Characterizations of Polynomial Complexity Classes with a Better Intensionality

Jean-Yves Marion, Romain Péchoux

Characterizations of Polynomial Complexity Classes with a Better Intensionality

Jean-Yves Marion
École Nationale Supérieure des Mines de Nancy, INPL,
Nancy-Université, Loria, Carte team B.P. 239 54506
Vandœuvre-lès-Nancy Cedex, France
Jean-Yves.Marion@loria.fr

Romain Péchoux
École Supérieure d'Informatique et Applications de
Lorraine, UHP, Nancy-Université, Loria, Carte team
B.P. 239 54506 Vandœuvre-lès-Nancy Cedex, France
Romain.Pechoux@loria.fr

Abstract

In this paper, we study characterizations of polynomial complexity classes using first order functional programs and we try to improve their intensionality, that is the number of natural algorithms captured. We use polynomial assignments over the reals. The polynomial assignments used are inspired by the notions of quasi-interpretation and sup-interpretation, and are decidable when considering polynomials of bounded degree ranging over real numbers. Contrarily to quasi-interpretations, the considered assignments are not required to have the subterm property. Consequently, they capture a strictly larger number of natural algorithms (including quotient, gcd, duplicate elimination from a list) than previous characterizations using quasi-interpretations.

Categories and Subject Descriptors F.2.0 [*ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY*]: General

General Terms Theory, Verification

Keywords static analysis, resource upper bounds, quasi-interpretation, sup-interpretation

1. Introduction

The resource control is a fundamental issue for critical systems which is studied by the *implicit computational complexity* community in four distinct approaches that we briefly review. The first one deals with linear type disciplines in order to restrict computational time (Girard 1998; Lafont 2004; Baillot and Mogbil 2004; Gaboardi and Ronchi Della Rocca 2007; Gaboardi et al. 2008). The second approach (Hofmann 2002), introduced a resource atomic type, into linear type systems, for higher-order functional programming. The third one considers imperative programming languages in (Niggl and Wunderlich 2006; Kristiansen and Jones 2005) and complexity analysis of Java (Albert et al. a,b). The fourth approach is the one on which we focus in this paper which considers polynomial assignments, quasi-interpretations (Bonfante et al. 2007) and sup-interpretations (Marion and Péchoux 2006). It is related to polynomial interpretations (Lankford 1979).

One of the key challenges is to create complexity tools in order to analyze the computational complexity of programs automatically. In this paper, we focus on polynomial upper bounds on program outputs (quasi-interpretations and sup-interpretation). In order to predict the complexity of the function computed by a program, we first have to prove that it terminates and, then, to put drastic restrictions on either its syntax or its semantics. Nowadays, the motivation behind such an analysis is to increase the intensionality of program complexity static analysis, that is, to enlarge the number of captured algorithms. In particular, suitable characterizations of complexity classes have to include a broad set of programs which are relevant from a programmer perspective.

Polynomial assignments are strongly related to polynomial interpretations of (Lankford 1979; Manna and Ness 1970). However, we no longer consider polynomials over natural numbers but polynomials over real numbers using the work of (Dershowitz 1982). We compensate for the loss of well-foundedness over real numbers by putting restriction on the considered quasi-orderings. Polynomial assignments are also related to quasi-interpretations (Bonfante et al. 2007) and sup-interpretations (Marion and Péchoux 2006). These two notions provide upper bounds on the size of the values computed by a function symbol. It is demonstrated in (Bonfante et al. 2007) that these notions combined with the product extension and the lexicographic extension of Recursive Path Orderings (RPO) characterize the set of functions computable in polynomial time and, respectively, the set of functions computable in polynomial space.

In this paper, we make a step forward in the study of intensional computational complexity by giving new characterizations of the sets of functions computable in polynomial time and in polynomial space. These characterizations allow us to capture distinct natural algorithms (quotient, gcd, duplicate elimination from a list...) which where not captured by previous methods of (Bonfante et al. 2007). Moreover, we present characterizations having more flexiblity than the ones of (Bonfante et al. 2007) since they rely on polynomials over reals and without the subterm property (i.e. without $\forall i,\ P(\ldots, X_i, \ldots) \geq X_i$). As a consequence, the considered assignment have a stronger intensionality than quasi-interpretations which are subterm assignments.

Using a result of Tarski (Tarski 1951), we have demonstrated in (Bonfante et al. 2005), that the polynomial assignment synthesis, which consists in finding an assignment for a given program, is exponential in the number of variables. From a practical point of view, this is not a serious drawback since our study is related to program static analysis. Consequently, the search for suitable assignments can be performed offline and without time restrictions. Some heuristics for quasi-interpretation synthesis were already developed in a software called CROCUS, available at `http://libresource.inria.fr//projects/crocus`, which captures a lot of programs efficiently (about 60% of the TPDB pro-

PPDP'08, July 16–18, 2008, Valencia, Spain.

grams are analyzed in less than 1 second `http://www.loria.fr/~bonfante/crocus/tpdb.html`).

The paper is organized as follows. Section 2 describes the syntax and the semantics of the language and some preliminary notions on first order functional programs. Section 3 defines the polynomial assignments and the notion of quasi-interpretation. Section 4 improves the results of (Bonfante et al. 2007) by providing better intensional characterizations of the sets of functions computable in polynomial space and polynomial time using dependency pairs, quasi-interpretations and Recursive Path Orderings. Finally, section 5 provides two characterizations of the sets of functions computable in polynomial space and in polynomial time only using polynomial assignments without the subterm property. We illustrate by many examples that algorithms captured are distinct from the ones of section 4.

2. Preliminaries

2.1 Syntax

A program is defined formally as a quadruple $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$ with $\mathcal{X}$, $\mathcal{C}$ and $\mathcal{F}$ finite disjoint sets which represent respectively the variables, the constructor symbols and the function symbols and $\mathcal{R}$ a finite set of rules defined by:

$$\begin{array}{lll} \texttt{(Values)} & \mathcal{T}(\mathcal{C}) \ni v & ::= \mathbf{c} \mid \mathbf{c}(v_1, \cdots, v_n) \\ \texttt{(Terms)} & \mathcal{T}(\mathcal{C}, \mathcal{F}, \mathcal{X}) \ni t & ::= \mathbf{c} \mid x \mid \mathbf{c}(t_1, \cdots, t_n) \\ & & \mid \mathtt{f}(t_1, \cdots, t_n) \\ \texttt{(Patterns)} & \mathcal{P} \ni p & ::= \mathbf{c} \mid x \mid \mathbf{c}(p_1, \cdots, p_n) \\ \texttt{(Rules)} & \mathcal{R} \ni r & ::= \mathtt{f}(p_1, \cdots, p_n) \to t \end{array}$$

Throughout the paper, we suppose that $x \in \mathcal{X}$, $\mathtt{f} \in \mathcal{F}$, $\mathbf{c} \in \mathcal{C}$, $v, v_1, \cdots, v_n \in \mathcal{T}(\mathcal{C})$, $t, t_1, \cdots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{F}, \mathcal{X})$ and $p, p_1, \cdots, p_n \in \mathcal{P}$.

