LIF

Laboratoire d'Informatique Fondamentale
de Marseille

Unité Mixte de Recherche 6166
CNRS - Université de Provence - Université de la Méditerranée

Max-plus quasi-interpretations

Roberto M. Amadio

Rapport/Report 10-2002

10 Décembre, 2002

Max-plus quasi-interpretations

Roberto M. Amadio

Laboratoire d'Informatique Fondamentale

UMR 6166

CNRS - Université de Provence - Université de la Méditerranée

CMI, 39 rue Joliot-Curie, F-13453, Marseille, France.

amadio@cmi.univ-mrs.fr

Abstract/Résumé

Quasi-interpretations are a tool to bound the size of the values computed by a first-order functional program (or a term rewriting system) and thus a mean to extract bounds on its computational complexity. We study the synthesis of quasi-interpretations selected in the space of polynomials over the *max-plus* algebra determined by the non-negative rationals extended with $-\infty$ and equipped with binary operations for the maximum and the addition. We prove that in this case the synthesis problem is NP-hard, and in NP for the particular case of *multi-linear* quasi-interpretations when programs are specified by rules of bounded size. The relevance of multi-linear quasi-interpretations is discussed by comparison to certain syntactic and type theoretic conditions proposed in the literature to control time and space complexity.
Keywords: Functional languages and term rewriting. Function algebras and implicit computational complexity. Static analysis. Polynomial interpretations and max-plus algebras.

Les quasi-interprétations sont un outil pour borner la taille des valeurs calculées par un programme fonctionnel du premier ordre (ou un système de réécriture de termes) et ainsi un moyen pour extraire des bornes sur sa complexité. Nous étudions la synthése des polynômes sur l'algèbre max-plus determinée par le rationnels non-negatifs étendus avec $-\infty$. Nous demontrons que dans ce cas le problème de la synthèse est NP-difficile et dans NP pour le cas particulier des quasi-interpretations multi-linéaires quand les programmes sont specifiés par des règles de taille bornée. L'intérêt des quasi-interprétation multi-linéaires est discuté par comparaison à certaines restrictions syntaxiques et de typage proposées dans la litterature pour contrôler la complexité en temps et en espace.
Mots clefs : Langages fonctionnels et réécriture de termes. Algèbres de fonctions et complexité implicite. Analyse statique. Interprétations polynômiales et algèbre max-plus.

Relecteurs/Reviewers: Silvano Dal Zilio, Denis Lugiez.

Notes: This work was done while on leave at the *Ludwig-Maximilian Universität München* and was partially supported by the *IST-Global Computing Mobile Resource Guarantee Project* and the *Action Spécifique Méthodes formelles pour la mobilité.* The author benefitted from a number of discussions with Martin Hofmann.

1 Introduction

1.1 Motivations

The extraction of complexity bounds from first-order functional programs has a long history and a variety of motivations. In a fundamental perspective one is interested in providing *functional algebras* characterizations of small complexity classes (see [Clo95] for a survey). Cobham's characterization of polynomial time by bounded recursion on notation [Cob65] is an early example. Descriptive complexity theory [Imm99] is a related thread of work relying on the tools of finite model theory and the ideas of logic programming.

In a more applied perspective, one is interested in estimating the resources required by a program for its execution. This is particularly interesting in the framework of mobile and/or embedded code. A popular implementation schema called *proof-carrying code* [Nec97] requires the mobile code to come with a proof of its compliance to a particular security policy. This proof must be generated by the producer of the code and must be easily checkable by the user of the code. One of the requirements for this implementation schema to be effective is that the proof of compliance must be generated in an automatic or quasi-automatic way. It seems clear that it is easier to generate such proofs when the code is expressed in a high-level language rather than in a low level machine language (see, *e.g.*, [San01] for an elaboration of this point).

1.2 Technical approaches

Cobham's characterization of polynomial time functions is based on definitions by primitive recursion on binary notation where the size of the result of the defined function is explicitly bounded by a polynomial. From a programming point of view, the annoying aspect of bounded recursion on notation is that the programmer has *to find* the bound while defining functions by *primitive recursion*. In other terms, bounded recursion on notation does not offer much support for automatically finding this bound and moreover imposes the use of primitive recursion.

Some years ago, Bellantoni-Cook [BC92] and Leivant [Lei94] have introduced a notion of *ramification*. In [BC92] this is expressed as a distinction between *normal* and *safe* arguments and a restriction on the way primitive recursion can be applied and functions composed ([Lei94] introduces a related notion of *tier*). By complying to this programming discipline the programmer is relieved from the problem of explicitly providing a bound. This bound is implicit in the constraints imposed by ramification and when needed can be explicitly computed.

It has been observed [Cas97] that this programming discipline rules out many natural algorithms. To improve the situation, one approach, followed by Marion *et al.* [Mar00, MM00, BMM01] has been to extend the notion of ramification to various types of *recursive path orders* (a family of simplification ordering such as lexicographic, multi-set,...). In a quite different direction, Jones [Jon97] and Hofmann [Hof02] have proposed to consider *general* recursive programs for which a bound can be found on the size of the values handled during the computation. Jones obtains this bound by imposing a (quite severe) syntactic restriction on the programs, while Hofmann introduces a 'linear' type system with a *resource* type that guarantees that values obtained in the course of the computation are *non-size increasing.*

1.3 Quasi-interpretations

Quasi-interpretations have been proposed by Marion *et al.* in the context of the work on ramified recursive path orders we mentioned above. More precisely, quasi-interpretations are extracted from a termination proof in the *ramified* path order and they are then used as a tool to bound the *space* required to compute the result of the program. Alternatively, quasi-interpretations have been considered in combination with recursive path orderings (see section 6.3).

Quasi-interpretations are obviously inspired by *polynomial simplification interpretations* which are one of the traditional tools used in proving the *termination* of *term rewriting systems* (TRS), see, *e.g.*, [BN98]. The limit of (quasi-)interpretations is that first, one has to *synthesize* one, and second, one has to *verify* that the interpretation fits a given TRS. We recall that for polynomial interpretations over natural numbers the verification problem is already undecidable as a consequence of the undecidability of Hilbert's 10^{th} problem. When working over the reals the situation improves a bit since both the verification and the synthesis up to a given degree of polynomials are decidable by appeal to Tarski's decision procedure which has however a high complexity.

1.4 Contribution

In this paper we address the problem of the automatic *synthesis* of quasi-interpretations for general recursive programs (and not just those admitting a termination proof by ramified path order). For the efficiency reasons mentioned above, we propose to restrict our attention to *multi-variate* polynomials over the so called max-plus algebra [BCOQ92]. We anticipate that polynomials over the max-plus algebra have a growth rate that is *linear* in the size of the argument. This growth rate is indeed a very severe restriction if we think in terms of traditional interpretations, *i.e.*, of the –time– taken by the computation to terminate. However, as pointed out above we are interested in *quasi*-interpretations as a mean to bound the –space– needed to compute a function. In particular, following Hofmann's work on type systems ensuring non-size increasing computations [Hof00], one can still accommodate within this framework all functions computable in exponential time whose output's size is bounded by the input's size; this is a respectable class of functions including for instance all decision problems decidable in time $2^{O(n)}$.

We propose to study the synthesis of quasi-interpretations selected in the space of polynomials over the *max-plus* algebra determined by the non-negative rationals extended with $-\infty$ and equipped with binary operations for the maximum and the addition. We prove that in this case the synthesis problem is NP-hard and in NP for the particular case of *multi-linear* quasi-interpretations when the size of the rules is bound by a constant (section 5).

We also relate multi-linear quasi-interpretations to certain syntactic and type-theoretic restrictions that have been proposed in the literature [Jon97, Hof02] in order to limit the computational complexity of the functions represented by a first-order functional language (section 6).

1.5 Related work

The idea of using Pressburger arithmetic over natural numbers occurs in the framework of *sized types* [HPS96, Par00]. In that context functions definitions are explicitly annotated with a function over natural numbers which is essentially a term of Pressburger arithmetic with successor and addition (the *max* operator is not considered). In our terminology, this means that what is addressed is a verification/type-checking problem. By contrast, we are interested in the synthesis/type-inference of a quasi-interpretation and look for (relatively) efficient methods.

In a recent work Hofmann and Jost [HJ03] propose to annotate typing judgments of a first-order functional language in order to infer bounds on the size of the 'heap' memory necessary to the evaluation of expressions. These annotations have the form $x : \tau, f \vdash e : \tau', g$ and the following interpretation: to evaluate e with input value v a heap of size at least $f(|v|)$ (where $|v|$ is the size of the value) is required and if the evaluation terminates with a value v' then a heap of size at least $g(|v'|)$ is available. This work refers to a specific evaluation strategy in which roughly heap memory is required to introduce a new constructor and heap memory is released when performing pattern matching. A method relying on linear programming is proposed to determine for a given program whether an annotation exists over the space of (standard) linear affine functions (no *max*). Quasi-interpretations also offer a bound on the 'heap' memory needed to evaluate a program (cf. theorem 6) but with respect to an evaluation strategy different from the one considered in [HJ03].

2 A first-order functional language

We consider a first-order, simply typed functional language operating over inductively defined data types according to a call-by-value evaluation strategy. A *program* in this context is given by a collection of data types declarations, a collection of mutually recursive function definitions' relying on pattern matching, and a function symbol which is designated as initial. Following Marion *et al.*, an alternative framework for this study could be term-rewriting rules with a distinction among constructors and function symbols.[1]

2.1 Types

The collection of types is the least set such that

$$\mu t.(\mathsf{c}_1 : \tau_{1,1}, \cdots, \tau_{1,n_1} \to t, \ldots, \mathsf{c}_m : \tau_{m,1}, \cdots, \tau_{m,n_m} \to t)$$

is a type provided, $\tau_{i,j}$ is either a type or the bound type variable t, and all constants c_i are distinct and do not occur in previously defined types $\tau_{i,j}$. Traditionally, the symbols $\mathsf{c}_1, \ldots, \mathsf{c}_n$ are the *constructors* of the data type. We assume that for a given program, constructors names are chosen so that no confusion may arise on what type a constructor belongs to. These types allow for the definition of basic data structures. For instance, we can regard $bool \equiv \mu t.(\mathsf{tt} : t, \mathsf{ff} : t)$ as the type of booleans, $tnat \equiv \mu t.(\mathsf{0} : t, \mathsf{s} : t \to t)$ as the type of tally natural numbers, and $tnatlist \equiv \mu t.(\mathsf{nil} : t, \mathsf{cons} : tnat, t \to t)$ as the type of lists of tally natural numbers.

2.2 Expressions

We reserve: $\mathsf{c}, \mathsf{c}', \ldots$ for constructor symbols, $f, f', \ldots$ for function symbols, and $x, x' \ldots$ for first-order variables. Moreover, we introduce the syntactic categories of *values*, *patterns*, and

[1] In this perspective, note that we work with orthogonal TRS and that for these TRS termination is equivalent to innermost termination [Gra96].

expressions as follows:

$$\begin{array}{ll} v ::= \mathsf{c}(v,\ldots,v) & \text{(values)} \\ p ::= x \mid \mathsf{c}(p,\ldots,p) & \text{(patterns)} \\ e ::= x \mid \mathsf{c}(e,\ldots,e) \mid f(e,\ldots,e) & \text{(expressions).} \end{array}$$

We denote with $Var(e)$ the collection of variables occurring in the expression e. Note that values are closed patterns, *i.e.*, $Var(v) = \emptyset$, and patterns are expressions without function symbols.

We denote with $[\tau/t]\tau'$ and $[e/x]e'$ the substitution in types and expressions, respectively. A *signature* Σ attributes to every function symbol f a functional type $\Sigma(f) \equiv \tau_1,\ldots,\tau_n \to \tau$. As usual if u is either a constructor or a function symbol we denote with $arity(u)$ the number of expected arguments as specified by its type. A *context* Γ is a finite list $x_1 : \tau_1,\ldots,x_n : \tau$ where $x_i \neq x_j$ if $i \neq j$. We use the judgment $\Gamma \vdash_\Sigma e : \tau$ to state that the expression e has type τ with respect to the signature Σ and the context Γ. Provable typing judgment are defined by the following inference system:

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash_\Sigma x : \tau} \qquad \frac{\tau \equiv \mu t.(\ldots,\mathsf{c} : \tau_1,\ldots,\tau_n \to t,\ldots) \quad \Gamma \vdash_\Sigma e_i : [\tau/t]\tau_i \quad i = 1,\ldots,n}{\Gamma \vdash_\Sigma \mathsf{c}(e_1,\ldots,e_n) : \tau}$$

$$\frac{\Sigma(f) = \tau_1,\ldots,\tau_n \to \tau \quad \Gamma \vdash_\Sigma e_i : \tau_i \quad i = 1,\ldots,n}{\Gamma \vdash_\Sigma f(e_1,\ldots,e_n) : \tau} \,.$$

2.3 Functions' definitions

Function symbols are defined by a finite system of mutually recursive equations so that each function symbol is defined by exactly one equation. If $\Sigma(f) = \tau_1,\ldots,\tau_n \to \tau$ then the equation defining f has the shape:

$$\begin{array}{ll} f(x_1,\ldots,x_n) = & \\ x_1 = p_{1,1},\ldots,x_n = p_{1,n} & \Rightarrow e_1 \\ \cdots & \\ x_1 = p_{m,1},\ldots,x_n = p_{m,n} & \Rightarrow e_m \end{array}$$

where the formal parameters $x_1,\ldots,x_n$ are distinct and

(1) Patterns are *linear*, *i.e.* in $p_{i,j}$ no variable occurs more than once and $Var(p_{i,j}) \cap Var(p_{i,j'}) = \emptyset$ if $j \neq j'$. We assume that if $Var(p_{i,j}) = \emptyset$ then $p_{i,j}$ is a constant constructor.[2] In the examples, we take the freedom of omitting trivial patterns of the shape $x_i = x_i$.

