Max-plus quasi-interpretations

Roberto M. Amadio[1]

Laboratoire d'Informatique Fondamentale, Marseille
amadio@cmi.univ-mrs.fr

Abstract. 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.

1 Introduction

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). 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 (see, *e.g.*, [San01] for an elaboration of this point).

Cobham's characterization of polynomial time functions [Cob65] 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 an 'affine' type system with a *resource* type that guarantees that values obtained in the course of the computation are *non-size increasing*.

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. For polynomial interpretations even the verification problem has a high complexity.

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). We share our goal if not the technical approach with a recent work by Hofmann and Jost [HJ03] (see also [HPS96,Par00] for earlier work on the verification problem). In particular, quasi-interpretations offer a bound on the 'heap' memory needed to evaluate a program (cf. theorem 1).

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).

Most proofs are omitted and can be found in [Amadio02].

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 between constructors and function symbols.[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.

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 *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

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

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:

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

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

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:

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

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 practice, one may 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 equations $p_{i,1} = p_{j,1}, \ldots, p_{i,n} = p_{j,n}$ are not unifiable. In particular, this entails that the programs we consider are *deterministic*.

(3) $Var(e_k) \subseteq \bigcup_{j=1,\ldots,n} p_{k,j}$, that is expressions' variables are contained in patterns' variables.

(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 $x_1 = p_1, \ldots, x_n = p_n \Rightarrow e$.

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:

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

assuming the function f is defined as in section 2 and σ denotes a pattern-matching substitution.

[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.

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]

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

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 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 defined in [BMM01]. However, we work over the non-negative rationals rather than over the natural numbers and we force the interpretation 0 for constants. This last condition allows to simplify some interpretations by neglecting the space needed to store constant values. The following proposition summarizes the basic properties of quasi-interpretations.

Proposition 1. *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* $|v| \leq q_f(d|v_1|, \ldots, d|v_n|)$.

[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.

A point that seems to have gone unnoticed is that quasi-interpretations –by themselves– already provide a bound on the complexity of the program.

Theorem 1. *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 $2^{O(q_{f(v_1,\ldots,v_n)})}$.*

PROOF HINT. Let $B = q_{f(v_1,\ldots,v_n)}$. The size of the values computed during the computation of $f(v_1, \ldots, v_n)$ is in $O(B)$. It follows that the number of values to which a function can be applied in the course of the computation is in $2^{O(B)}$.

Let E be a 'call-by-value' evaluation context E defined as follows:

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

Any closed expression e which is not a value admits a unique decomposition in an evaluation context and a function application. We define an evaluation function *Eval* that performs an innermost leftmost evaluation of an expression.

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

where we assume that the function f is defined as in section 2 and that invoking $\mathit{Return}\perp$ stops the computation returning $\perp$ 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 $\mathit{max}(k, 1)$. It follows that also the size of the expressions involved in the evaluation has size in $O(B)$. Then 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 $O(B)$. Finally, by a well-known result by S. Cook [Coo71], the function can also be implemented to run on a Turing Machine in time $2^{O(B)}$ using a 'table' of size in $2^{O(B)}$ to store intermediate results (a so called *memoization* technique). $\square$

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:

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

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. Consequently, for α natural number, αx can be interpreted in the max-plus algebra as x raised to the exponent α.

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 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

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

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 $i = 1, \ldots, n$

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

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 5 (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 [Amadio02]. 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

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

$\mathbf{Q}^{+}_{max}$. 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.

Theorem 2. *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.*

PROOF HINT. This is based on a rather technical reduction from 3-SAT. As a first step, we produce rules forcing the quasi-interpretation of a function symbol f with n arguments to be $q_f = max(x_1, \ldots, x_n)$. If a non-uniform choice of constructors is allowed then we code literals as coefficients of unary constructors. Otherwise, some more work is needed to show that for a binary function symbol g one can force the quasi-interpretation $q_g = max(x_1 + a_1, x_2 + a_2)$. Then we use the coefficients a_1, a_2 to represent a boolean variable. $\square$

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:

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

The following is easily checked.

Proposition 2 (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*

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

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 3. *Let P_1 be a multi-linear polynomial and P_2 be a polynomial. If $P_1 \geq P_2$ then P_2 must be multi-linear.*

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.

Proposition 4 (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 $K_V = \{k \in \{1, \ldots, n\} \mid V \cap Var(p_k) \neq \emptyset\}$.

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 3 and 4(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.

Proposition 5 (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:*

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

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

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

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

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

Fig. 1. Constraints enforcing multi-linearity of q_e

(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.*

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.

Proposition 6 (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:*

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

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

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 2. The 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 4(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 seems 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 10 and 12.