2.2 Call-by-value semantics

The domain of computation of a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$ is the constructor algebra $\mathcal{T}(\mathcal{C})$. A (normalized) substitution σ is a mapping from variables to values of $\mathcal{T}(\mathcal{C})$. We consider a call-by-value semantics which is displayed in figure 1. The meaning of $e \downarrow v$ is that e evaluates to the value $v \in \mathcal{T}(\mathcal{C})$. Throughout the paper, we only consider programs having disjoint and linear patterns. So each program is confluent (Huet 1980).

$$\frac{t_1 \downarrow w_1 \ldots t_n \downarrow w_n}{\mathbf{c}(t_1, \cdots, t_n) \downarrow \mathbf{c}(w_1, \cdots, w_n)}\ \mathbf{c} \in \mathcal{C}$$

$$\frac{e_i \downarrow w_i \quad \mathtt{f}(p_1, \ldots, p_n) \to e \in \mathcal{R} \quad \exists \sigma, p_i\sigma = w_i \quad e\sigma \downarrow u}{\mathtt{f}(e_1, \ldots, e_n) \downarrow u}\ \mathtt{f} \in \mathcal{F}$$

Figure 1. Call-by-value semantics of a program p

Given a term t and a substitution σ, if $t\sigma \downarrow w$ and w is in $\mathcal{T}(\mathcal{C})$ then $[\![t\sigma]\!] = w$, otherwise $[\![t\sigma]\!]$ is undefined and we set $[\![t\sigma]\!] = \bot$.

Because it is convenient, we will sometimes refer to $\xrightarrow{*}$ as the reflexive and transitive closure of the rewrite relation $\to$.

example 1. *Consider the following program which computes the quotient:*

$$\begin{array}{rl} \mathtt{minus}(\mathbf{0}, z) & \to \mathbf{0} \\ \mathtt{minus}(\mathbf{S}(z), \mathbf{0}) & \to \mathbf{S}(z) \\ \mathtt{minus}(\mathbf{S}(u), \mathbf{S}(v)) & \to \mathtt{minus}(u, v) \\ \mathtt{quo}(\mathbf{0}, \mathbf{S}(z)) & \to \mathbf{0} \\ \mathtt{quo}(\mathbf{S}(y), \mathbf{S}(z)) & \to \mathbf{S}(\mathtt{quo}(\mathtt{minus}(y, z), \mathbf{S}(z))) \end{array}$$

We have for every positive natural number m,

$$[\![\mathtt{quo}(\mathbf{S}^n(\mathbf{0}), \mathbf{S}^m(\mathbf{0}))]\!] = \mathbf{S}^{\lceil n/m \rceil}(\mathbf{0})$$

where $\mathbf{S}^{n+1}(\mathbf{0}) = \mathbf{S}(\mathbf{S}^n(\mathbf{0}))$ and $\mathbf{S}^0(\mathbf{0}) = \mathbf{0}$.

2.3 Call-tree

In this section, we define the notion of *call-tree* which will be used in the proofs of this paper to ensure that program computed values remain polynomially bounded.

A *context* is a term $\mathsf{C}[\Box_1, \cdots, \Box_r]$ with only one occurrence of each hole $\Box_i$, where the holes $\Box_i$ are fresh symbols. The substitution of each hole $\Box_i$ by a term d_i is denoted $\mathsf{C}[d_1, \cdots, d_r]$.

For example, if $\mathsf{C}[\Box] = \mathtt{minus}(\Box, \mathbf{S}(v))$ then $\mathsf{C}[\mathbf{S}(u)] = \mathtt{minus}(\mathbf{S}(u), \mathbf{S}(v))$.

definition 1 (State)**.** *A state $\langle \mathtt{f}, v_1, \cdots, v_n \rangle$ is a tuple where $\mathtt{f}$ is a function symbol of arity n and $v_1, \ldots, v_n$ are values.*

Assume that $\eta_1 = \langle \mathtt{f}, v_1, \cdots, v_n \rangle$ and $\eta_2 = \langle \mathtt{g}, u_1, \cdots, u_m \rangle$ are two states, there is a transition *between η_1 and η_2, denoted $\eta_1 \rightsquigarrow \eta_2$, if there are a rule of the shape $\mathtt{f}(p_1, \cdots, p_n) \to \mathsf{C}[\mathtt{g}(e_1, \cdots, e_m)] \in \mathcal{R}$, with $\mathsf{C}[\Box]$ a context, and a substitution σ such that:*

1. *$p_i\sigma = v_i$, for $i \in \{1, \ldots, n\}$,*
2. *$[\![e_j\sigma]\!] = u_j$, for $j \in \{1, \ldots, m\}$.*

Now, we define the notion of call-tree which corresponds to the tree of function calls in one execution of a program:

definition 2 (Call-tree)**.** *A call-tree corresponding to the evaluation of $\mathtt{f}(v_1, \ldots, v_n)$ is the finite or infinite tree whose nodes are labeled by the states, whose root is labeled by $\langle \mathtt{f}, v_1, \cdots, v_n \rangle$ and where the children of a node labeled by η_1 are the nodes labeled by η_2 such that $\eta_1 \rightsquigarrow \eta_2$.*

definition 3 (Path)**.** *A path is a finite or infinite set of nodes labeled by the states $\eta_1, \cdots, \eta_k$ such that $\eta_1 \rightsquigarrow \eta_2 \ldots \rightsquigarrow \eta_{k-1} \rightsquigarrow \eta_k$.*

definition 4 (Sub-tree)**.** *Given two call-trees T and T', T' is a sub-tree of T if every node of T' is a node of T and every transition of T' is a transition of T.*

Throughout the paper, we will refer to $\overset{+}{\rightsquigarrow}$ and $\overset{*}{\rightsquigarrow}$ as the transitive closure and as the transitive and reflexive closure of $\rightsquigarrow$.

definition 5 (Size and length)**.** *The size of a term t is defined by:*

$$|t| = 0 \qquad \text{if } t \text{ is a symbol of arity } 0$$

$$|t| = \sum_{i \in \{1, \ldots, n\}} |t_i| + 1 \qquad \text{if } t = b(t_1, \cdots, t_n),\ b \in \mathcal{C} \cup \mathcal{F}.$$

The size of a state $\langle \mathtt{f}, u_1, \cdots, u_n \rangle$ is defined by:

$$|\langle \mathtt{f}, u_1, \cdots, u_n \rangle| = \sum_{i \in \{1, \ldots, n\}} |u_i|$$

Given a finite path B of a call-tree, the length of the path $\mathtt{lg}(B)$ is the number of states in the path, i.e. if $B = \eta_1 \rightsquigarrow \ldots \rightsquigarrow \eta_{i-1} \rightsquigarrow \eta_i$, then $\mathtt{lg}(B) = i$, and the size of the path is the sum of the sizes of all its states, i.e.

$$|B| = \sum_{j \in \{1, \ldots, \mathtt{lg}(B)\}} |\eta_j|$$

Given a finite call-tree T, whose nodes are labeled by the states $s_1, \cdots, s_m$, the size of the call-tree is defined by

$$|T| = \sum_{j \in \{1, \ldots, m\}} |s_j|$$

A state may be seen as a stack frame since it contains a function call and its respective arguments. A call-tree of root $\langle \mathtt{f}, v_1, \cdots, v_n \rangle$

represents all the stack frames which will be pushed on the stack when we compute $\mathtt{f}(v_1, \ldots, v_n)$. Notice that both a path and a call-tree may be infinite if the program is not terminating.