(2) Patterns do not superpose, *i.e.*, if $i \neq j$ then the set of equations $\{p_{i,1} = p_{j,1},\ldots,p_{i,n} = p_{j,n}\}$ is not unifiable. In particular, this entails that the programs we consider are *deterministic.*

(3) Expressions' variables are contained in patterns' variables, *i.e.*, $Var(e_k) \subseteq \bigcup_{j=1,\ldots,n} p_{k,j}$.

(4) Patterns and expressions are well typed, *i.e.*, for $i = 1,\ldots,m$ there are contexts Γ_i such that:

$$\Gamma_i \vdash_\Sigma p_{i,j} : \tau_j \text{ for } j = 1,\ldots,n \text{ and } \Gamma_i \vdash_\Sigma e_i : \tau \,.$$

We call *rule* a clause of the shape $x_1 = p_1,\ldots,x_n = p_n \Rightarrow e$.

[2] This is a technical condition that does not impair the expressivity of the language as general patterns without free variables can be simulated by introducing auxiliary function symbols.

2.4 Evaluation

A program is a finite collection of inductive types and a finite system of functions' definitions with a selected *main* function. Expression evaluation follows a call-by-value strategy which is specified as follows:

$$(cst)\ \frac{e_j \mapsto v_j \quad j = 1, \ldots, n}{\mathsf{c}(e_1, \ldots, e_n) \mapsto \mathsf{c}(v_1 \ldots, v_n)} \qquad (fun)\ \frac{e'_j \mapsto v_j, \quad \sigma p_{i,j} = v_j, j = 1, \ldots, n \quad \sigma(e_i) \mapsto v}{f(e'_1, \ldots, e'_n) \mapsto v}$$

assuming the function f is defined as in section 2.3 and σ denotes a pattern-matching substitution.

3 Quasi-interpretations

In this section we introduce the notion of quasi-interpretation and its basic properties.

Definition 1 (size) *The size of a value v is defined by* [3]

$$|c(v_1, \ldots, v_n)| = \begin{cases} 0 & \text{if } n = 0 \\ 1 + \Sigma_{i=1,\ldots,n}|v_i| & \text{if } n > 0 \ . \end{cases}$$

Definition 2 (assignment) *Given a program, an assignment associates:*

(1) *To every constructor* c *with* k *arguments a function* $q_{\mathsf{c}} : (\mathbf{Q}^+)^k \to \mathbf{Q}^+$ *such that:*

(1.1) $q_{\mathsf{c}} = 0$ *if* c *has arity* 0 *and*

(1.2) $q_{\mathsf{c}} = d + \Sigma_{i=1,\ldots,n} x_i$ *for some* $d \geq 1$*, otherwise.*

(2) *To every function symbol* f *with* k *arguments a function* $q_f : (\mathbf{Q}^+)^k \to \mathbf{Q}^+$ *such that:*

(2.1) $q_f(n_1, \ldots, n_k) \geq n_i$ *for* $i = 1, \ldots, k$ *and*

(2.2) $q_f(n_1, \ldots, n_k) \geq q_f(m_1, \ldots, m_k)$ *if* $n_i \geq m_i$ *for* $i = 1, \ldots, k$.

Definition 3 (extension of the assignment) *Given an assignment and an expression e with $Var(e) = \{x_1, \ldots, x_k\}$ we can define a function $q_e : (\mathbf{Q}^+)^k \to \mathbf{Q}^+$ by induction on e as follows:*

$$q_x = x, \quad q_{\mathsf{c}(e_1,\ldots,e_n)} = q_{\mathsf{c}}(q_{e_1}, \ldots, q_{e_n}), \quad q_{f(e_1,\ldots,e_n)} = q_f(q_{e_1}, \ldots, q_{e_n}) \ .$$

Definition 4 (quasi-interpretation) *Given a program, the related assignment q is a quasi interpretation if for every function definition of the shape presented in section 2.3 the following condition holds for $i = 1, \ldots, m$ (where functions are ordered pointwise):*

$$q_f(q_{p_{i,1}}, \ldots, q_{p_{i,n}}) \geq q_{e_i} \ . \tag{1}$$

The notion of assignment we consider is obviously inspired by the *simplification interpretation method* used in termination proofs of TRS. The specific conditions on constructors correspond to the notion of *kind* 0 quasi-interpretation presented in [BMM01]. However, we work over the non-negative rationals rather than over the natural numbers and we force the

[3] The alternative definition of size where we assign a positive size to constants is equivalent to the present one within a constant multiplicative factor.

interpretation 0 for constants. This last condition allows to simplify some interpretations by neglecting the space needed to store constant values. It is also instrumental to the simple form of the satisfaction problem we will derive (cf. remark 12). The following proposition summarizes the basic properties of quasi-interpretations (the standard proof is delayed to appendix B.1).

Proposition 5 *Suppose q is a quasi-interpretation for a given program. Then:*

(1) *There is a constant d such that for any value v, $|v| \leq q_v \leq d|v|$.*

(2) *If $e \mapsto v$ then $q_e \geq q_v \geq |v|$. In particular, if $f(v_1, \ldots, v_n) \mapsto v$ then $|v| \leq q_f(d|v_1|, \ldots, d|v_n|)$.*

We remark that quasi-interpretations –by themselves– already provide a bound on the complexity of the program.

Theorem 6 *Suppose q is a quasi-interpretation for a given program. Then there is an evaluation strategy that given a function symbol f with arguments $v_1, \ldots, v_n$ returns the value v iff $f(v_1, \ldots, v_n) \mapsto v$ and a special symbol $\perp$ otherwise. The procedure runs in time $2^{O(q_{f(v_1,\ldots,v_n)})}$.*

The proof (appendix B.2) proceeds by presenting an evaluator that can be run on a *bounded auxiliary push-down machine* (APDA) (the bound depending on the quasi-interpretation). By a well-known result of S. Cook [Coo71], a bounded APDA can be simulated by a Turing Machine in exponential time using a 'table' to store intermediate results.

4 Max-plus polynomials and synthesis problem

We consider the set $\mathbf{Q}^+ \cup \{-\infty\}$ equipped with two internal composition laws *max* and *plus* (denoted $+$) where it is understood that:

$$max(-\infty, x) = max(x, -\infty) = x \quad -\infty + x = x + (-\infty) = -\infty \ .$$

We briefly refer to this structure as $\mathbf{Q}^+_{max}$. We note that $\mathbf{Q}^+_{max}$ is a commutative and idempotent monoid for *max* with neutral element $-\infty$ and a commutative monoid for *plus* with neutral element 0. Moreover, *plus* distributes over *max*: $x + max(y, z) = max(x + y, x + z)$. In the max-plus literature one regards *max* as an addition and *plus* as a multiplication and therefore the following notation is adopted: $x \oplus y = max(x, y)$, $x \otimes y = x + y$. Exponentiation x^α with $\alpha \geq 0$ stands for $x \otimes \cdots \otimes x$ α times and thus corresponds to the product αx in the usual mathematical notation. Note in particular that $x^0 = 0$. In the following we will just use quite elementary properties of max-plus algebras and so we find it more convenient to stick to the usual mathematical notation using $max(x, y)$, $x + y$, and αx for 'addition', 'multiplication', and 'exponentiation' in the max-plus algebra.

A *monomial* with coefficient in $a \in \mathbf{Q}^+_{max}$ and indeterminates $x_1, \ldots, x_n$ can be written as

$$\alpha_1 x_1 + \cdots + \alpha_n x_n + a \tag{2}$$

where $\alpha_i \in \mathbf{N}$. We say that a monomial has *degree* d if $\alpha_i \leq d$ for $i = 1, \ldots, n$.[4] A *polynomial* is now written as $max_{i \in I} m_i$ where m_i are monomials of the type specified above. We say that

[4]This is a slightly improper terminology; we should say that the monomial restricted to any of its indeterminates has degree at most d.

a polynomial has degree d if all monomials m_i have degree d. A polynomial of degree d with n indeterminates can be represented as

$$max_{I:\{1,\ldots,n\}\to\{0,\ldots,d\}}(I(1)x_1 + \cdots + I(n)x_n + a_I) \quad (3)$$

and it is therefore specified by the $(d+1)^n$ coefficients $\{a_I \mid I : \{1,\ldots,n\} \to \{0,\ldots,d\}\}$.

An *assignment* (cf. definition 2) of max-plus polynomials of degree d is determined as follows:

(1) For every constructor c with positive arity a coefficient a^{c} subject to the constraint $a^{\mathsf{c}} \geq 1$.

(2) For every function symbol f with $n \geq 1$ arguments a set of coefficients $\{a^f_I \mid I : \{1,\ldots,n\} \to \{0,\ldots,k\}\}$ subject to the constraints for $i = 1,\ldots,n$

$$max\{a^f_I \mid I(i) \geq 1\} \geq 0 \quad (4)$$

This last constraint is necessary and sufficient for the condition (2.1) $q_f(n_1,\ldots,n_k) \geq n_i$ of definition 2 to hold. We note that the following monotonicity condition (2.2) is always satisfied.

Definition 7 (synthesis problem) *Given a program, the synthesis problem amounts to determine whether there is a polynomial max-plus quasi-interpretation.*

If the program includes l constructors of positive arity and m functional symbols of arity at most n an assignment of polynomials of degree at most d is determined by at most $l+m(d+1)^n$ coefficients. The assignment is a quasi-interpretation iff it satisfies the constraints above and those induced by the condition (1) $q_f(q_{p_{i,1}},\ldots,q_{p_{i,n}}) \geq q_{e_i}$.

Some simple instances of max-plus polynomial quasi-interpretations are given in appendix A. Of course, the rule of the game is to get quasi-interpretations as small as possible and in this respect the *max* operator is quite useful. Moreover, in many examples where a variable occurs several times on the right-hand side, it is simply not possible to find a (max-plus) quasi-interpretation that does not rely on *max*. We note that in general the existence of a polynomial max-plus interpretation of a *given degree* can be reduced to the validity of an $\exists\forall$ formula in Pressburger arithmetic over $\mathbf{Q}^+_{max}$. This problem can be attacked with tools such as the OMEGA test [Pug92].

We can also consider a restricted problem where one looks for polynomials over $\mathbf{N}_{max} = \mathbf{N} \cup \{-\infty\}$ (with coefficients in $\mathbf{N}_{max}$). In this case, the satisfaction of a $\exists\forall$ formulae can also be attacked with automata theoretic tools (see, *e.g.*, [WB00]). Interestingly, Pressburger arithmetic over $\mathbf{N}_{max}$ still has the nice properties of usual Pressburger arithmetic. In particular, the property that definable sets are semi-linear (or equivalently rational) is preserved [GR02].

As a lower bound on the complexity of the synthesis problem, we state the following theorem which is based on a non-trivial reduction from 3-SAT (proof in appendix B.3).

Theorem 8 *The synthesis problem is* NP*-hard and it remains so if any combination of the following restrictions is considered:*

(1) *Rules of bounded size (for a small bound).*

(2) *Max-plus polynomials of bounded degree* $d \geq 1$.

(3) *Uniform choice of the coefficients of the constructors:* $a^{\mathsf{c}} = a^{\mathsf{c}'}$ *for all constructors* c,c' *of positive arity.*

5 Synthesis of multi-linear quasi-interpretations

We consider the synthesis problem when the degree is 1. We start by pointing out some specific properties of this case (section 5.1), then we give effective methods to compute and compare quasi-interpretations (section 5.2), and finally we show how the generated conditions can be reduced to linear programming (section 5.3).