We reserve $x_1, \ldots, x_n$ for the variables corresponding to the coefficients a_I^f 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 $\mathcal{S}(\boldsymbol{x}, \boldsymbol{y})$ 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 7 (right *max*-elimination). *The system $\mathcal{S}(\boldsymbol{x}, \boldsymbol{y})$ can be transformed in polynomial time into a system $\mathcal{S}_1(\boldsymbol{x}, \boldsymbol{x'}, \boldsymbol{y})$ over $\mathbf{Q}^+_{max}$ with additional auxiliary variables $\boldsymbol{x'}$ such that:*

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

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

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

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

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 $x'_{i,j} \geq B_{i,j}$. □

Proposition 8 ($-\infty$-elimination). *The system $\mathcal{S}_1(\boldsymbol{x}, \boldsymbol{x'}, \boldsymbol{y})$ over $\mathbf{Q}^+_{max}$ obtained in proposition 7 can be transformed in polynomial time into a system $\mathcal{S}_2(\boldsymbol{x''}, \boldsymbol{y})$ over $\mathbf{Q}^+$ where (i) $\{\boldsymbol{x''}\} \subset \{\boldsymbol{x}, \boldsymbol{x'}\}$, (ii) the constraints have the shape (b) and (c) in proposition 7, and assuming $\{\boldsymbol{z}\} = \{\boldsymbol{x}, \boldsymbol{x'}\} \backslash \{\boldsymbol{x''}\}$ an assignment ρ satisfies $\mathcal{S}_1$ iff $\rho[-\boldsymbol{\infty}/\boldsymbol{z}]$ 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 $\bot$ 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 $\bot \notin S'$ 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 $X_0 = \{x \mid (x \Rightarrow \bot) \in S'\}$. 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 $(x \Rightarrow \bot)$. $\Box$

Our method also applies to multi-linear polynomial over $\mathbf{N}_{max}$: if the system of inequalities admits a solution over $\mathbf{Q}^+$ then by multiplying by the least common denominator we obtain a solution in $\mathbf{N}$. Also, since we reduce to a linear programming problem, it is possible to look for *minimal* solutions with respect to a given linear cost function. We summarize below our analysis for programs whose rules have *bounded* size. In general, the method is exponential in the size of the rules since the number of coefficients we have to determine is exponential in the number of the variables in a rule. However, in practice one expects the size of the rules to be a slow growing function in the size of the program.

Theorem 3. *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 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 9. *A program conforming to Jones' restriction admits the following multi-linear quasi-interpretation (assuming $arity(\mathsf{c}) = arity(f) = n \geq 1$):*

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

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

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

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 9 falls in this family. We refer to this class of quasi-interpretations as *max*-multi-linear. The following is proven by noting that under the conditions (13) the *max* operation is not needed on the left-hand side of an inequality.

Proposition 10. *The synthesis problem over max-multi-linear interpretations can be solved in polynomial time.*

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 $r(f) < \mathit{arity}(f)$ be the number of arguments of resource type. As usual patterns and expressions in functions' definitions have to be well typed (cf. section 2). 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:

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

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:

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

Proposition 11. *If a program has an affine typing then its erasure admits the following multi-linear quasi-interpretation:*

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

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

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

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

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 11 falls in this family. We refer to this class of quasi-interpretations as *sum*-multi-linear. The proof strategy for the following proposition is the same as for proposition 10.

Proposition 12. *The synthesis problem over sum-multi-linear interpretations can be solved in polynomial time.*

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 1 that a program admitting a polynomial max-plus interpretation can 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 an exponential time Turing Machine by a linearly bounded auxiliary pushdown automaton in Cook's theorem [Coo71]).

The exponential upper bound derived in theorem 1 might be too loose in practice. To improve the situation, 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, one can program a checker for the validity of quantified boolean formulas 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 1.

7 Conclusion

Cobham's celebrated result [Cob65] is about primitive recursive functions with polynomial bounds on the size of the results. Here we have looked at general recursive functions with max-plus polynomial bounds on the size of the results. We have shown that the synthesis problem in the multi-linear case is NP-complete (when rules have bounded size). This case appears to be 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.

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

Acknowledgement 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.

References

[Amadio02] R. Amadio. Max-plus quasi-interpretations. Research Report *Laboratoire d'Informatique Fondamentale de Marseille* 10-2002, December 2002.

[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 *Ershov 4th Int. Conf.*, Springer LNCS, 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.

[Gra96] B. Gramlich. On proving termination by innermost termination. In *Proc. RTA*, Springer Lecture Notes in Comp. Sci. 1103, 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.

[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 *Proc. LPAR*, Springer Lecture Notes in Comp. Sci. 1955, 2000.

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