2.4 Dependency pairs

In this section, we briefly review the notion of dependency pair introduced by Arts and Giesl (Arts and Giesl 2000) in order to analyze the termination of programs automatically. Since this paper does not focus on termination, the notions introduced differ slightly from the original dependency pair method. These deviations will be pointed out when necessary.

definition 6 (Dependency pair)**.** *Given a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$, a dependency pair (DP) is a couple*

$$(\mathtt{f}(p_1, \cdots, p_n), \mathtt{g}(e_1, \cdots, e_m))$$

satisfying $\mathtt{f}, \mathtt{g} \in \mathcal{F}$ and there is a context $\mathsf{C}[\Box]$ such that:

$$\mathtt{f}(p_1, \cdots, p_n) \to \mathsf{C}[\mathtt{g}(e_1, \cdots, e_m)] \in \mathcal{R}$$

example 2. *Consider the program of example 1: $(\mathtt{quo}(\mathbf{S}(y), \mathbf{S}(z)), \mathtt{minus}(y, z))$, is a dependency pair. Indeed, there is a rule $\mathtt{quo}(\mathbf{S}(y), \mathbf{S}(z)) \to \mathsf{C}[\mathtt{minus}(y, z)]$ with a context $\mathsf{C}[\Box] = \mathbf{S}(\mathtt{quo}(\Box, \mathbf{S}(z)))$.*

remark 1. *Contrarily to the original DP method, we do not distinguish a function symbol $\mathtt{f}$ from the tuple symbol $\mathtt{f}^{\#}$. Indeed there is no real need to distinguish the two symbols since we do not focus on termination.*

definition 7 (SDP graph)**.** *Given a program $\mathtt{p}$, we define its syntactic dependency pair graph (SDP graph) by:*

- *The nodes are the dependency pairs.*
- *There is an arc from (e, e') to (d, d') if and only if $e' = \mathtt{f}(e_1, \cdots, e_n)$ and $d = \mathtt{f}(p_1, \cdots, p_m)$.*

A cycle of dependency pairs is defined to be a cycle in the SDP graph. We say that the dependency pair is involved *in a cycle if it belongs to a cycle in the SDP graph.*

remark 2. *Contrarily to the original DP method, the notion of graph we consider is based on a syntactical condition (i.e. the uppermost symbols are the same) instead of a semantics condition (In the original paper, there is an arc from (e, e') to (d, d') if there is a substitution σ such that $e'\sigma \to^* d\sigma$). This restriction is made since we use a call-by-value strategy and we are more concerned in the decidability of our method than in its completeness.*

Now, we recall the notion of reduction pair of (Kusakari et al. 1999) and we suggest the new notion of strong reduction pair.

definition 8 (Strong reduction pair)**.** *A reduction pair is a couple $(\succsim, \succ)$ such that:*

- *$\succsim$ is a quasi-ordering (reflexive and transitive relation) weakly monotonic (i.e. $\forall b\ s \succsim t$ implies $b(\ldots, s, \ldots) \succsim b(\ldots, t, \ldots)$) and stable by substitution (i.e. $\forall \sigma,\ s \succsim t$ implies $s\sigma \succsim t\sigma$),*
- *and $\succ$ is a well-founded ordering stable by substitution,*

which satisfy $\succ \circ \succsim \subseteq \succ$ or $\succsim \circ \succ \subseteq \succ$.
A strong reduction pair *is a reduction pair $(\succsim, \succ)$ which satisfies both $\succ \circ \succsim \subseteq \succ$ and $\succsim \circ \succ \subseteq \succ$.*

remark 3. *Reduction pair seems to be a too weak notion for program complexity analysis even if it is a suitable notion for showing program termination. To understand the intuition behind strong reduction pairs, notice that if an infinite decreasing sequence of the shape $u_1 \succsim u_2 \succ u_3 \succsim u_4 \succsim \ldots$ involves an infinite number of occurrences of $\succ$ then the notion of reduction pair is powerful enough in order to extract an infinite decreasing sub-sequence with respect to $\succ$. This contradicts the fact that $\succ$ is well-founded and consequently there is no such infinite sequence and the program is terminating. When considering complexity, the extraction of a sub-sequence is not sufficient, we want to keep "upper-bounds" on each expression of a sequence and this requirement justifies the introduction of strong reduction pairs.*

definition 9. *Given a reduction pair $(\succsim, \succ)$, a program $\mathtt{p}$ is in $DP(\succsim, \succ)$ if:*

1. *For each rule $\mathtt{f}(p_1, \cdots, p_n) \to e \in \mathcal{R}$, $\mathtt{f}(p_1, \cdots, p_n) \succsim e$.*
2. *For each dependency pair (s, t), $s \succsim t$.*
3. *For each cycle C of the SDP graph, there is a dependency pair (s, t) involved in C such that $s \succ t$*

theorem 1. *A program $\mathtt{p} \in DP(\succsim, \succ)$ is terminating.*

Proof. This result is a consequence of (Arts and Giesl 2000). □

3. Assignments

In this section, we define max-polynomial assignments over non negative real numbers following the terminology of (Dershowitz 1982). A max-polynomial assignment associates a max-polynomial function to every symbol of a program. Throughout the paper, $\geq$ denotes the natural ordering on real numbers.

definition 10 (Assignment)**.** *An assignment of a symbol $b \in \mathcal{F} \cup \mathcal{C}$ of arity n is a function $(\!|b|\!) : (\mathbb{R}_{\geq 0})^n \to \mathbb{R}_{\geq 0}$.*

For each variable $x \in \mathcal{X}$, we define $(\!|x|\!) = X$ with X a fresh variable ranging over $\mathbb{R}_{\geq 0}$.

We extend assignments $(\!|-|\!)$ to terms canonically. Given a term $b(t_1, \cdots, t_n)$ with m variables, its assignment is a function $(\mathbb{R}_{\geq 0})^m \to \mathbb{R}_{\geq 0}$ defined by:

$$(\!|b(t_1, \cdots, t_n)|\!) = (\!|b|\!)((\!|t_1|\!), \cdots, (\!|t_n|\!))$$

A program assignment is an assignment $(\!|-|\!)$ defined for each symbol of the program.

example 3. *The function $(\!|-|\!)$ defined by $(\!|\mathtt{quo}|\!)(U, V) = U + V$, $(\!|\mathtt{minus}|\!)(U, V) = U + V$, $(\!|\mathbf{0}|\!) = 0$ and $(\!|\mathbf{S}|\!)(U, V) = U + V + 1$ is an assignment of the program of example 1.*

Now we define the notion of additive assignments which guarantees that the assignment of a value remains affinely bounded by its size.

definition 11 (Additive assignment)**.** *An assignment of a symbol $\mathbf{c}$ of arity n is additive if:*

$$(\!|\mathbf{c}|\!)(X_1, \cdots, X_n) = \sum_{i=1}^{n} X_i + \alpha_{\mathbf{c}}, \text{ with } \alpha_{\mathbf{c}} \geq 1 \quad \text{if } n > 0,$$
$$(\!|\mathbf{c}|\!) = 0 \quad \text{otherwise.}$$

The assignment $(\!|-|\!)$ of a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$ is called additive *assignment if each constructor symbol of $\mathcal{C}$ has an additive assignment.*

lemma 1. *Given an additive assignment $(\!|-|\!)$, there is a constant α such that for each value v, the following inequality is satisfied:*

$$|v| \leq (\!|v|\!) \leq \alpha \times |v|$$

Proof. Define $\alpha = \max_{\mathbf{c} \in \mathcal{C}}(\alpha_{\mathbf{c}})$ where $\alpha_{\mathbf{c}}$ is taken to be the constant of definition 11, if $\mathbf{c}$ is of strictly positive arity, and $\alpha_{\mathbf{c}} = 0$ otherwise. If v is a constructor symbol of arity 0 then $|v| = (\!|v|\!) = 0$, else $v = \mathbf{c}(v_1, \cdots, v_n)$ and we show the result by induction on

the size. Suppose that $|v_i| \leq (\!|v_i|\!) \leq \alpha \times |v_i|$ (I.H.):

$$\begin{aligned}|\mathbf{c}(v_1, \cdots, v_n)| &= \sum_{i \in \{1,\ldots,n\}} |v_i| + 1 \\ &\leq \sum_{i \in \{1,\ldots,n\}} (\!|v_i|\!) + \alpha_{\mathbf{c}} = (\!|\mathbf{c}(v_1, \cdots, v_n)|\!) \\ &\leq \sum_{i \in \{1,\ldots,n\}} \alpha \times |v_i| + \alpha = \alpha \times |\mathbf{c}(v_1, \cdots, v_n)|\end{aligned}$$