5.1 Multi-linear assignments

Following a rather standard terminology, we will refer to monomials (polynomials) of degree 1 as *multi-linear* monomials (polynomials). We note that a multi-linear polynomial in n indeterminates is specified by 2^n coefficients $\{a_I \mid I \subseteq \{1, \ldots, n\}\}$ and can be written as follows:

$$max_{I \subseteq \{1,\ldots,n\}}(\Sigma_{i \in I} x_i + a_I) \ . \tag{5}$$

Equivalently, if the multi-linear polynomial depends on the variables $x_1, \ldots, x_n$ we will also write:

$$max_{V \subseteq \{x_1,\ldots,x_n\}}(\Sigma_{v \in V} v + a_V) \ . \tag{6}$$

Proposition 9 (normal form) *For every multi-linear polynomial $P(x_1, \ldots, x_n)$ there is an equivalent multi-linear polynomial $P'(x_1, \ldots, x_n)$ with coefficients $\{a'_I \mid I \subseteq \{1, \ldots, n\}\}$ satisfying the condition*

$$J \subseteq K \subseteq \{1, \ldots, n\} \quad \Rightarrow a'_J \geq a'_K \ . \tag{7}$$

Proof. We define $a'_I = max\{a_J \mid I \subseteq J\}$. Clearly $P \leq P'$ and P' satisfies the condition (7). It remains to prove $P \geq P'$. It is enough to show that for $K \subseteq \{1, \ldots, n\}$, $P(x_1, \ldots, x_n) \geq \Sigma_{i \in K} x_i + a'_K$. But $a'_K = a_{J_o}$ for some $J_o \supseteq K$ and $P(x_1, \ldots, x_n) \geq \Sigma_{i \in J_o} x_i + a'_{J_o} \geq \Sigma_{i \in K} x_i + a'_K$. $\Box$

In the following we assume that a program has been fixed and that $\mathsf{c}_1, \ldots, \mathsf{c}_l$ are the constructors of positive arity occurring in the program. We say that a multi-linear polynomial is in *normal form* if its coefficients satisfy condition (7). For such polynomials condition (4) on assignments can be reformulated as $a^f_{\{i\}} \geq 0$ for every function f with $arity(f) = n$. Given a multi-linear assignment we will show that $q_{f(p_1,\ldots,p_n)}$ is always a multi-linear polynomial; a property that may fail for a general expression e.

Proposition 10 *Let P_1 be a multi-linear polynomial and P_2 be a polynomial over $x_1, \ldots, x_n$. If $P_1 \geq P_2$ then P_2 must be multi-linear.*

Proof. If P_2 is not multi-linear then there is an argument x_i such that on entry $X_i \equiv (0, \ldots, 0, x_i, 0 \ldots, 0)$, $P_2(X_i) \geq 2x_i$. On the other hand $P_1(X_i) = x_i + n$ for some $n \in \mathbf{Q}^+_{max}$. Clearly $P_1(X_i) \geq P_2(X_i)$ fails for sufficiently large x_i. $\Box$

5.2 Computing multi-linear quasi-interpretations

We explicitly compute the shape of the different polynomials arising from a multi-linear assignment. The proofs require some involved notation but just rely on elementary arithmetic considerations and are delayed to appendix B.

Proposition 11 (left-hand-side) (1) *Suppose q is a multi-linear assignment and p is a pattern in a function definition then q_p is a multi-linear polynomial of the shape*

$$q_p = \Sigma_{v \in Var(p)} v + \Sigma_{j=1,\ldots,l} \alpha_j a^{\mathsf{c}_j} \tag{8}$$

for some $\alpha_j \in \mathbf{N}$.
(2) *Suppose q is a multi-linear assignment and the function f contains the rule $x_1 = p_1, \ldots, x_n = p_n \Rightarrow e$. Then $q_{f(p_1,\ldots,p_n)}$ is always a multi-linear function and assuming $q_{p_i} = \Sigma_{v \in Var(p_i)} v + \Sigma_{j=1,\ldots,l} \alpha_{i,j} a^{\mathsf{c}_j}$ then the coefficient b_V for $V \subseteq \bigcup_{i=1,\ldots,n} Var(p_i)$ is given by*

$$b_V = a^f_{K_V} + \Sigma_{j=1,\ldots,l} (\Sigma_{k \in K_V} \alpha_{k,j}) a^{\mathsf{c}_j} \tag{9}$$

where $K_V = \{k \in \{1,\ldots,n\} \mid V \cap Var(p_k) \neq \emptyset\}$.

Remark 12 *It is interesting that the coefficients b_V can be expressed without the max operation. Let us see why a weakening of the rules on the patterns compromises this property. Take f with $arity(f) = 2$ and consider the pattern $x_1 = x_1, x_2 = v$ for some value v such that $q_v > 0$. Then the coefficients of the multi-linear polynomial $q_f(x_1, v)$ are*

$$b_\emptyset = max(a^f_\emptyset, a^f_{\{2\}} + q_v) \qquad b_{\{1\}} = max(a^f_{\{1\}}, a^f_{\{1,2\}} + q_v)$$

where by the constraints on the assignments we may assume that $a^f_\emptyset \geq a^f_{\{1\}}, a^f_{\{2\}} \geq a^f_{\{1,2\}}$ and $a^f_{\{1\}}, a^f_{\{2\}} \geq 0$. However, these constraints are not sufficiently strong to get rid of the max operation.

We now turn to the polynomial q_e. This polynomial is obtained by arbitrary composition of multi-linear polynomials and may fail to be multi-linear. However, in this case we know from propositions 10 and 11(2) that the inequality $q_{f(p_1,\ldots,p_n)} \geq q_e$ cannot hold. So our next task is to generate constraints that are necessary and sufficient to guarantee that q_e is multi-linear. To this end, we introduce in figure 1 a little formal system with judgments of the shape (e, C) where e is an expression and C is a set of constraints on the coefficients of the functions occurring in e. As usual we introduce a special constraint $\perp$ with the hypothesis that no assignment can satisfy it.

Example 13 *For the expression $e \equiv f(\mathsf{c}(x, y), g(x))$ we obtain $\vdash (e, \{a^f_{1,2} = -\infty\})$. On the other hand, for the expression $e \equiv \mathsf{c}(x, x)$ we obtain $\vdash (e, \{\perp\})$.*

Proposition 14 (right-hand-side) *Suppose q is a multi-linear assignment in normal form.*
(1) *If $q_{e_i} = max_{U_i \subseteq V_i} (\Sigma_{v \in U_i} v + a^i_{U_i})$, $i = 1, \ldots, n$, are multi-linear polynomials where $V_i = Var(e_i)$ and $V = \bigcup_{i=1,\ldots,n} V_i$. Then:*
(1.1) *$q_{\mathsf{c}(e_1,\ldots,e_n)}$ is a multi-linear polynomial iff $i \neq j$ implies $V_i \cap V_j = \emptyset$, and in this case the coefficients b_U for $U \subseteq V$ are determined by:*

$$b_U = \Sigma_{i=1,\ldots,n} a^i_{U \cap V_i} + a^{\mathsf{c}} \tag{10}$$

(1.2) *Whenever $q_{f(e_1,\ldots,e_n)}$ is a multi-linear polynomial the coefficients b_U for $U \subseteq V$ are determined by:*

$$b_U = max_{I \subseteq \{1,\ldots,n\}, \downarrow I, U \subseteq \bigcup_{i \in I} V_i} (\Sigma_{i \in I} a^i_{U \cap V_i} + a^f_I) \tag{11}$$

where by definition $\downarrow I$ if $i, j \in I$ and $i \neq j$ implies $V_i \cap V_j = \emptyset$.
(2) *If $\vdash (e, C)$. Then q_e is multi-linear iff q satisfies C.*

$$\frac{}{(x,\emptyset)}$$

$$\frac{(e_i,C_i),\ i=1,\ldots,n \quad Var(e_i)\cap Var(e_j)=\emptyset \quad \text{for all } i\neq j}{(\mathsf{c}(e_1,\ldots,e_n),\bigcup_{i=1,\ldots,n} C_i)}$$

$$\frac{(e_i,C_i),\ i=1,\ldots,n \quad Var(e_i)\cap Var(e_j)\neq\emptyset \quad \text{for some } i\neq j}{(\mathsf{c}(e_1,\ldots,e_n),\bigcup_{i=1,\ldots,n} C_i\cup\{\bot\})}$$

$$\frac{(e_i,C_i),\ i=1,\ldots,n}{(f(e_1,\ldots,e_n),\{a^f_{i,j}=-\infty \mid i\neq j,\, Var(e_i)\cap Var(e_j)\neq\emptyset\}\cup\bigcup_{i=1,\ldots,n} C_i)}$$

Figure 1: Constraints enforcing multi-linearity of q_e

Thus given a rule $x_1 = p_1,\ldots,x_n = p_n \Rightarrow e$ and a generic multi-linear assignment q we determine the conditions under which q_e is multi-linear and then formally compute its coefficients. Next, we have to find necessary and sufficient conditions on the coefficients to compare multi-linear polynomials.

Example 15 *Consider again the expression* $e \equiv f(\mathsf{c}(x,y),g(x))$. *First of all we note the following constraints on the coefficients:*

$$\begin{array}{ll} a^f_\emptyset \geq a^f_{\{1\}}, a^f_{\{2\}} \geq 0 & a^f_{\{1,2\}} = -\infty \\ a^g_\emptyset \geq a^g_{\{1\}} \geq 0 & a^{\mathsf{c}} \geq 1 \end{array}$$

Then we compute q_e *as follows:*

$$\begin{array}{ll} q_{\mathsf{c}(x,y)} & = a^{\mathsf{c}} + x + y \\ q_{g(x)} & = max(a^g_\emptyset, a^g_{\{1\}} + x) \\ q_{f(x_1,x_2)} & = max(a^f_\emptyset, a^f_{\{1\}} + x_1, a^f_{\{2\}} + x_2) \\ q_e & = max(a^f_\emptyset, a^f_{\{1\}} + a^{\mathsf{c}} + x + y, a^f_{\{2\}} + max(a^g_\emptyset, a^g_{\{1\}} + x)) \\ & = max(a^f_\emptyset, a^g_\emptyset + a^f_{\{2\}}, a^g_{\{1\}} + a^f_{\{2\}} + x, a^f_{\{1\}} + a^{\mathsf{c}} + x + y)\ . \end{array}$$

Proposition 16 (comparison) *Suppose* P_1 *and* P_2 *are multi-linear polynomials with* n *indeterminates and coefficients* $\{a_I \mid I \subseteq \{1,\ldots,n\}\}$ *and* $\{b_I \mid I \subseteq \{1,\ldots,n\}\}$, *respectively. Then*

(1) $P_1 \geq P_2$ *iff the following condition holds:*

$$max\{a_K \mid K \supseteq J\} \geq b_J \quad \text{for all } J \subseteq \{1,\ldots,n\}\ . \tag{12}$$

(2) *If moreover,* P_1 *is in normal form then the condition (12) is equivalent to* $a_J \geq b_J$ *for all* $J \subseteq \{1,\ldots,n\}$.

Remark 17 *For max-plus polynomials of degree higher than* 1 *this simple comparison criteria fails. For instance,* $max(2x,2y) \geq x+y$.

To summarize, we have shown how to generate a system $\mathcal{S}$ of inequality constraints on the coefficients of multi-linear polynomials so that the constraints can be satisfied in $\mathbf{Q}^+_{max}$ iff the corresponding polynomials determine a multi-linear assignment and a quasi-interpretation.

5.3 Reduction to linear programming

For programs with rules of *bounded* size we show that the synthesis problem can be solved in non-deterministic polynomial time thus matching the lower bound given by theorem 8.

The comparison criteria (12) introduces inequalities of the shape $max(A_1, \ldots, A_m) \geq B$, where A_i are coefficients of the type specified by proposition 11(2) *not* containing the *max* operation. We remove the *max* by non-deterministically *guessing* the maximum A_i among $A_1, \ldots, A_m$ and transforming the inequality into $A_i \geq B$. We show next that the resulting system can be solved in deterministic polynomial time. Thus the quest of synthesis problems with (deterministic) *polynomial* time complexity seem to depend crucially on the possibility of removing the *max* operation on the left-hand side of the inequalities generated by the comparison criteria. Some interesting cases where this is actually possible are discussed in propositions 23 and 25.

We reserve $x_1, \ldots, x_n$ for the variables corresponding to the coefficients a^f_I or for auxiliary variables, and $y_1, \ldots, y_l$ for the variables corresponding to the coefficients a^{c_j} which are all subject to the constraint $y \geq 1$. Let $\mathcal{S}(\vec{x}, \vec{y})$ be the system of inequalities over $\mathbf{Q}^+_{max}$ that we have derived for the synthesis problem over multi-linear polynomials after elimination of the *max* on the left-hand side.

