at all at multivariate rules. Also, it seems worthwhile to examine if besides idempotent systems other prs families can related to various types of arithmetic circuits and counting problems in Boolean circuits.

Acknowledgment. We are grateful to Sven Kosub (Würzburg) and Thomas Thierauf (Ulm) for helpful discussions.

References

[AAB+99] E. Allender, A. Ambainis, D. Mix Barrington, S. Datta, and H. LêThanh. Bounded depth arithmetic circuits: Counting and closure. In *Proceedings 26th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, Berlin Heidelberg, 1999. Springer Verlag. To appear.

[AAD97] M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and arithmetic circuits. In *Proceedings 12th Computational Complexity*, pages 134–148. IEEE Computer Society, 1997.

[All98] E. Allender. Making computation count: arithmetic circuits in the nineties. *SIGACT News*, 28(4):2–15, 1998.

[AML98] A. Ambainis, D. A. Mix Barrington, and H. LêThanh. On counting AC^0 circuits with negative constants. In *Proceedings 23rd Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 1450, pages 409–417, Berlin, 1998. Springer-Verlag.

[BF91] L. Babai and L. Fortnow. Arithmetization: a new method in structural complexity theory. *Computational Complexity*, 1:41–66, 1991.

[CMTV98] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[Gol99] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics.* Springer Verlag, Berlin Heidelberg, 1999.

[IM83] O. Ibarra and S. Moran. Probabilistic algorithms for deciding equivalence of straight-line programs. *Journal of the ACM*, 30:217–228, 1983.

[Jia92] J. Jiao. Some questions concerning circuit counting classes and other low-level complexity classes. Master's essay, 1992.

[Kal88] E. Kaltofen. Greatest common divisors of polynomials given by straight-line programs. *Journal of the ACM*, 35:231–264, 1988.

[Lan93] K. J. Lange. Unambiguity of circuits. *Theoretical Computer Science*, 107:77–94, 1993.

[LêT98] H. LêThanh. *Circuits arithmétiques de profondeur constante*. PhD thesis, Université de Paris-Sud, 1998.

[Mat93] Y. V. Matiasjevič. *Hilbert's Tenth Problem.* Foundations of Computing Series. MIT Press, Cambridge, MA, 1993.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[Sch80] J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the Association for Computing Machinery*, 27:701–717, 1980.

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proceedings 5th ACM Symposium on the Theory of Computing*, pages 1–9. ACM Press, 1973.

[Sud78] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the Association for Computing Machinery*, 25:405–414, 1978.

[Val79] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8(3):411–421, 1979.

[Ven92] H. Venkateswaran. Circuit definitions of non-deterministic complexity classes. *SIAM Journal on Computing*, 21:655–670, 1992.

[Vin91a] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proceedings 6th Structure in Complexity Theory*, pages 270–284. IEEE Computer Society Press, 1991.

[Vin91b] V. Vinay. *Semi-Unboundedness and Complexity Classes.* PhD thesis, Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India, 1991.

[VSBR83] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12:641–644, 1983.

[VT89] H. Venkateswaran and M. Tompa. A new pebble game that characterizes parallel complexity classes. *SIAM J. on Computing*, 18:533–549, 1989.

[Wag84] K. W. Wagner. The complexity of problems concerning graphs with regularities. Technical report, Friedrich-Schiller-Universität Jena, 1984. Extended abstract in *Proceedings 11th Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 176, pages 544–552, 1984.

[Wag86] K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23:325–356, 1986.

[Yan00] K. Yang. Magic circuit is PSPACE-complete. In *Proceedings 15th Computational Complexity Conference*. IEEE Computer Society Press, 2000. To appear.

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of EUROSAM 79*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer Verlag, 1979.