The first inequality is a consequence of the combination of I.H. and the fact that $\alpha_{\mathbf{c}} \geq 1$. The second inequality is a consequence of the combination of I.H. and the fact that $\alpha_{\mathbf{c}} \leq \alpha$. □

definition 12 (Monotonic assignment)**.** *An assignment is (weakly)* monotonic *if for any symbol b, $(\!|b|\!)$ is an increasing (not necessarily strictly) function with respect to each variable. That is, for every symbol b and all X_i, Y_i of $\mathbb{R}_{\geq 0}$ such that $X_i \geq Y_i$, we have:*

$$(\!|b|\!)(\ldots, X_i, \ldots) \geq (\!|b|\!)(\ldots, Y_i, \ldots)$$

We start by showing some properties on monotonic assignments:

proposition 1. *Given a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$ and a monotonic assignment $(\!|-|\!)$ such that $\forall l \to r \in \mathcal{R},\ (\!|l|\!) \geq (\!|r|\!)$, then for every term t and every substitution σ if $t\sigma \to^* u$, we have: $(\!|t\sigma|\!) \geq (\!|u|\!)$*

Proof. Given a program $\mathtt{p}$ which admits an additive, max-polynomial and monotonic assignment $(\!|-|\!)$ such that $\forall l \to r \in \mathcal{R},\ (\!|l|\!) \geq (\!|r|\!)$, we show this result by induction on the derivation length. Notice that $(\!|l|\!) \geq (\!|r|\!)$ implies $\forall \sigma,\ (\!|l\sigma|\!) \geq (\!|r\sigma|\!)$ (closure by substitution). Consider a term t and suppose, by Induction Hypothesis (I.H.), that if $t\sigma \to^i u$, where $\to^i$ denotes i rewrite steps, then $(\!|t\sigma|\!) \geq (\!|u|\!)$. The inequality is clearly satisfied for $i = 0$. Now, take a reduction of length $i + 1$ of the shape $t\sigma \to^i u \to w$. We know that there are a context $\mathsf{C}[-]$, a rewrite rule $l \to r$ and a substitution σ' such that $u = \mathsf{C}[l\sigma']$ and $w = \mathsf{C}[r\sigma']$, so:

$$\begin{aligned}(\!|t\sigma|\!) &\geq (\!|u|\!) && \text{By I.H.} \\ &= (\!|\mathsf{C}[l\sigma']|\!) && \text{Since } u = \mathsf{C}[l\sigma'] \\ &= (\!|\mathsf{C}|\!)((\!|l\sigma'|\!)) && \text{By definition of } (\!|-|\!) \\ &\geq (\!|\mathsf{C}|\!)((\!|r\sigma'|\!)) && \text{By monotonicity of } (\!|-|\!) \\ &= (\!|w|\!) && \text{By definition of } (\!|-|\!) \text{ again}\end{aligned}$$

□

definition 13 (Subterm assignment)**.** *An assignment is* subterm *if for any symbol b of arity $n > 0$, we have*

$$\forall i \in \{1, \ldots, n\}, (\!|b|\!)(\ldots, X_i, \ldots) \geq X_i$$

definition 14 (Max-poly)**.** *Let **Max-Poly**$\{\mathbb{R}_{\geq 0}\}$ be the smallest class which contains the constant functions over $\mathbb{R}_{\geq 0}$, the projections, $\max$, $+$, $\times$ and which is closed under composition. An assignment $(\!|-|\!)$ is said to be max-polynomial if for each symbol $b \in \mathcal{F} \cup \mathcal{C}$, $(\!|b|\!)$ is a function in **Max-Poly**$\{\mathbb{R}_{\geq 0}\}$.*

example 4. *The assignment of example 3 is monotonic, subterm, max-polynomial and additive.*

definition 15 (Quasi-interpretation (QI))**.** *Given a program $\mathtt{p}$, $\mathtt{p}$ admits a quasi-interpretation $(\!|-|\!)$ if $(\!|-|\!)$ is a* monotonic *and* subterm *assignment which satisfies:*

$$\forall l \to r \in \mathcal{R},\ (\!|l|\!) \geq (\!|r|\!)$$

definition 16 ($QI^{\text{add}}_{\text{poly}}$)**.** *Let $QI^{\text{add}}_{\text{poly}}$ be the set of programs which admit an additive and max-polynomial quasi-interpretation.*

example 5. *The following program:*

$$\begin{aligned}\mathtt{minus}(\mathbf{0}, z) &\to \mathbf{0} \\ \mathtt{minus}(\mathbf{S}(z), \mathbf{0}) &\to \mathbf{S}(z) \\ \mathtt{minus}(\mathbf{S}(u), \mathbf{S}(v)) &\to \mathtt{minus}(u, v)\end{aligned}$$

admits the additive and max-polynomial quasi-interpretation $(\!|-|\!)$ defined by $(\!|\mathbf{S}|\!)(U, V) = U + V + 1$, $(\!|\mathtt{minus}|\!)(U, V) = U + V$, and $(\!|\mathbf{0}|\!) = 0$. Indeed, we check that:

$$\begin{aligned}(\!|\mathtt{minus}(\mathbf{0}, z)|\!) &= Z \geq 0 = (\!|\mathbf{0}|\!) \\ (\!|\mathtt{minus}(\mathbf{S}(z), \mathbf{0})|\!) &= Z + 1 \geq Z + 1 = (\!|\mathbf{S}(z)|\!) \\ (\!|\mathtt{minus}(\mathbf{S}(u), \mathbf{S}(v))|\!) &= U + V + 2 \geq U + V = (\!|\mathtt{minus}(u, v)|\!)\end{aligned}$$

Consequently, this program belongs to QI^{add}_{poly}.

However there are natural programs that do not admit any (additive and) polynomial quasi-interpretation because of the subterm property:

example 6. *The program of example 1 does not have any quasi-interpretation. Ad absurdum, suppose that it admits an additive quasi-interpretation $(\!|-|\!)$. In particular, we have to check that $(\!|\mathtt{quo}(\mathbf{S}(y), \mathbf{S}(z))|\!) \geq (\!|\mathbf{S}(\mathtt{quo}(\mathtt{minus}(y, z), \mathbf{S}(z)))|\!)$. Suppose that $(\!|\mathbf{S}|\!)(X) = X + k$, with $k \geq 1$. We have to find $(\!|\mathtt{quo}|\!)$ such that:*

$$\begin{aligned}(\!|\mathtt{quo}|\!)(Y + k, Z + k) &\geq (\!|\mathtt{quo}|\!)((\!|\mathtt{minus}(y, z)|\!), Z + k) + k \\ &> (\!|\mathtt{quo}|\!)(\max(Y, Z), Z + k)\end{aligned}$$

The last strict inequality holds by subterm property. For Z large enough (take $Z > Y + k$), we obtain a contradiction since $(\!|\mathtt{quo}|\!)(Y + k, Z + k) > (\!|\mathtt{quo}|\!)(Y + k, Z + k)$.

4. Characterizations using RPO and QI

In this section, we provide characterizations of the sets of functions computable in polynomial time and polynomial space with the help of recursive path orderings (RPO) and quasi-interpretations. These characterizations improve the intensionality of the ones presented in (Bonfante et al. 2007) since the set of programs captured by (Bonfante et al. 2007) is a proper subset of the characterizations presented in this section (as illustrated by example 8).