Proposition 18 (right *max*-elimination) *The system $\mathcal{S}(\vec{x}, \vec{y})$ can be transformed in polynomial time into a system $\mathcal{S}_1(\vec{x}, \vec{x'}, \vec{y})$ over $\mathbf{Q}^+_{max}$ with additional auxiliary variables $\vec{x'}$ such that:*

(1) *The inequalities in $\mathcal{S}_1$ have one of the following 3 shapes assuming $\vec{x}, \vec{x'} \equiv x_1, \ldots, x_n$ and $\vec{y} \equiv y_1, \ldots, y_l$.*

$$(a) \quad x = -\infty \quad \textit{provided } x \in \{\vec{x}\} \qquad (b) \quad y \geq 1 \quad \textit{for all } y \in \{\vec{y}\}$$
$$(c) \quad x + \Sigma_{j=1,\ldots,l}\alpha_j y_j \geq \Sigma_{j=1,\ldots,n}\beta_j x_j + \Sigma_{j=1,\ldots,l}\gamma_j y_j \quad \textit{where: } \alpha_j, \beta_j, \gamma_j \in \mathbf{N}, x \in \{\vec{x}, \vec{x'}\} \ .$$

(2) *An assignment ρ satisfies $\mathcal{S}$ iff for some $\vec{w}$, $\rho[\vec{w}/\vec{x'}]$ satisfies $\mathcal{S}_1$.*

Proof hint. An inequality $A \geq max_{i\in I}(\Sigma_{j\in J_i} B_{i,j} + C)$ can be transformed into

$$A \geq x' \qquad x' \geq \Sigma_{j\in J_i} x'_{i,j} + C \text{ for } i \in I \qquad x'_{i,j} \geq B_{i,j} \text{ for } i \in I, j \in J_i$$

where $x', x'_{i,j}$ are fresh variables. It can be easily verified that the derived system is satisfiable iff the initial one is. If $B_{i,j}$ contains again the *max* operator then we apply recursively the transformation to the inequality $x'_{i,j} \geq B_{i,j}$. $\square$

Proposition 19 ($-\infty$-elimination) *The system $\mathcal{S}_1(\vec{x}, \vec{x'}, \vec{y})$ over $\mathbf{Q}^+_{max}$ obtained in proposition 18 can be transformed in polynomial time into a system $\mathcal{S}_2(\vec{x''}, \vec{y})$ over $\mathbf{Q}^+$ where (i) $\{\vec{x''}\} \subset \{\vec{x}, \vec{x'}\}$, (ii) the constraints have the shape (b) and (c) in proposition 18, and assuming $\{\vec{z}\} = \{\vec{x}, \vec{x'}\} \backslash \{\vec{x''}\}$ an assignment ρ satisfies $\mathcal{S}_1$ iff $\rho[\vec{-\infty}/\vec{z}]$ satisfies $\mathcal{S}_2$.*

Proof hint. We describe the proof strategy in a simplified case. Consider the conjunction of boolean formulae of the shape $\bigvee_{j\in J} x_j$ or $x \Rightarrow \bigvee_{j\in J} x_j$. Its satisfiability can be decided by applying the following rules:

$$\begin{array}{ll}
S, x, (x \Rightarrow \bigvee_{j\in J} x_j) & \to S, x, \bigvee_{j\in J} x_j \\
S, x, x \vee \bigvee_{j\in J} x_j & \to S, x \quad \text{if } J \neq \emptyset \\
S, (x \Rightarrow \bot), x \vee \bigvee_{j\in J} x_j & \to S, (x \Rightarrow \bot), \bigvee_{j\in J} x_j \quad \text{if } J \neq \emptyset \\
S, x', (x \Rightarrow (x' \vee \bigvee_{j\in J} x_j)) & \to S, x' \\
S, (x' \Rightarrow \bot), (x \Rightarrow (x' \vee \bigvee_{j\in J} x_j)) & \to S, (x' \Rightarrow \bot), (x \Rightarrow \bigvee_{j\in J} x_j)
\end{array}$$

where as usual $\perp$ stands for the empty disjunction, disjunction is treated as an associative and commutative operator, and ',' stands for conjunction. These simplification rules obviously terminate in a system S' that is satisfiable iff the original one is. Moreover, if $\perp \notin S'$ then the boolean variables X can be *partitioned* in three sets X_1, X_0, X_2 where $X_1 = \{x \mid s \in S'\}$ and $X_0 = \{x \mid (x \Rightarrow \perp) \in S'\}$. Then a satisfying assignment is obtained by taking $\rho(x) = 1$ if $x \in X_1 \cup X_2$ and $\rho(x) = 0$ if $x \in X_0$. This proof strategy is repeated for the system over $\mathbf{Q}^+_{max}$ where the constraint $x = -\infty$ corresponds to x and the constraint $x \geq 0$ to $(x \Rightarrow \perp)$. A proper generalization of the rules above is presented in appendix B.7. □

Remark 20 (optimality and integer solutions) (1) *Once the problem is reduced to linear programming we may look for a solution which is* optimal *with respect to a given linear cost function. For instance, we may minimize the function* $\Sigma_{x\in\{\vec{x''}\}} x + \Sigma_{j=1,\ldots,l} y_j$.

(2) *The transformations we have presented apply equally well to multi-linear polynomials over* $\mathbf{N}_{max}$. *It is interesting to note that at the final step we can still rely on linear programming. Indeed, if the system of inequalities over* $\mathbf{Q}^+$ *admits a solution* $s = (n_1/d_1, \ldots, n_k/d_k)$ *then multiplying* s *by the least common denominator we obtain a solution in* $\mathbf{N}$ *because of the particular shape (b) and (c) of the constraints generated by* $-\infty$*-elimination. As usual, the rational solutions may provide a better upper bound than the integer ones.*

We summarize our analysis for programs whose rules have *bounded* size. The proof given in section B.8 also shows that the complexity of the method is exponential in the size of the rule. This is not surprising since the number of coefficients we have to determine is exponential in the number of variables in a rule.

Theorem 21 *The synthesis problem over multi-linear polynomials for programs with rules of bounded size is* NP*-complete.*

6 Expressivity

We present within our framework a 'no cons' syntactic restriction and a 'type system for in-place update' that have been proposed in the literature to control the time and space complexity. We also mention some upper and lower bounds on the complexity of functions representable by programs admitting a max-plus quasi-interpretation.

6.1 No cons syntactic condition

Jones' syntactic condition [Jon97] concerns first-order functional programs defined over the type of booleans $bool \equiv \mu t.(\mathsf{tt} : t, \mathsf{ff} : t)$ and the type of lists of booleans $blist \equiv \mu t.(\mathsf{nil} : t, \mathsf{cons} : bool, t \to t)$. The syntactic restriction requires that in a function definition of the shape presented in section 2.3 the cons constructor does not appear in the expressions e_i on the right-hand side of pattern matching. The following can be easily checked.

Proposition 22 *A program conforming to Jones' restriction admits the following multi-linear quasi-interpretation assuming* $arity(\mathsf{c}) = arity(f) = n \geq 1$*:*

$$q_{\mathsf{c}} = 1 + \Sigma_{i=1,\ldots,n} x_i \qquad q_f = max(x_1, \ldots, x_n)\ .$$

PROOF. We have $q_{f(p_1,\ldots,p_n)} = max_{i=1,\ldots,n}(\Sigma_{v\in Var(p_i)}v + d_i)$ for some $d_i \geq 0$. On the other hand, if no cons can occur in the expression e then $q_e = max\{v \mid v \in Var(e)\}$ and by definition of rule, $Var(e) \subseteq \bigcup_{i=1,\ldots,n} Var(p_i)$. $\Box$

We consider a restricted class of multi-linear quasi-interpretations where:

$$q_{\mathsf{c}} = a + \Sigma_{i=1,\ldots,n}x_i, \quad a \geq 1 \qquad q_f = max(x_1 + a^f, \ldots, x_n + a^f), \quad a^f \geq 0, \tag{13}$$

for $arity(f) = arity(\mathsf{c}) = n \geq 1$. We note that (i) all constructors have the *same* coefficient a, (ii) every function is determined by exactly *one* coefficient a^f, and (iii) the interpretation in proposition 22 falls in this family. We refer to this class of quasi-interpretations as *max-multi-linear*.

Proposition 23 *The synthesis problem over max-multi-linear interpretations can be solved in polynomial time.*

PROOF. It is enough to note that under the conditions (13) the *max* operation is not needed on the left-hand side of an inequality. First we note that for a pattern p_i, $q_{p_i} = \alpha_i a + \Sigma_{v\in Var(p_i)}v$ for some $\alpha_i \in \mathbf{N}$. Thus

$$q_{f(p_1,\ldots,p_n)} = max_{i=1,\ldots,n}(a^f + \alpha_i a + \Sigma_{v\in Var(p_i)}v) \ .$$

Now let $V \subseteq \bigcup_{i=1,\ldots,n} Var(p_i)$.

- If $V = \emptyset$ then the comparison condition (12) on the coefficients is expressed as:

$$max_{i=1,\ldots,n}(a^f + \alpha_i a) = a^f + a(max_{i=1,\ldots,n}(\alpha_i)) \geq b_\emptyset$$

noting that α_i are natural numbers and their maximum can be easily determined.

- If $\emptyset \neq V \subseteq Var(p_i)$ then by the linearity of the patterns i is unique and the comparison condition (12) on the coefficients is expressed as: $a^f + \alpha_i a \geq b_V$.
- Finally, if $V \not\subseteq Var(p_i)$ for all i then it must be that $-\infty = b_V$. $\Box$

6.2 Type system for in-place update

Hofmann [Hof00] proposes a first-order functional language that can be compiled into code not requiring dynamic heap memory allocation. This is achieved by means of an –empty– 'resource type' $\diamond$ and 'affine' typing rules. Elements of resource type have to be understood as memory cells. Constructors of inductive types require an argument of resource type. Also functions may take as arguments elements of resource type. We look at a little fragment of this type system[5] composed of programs over the types:

$$\begin{array}{ll} \diamond \equiv \mu t.() & \text{(resource type)} \\ W \equiv \mu t.(\epsilon : t, \mathsf{0} : \diamond, t \to t, \mathsf{1} : \diamond, t \to t) & \text{(binary words).} \end{array}$$

For every function f we assume $\Sigma(f)$ has the shape $(\diamond, \ldots, \diamond, W, \ldots, W) \to W$ and let $r(f) < arity(f)$ be the number of arguments of resource type. As usual patterns and expressions in

[5] In particular, we neglect enumerated, product, and higher-order types.

functions' definitions have to be well typed (cf. section 2.3). This means that assuming $\Sigma(f) = (\tau_1, \ldots, \tau_n) \to W$ for every rule in the definition of f there is a context Γ such that:

$$\Gamma \vdash_\Sigma p_{i,j} : \tau_j \text{ for } j = 1, \ldots, n \qquad \Gamma \vdash_\Sigma e_i : W \ . \tag{14}$$

Without loss of generality, we may assume that Γ contains only the variables occurring in the patterns $p_{i,j}$. Now we say that the typing is *affine* if in the typing of e_i the hypotheses in the context Γ are used at most once. Note that the typing of the patterns is always affine since we deal with linear patterns.

Resource arguments can be regarded as annotations for the compiler but no real computation is performed on them. Indeed, it is not even possible to create (closed) values of resource type. However, there is an obvious way to *erase* resource arguments and obtain the 'intended' program. In our simple case, the resulting program will operate over the type $w \equiv \mu t.(\epsilon : t, \mathsf{0} : t \to t, \mathsf{1} : t \to t)$. The erasure function er is defined as follows over expressions:

$$\begin{gathered} er(x) = x \quad er(\epsilon) = \epsilon \quad er(\mathsf{0}(x,e)) = \mathsf{0}(er(e)) \quad er(\mathsf{1}(x,e)) = \mathsf{1}(er(e)) \\ er(f(e_1, \ldots, e_n)) = f(er(e_{r(f)+1}), \ldots, er(e_n)) \ . \end{gathered}$$

Proposition 24 *If a program has an affine typing then its erasure admits the following multi-linear quasi-interpretation:*

$$q_\mathsf{0} = q_\mathsf{1} = x + 1, \quad q_f = \Sigma_{i=1,\ldots,n} x_i + r(f) \ .$$

Proof hint. We define a function R on expressions that counts the number of arguments of resource type:

$$\begin{gathered} R(x) = R(\epsilon) = 0, \qquad R(\mathsf{0}(x,e)) = R(\mathsf{1}(x,e)) = 1 + R(e), \\ R(f(e_1, \ldots, e_n)) = r(f) + \Sigma_{i=1,\ldots,n} R(e_i) \ . \end{gathered}$$

The only expressions of resource type that can occur in an expression e on the right hand side of a rule are the variables of resource type that we find in the pattern. These are the formal parameters of resource type of the function, say f, plus the variables of resource type arising in the patterns using the constructors $\mathsf{0}$ and $\mathsf{1}$. Thus $r(f) + \Sigma_{i=r(f)+1,\ldots,n} R(p_i)$ if $arity(f) = n$. Note that this is precisely the coefficient of the polynomial $P = q_{er(f(p_1,\ldots,p_n))}$. On the other hand, let $R(\Gamma) = \sharp\{x \mid x : \diamond \in \Gamma\}$ be the number of variables of resource type in a context Γ. Suppose $\Gamma \vdash^{af}_\Sigma e$ is an affine typing of the expression e. Then it can be easily checked by induction on the typing that $q_{er(e)} \leq d + \Sigma_{v \in Var(er(e))} v$ for some $d \leq R(\Gamma)$. Then the assertion follows since the context Γ selected in (14) satisfies $R(\Gamma) = r(f) + \Sigma_{i=r(f)+1,\ldots,n} R(p_i)$. □

We consider a restricted class of multi-linear quasi-interpretations where:

$$q_f = a^f + \Sigma_{i=1,\ldots,n} x_i \quad a^f \geq 0 \tag{15}$$

for $arity(f) = n \geq 1$. We note that (i) constructors are subject to the general conditions of assignments, (ii) every function is determined by exactly *one* coefficient a^f, and (iii) the interpretation in proposition 24 falls in this family. We refer to this class of quasi-interpretations as *sum*-multi-linear.