4.1 Recursive Path Orderings

Given a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$ and a quasi-ordering $\geq_{\mathcal{F}}$ on $\mathcal{F}$, we define $\approx_{\mathcal{F}}$ by, $\mathtt{f} \approx_{\mathcal{F}} \mathtt{g}$ iff $\mathtt{f} \geq_{\mathcal{F}} \mathtt{g}$ and $\mathtt{g} \geq_{\mathcal{F}} \mathtt{f}$, and $>_{\mathcal{F}}$ by, $\mathtt{f} >_{\mathcal{F}} \mathtt{g}$ iff $\mathtt{f} \geq_{\mathcal{F}} \mathtt{g}$ and not $\mathtt{g} \geq_{\mathcal{F}} \mathtt{f}$.

We associate to each function symbol $\mathtt{f}$ a status $st(\mathtt{f})$ which ranges over $\{p, l\}$ and which satisfies the condition $st(\mathtt{f}) = st(\mathtt{g})$ if $\mathtt{f} \approx_{\mathcal{F}} \mathtt{g}$ holds.

definition 17. *The product extension $\succ^p$ and lexicographic extension $\succ^l$ of a partial ordering $\succeq$ and its strict part $\succ$ are defined by:*

- *Product extension: $\{m_1, \cdots, m_k\} \succ^p \{n_1, \cdots, n_k\}$ iff (i) $\forall i \leq k, m_i \succeq n_i$ and (ii) $\exists j \leq k$ such that $m_j \succ n_j$.*
- *Lexicographic extension: $\{m_1, \cdots, m_k\} \succ^l \{n_1, \cdots, n_l\}$ iff $\exists j$ such that $\forall i < j,\ m_i \succeq n_i$ and $m_j \succ n_j$.*

definition 18 (RPO)**.** *Given a preorder $\geq_{\mathcal{F}}$ and a status st, the Recursive Path Ordering $\succ_{rpo}$ is defined by the rules of figure 1.*

$\succ^{prod}_{rpo}$ is the restriction of $\succ_{rpo}$ where each function symbol has the product status, i.e.$\forall \mathtt{f} \in \mathcal{F},\ st(\mathtt{f}) = p$. We define $\succeq_{rpo}$ and $\succeq^{prod}_{rpo}$ to be the reflexive closures of $\succ_{rpo}$ and, respectively, $\succ^{prod}_{rpo}$ (i.e. $t \succeq_{rpo} s$ iff $t = s$ or $t \succ_{rpo} s$).

Notice that $(\succeq_{rpo}, \succ_{rpo})$ and, a fortiori, $(\succeq^{\text{prod}}_{rpo}, \succ^{\text{prod}}_{rpo})$ are strong reduction pairs since $\succ_{rpo}$ is a well-founded, stable and

$$\frac{\{t_1,\cdots,t_n\} \succ_{rpo}^{p} \{t'_1,\cdots,t'_n\}}{\mathbf{c}(t_1,\cdots,t_n) \succ_{rpo} \mathbf{c}(t'_1,\cdots,t'_n)}\,\mathbf{c}\in\mathcal{C} \qquad \frac{t' = t_i \text{ or } t_i \succ_{rpo} t'}{b(\ldots,t_i,\ldots) \succ_{rpo} t'}\,b\in\mathcal{C}\cup\mathcal{F}$$

$$\frac{\forall i\ \mathtt{f}(t_1,\cdots,t_n) \succ_{rpo} t'_i}{\mathtt{f}(t_1,\cdots,t_n) \succ_{rpo} \mathbf{c}(t'_1,\cdots,t'_m)}\,\mathbf{c}\in\mathcal{C},\ \mathtt{f}\in\mathcal{F} \qquad \frac{\forall i\ \mathtt{f}(t_1,\cdots,t_n) \succ_{rpo} t'_i \qquad \mathtt{f} >_{\mathcal{F}} \mathtt{g}}{\mathtt{f}(t_1,\cdots,t_n) \succ_{rpo} \mathtt{g}(t'_1,\cdots,t'_m)}\,\mathtt{g},\mathtt{f}\in\mathcal{F}$$

$$\frac{\{t_1,\cdots,t_n\} \succ_{rpo}^{st(\mathtt{f})} \{t'_1,\cdots,t'_n\} \qquad \mathtt{f}\approx_{\mathcal{F}}\mathtt{g} \qquad \forall i\ \mathtt{f}(t_1,\cdots,t_n) \succ_{rpo} t'_i}{\mathtt{f}(t_1,\cdots,t_n) \succ_{rpo} \mathtt{g}(t'_1,\cdots,t'_n)}\,\mathtt{g},\mathtt{f}\in\mathcal{F}$$

Figure 2. Definition of $\succ_{rpo}$

monotonic strict ordering (Dershowitz 1982; Kamin and Lévy 1980).

4.2 Properties of $DP(\succsim,\succ_{rpo}) \cap QI^{\mathbf{add}}_{\mathbf{poly}}$

If we consider programs in QI^{add}_{poly} the size of every computed value is polynomially bounded by the input size:

lemma 2. *Given a program $\langle\mathcal{X},\mathcal{C},\mathcal{F},\mathcal{R}\rangle \in QI^{\text{add}}_{\text{poly}}$ and a function symbol $\mathtt{f}\in\mathcal{F}$, there is a polynomial P such that for every state η of a call-tree of root $\langle\mathtt{f},v_1,\cdots,v_n\rangle$, we have:*

$$|\eta| \le P(|\langle\mathtt{f},v_1,\cdots,v_n\rangle|)$$

Proof. Given a node $\langle\mathtt{g},s_1,\cdots,s_m\rangle$ of the call-tree of root $\langle\mathtt{f},v_1,\cdots,v_n\rangle$, we know that there is a derivation of the shape $\mathtt{f}(v_1,\cdots,v_n) \to^* \mathsf{C}[\mathtt{g}(s_1,\cdots,s_m)]$, for some context $\mathsf{C}[\Box]$. By proposition 1, $(\!|\mathtt{f}(v_1,\cdots,v_n)|\!) \ge (\!|\mathsf{C}[\mathtt{g}(s_1,\cdots,s_m)]|\!)$ and, by subterm property, we obtain:

$$(\!|\mathtt{f}(v_1,\cdots,v_n)|\!) \ge \max_{j\in\{1,\ldots,m\}} (\!|s_j|\!)$$

If k is the maximal arity of a symbol, applying lemma 1, then $P(X) = k\times(\!|\mathtt{f}|\!)(\alpha\times X,\ldots,\alpha\times X)$ is the required polynomial. □

We can extend this result to paths of every call-tree when considering programs of $DP(\succsim,\succ_{rpo})\cap QI^{add}_{poly}$, where $(\succsim,\succ_{rpo})$ is a strong reduction pair. By theorem 1, every program of $DP(\succsim,\succ_{rpo})$ is terminating and, consequently, has only finite call-trees. For that purpose, we first establish an intermediate lemma which uses the properties of strong reduction pairs:

lemma 3. *Assume that $\succsim$ is a relation s.t. $(\succsim,\succ_{rpo})$ is a strong reduction pair.*
Given a program $\mathtt{p}\in DP(\succsim,\succ_{rpo})$ and a call-tree corresponding to one execution of this program and containing a path of the shape $\langle\mathtt{f},v_1,\cdots,v_n\rangle \overset{+}{\rightsquigarrow} \langle\mathtt{f},u_1,\cdots,u_n\rangle$, we have:

$$\mathtt{f}(v_1,\cdots,v_n) \succ_{rpo} \mathtt{f}(u_1,\cdots,u_n)$$

Proof. Since the path corresponds to a cycle in the SDP graph, by definitions 9 and 2, there are two consecutive states $\langle\mathtt{h},v'_1,\cdots,v'_l\rangle$ and $\langle\mathtt{g},s_1,\cdots,s_m\rangle$ of the path, a rule of the shape $\mathtt{h}(p_1,\cdots,p_l)\to\mathsf{C}[\mathtt{g}(e_1,\cdots,e_m)]\in\mathcal{R}$ and a substitution σ such that:
$\mathtt{h}(p_1,\cdots,p_l)\sigma \succ_{rpo} \mathtt{g}(e_1,\cdots,e_m)\sigma$, $p_i\sigma = v'_i$ and $[\![e_j\sigma]\!] = s_j$. Moreover, it forces $\mathtt{h}\approx_{\mathcal{F}}\mathtt{g}\approx_{\mathcal{F}}\mathtt{f}$, $st(\mathtt{h}) = st(\mathtt{g}) = st(\mathtt{f})$ and $l=m=n$. Consequently,

$$\{p_1,\cdots,p_n\} \succ_{rpo}^{st(\mathtt{h})} \{e_1,\cdots,e_n\}$$

We let the reader check that, for any substitution σ:

$$\{p_1\sigma,\ldots,p_n\sigma\} \succ_{rpo}^{st(\mathtt{h})} \{e_1\sigma,\ldots,e_n\sigma\} \Rightarrow$$
$$\{p_1\sigma,\ldots,p_n\sigma\} \succ_{rpo}^{st(\mathtt{h})} \{[\![e_1\sigma]\!],\ldots,[\![e_n\sigma]\!]\}$$

Finally, we obtain that $\mathtt{h}(v'_1,\cdots,v'_n) \succ_{rpo} \mathtt{g}(s_1,\cdots,s_n)$.

By definitions 2 and 9, for every transition $\langle\mathtt{h},v'_1,\cdots,v'_n\rangle \rightsquigarrow \langle\mathtt{g},s_1,\cdots,s_n\rangle$ of the path, there are a rule $\mathtt{h}(p_1,\cdots,p_n)\to\mathsf{C}[\mathtt{g}(e_1,\cdots,e_n)]$ and a substitution σ s.t. $\mathtt{h}(p_1,\cdots,p_n)\sigma \succsim \mathtt{g}(e_1,\cdots,e_n)\sigma$, $p_i\sigma = v'_i$ and $[\![e_j\sigma]\!] = s_j$. Thus, we have:

$$\begin{aligned}
\mathtt{h}(v'_1,\cdots,v'_n) &= \mathtt{h}(p_1,\cdots,p_n)\sigma \\
&\succsim \mathtt{g}(e_1\sigma,\ldots,e_n\sigma) && \succsim \text{ is stable}\\
&\succsim \mathtt{g}([\![e_1\sigma]\!],\ldots,[\![e_n\sigma]\!]) && \text{Since } \succsim \text{ is monotonic}\\
&= \mathtt{g}(s_1,\cdots,s_n) && \text{Since } [\![e_j\sigma]\!] = s_j
\end{aligned}$$

There are two states $\langle\mathtt{h},v'_1,\cdots,v'_n\rangle$ and $\langle\mathtt{g},s_1,\cdots,s_n\rangle$ in the path $\langle\mathtt{f},v_1,\cdots,v_n\rangle \overset{*}{\rightsquigarrow} \langle\mathtt{h},v'_1,\cdots,v'_n\rangle \rightsquigarrow \langle\mathtt{g},s_1,\cdots,s_n\rangle \overset{*}{\rightsquigarrow} \langle\mathtt{f},u_1,\cdots,u_n\rangle$ such that the following inequalities hold:

$$\begin{aligned}
\mathtt{f}(v_1,\cdots,v_n) &\succsim \ldots\\
&\succsim \mathtt{h}(v'_1,\cdots,v'_n)\\
&\succ_{rpo} \mathtt{g}(s_1,\cdots,s_n)\\
&\succsim \ldots\\
&\succsim \mathtt{f}(u_1,\cdots,u_n)
\end{aligned}$$

Since $(\succsim,\succ_{rpo})$ is a strong reduction pair, the inequalities collapse into $\mathtt{f}(v_1,\cdots,v_n)\succ_{rpo}\mathtt{f}(u_1,\cdots,u_n)$. □

lemma 4. *Assume that $\succsim$ is a relation s.t. $(\succsim,\succ_{rpo})$ is a strong reduction pair.*
Given a program $\langle\mathcal{X},\mathcal{C},\mathcal{F},\mathcal{R}\rangle \in DP(\succsim,\succ_{rpo})\cap QI^{\text{add}}_{\text{poly}}$ and a function symbol $\mathtt{f}\in\mathcal{F}$, there is a polynomial P such that for every path B of a call-tree of root $\langle\mathtt{f},v_1,\cdots,v_n\rangle$, we have:

$$|B| \le P(|\langle\mathtt{f},v_1,\cdots,v_n\rangle|)$$

Proof. By lemma 3, we know that for each path of the shape $\langle\mathtt{h},v'_1,\cdots,v'_l\rangle \overset{+}{\rightsquigarrow} \langle\mathtt{h},w_1,\cdots,w_l\rangle$, we have

$$\mathtt{h}(v'_1,\cdots,v'_l) \succ_{rpo} \mathtt{h}(w_1,\cdots,w_l)$$
$$\{v'_1,\cdots,v'_l\} \succ_{rpo}^{st(\mathtt{h})} \{w_1,\cdots,w_l\}$$

Notice that $\{v'_1,\cdots,v'_l\} \succ_{rpo}^{st(\mathtt{h})} \{w_1,\cdots,w_l\}$ implies that $\{v'_1,\cdots,v'_l\} \rhd^{st(\mathtt{h})} \{w_1,\cdots,w_l\}$, where $\rhd$ is the homeomorphic embedding on terms. Consequently, there are at most $\prod_{1\le i\le l}|v'_i|$ states corresponding to the function symbol $\mathtt{h}$ in every path of the call-tree.

We know by lemma 2 that there is a polynomial R such that, for every state η, $|\eta| \leq R(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|)$. Consequently, if k is the maximal arity of a symbol then we have at most $\prod_{j\in\{1,\ldots,m\}} |s_i| \leq (R(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|))^k$ states corresponding to the function symbol $\mathtt{g}$ in a path starting from $\langle \mathtt{g}, s_1, \cdots, s_m\rangle$. If $\#\mathcal{F}$ is the cardinal of the set $\mathcal{F}$, we have $\mathtt{lg}(B) \leq \#\mathcal{F} \times (R(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|))^k$, for every path B of the call-tree. Finally, $\#\mathcal{F} \times (R(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|))^k \times k \times R(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|)$ is an upper bound on the size of every path of the call-tree, and we set $P'(X) = \#\mathcal{F}\times k\times (R(X))^{k+1}$. Since P' is in **Max-Poly** $\{\mathbb{R}_{\geq 0}\}$, there is a polynomial P such that $\forall X,\ P(X) \geq P'(X)$. □

lemma 5. *Assume that $\succsim$ is a relation s.t. $(\succsim, \succ_{rpo}^{prod})$ is a strong reduction pair. Given a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}\rangle \in DP(\succsim, \succ_{rpo}^{prod}) \cap QI_{\text{poly}}^{\text{add}}$ and a function symbol $\mathtt{f} \in \mathcal{F}$, there is a polynomial P such that for every sub-tree T of a call-tree of root $\langle \mathtt{f}, v_1, \cdots, v_n\rangle$ having pairwise distinct states, we have:*

$$|T| \leq P(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|)$$

Proof. We have demonstrated in the proof of lemma 4 that, for each path $\langle \mathtt{f}, v_1, \cdots, v_n\rangle \overset{+}{\rightsquigarrow} \langle \mathtt{f}, u_1, \cdots, u_n\rangle$, we have

$$\{v_1, \cdots, v_n\} \rhd^{st(\mathtt{f})} \{u_1, \cdots, u_n\}$$

where $\rhd$ is the homeomorphic embedding on terms.

Since $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}\rangle \in DP(\succsim, \succ_{rpo}^{\text{prod}})$, we have $st(\mathtt{f}) = p$ and there at most $\prod_{i\in\{1,\ldots,n\}} |v_i|$ distinct states corresponding to the function symbol $\mathtt{f}$ in a path of the call-tree. As a consequence, there are at most $\prod_{i\in\{1,\ldots,n\}} |v_i|$ states corresponding to the function symbol $\mathtt{f}$ in every sub-tree T having pairwise distinct states. Following the reasoning of the proof of lemma 4, we obtain that $\#\mathcal{F} \times k \times (R(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|))^{k+1}$ is an upper bound on the size of T, for some polynomial R, and we set $P'(X) = \#\mathcal{F}\times k\times (R(X))^{k+1}$. Since P' is in **Max-Poly** $\{\mathbb{R}_{\geq 0}\}$, there is a polynomial P such that $\forall X,\ P(X) \geq P'(X)$. □

theorem 2. *Given a relation $\succsim$ such that $(\succsim, \succ_{rpo})$ is a strong reduction pair, the set of functions computed by programs of:*

1. $DP(\succsim, \succ_{rpo}^{prod}) \cap QI_{\text{poly}}^{\text{add}}$ *is exactly the set of functions* FPTIME
2. $DP(\succsim, \succ_{rpo}) \cap QI_{\text{poly}}^{\text{add}}$ *is exactly the set of functions* FPSPACE

Proof. (1) By lemma 5, we have shown that the size of every subtree of the call-tree, whose states are pairwise distinct, is polynomially bounded by the input size. We evaluate the program in polynomial time using a call-by-value interpreter with cache. This technique implies that we never evaluate the same function call twice. Conversely, we simulate a polynomial time Register Machine as in (Bonfante et al. 2007). (2) By lemma 4, we know that the size of each path of the call-tree is polynomially bounded by the input size. Evaluating the program in the depth of the call-tree, we obtain that $DP(\succsim, \succ_{rpo}) \cap QI_{poly}^{add}$ is included in FPSPACE. Conversely, we simulate a polynomial time Parallel Register Machine as in (Bonfante et al. 2007). □

In particular, we obtain the characterizations of (Bonfante et al. 2007) as a corollary:

corollary 1 ((Bonfante et al. 2007))**.** *The set of functions computed by programs of:*

1. $DP(\succsim_{rpo}^{prod}, \succ_{rpo}^{prod}) \cap QI_{\text{poly}}^{\text{add}}$ *is exactly the set* FPTIME
2. $DP(\succsim_{rpo}, \succ_{rpo}) \cap QI_{\text{poly}}^{\text{add}}$ *is exactly the set* FPSPACE

example 7. *The program of example 5 is in $DP(\succsim_{rpo}^{prod}, \succ_{rpo}^{prod}) \cap QI_{\text{poly}}^{\text{add}}$ by taking the additive and polynomial quasi-interpretation provided in example 5.*

Notice that the characterizations of corollary 1 are strictly included in the ones of theorem 2 as illustrated by the following example:

example 8. *Define $\mathtt{f}(p_1, \cdots, p_n) \succsim \mathtt{g}(e_1, \cdots, e_n)$ by:*

$$(\mathtt{f}(p_1, \cdots, p_n) \succ_{rpo} \mathtt{g}(e_1, \cdots, e_n))$$
$$\vee\ ((\mathtt{f} \approx_{\mathcal{F}} \mathtt{g}) \wedge (\{p_1, \cdots, p_n\} \succsim_{rpo}^{st(\mathtt{f})} \{e_1, \cdots, e_n\}))$$

and consider the program defined by the rules $\mathtt{f}(x) \to \mathtt{g}(x)$ and $\mathtt{g}(\boldsymbol{a}(x)) \to \mathtt{f}(x)$. Taking $\mathtt{f} \approx_{\mathcal{F}} \mathtt{g}$ and $st(\mathtt{f}) = p$, we have $\mathtt{f}(x) \succsim \mathtt{g}(x)$ and $\mathtt{g}(\boldsymbol{a}(x)) \succ_{rpo} \mathtt{f}(x)$. However, $\mathtt{f}(x) \succ_{rpo} \mathtt{g}(x)$ does not hold. Consequently, this program belongs to $DP(\succsim, \succ_{rpo})$ but does not belong to $DP(\succsim_{rpo}^{prod}, \succ_{rpo}^{prod})$.

These characterizations capture interesting algorithms and, in particular, algorithms having an exponential derivation length but being computable in polynomial time using dynamic programming techniques:

example 9. *Given two words over the alphabet $\{\mathbf{a}, \mathbf{b}, \epsilon\}$, the following program computes the length of their longest common subsequence*

$$\begin{aligned}
\max(\mathbf{0}, y) &\to y\\
\max(x, \mathbf{0}) &\to x\\
\max(\mathbf{S}(x), \mathbf{S}(y)) &\to \mathbf{S}(\max(x, y))\\
\mathtt{lcs}(\epsilon, y) &\to \mathbf{0}\\
\mathtt{lcs}(\mathbf{i}(x), \mathbf{i}(y)) &\to \mathbf{S}(\mathtt{lcs}(x, y))\\
\mathtt{lcs}(\mathbf{i}(x), \mathbf{j}(y)) &\to \max(\mathtt{lcs}(\mathbf{i}(x), y), \mathtt{lcs}(x, \mathbf{j}(y)))
\end{aligned}$$

where $\mathbf{i}, \mathbf{j} \in \{\mathbf{a}, \mathbf{b}\}$ and $\mathbf{i} \neq \mathbf{j}$. This program belongs to $DP(\succsim_{rpo}, \succ_{rpo})$ and admits the following additive and polynomial quasi-interpretation $(\!|\mathbf{0}|\!) = 0$, $(\!|\mathbf{a}|\!)(X) = (\!|\mathbf{b}|\!)(X) = (\!|\mathbf{S}|\!)(X) = X + 1$, $(\!|\max|\!)(X, Y) = max(X, Y)$, $(\!|\mathtt{lcs}|\!)(X, Y) = X + Y$. For example, for the last rule we check that:

$$\begin{aligned}
(\!|\mathtt{lcs}(\mathbf{i}(x), \mathbf{j}(y))|\!) &= X + 1 + Y + 1\\
&\geq X + Y + 1\\
&= (\!|\max(\mathtt{lcs}(\mathbf{i}(x), y), \mathtt{lcs}(x, \mathbf{j}(y)))|\!)
\end{aligned}$$