Proposition 25 *The synthesis problem over sum-multi-linear interpretations can be solved in polynomial time.*

PROOF. The proof strategy is the same as in proposition 23. Now

$$q_{f(p_1,\ldots,p_n)} = a^f + \Sigma_{j=1,\ldots,l}(\Sigma_{i=1,\ldots,n}\alpha_{i,j})a^{\mathsf{c}_j} + \Sigma_{v\in\bigcup_{i=1,\ldots,n} Var(p_i)}v$$

and if $V \subseteq \bigcup_{i=1,\ldots,n} Var(p_i)$ the comparison condition is simply $a^f + \Sigma_{j=1,\ldots,l}(\Sigma_{i=1,\ldots,n}\alpha_{i,j})a^{\mathsf{c}_j} \geq b_V$. □

6.3 Lower and upper bounds on complexity

If a program admits a polynomial max-plus interpretation of degree k, then $q_{f(x_1,\ldots,x_n)} \leq k(\Sigma_{i=1,\ldots,n}x_i) + c$ for some constant c. Hence $q_f(v_1,\ldots,v_n)$ is in $O(\Sigma_{i=1,\ldots,n}|v_i|)$. Then it follows from theorem 6 that a program admitting a polynomial max-plus interpretation can be be evaluated in time $2^{O(n)}$ on data of size n.

For a lower bound, we refer to [Hof00] where it is shown that 'non-size increasing' recursive programs can simulate Turing machines (TM) running in time $2^{O(n)}$. This is inspired by the simulation of TM by APDA in Cook's theorem [Coo71]: define a recursive function T that, given an input $x \in \{0,1\}^*$, a number of steps $s \in \mathbf{N}$, and a position $p \in \mathbf{Z}$, computes a pair (q,i) such that: q is the state at which the machine will be after s steps starting with input x, and i is the character at position p on the tape of the TM after s steps. Because, the machine runs in time $2^{O(|x|)}$ it is possible to represent the number of steps and the position in space in $O(|x|)$. Then one can implement basic arithmetic operation like increment and decrement modulo 2^{cn}, and test for zero as size-preserving operations on lists. Finally, the definition of $T(x,s+1,p)$ can be given recursively in terms of $T(x,s,p-1)$, $T(x,s,p)$, and $T(x,s,p+1)$ by means of a straightforward case analysis. Quasi-interpretations can be combined with various methods enforcing *program termination*. In particular, in [BMM01] it is shown that a program terminating by *lexicographic path-order* (lpo[6]) and admitting a polynomially bounded quasi-interpretation (polynomial in the usual sense) can be evaluated in PSPACE. For a lower bound, we refer to the encoding of quantified boolean formulas (qbf) in appendix A that terminates by lpo and admits a multi-linear max-plus quasi-interpretation. By imposing further conditions on the termination method (product path-order) it is also possible to characterize PTIME [Mar00]. To prove these results, one can still rely on the basic evaluation strategies presented in the proof of theorem 6 in appendix B.2.

7 Conclusion

Polynomial interpretations are a classical topic. We have taken a fresh look at them focusing on space rather than on time bounds and shifting from the $(+,\times)$ algebra to the $(max,+)$ one. We have shown that the synthesis problem in the multi-linear case is NP-complete. This case appears to be as a reasonable compromise between complexity of the decision procedure and power of the analysis. The synthesis problem for max-plus polynomials of degree higher than 1 remains to be analyzed. Also it remains to be seen whether the approach can be extended to more complex functional languages including, *e.g.*, higher-order or coinductive types.

[6] A popular termination method that can be synthesized in non-deterministic polynomial time; see, *e.g.*, [BN98].

References

[BC92] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

[BCOQ92] F. Baccelli, G. Cohen, G. Olsder, and J.-P. Quadrat. *Synchronization and linearity.* Wiley, 1992.

[BMM01] G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. On termination methods with space bound certifications. In *Andrei Ershov Fourth International Conference "Perspectives of System Informatics"*, Lecture Notes in Computer Science. Springer, 2001.

[BN98] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[Cas97] V. Caseiro. *Equations for defining polytime functions.* PhD thesis, University of Oslo, 1997.

[Clo95] P. Clote. Computation models and function algebras. In *Proc. Logic and computational complexity, Springer Lecture Notes in Comp. Sci. 960*, 1995.

[Cob65] A. Cobham. The intrinsic computational difficulty of functions. In *Proc. Logic, Methodology, and Philosophy of Science II, North Holland*, 1965.

[Coo71] S. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18(1):4–18, 1971.

[GR02] S. Gaubert and K. Ricardo. Rational semimodules over the Max-Plus semiring and geometric approach of discrete event systems. Technical report, 2002. RR-4519, INRIA.

[Gra96] B. Gramlich. On proving termination by innermost termination. In *Proc. 7th Int. Conf. on Rewriting Techniques and Applications (RTA'96)*, volume 1103 of *Lecture Notes in Computer Science*, pages 93–107. Springer-Verlag, 1996.

[HJ03] M. Hofmann and S. Jost. Static prediction of heap space usage for first-order functional programs. In *Proc. ACM POPL*, 2003.

[Hof00] M. Hofmann. A type system for bounded space and functional in-place update. *Nordic Journal of Computing*, 7(4):258–289, 2000.

[Hof02] M. Hofmann. The strength of non size-increasing computation. In *Proc. ACM POPL*, 2002.

[HPS96] R. Hughes, L. Pareto, and A. Sabry. Proving the correctness of reactive systems using sized types. In *Proc. ACM POPL*, 1996.

[Imm99] N. Immerman. *Descriptive complexity.* Springer, 1999.

[Jon97] N. Jones. *Computability and complexity, from a programming perspective.* MIT-Press, 1997.

[Lei94] D. Leivant. Predicative recurrence and computational complexity i: word recurrence and poly-time. *Feasible mathematics II, Clote and Remmel (eds.)*, Birkhäuser:320–343, 1994.

[Mar00] J.-Y. Marion. *Complexité implicite des calculs, de la théorie à la pratique.* PhD thesis, Universitè Nancy, 2000. Habilitation à diriger des recherches.

[MM00] J.-Y. Marion and J.-Y. Moyen. Efficient first order functional program interpreter with time bound certifications. In *LPAR*, volume 1955 of *Lecture Notes in Computer Science*, pages 25–42. Springer, Nov 2000.

[Nec97] G. Necula. Proof carrying code. In *Proc. ACM POPL*, 1997.

[Par00] L. Pareto. *Types for crash prevention.* PhD thesis, Chalmers University of Technology, 2000.

[Pug92] W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. In *Communications of the ACM, 102-114*, 1992.

[San01] D. Sannella. Mobile resource guarantee. Ist-global computing research proposal, U. Edinburgh, 2001. http://www.dcs.ed.ac.uk/home/mrg/.

[WB00] P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In *Proc. TACAS, Springer Lecture Notes in Comp. Sci. 1785*, 2000.

A Examples of programs and quasi-interpretations

We provide a few examples of programs that can be defined in the language specified in section 2. Both the insertion sort and the common subsequence algorithms are considered in the literature [Mar00, Hof00] as situations where the constraints induced by ramification lead to unnatural programming. Qbf is a PSPACE-complete problem admitting a multi-linear quasi-interpretation[7] (a similar encoding can be found in in [Mar00]).

Example 26 (insertion sort) *We define a program that sorts lists of tally numbers. We assume the types bool, tnat, and tnatlist as in section 2.1. Then we define the following system of recursive functions:*

$$\begin{array}{ll}
\mathit{sort}(l) = & \\
l = \mathsf{nil} & \Rightarrow \mathsf{nil} \\
l = \mathsf{cons}(x, l') & \Rightarrow \mathit{insert}(x, \mathit{sort}(l')) \\
 & \\
\mathit{insert}(x, l) = & \\
l = \mathsf{nil} & \Rightarrow \mathsf{cons}(x, \mathsf{nil}) \\
l = \mathsf{cons}(y, l') & \Rightarrow \mathit{if}(\mathit{lesseq}(x, y), \mathsf{cons}(x, \mathsf{cons}(y, l')), \mathsf{cons}(y, \mathit{insert}(x, l'))) \\
 & \\
\mathit{if}(x, y, z) = & \\
x = \mathsf{tt} & \Rightarrow y \\
x = \mathsf{ff} & \Rightarrow z \\
 & \\
\mathit{lesseq}(x, y) = & \\
x = 0 & \Rightarrow \mathsf{tt} \\
x = \mathsf{s}(x'), y = 0 & \Rightarrow \mathsf{ff} \\
x = \mathsf{s}(x'), y = \mathsf{s}(y') & \Rightarrow \mathit{lesseq}(x', y') \ .
\end{array}$$

The program admits the following quasi-interpretation:

$$\begin{array}{lll}
q_{\mathsf{s}} = x + 1, & q_{\mathsf{cons}} = x + l + 1, & q_{sort} = l, \\
q_{insert} = x + l + 1, & q_{if} = max(x, y, z), & q_{lesseq} = max(x, y) \ .
\end{array}$$

Example 27 (common subsequence) *We define a program that computes the length of a longest common subsequence of two binary words. The length is represented by a tally natural number and the words by lists of booleans. The definition of the if function is borrowed from the previous example.*

$$\begin{array}{ll}
\mathit{lcs}(x, y) = & \\
x = \mathsf{nil} & \Rightarrow 0 \\
x = \mathsf{cons}(x', l), y = \mathsf{nil} & \Rightarrow 0 \\
x = \mathsf{cons}(x', l), y = \mathsf{cons}(y', l') & \Rightarrow \mathit{if}(\mathit{eq}(x', y'), \mathsf{s}(\mathit{lcs}(l, l')), \\
 & \qquad \mathit{max}(\mathit{lcs}(\mathsf{cons}(x', l), l'), \mathit{lcs}(l, \mathsf{cons}(y', l'))))
\end{array}$$

$$\begin{array}{llll}
\mathit{eq}(x, y) = & & \mathit{max}(x, y) = & \\
x = \mathsf{tt}, y = \mathsf{tt} & \Rightarrow \mathsf{tt} & x = 0 & \Rightarrow y \\
x = \mathsf{ff}, y = \mathsf{ff} & \Rightarrow \mathsf{tt} & x = \mathsf{s}(x'), y = 0 & \Rightarrow \mathsf{s}(x') \\
x = \mathsf{tt}, y = \mathsf{ff} & \Rightarrow \mathsf{ff} & x = \mathsf{s}(x'), y = \mathsf{s}(y') & \Rightarrow \mathsf{s}(\mathit{max}(x', y')) \ . \\
x = \mathsf{ff}, y = \mathsf{tt} & \Rightarrow \mathsf{ff} & &
\end{array}$$

[7] Qbf is known to be solvable in linear space.

The program admits the following quasi-interpretation:

$$q_{\mathsf{s}} = x+1, \quad q_{\mathsf{cons}} = x+l+1, \quad q_{lcs} = max(x,y),$$
$$q_{if} = max(x,y,z), \quad q_{eq} = max(x,y), \quad q_{max} = max(x,y) \ .$$

Example 28 (qbf) *We define a program that verifies the validity of a closed quantified boolean formula (qbf). Truth values are represented by the type bool. Names of variables are coded as tally natural numbers and we use a list of tally natural numbers to represent the variables that are assigned the truth value* tt*. Finally, qbf formulas are elements of the type*

$$\begin{array}{ll} form & \equiv \mu t.(\ \ \mathsf{v} : tnat \to t, \mathsf{n} : t \to t \\ & \qquad \mathsf{a} : t,t \to t, \mathsf{o} : t,t \to t, \\ & \qquad \mathsf{all} : tnat, t \to t, \mathsf{ex} : tnat, t \to t) \ . \end{array}$$

We leave to the reader the definition of the boolean functions and, or, not, and of the test for equality of tally numbers eq. We also need a function that checks for membership of an element in a list

$$\begin{array}{ll} mem(x,l) = & \\ l = \mathsf{nil} & \Rightarrow \mathsf{ff} \\ l = \mathsf{cons}(y,l') & \Rightarrow or(eq(x,y), mem(x,l')) \ . \end{array}$$

The main program checks a formula with respect to a list of variables that have been affected the value tt.

$$\begin{array}{ll} qbf(\phi) = & check(\phi, nil) \\ & \\ check(\phi,l) = & \\ \phi = \mathsf{v}(x) & \Rightarrow mem(x,l) \\ \phi = \mathsf{n}(\phi') & \Rightarrow not(check(\phi',l)) \\ \phi = \mathsf{a}(\phi',\phi'') & \Rightarrow and(check(\phi',l), check(\phi'',l)) \\ \phi = \mathsf{o}(\phi',\phi'') & \Rightarrow or(check(\phi',l), check(\phi'',l)) \\ \phi = \mathsf{all}(x,\phi') & \Rightarrow and(check(\phi', \mathsf{cons}(x,l)), check(\phi',l)) \\ \phi = \mathsf{ex}(x,\phi') & \Rightarrow or(check(\phi', \mathsf{cons}(x,l)), check(\phi',l)) \ . \end{array}$$

The program admits the following quasi-interpretation:

$$\begin{array}{ll} q_{\mathsf{v}} = q_{\mathsf{n}} = x+1, & q_{\mathsf{a}} = q_{\mathsf{o}} = q_{\mathsf{all}} = q_{\mathsf{ex}} = x+y+1, \\ q_{not} = q_{qbf} = x, & q_{and} = q_{or} = q_{eq} = q_{mem} = max(x,y), \\ q_{check} = \phi + l, & q_{max} = max(x,y) \ . \end{array}$$

B Proofs

B.1 Proof of proposition 5

(1) We take d as the largest additive coefficient d' occurring in the interpretation $\Sigma_{i=1,\ldots,n} x_i + d'$ of a constructor of positive arity n. Then the assertion is proven by induction on the structure of the value v.

(2) First we note that for all expressions e with $Var(e) = \{x_1, \ldots, x_n\}$ and for all substitutions σ over $Var(e)$ the following identity holds:

$$q_{\sigma e} = q_e(q_{\sigma(x_1)}, \ldots, q_{\sigma(x_n)}) \ . \tag{16}$$

Then we proceed by induction on the definition of the evaluation relation $\mapsto$. Let us consider the case where the last rule applied is (*fun*). By inductive hypothesis, $q_{e'_j} \geq q_{v_j}$ for $j = 1, \ldots, n$. Hence by the monotonicity property (2.2) of an assignment

$$q_f(q_{e'_1}, \ldots, q_{e'_n}) \geq q_f(q_{v_1}, \ldots, q_{v_n}) \ . \tag{17}$$

Since q is a quasi-interpretation, we know that

$$q_{f(p_{i,1},\ldots,p_{i,n})} \geq q_{e_i} \ . \tag{18}$$

Thus we obtain

$$\begin{array}{lll} q_{f(e'_1,\ldots,e'_n)} & \geq q_{f(q_{v_1},\ldots,q_{v_n})} & \text{by (17)} \\ & = q_{\sigma(f(p_{i,1},\ldots,p_{i,n}))} & \text{by definition of rule } (\mathit{fun}) \\ & \geq q_{\sigma e_i} & \text{by (18) and (16)} \\ & \geq q_v & \text{by inductive hypothesis.} \end{array}$$

B.2 Proof of theorem 6

By property (2.1) of assignments, we note that if e' is a subexpression of the expression e then $q_e \geq q_{e'}$. Let $B = q_{f(v_1,\ldots,v_n)}$. It follows from the remark above and (1) that any value v' obtained in the course of the computation of $f(v_1, \ldots, v_n)$ is such that $|v'| \leq B$. Note that both the number of constructors in the program and the arity of a function are bound by a constant. It follows that the number of values to which a function can be applied in the course of the computation is in $2^{O(B)}$.

A 'call-by-value' evaluation context E for an expression e is defined as follows:

$$E ::= [\ ] \mid \mathsf{c}(v_1, \ldots, v_{i-1}, E, e_{i+1}, \ldots, e_n) \mid g(v_1, \ldots, v_{i-1}, E, e_{i+1}, \ldots, e_n) \ .$$

It is easy to verify that any closed expression e which is not a value admits a unique decomposition in an evaluation context E and a function application $g(v_1, \ldots, v_n)$ so that $e \equiv E[g(v_1, \ldots, v_n)]$.

We define an evaluation function *Eval* that performs an innermost leftmost evaluation of an expression.

$$\begin{array}{ll} \mathit{Eval}(e) = \mathsf{case} & \\ e \text{ value} \ : \ e & \\ e \equiv E[f(v_1, \ldots, v_n)] \text{ and } \exists\, \sigma, i \ \ (\sigma(p_{i,j}) = v_j, j = 1, \ldots, n) & : \mathsf{let}\ v' = \mathit{Eval}(\sigma(e_i))\ \mathsf{in} \\ & \quad \mathit{Eval}(E[v']) \\ \mathsf{else} \ : \ \mathit{Return}\ \bot & \end{array}$$

where we assume that the function f is defined as in section 2.3 and that invoking *Return*$\bot$ stops the computation returning $\bot$ as result. Let k be the maximum number of function symbols that occur in an expression on the right hand side of $\Rightarrow$ in a function definition. We note that the evaluation function initially applied to the expression $f(v_1, \ldots, v_n)$ maintains the invariant that the number of function symbols in an argument e is bound by k. It is easy to see that the size of an expression e such that $q_e \leq B$ and containing at most k function symbols has size in $O(B)$. It follows that both the expressions and the values involved in the evaluation have size in $O(B)$. Hence a stack frame for *Eval* has size in $O(B)$ and *Eval* can be implemented on an *auxiliary deterministic pushdown automata* with auxiliary memory which is in $O(B)$. Then by a well-known result by S. Cook [Coo71], the function can also be implemented to run on a Turing Machine in time $2^{O(B)}$.

Classically, this transformation relies on a technique called *memoization* that saves computed results and thus avoids recomputing several times a function with the same arguments.

A simple description of this idea is given by the evaluator below $Eval_m$ that relies on a global table T which is initially empty and is accessed with two procedures *Insert* and *Update*:

$$
\begin{array}{l}
Eval_m(e) = \mathsf{case} \\
e \text{ value } : \; e \\
e \equiv E[f(v_1,\ldots,v_n)] \text{ and } \exists\, \sigma, i \;\; (\sigma(p_{i,j}) = v_j, j = 1,\ldots,n) : \\
\qquad (new, v'') := Insert(f(v_1,...,v_n)); \\
\qquad \mathsf{case} \\
\qquad new : \; \mathsf{let}\ v' = Eval_m(\sigma(e_i))\ \mathsf{in} \qquad (1) \\
\qquad \qquad Update(f(v_1,...,v_n), v'); \\
\qquad \qquad Eval_m(E[v']) \\
\qquad \neg new, v'' \neq \bot : \; Eval_m(E[v'']) \qquad (2) \\
\qquad \mathsf{else} \; : \; Return \perp \\
\mathsf{else} \; : \; Return \perp .
\end{array}
$$

The *Insert* and *Return* procedures are defined as follows:

$$
\begin{array}{ll}
Insert(f(v_1,\ldots,v_n)) = \mathsf{case} & \\
(f(v_1,\ldots,v_n), v) \in T & : (\mathsf{false}, v) \\
\mathsf{else} & : T := T \cup \{(f(v_1,\ldots,v_n), \perp)\}; (\mathsf{true}, \perp) \\
 & \\
Update(f(v_1,\ldots,v_n), v) & = T := T \backslash \{(f(v_1,\ldots,v_n), \perp)\} \cup \{(f(v_1,\ldots,v_n), v)\} \; .
\end{array}
$$

The table can be implemented so that these procedures run in time in $O(B)$. Since the table T can contain at most $2^{O(B)}$ entries, branch (1) can be taken at most $2^{O(B)}$ times. On the other hand, branch (2) decreases by one the number of function symbols in the evaluated expression. This number being bound by a constant, we can take branch (2) only a constant number of times before running again branch (1). We conclude that the evaluation strategy runs in time $2^{O(B)}$. □

B.3 Proof of theorem 8

We present a polynomial reduction from 3-SAT. We need the following lemma that we prove later.

Lemma 29 *To any function symbol f with n arguments we can associate rules of bounded size and in number polynomial in n so that a max-plus polynomial assignment q satisfies the constraints induced by the rules iff $q_f = max(x_1,\ldots,x_n)$.*

In the following we denote with f_n a function defined as specified by the lemma so that any quasi-interpretation satisfies $q_{f_n} = max(x_1,\ldots,x_n)$.

Given a formula in 3-CNF we introduce a pair of unary constructors c and $\overline{\mathsf{c}}$ for every variable c occurring in the formula. We also assume a unary constructor d and a constant b Associated to this constructor we have a coefficient $a^{\mathsf{d}} \geq 1$. We want to force the property:

$$a^{\mathsf{c}}, a^{\overline{\mathsf{c}}} \in \{a^{\mathsf{d}}, 2a^{\mathsf{d}}\}, \quad a^{\mathsf{c}} = a^{\mathsf{d}} \text{ iff } a^{\overline{\mathsf{c}}} = 2a^{\mathsf{d}} \; .$$

Since $a^{\mathsf{d}} \geq 1$ we know $a^{\mathsf{d}} < 2a^{\mathsf{d}}$ and we will use a^{d} to represent the boolean value 0 and $2a^{\mathsf{d}}$ to represent the boolean value 1.

To this end, we write rules of bounded size and in number polynomial in the size of the formula as follows:

$$\begin{array}{lll} f_1(\mathsf{l}(x)) \Rightarrow \mathsf{d}(x) & f_1(\mathsf{d}(\mathsf{d}(x)) \Rightarrow \mathsf{l}(x) & \mathsf{l} = \mathsf{c} \text{ or } \mathsf{l} = \overline{\mathsf{c}} \\ f_2(\mathsf{c}(x), \overline{\mathsf{c}}(y)) \Rightarrow \mathsf{d}(\mathsf{d}(\mathsf{b})) & f_1(\mathsf{d}(\mathsf{d}(\mathsf{d}(x)))) \Rightarrow \mathsf{c}(\overline{\mathsf{c}}(x)) & \end{array}$$

To express that a disjunction $(l_1 \vee l_2 \vee l_3)$ in the original formula is satisfied we add the rule:

$$f_1(\mathsf{l}_1(\mathsf{l}_2(\mathsf{l}_3(x)))) \Rightarrow \mathsf{d}(\mathsf{d}(\mathsf{d}(\mathsf{d}(x)))) \tag{19}$$

which forces $a^{\mathsf{l}_i} = 2a^{\mathsf{d}}$ for some $i \in \{1, 2, 3\}$.

• Let ϕ be a boolean formula in conjunctive normal form with disjunctions composed of three literals and let P_ϕ be the collection of associated functions' definitions. We check that an assignment ρ satisfies ϕ iff there is a max-plus polynomial quasi-interpretation q that satisfies P_ϕ.

($\Rightarrow$) Suppose ρ satisfies ϕ. We define a quasi interpretations q_ϕ for the associated program such that:

$$\begin{array}{llll} q_f & = max(x_1, \ldots, x_n) \text{ if } arity(f) = n & q_{\mathsf{d}} & = x + k \text{ with } k \geq 1 \\ q_{\mathsf{c}} & = \begin{cases} k & \text{if } \rho(c) = 0 \\ 2k & \text{if } \rho(c) = 1 \end{cases} & q_{\overline{\mathsf{c}}} & = \begin{cases} k & \text{if } \rho(c) = 1 \\ 2k & \text{if } \rho(c) = 0 \end{cases}. \end{array}$$

($\Leftarrow$) Suppose q is a quasi-interpretation for P_ϕ. By the lemma we know that $q_f = max(x_1, \ldots, x_n)$. Also if $q_{\mathsf{d}} = x + k$ then for all c: $q_{\mathsf{c}} = q_{\mathsf{d}}$ or $q_{\overline{\mathsf{c}}} = q_{\mathsf{d}}$ and $q_{\mathsf{c}} = x + 2k$ or $q_{\overline{\mathsf{c}}} = x + 2k$. Then the following boolean assignment is well-defined:

$$\rho_q(c) = \begin{cases} 0 & \text{if } q_{\mathsf{c}} = x + k \\ 1 & \text{if } q_{\mathsf{c}} = x + 2k \end{cases}.$$

By the condition induced by the rule (19) it follows that ρ_q satisfies the formula ϕ.

• We now turn to the proof of the lemma 29. Let P be the max-plus polynomial assigned to f. The polynomial can be written as $P = max_{i \in I}(\Sigma_{j=1,\ldots,n}\alpha_{i,j}x_j + a_i)$ where $\alpha_{i,j} \in \mathbf{N}$ and $a_i \geq 0$.

Again we assume to have some constructors d, b available. Consider the following rule:

$$e \equiv f(\mathsf{b}, \ldots, \mathsf{b}, \mathsf{d}(x), \mathsf{b}, \ldots, \mathsf{b}) \Rightarrow f(\mathsf{b}, \ldots, \mathsf{b}, f(\mathsf{b}, \ldots, \mathsf{b}, \mathsf{d}(x), \mathsf{b}, \ldots, \mathsf{b}), \mathsf{b}, \ldots, \mathsf{b}) \equiv e' \tag{20}$$

where the expression $\mathsf{d}(x)$ occurs as the j^{th} argument. We claim that if the assignment satisfies this rule then $\alpha_{i,j} \in \{0, 1\}$ for all $i \in I$. Suppose $\alpha_{k,j} = max\{\alpha_{i,j} \mid i \in I\}$. By the condition on assignment it must be that $\alpha_{k,j} \geq 1$. We may assume that k is chosen so that $\alpha_{k,j} = \alpha_{k',j}$ implies $a_k \geq a_{k'}$.

For x large enough, $q_e = \alpha_{k,j}(x + a^{\mathsf{d}}) + a_k$ where $a^{\mathsf{d}} \geq 1$ is the coefficient associated to the constructor. On the other hand, $q_{e'} = \alpha_{k,j}(q_e) + a_k$. The inequality $q_e \geq q_{e'}$ forces $\alpha_{k,j} = 1$. Thus now for x large enough the condition simplifies into $x + a^{\mathsf{d}} + a_k \geq x + a^{\mathsf{d}} + 2a_k$ which forces $a_k = 0$.

Thus, by introducing rules of type (20) for every argument, we can show that: (i) $\alpha_{i,j} \in \{0, 1\}$ and (ii) $\alpha_{i,j} = 1$ implies $a_i = 0$.

• Next we want to force the property that $P = max(a, x_1, \ldots, x_n)$ for some a. To this end we add the rule

$$e_1 \equiv f(\mathsf{c}(x_1), \ldots, \mathsf{c}(x_n)) \Rightarrow f(f(\mathsf{c}(x_1), \ldots, \mathsf{c}(x_n)), \ldots, f(\mathsf{c}(x_1), \ldots, \mathsf{c}(x_n))) \equiv e_2 \tag{21}$$

for some fresh constructor c. Clearly, if $q_{e_1} \geq q_{e_2}$ then P cannot add two arguments.

• Finally, to force $a = 0$ we consider the following rule:

$$f(\mathsf{e}(\mathsf{b}, x), \mathsf{b}, \ldots, \mathsf{b}) \Rightarrow \mathsf{e}(f(\mathsf{b}, \ldots, \mathsf{b}), f(\mathsf{b}, \ldots, \mathsf{b})) \ . \tag{22}$$

This requires $max(a, x + a^{\mathsf{e}}) \geq a^{\mathsf{e}} + 2a$. For $x = 0$ this means $max(a, a^{\mathsf{e}}) \geq a^{\mathsf{e}} + 2a$. Since $a \geq a^{\mathsf{e}} \geq 1$ this forces $a^{\mathsf{e}} \geq a$ and $a^{\mathsf{e}} \geq a^{\mathsf{e}} + 2a$. And the latter implies $a = 0$.

• In the case all constructors of positive arity are assigned the same coefficient, say $k \geq 1$, we need a more elaborate proof strategy. We note that lemma 29 still holds since its proof does not require constructors of positive arity with distinct coefficients. So we can still use f_n as a function symbol such that $q_{f_n} = max(x_1, \ldots, x_n)$. In the following we will write rules using always the same constructor symbols c and b. In case the patterns superpose, it is intended that the constructor symbols are suitably renamed.

• Suppose f is a function symbol of arity n and consider the rule:

$$f_1(\mathsf{c}(\mathsf{c}(x))) \Rightarrow f(x, \ldots, x) \ . \tag{23}$$

If $q_f = max_{i \in I}(\Sigma_{j=1,\ldots,n} \alpha_{i,j} x_j + a_i)$ is a max-plus polynomial (where $\alpha_{i,j} \in \mathbf{N}$ and $a_i \geq 0$) then the rule (23) forces the following conditions:

$$\forall i \in I \ \ 1 \geq \Sigma_{j=1,\ldots,n} \alpha_{i,j} \text{ and } 2k \geq a_i \ . \tag{24}$$

Thus q_f must be a multi-linear polynomial of the shape:

$$q_f = max(a_0, a_1 + x_1, \ldots, a_n + x_n) \ , \tag{25}$$

and we can assume $2k \geq a_0 \geq a_i \geq 0$, for $i = 1, \ldots, n$.

• Next add rules of the following shape for the same function symbol f:

$$f(\mathsf{b}, \ldots, \mathsf{b}, \mathsf{c}(x), \mathsf{b}, \ldots, \mathsf{b}) \Rightarrow \mathsf{c}(\mathsf{c}(x)) \ . \tag{26}$$

If $\mathsf{c}(x)$ occurs as the j^{th} argument then we require $max(a_0, \ldots, a_{j-1}, x + k + a_j, a_{j+1}, \ldots, a_n) \geq 2k + x$ which forces $a_j \geq k$. By varying the position of $\mathsf{c}(x)$ between the first and the last argument of f we obtain the condition

$$a_i \geq k \text{ for } i = 1, \ldots, n \ . \tag{27}$$

• Now add a rule of the following shape for the same function symbol f:

$$f(\mathsf{c}(\mathsf{c}(x_1)), \ldots, \mathsf{c}(\mathsf{c}(x_n)) \Rightarrow \mathsf{c}(\mathsf{c}(\mathsf{c}(\mathsf{c}(\mathsf{b})))) \ . \tag{28}$$

This requires $max(a_0, 2k + a_1 + x_1, \ldots, 2k + a_n + x_n) \geq 4k$. Since by condition (24) $a_0 \leq 2k$, this is equivalent to

$$a_0 = max(a_1, \ldots, a_n) = 2k \ . \tag{29}$$

• For a function symbol f of arity 2 we add a rule of the following shape:

$$f_2(\mathsf{c}(\mathsf{c}(\mathsf{c}(x))), \mathsf{c}(\mathsf{c}(\mathsf{c}(\mathsf{c}(\mathsf{b}))))) \Rightarrow f(f(\mathsf{b}, x), \mathsf{b}) \; . \tag{30}$$

This requires $max(x+3k, 4k) \geq max(x+a_1+a_2, a_0, a_2, a_1+a_0, 2a_1)$ and since $2k \geq a_0 \geq a_1, a_2$ this is equivalent to $3k \geq a_1 + a_2$ which coupled with condition (29) can be expressed as:

$$(a_1 = k \wedge a_2 = 2k) \vee (a_1 = 2k \wedge a_2 = k) \; . \tag{31}$$

The goal here is to represent a boolean variable with the coefficients a_1, a_2 of the binary function f so that the variable evaluates to 1 iff $a_1 = 2k$.

• Given a formula ϕ in 3-CNF, for every propositional variable u we introduce a binary function symbol u subject to the conditions (24,27,29,31). Thus $q_u = max(a_1 + x_1, a_2 + x_2)$ and $(a_1 = k \wedge a_2 = 2k) \vee (a_1 = 2k \wedge a_2 = k)$.

• For every 3-disjunction d in the formula ϕ we introduce a ternary function symbol d subject to the conditions (24,27). Thus $q_d = max(b_0, b_1 + x_1, b_2 + x_2, b_3 + x_3)$ with $2k \geq b_0 \geq b_i \geq k$ for $i = 1, \ldots, 3$. If the first literal of the disjunction d is the propositional variable u then we want to force $b_1 = a_1$. This can be done with the rules:

$$d(\mathsf{c}(\mathsf{c}(x_1)), \mathsf{b}, \mathsf{b}) \Rightarrow u(\mathsf{c}(\mathsf{c}(x_1)), \mathsf{b}) \quad u(\mathsf{c}(\mathsf{c}(x_1)), \mathsf{b}) \Rightarrow d(\mathsf{c}(\mathsf{c}(x_1)), \mathsf{b}, \mathsf{b}) \; . \tag{32}$$

On the other hand, if the first literal of the disjunction is $\overline{u}$ then we want to force $b_1 = a_2$. Thus we write:

$$d(\mathsf{c}(\mathsf{c}(x)), \mathsf{b}, \mathsf{b}) \Rightarrow u(\mathsf{b}, \mathsf{c}(\mathsf{c}(x))) \quad u(\mathsf{b}, \mathsf{c}(\mathsf{c}(x))) \Rightarrow d(\mathsf{c}(\mathsf{c}(x)), \mathsf{b}, \mathsf{b}) \; . \tag{33}$$

We add this type of rules for every disjunction d and for every argument of the associated function symbol.

• To express the fact that every disjunction d evaluates to 1 we require that at least one of the coefficients of the associated ternary function evaluates to $2k$. This is expressed by a variant of the rule (28) as follows:

$$d(\mathsf{c}(\mathsf{c}(x_1)), \mathsf{c}(\mathsf{c}(x_2)), \mathsf{c}(\mathsf{c}(x_3))) \Rightarrow \mathsf{c}(\mathsf{c}(\mathsf{c}(\mathsf{c}(\mathsf{b})))) \; . \tag{34}$$

Then satisfying boolean assignments and quasi-interpretations can be related along the lines of what has been discussed above. □

B.4 Proof of proposition 11

(1) By induction on the structure of p.

$p \equiv \mathsf{c}$ Then $q_{\mathsf{c}} = 0$ and we take $\alpha_j = 0$ for $j = 1, \ldots, l$.

$p \equiv x$ Then $q_x = x$ and we take as in the previous case $\alpha_j = 0$ for $j = 1, \ldots, l$.

$p \equiv \mathsf{c}_k(p_1, \ldots, p_n)$ By hypothesis on the shape of patterns in functions' definitions we know that $Var(p_i) \cap Var(p_j) = \emptyset$ if $i \neq j$. By inductive hypothesis, $q_{p_i} = \Sigma_{v \in Var(p_i)} v + \Sigma_{j=1,\ldots,l} \alpha_{i,j} a^{\mathsf{c}_j}$ for some $\alpha_{i,j} \in \mathbf{N}$. Then

$$\begin{aligned} q_p &= q_{p_1} + \cdots q_{p_n} + a^{\mathsf{c}_k} \\ &= \Sigma_{v \in Var(p_1)} v + \cdots + \Sigma_{v \in Var(p_n)} v + \Sigma_{j=1,\ldots,l} \alpha_{1,j} a^{\mathsf{c}_j} + \cdots + \Sigma_{j=1,\ldots,l} \alpha_{n,j} a^{\mathsf{c}_j} + a^{\mathsf{c}_k} \\ &= \Sigma_{v \in Var(p)} v + \Sigma_{j=1,\ldots,l} \alpha'_j a^{\mathsf{c}_j} \end{aligned}$$

$$\text{where } \alpha'_j = \begin{cases} \Sigma_{i=1,\ldots,n}\alpha_{i,j} & \text{if } j \neq k \\ \Sigma_{i=1,\ldots,n}\alpha_{i,k} + 1 & \text{if } j = k \ . \end{cases}$$

(2) We start by computing:

$$\begin{aligned} q_{f(p_1,\ldots,p_n)} &= max_{I\subseteq\{1,\ldots,n\}}(a^f_I + \Sigma_{i\in I}(\Sigma_{v\in Var(p_i)}v + \Sigma_{j=1,\ldots,l}\alpha_{i,j}a^{\mathsf{c}_j})) \\ &= max_{I\subseteq\{1,\ldots,n\}}(\Sigma_{v\in\bigcup_{i\in I} Var(p_i)}v + a^f_I + \Sigma_{j=1,\ldots,l}(\Sigma_{i\in I}\alpha_{i,j})a^{\mathsf{c}_j}) \end{aligned}$$

Now, let P be the multi-linear polynomial determined by the coefficients b_V. We show that $q_{f(p_1,\ldots,p_n)} = P$.

$q_{f(p_1,\ldots,p_n)} \leq P$. Fix $I \subseteq \{1,\ldots,n\}$ and take $V = \bigcup_{i\in I} Var(p_i)$. Then $K_V = \{k \mid V \cap Var(p_k) \neq \emptyset\} \subseteq I$, as by the conditions on patterns, $Var(p_i) \cap Var(p_k) \neq \emptyset$ implies $i = k$.

It follows, by the normalization constraint that $a^f_{K_V} \geq a^f_I$. We also note that if $i \in I\backslash K_V$ then $Var(p_i) = \emptyset$ and therefore $\alpha_{i,j} = 0$ for $j = 1,\ldots,l$. Hence

$$\Sigma_{i\in I}\alpha_{i,j} = \Sigma_{i\in K_V}\alpha_{i,j} + \Sigma_{i\in I\backslash K_V}\alpha_{i,j} = \Sigma_{i\in K_V}\alpha_{i,j} \ .$$

$P \leq q_{f(p_1,\ldots,p_n)}$. Given V and the related set K_V we set $I = K_V$. Then $a^f_I = a^f_{K_V}$, $V \subseteq \bigcup_{i\in I} Var(p_i)$, and $\Sigma_{k\in K_V}\alpha_{k,j} = \Sigma_{i\in I}\alpha_{i,j}$. □

B.5 Proof of proposition 14

(1) We note that in general, $q_{e_i} \geq v$ if $v \in V_i$. Thus $q_{e_i} + q_{e_j}$ is multi-linear only if $V_i \cap V_j = \emptyset$.

(1.1) Since $a^{\mathsf{c}} \geq 1$, to compute $q_{\mathsf{c}(e_1,\ldots,e_n)}$ we have to assume that

$$V_i \cap V_j = \emptyset \text{ if } i \neq j \ . \tag{35}$$

Otherwise the resulting polynomial is not multi-linear. Then

$$q_{\mathsf{c}(e_1,\ldots,e_n)} = max_{U_i\subseteq V_i, i=1,\ldots,n}(\Sigma_{v\in\bigcup_{i=1,\ldots,n} U_i}v + \Sigma_{i=1,\ldots,n}a^i_{U_i} + a^{\mathsf{c}}) \ .$$

Let $U \subseteq V$. To determine the coefficient b_U of $q_{\mathsf{c}(e_1,\ldots,e_n)}$ we have to consider all families $U_1,\ldots,U_n$ such that $U_i \subseteq V_i$ for $i = 1,\ldots,n$ and $\bigcup_{i=1,\ldots,n} U_i = U$. This forces $U_i = U \cap V_i$. Thus

$$b_U = \Sigma_{i=1,\ldots,n}a^i_{U\cap V_i} + a^{\mathsf{c}} \ . \tag{36}$$

Therefore condition (35) is also sufficient to preserve multi-linearity.

(1.2) To compute $q_{f(e_1,\ldots,e_n)}$ suppose moreover that q_f is determined by the coefficients $\{a^f_I \mid I \subseteq \{1,\ldots,n\}\}$. It is necessary to assume that $a^f_I = -\infty$ whenever $\not\downarrow I$. Since we require that q_f is in normal form we may equivalently express this condition by stating that

$$a^f_{\{i,j\}} = -\infty \text{ whenever } i \neq j \text{ and } V_i \cap V_j \neq \emptyset \ . \tag{37}$$

Otherwise, the resulting polynomial is not multi-linear. Then

$$\begin{aligned} q_{f(e_1,\ldots,e_n)} &= max_{I\subseteq\{1,\ldots,n\},\downarrow I}(\Sigma_{i\in I}(max_{U_i\subseteq V_i}(\Sigma_{v\in U_i}v + a^i_{U_i} + a^f_I))) \\ &= max_{I\subseteq\{1,\ldots,n\},\downarrow I, U_i\subseteq V_i}(\Sigma_{v\in\bigcup_{i\in I} U_i}v + \Sigma_{i\in I}a^i_{U_i} + a^f_I) \ . \end{aligned}$$

Let $U \subseteq V$. To determine the coefficient b_U of $q_{f(e_1,\ldots,e_n)}$ we have to consider all the $I \subseteq \{1,\ldots,n\}$ such that (i) $\downarrow I$ and (ii) for $U_i \subseteq V_i$, $i \in I$, we have $U = \bigcup_{i\in I} U_i$. By (i), (ii) is actually equivalent to $U \subseteq \bigcup_{i\in I} V_i$ taking $U_i = U \cap V_i$. Thus

$$b_U = max_{I\subseteq\{1,\ldots,n\},\downarrow I, U\subseteq\bigcup_{i\in I} V_i}(\Sigma_{i\in I} a^i_{U\cap V_i} + a^f_I) \ . \tag{38}$$

Therefore condition (37) is also sufficient to preserve multi-linearity.

(2) Following the analysis above, we prove the assertion by induction on the proof of (e, C).

$e \equiv x$. Then $q_e \equiv x$ is multi-linear, $C = \emptyset$, and q satisfies C.

$e \equiv \mathsf{c}(e_1,\ldots,e_n)$. Suppose $\vdash (e_i, C_i)$ for $i = 1,\ldots,n$. We distinguish two cases.

$Var(e_i) \cap Var(e_j) = \emptyset$ if $i \neq j$. Then $\vdash (\mathsf{c}(e_1,\ldots,e_n), \bigcup_{i=1,\ldots,n} C_i)$. If q_e is multi-linear then q_{e_i} must be multi-linear since $q_{\mathsf{c}(e_1,\ldots,e_n)} \geq q_{e_i}$. Thus by inductive hypothesis, q satisfies C_i for $i = 1,\ldots,n$, that is q satisfies $\bigcup_{i=1,\ldots,n} C_i$. Vice versa, if q satisfies $\bigcup_{i=1,\ldots,n} C_i$ then by inductive hypothesis, q_{e_i} is multi-linear and by the computation above q_e is also multi-linear.

$Var(e_i) \cap Var(e_j) \neq \emptyset$ for $i \neq j$. Then $\vdash (\mathsf{c}(e_1,\ldots,e_n), \{\perp\} \cup \bigcup_{i=1,\ldots,n} C_i)$. Hence $q_{\mathsf{c}(e_1,\ldots,e_n)}$ cannot be multi-linear and q cannot satisfy $\{\perp\} \cup \bigcup_{i=1,\ldots,n} C_i$.

$e \equiv f(e_1,\ldots,e_n)$. Suppose $\vdash (e_i, C_i)$ for $i = 1,\ldots,n$. Again if q_e is multi-linear then q_{e_i} is multi-linear and by inductive hypothesis q satisfies C_i for $i = 1,\ldots,n$. Moreover, since q is multi-linear it must also satisfy condition (37). Vice versa, if q satisfies the constraints $\{a^f_{i,j} = -\infty \mid i \neq j, Var(e_i) \cap Var(e_j) \neq \emptyset\} \cup \bigcup_{i=1,\ldots,n} C_i$ then by inductive hypothesis q_{e_i} is multi-linear for $i = 1,\ldots,n$ and q_e is also multi-linear by the computation above. □

B.6 Proof of proposition 16

(1) Clearly, if the condition (12) holds then $P_1 \geq P_2$. Vice versa suppose $P_1 \geq P_2$ and consider a monomial $\Sigma_{i\in J} x_i + b_J$ in P_2 and the vector X_J whose components are specified by:

$$(X_J)_i = \begin{cases} x & \text{if } i \in J \\ 0 & \text{otherwise} \end{cases}$$

Then $P_1(X_J) \geq P_2(X_J) \geq (\sharp J)x + b_J$. For sufficiently large x this means that there is a $K \supseteq J$ such that $P_1(X_J) = (\sharp J)x + a_K \geq (\sharp J)x + b_J$. Which implies that $max\{a_I \mid I \supseteq J\} \geq a_K \geq b_J$.

(2) If P_1 is in normal form then $max\{a_I \mid I \supseteq J\} = a_J$ and the argument in (1) applies. □

B.7 Proof of proposition 19

Initially, the constraints have the shapes *(a–c)* specified in proposition 18. We also allow constraints of the shape $\Sigma_{j\in J} x_j = -\infty$. It will be convenient to add to the system the constraints $y \geq 0$ whenever $y \geq 1$ and write a sum $\Sigma_{j=1,\ldots,k} \alpha_j u_j$ where $\alpha_j \in \mathbf{N}$ as $\Sigma_{j\in J} x_j$ for a suitable j. Using this notation, we introduce the following simplification rules:

$$(0) \qquad \frac{S, x + \Sigma_{j\in J_1} y_j \geq \Sigma_{j\in J_2} y_j}{S, x \geq 0, x + \Sigma_{j\in J_1} y_j \geq \Sigma_{j\in J_2} y_j}$$

$$(1) \qquad \frac{S, (x = -\infty), x + \Sigma_{j\in J_1} y_j \geq \Sigma_{j\in J_2} x_j + \Sigma_{j\in J_3} y_j}{S, (x = -\infty), \Sigma_{j\in J_2} x_j = -\infty}$$

$$(2) \qquad \frac{S, (x = -\infty), (x + \Sigma_{j\in J} x_j = -\infty)}{S, (x = -\infty)}$$

$$(3) \qquad \frac{S, x \geq 0, \alpha \geq 1, (\alpha x + \Sigma_{j\in J} x_j = -\infty)}{S, x \geq 0, (\Sigma_{j\in J} x_j = -\infty)}$$

$$(4) \qquad \frac{S, (x' = -\infty), x + \Sigma_{j\in J_1} y_j \geq x' + \Sigma_{j\in J_2} x_j + \Sigma_{j\in J_3} y_j}{S, (x' = -\infty)}$$

$$(5) \qquad \frac{S, x'_k \geq 0 \text{ (for all } k \in K) \quad x + \Sigma_{j\in J} y_j \geq \Sigma_{k\in K} x'_k + \Sigma_{j\in J'} y_j}{S, x'_k \geq 0 \text{ (for all } k \in K), x \geq 0, x + \Sigma_{j\in J} y_j \geq \Sigma_{k\in K} x'_k + \Sigma_{j\in J'} y_j} \; .$$

To enforce a (quick) termination, rules (0) and (5) should be applied only if the constraint $x \geq 0$ is not already in the hypothesis and rule (3) should be applied by taking the factor α as large as possible. Let $\mathcal{S}'_1$ be the system resulting from the application of the rules above. Clearly an assignment satisfies the initial system iff it satisfies $\mathcal{S}'_1$. Let $X_1 = \{x \mid x = -\infty \text{ occurs in } \mathcal{S}'_1\}$ and $X_0 = \{x \mid x \geq 0 \text{ occurs in } \mathcal{S}'_1\}$. If $X_1 \cap X_0 \neq \emptyset$ then $0 = \infty$ occurs in $\mathcal{S}'_1$ and the initial system is not satisfiable.

Otherwise, let X_2 be composed of the variables that are neither in X_1 nor in X_0. The constraints in $\mathcal{S}'_1$ have one of the following forms:

$$(a) \quad y \geq 1, y \geq 0 \quad (b) \quad x \geq 0 \quad (c) \quad (x = \infty) \quad (d) \quad (\Sigma_{j\in J} x_j = -\infty) \text{ for } \sharp J \geq 2$$

$$(e) \quad x + \Sigma_{j\in J_1} y_j \geq \Sigma_{j\in J_2} x_j + \Sigma_{j\in J_3} y_j \; .$$

We note that in the constraint *(d)* it must be the case that $x_j \in X_2$ for all $j \in J$ (rules (2) and (3)), and that in the constraint *(e)* $x, x_j \notin X_1$ for all $j \in J_2$ (rules (1) and (4)). Now suppose the assignment ρ satisfies $\mathcal{S}'_1$ then we claim that ρ' defined by $\rho'(x) = \rho(x)$ if $x \in X_1 \cup X_0$ and $\rho'(x) = -\infty$ otherwise satisfies $\mathcal{S}'_1$. Indeed ρ' may behave differently from ρ only in the constraints of the shape *(d)* and *(e)*.

Since all the variables in the constraint *(d)* are in X_2, ρ' obviously satisfies this constraint. As for the constraint *(e)*: if $\exists\, j \in J_2 \ (x_j \in X_2)$ then ρ' satisfies the constraints. Otherwise, it must be that $\forall j \in J_2(x_j \in X_0)$ and then by rule (5) we know that $x \in X_0$ so that ρ' behaves as ρ on this constraint.

Thus the system $\mathcal{S}_2(\vec{x''})$ in the statement of the proposition can be obtained by restricting the system $\mathcal{S}'_1$ to the constraints that involve only the variables in X_0. □

B.8 Proof of theorem 21

NP-hardness follows from proposition 8. To establish that the problem can be solved in non-deterministic polynomial time we provide a rough upper bound to the size of the system of inequalities as a function of the size of the program. The size of a pattern p_i or of an expression e is defined as for values (definition 1). Let m be the size of the greatest pattern or expression. Let n be the maximum number of arguments of a function. Then $d = (n+1)m$ is an upper bound to the size of a rule. Note that by the hypothesis that the rules in the program have bounded size, d is bounded by some *constant*. Still, we will take d into account in the following to see how it affects the complexity.

Let r be the number of rules that compose a program and f be the number of functions in the program. Then $f \leq r$ and the size of the program is bound by rd. Let c be the number of constructors of positive arity. Clearly $c \leq rd$.

In the related synthesis problem, we have to determine at most $c + f2^n$ coefficients subject to a certain number of inequalities where we count the size of an inequality $u \geq v$ as the size of u plus the size of v. We have c inequalities of the form $a^c \geq 1$, at most fn inequalities of the form $a^f_i \geq 0$, at most $f2^{2n}$ inequalities of the shape $a^f_I \geq a^f_J$, and at most $f2^n$ inequalities of the shape $a^f_I = -\infty$. Hence the resulting system has a size in $O(f2^{2n})$.

It remains to determine the size of the system induced by the conditions $q_{f(p_1,\ldots,p_n)} \geq q_e$. The number of variables in the patterns is at most nm. Hence we have at most $r2^{nm} = r2^d$ inequalities of the shape $max\{b_V \mid V \supseteq U\} \geq b'_U$.

For each such inequality, we select non-deterministically the maximum on the left-hand side, say $b_{V'}$. Then we we have to bound the size of the coefficients $b_{V'}U$ and b'_U. The coefficient b_V is determined in proposition 11(2). We note that the multiplicative coefficient $\Sigma_{k\in K_V} \alpha_{k,j}$ is bound by nm and therefore it has size in $O(log(nm)) = O(log(d))$. It follows that the size of the coefficient $b_{V'}$ is in $O(c\ log(d))$. The form of the coefficient b'_U is determined in proposition 14. Let z_i denote an upper bound on the size of a coefficient b'_U for an expression of height i. Then $z_{i+1} \leq 2^n n z_i$. An expression e has size and hence height at most m, thus the size of b'_U is bound by $(2^n n)^m = 2^{nm} n^m \leq 2^{2nm}$. Assuming $c\ log(d) \leq 2^{2nm}$, we conclude that the size of the system is in $O(r2^{3d})$. We expect the factor 3 to be reducible but note that the mere fact that we try to determine a multi-linear polynomial with d indeterminates forces the resulting system to be exponential in d.

The last two steps presented in propositions 18 and 19 output a system whose size is polynomial in the size of the one in input and since the final system is composed of linear inequalities over $\mathbf{Q}^+$ it can be solved in polynomial time. $\square$