However, the property to be ordered by $\succ_{rpo}$ fails on many natural algorithms. In particular, a lot of algorithms which use function calls as arguments of a recursive call are not captured by this method:

example 10. *Consider the following program which eliminates duplicates from a list over an alphabet $\{\mathbf{a}_1, \cdots, \mathbf{a}_n\}$:*

$$\begin{aligned}
\mathtt{equ}(\mathbf{i}, \mathbf{j}) &\to \mathbf{ff}\\
\mathtt{equ}(\mathbf{i}, \mathbf{i}) &\to \mathbf{tt}\\
\mathtt{if}(\mathbf{ff}, n, \mathbf{c}(m, x)) &\to \mathbf{c}(m, \mathtt{rm}(n, x))\\
\mathtt{if}(\mathbf{tt}, n, \mathbf{c}(m, x)) &\to \mathtt{rm}(n, x)\\
\mathtt{rm}(n, \mathbf{nil}) &\to \mathbf{nil}\\
\mathtt{rm}(n, \mathbf{c}(m, x)) &\to \mathtt{if}(\mathtt{equ}(n, m), n, \mathbf{c}(m, x))\\
\mathtt{purge}(\mathbf{nil}) &\to \mathbf{nil}\\
\mathtt{purge}(\mathbf{c}(n, x)) &\to \mathbf{c}(n, \mathtt{purge}(\mathtt{rm}(n, x)))
\end{aligned}$$

where $\mathbf{i}, \mathbf{j} \in \{\mathbf{a}_1, \cdots, \mathbf{a}_n\}$ and $\mathbf{i} \neq \mathbf{j}$. It admits the following additive and max-polynomial quasi-interpretation $(\!|\mathbf{a}_k|\!) = (\!|\mathbf{nil}|\!) = 0$, $(\!|\mathtt{equ}|\!)(X, Y) = max(X, Y)$, $(\!|\mathbf{c}|\!)(X, Y) = X + Y + 1$, $(\!|\mathtt{if}|\!)(X, Y, Z) = \max(X, Y + Z)$, $(\!|\mathtt{rm}|\!)(X, Y) = X + Y$ and $(\!|\mathtt{purge}|\!)(X) = X^2$ but it fails to be $\succ_{rpo}$ ordered. Indeed, the last rule corresponds to a dependency pair of the shape

$(\mathtt{purge}(\mathbf{c}(n,x)), \mathtt{purge}(\mathtt{rm}(n,x)))$ *and we have* $\mathtt{rm}(n,x) \succ_{rpo} \mathbf{c}(n,x)$*, since* $\mathbf{c} \in \mathcal{C}$*.*

Moreover, many natural algorithms do not admit any additive quasi-interpretation because of the subterm property as illustrated by example 6.

So we strongly need new methods in order to capture such algorithms. Indeed, we want to increase the number of natural algorithms captured by our analysis in a perspective of automation. These new methods have to fulfill the following conditions: (i) remain decidable since our purpose is to analyze the complexity of programs automatically (ii) allow more flexibility than $\succ_{rpo}$ while analyzing recursive calls (iii) throw away the subterm property of quasi-interpretations.

5. Non-subterm assignments

In this section, we develop new methods in order to extend the intensionality (that is the number of captured algorithms) of the analysis performed in previous section. First, we define a reduction pair on assignments over real numbers:

definition 19. *Given a fixed constant $\delta > 0$, we define the strict well-founded ordering $>_\delta$ over $\mathbb{R}_{\geq 0}$ by: $x >_\delta y$ if $x \geq y+\delta$. Given an assignment $(\!|-|\!)$, we extend $\geq$ and $>_\delta$ to terms by $s \geq^{(\!|-|\!)} t$ iff $(\!|s|\!) \geq (\!|t|\!)$ and $s >_\delta^{(\!|-|\!)} t$ iff $(\!|s|\!) >_\delta (\!|t|\!)$.*

proposition 2. *$(\geq^{(\!|-|\!)}, >_\delta^{(\!|-|\!)})$ is a strong reduction pair.*

Proof. It is clear that $(\geq^{(\!|-|\!)}, >_\delta^{(\!|-|\!)})$ is a reduction pair. Just notice that $>_\delta^{(\!|-|\!)} \circ \geq^{(\!|-|\!)} \subseteq >_\delta^{(\!|-|\!)}$ and $\geq^{(\!|-|\!)} \circ >_\delta^{(\!|-|\!)} \subseteq >_\delta^{(\!|-|\!)}$. □

Now, we define a criterion which allows us to compensate the loss of the subterm property:

definition 20 (Bounded Recursive Calls (BRC))**.** *Given an assignment $(\!|-|\!)$, a program $\mathcal{p}$ is in $BRC((\!|-|\!))$ if for every dependency pair $(\mathtt{f}(p_1, \cdots, p_n), \mathtt{g}(e_1, \cdots, e_m))$ involved in a cycle of the SDP graph, we have:*

$$\sum_{j \in \{1,\ldots,n\}} (\!|p_j|\!) \geq \sum_{i \in \{1,\ldots,m\}} (\!|e_i|\!)$$

5.1 A criterion for FPSPACE

definition 21. *A program $\mathcal{p}$ is in $DP(\text{SPACE})$ if there is an additive, monotonic and max-polynomial assignment $(\!|-|\!)$ and a constant $\delta > 0$ such that:*

$$\mathcal{p} \in DP(\geq^{(\!|-|\!)}, >_\delta^{(\!|-|\!)}) \cap BRC((\!|-|\!))$$

example 11. *We can show that the program of example 1 belongs to $DP(\text{SPACE})$ since it belongs to $DP(\geq^{(\!|-|\!)}, >_1^{(\!|-|\!)}) \cap BRC((\!|-|\!))$ by taking the following assignment $(\!|-|\!)$:*

$$(\!|\mathbf{0}|\!) = 0 \qquad (\!|\mathbf{S}|\!)(X) = X + 1$$
$$(\!|\mathtt{minus}|\!)(X,Y) = X \qquad (\!|\mathtt{quo}|\!)(X,Y) = X + Y$$

Indeed, we let the reader check that for each rule $l \to r$, we have $(\!|l|\!) \geq (\!|r|\!)$. Moreover, for the two cycles in the SDP graph, we have:

$$\begin{aligned}(\!|\mathtt{quo}(\mathbf{S}(y), \mathbf{S}(z))|\!) &= Y + Z + 2\\ &>_1 Y + Z + 1\\ &= (\!|\mathtt{quo}(\mathtt{minus}(y,z), \mathbf{S}(z))|\!)\\ (\!|\mathtt{minus}(\mathbf{S}(u), \mathbf{S}(v))|\!) &= U + 1\\ &>_1 U\\ &= (\!|\mathtt{minus}(u,v)|\!)\end{aligned}$$

And for the BRC conditions, we have:

$$\begin{aligned}(\!|\mathbf{S}(y)|\!) + (\!|\mathbf{S}(\mathrm{z})|\!) &= Y + Z + 2\\ &\geq Y + Z + 1\\ &= (\!|\mathtt{minus}(y,z)|\!) + (\!|\mathbf{S}(z)|\!)\\ (\!|\mathbf{S}(u)|\!) + (\!|\mathbf{S}(v)|\!) &= U + V + 2\\ &\geq U + V\\ &= (\!|u|\!) + (\!|v|\!)\end{aligned}$$

Moreover, notice that this assignment is not a quasi-interpretation since it has not the subterm property:

$$(\!|\mathtt{minus}|\!)(X,Y) = X$$

example 12. *The program of example 9 is in $DP(\geq^{(\!|-|\!)}, >_1^{(\!|-|\!)}) \cap BRC((\!|-|\!)) \subset DP(\text{SPACE})$ by taking the assignment $(\!|-|\!)$ defined in example 9.*

example 13. *The program of example 10 is in $DP(\geq^{(\!|-|\!)}, >_1^{(\!|-|\!)}) \cap BRC((\!|-|\!)) \subset DP(\text{SPACE})$ by taking the assignment $(\!|-|\!)$ defined in example 10, except over* equ *where we take $(\!|\mathtt{equ}|\!)(X,Y) = 0$. Otherwise the program is not in $BRC((\!|-|\!))$ and, a fortiori not in $DP(\text{SPACE})$ since, for the sixth rule, we have to check:*

$$N + M + X + 1 \geq (\!|\mathtt{equ}|\!)(N,M) + N + M + X + 1$$

example 14. *Consider the following program computing the Quantified Boolean Formula problem (QBF) which is a* PSPACE*-complete problem:*

$$\begin{aligned}\mathtt{not}(\mathbf{tt}) &\to \mathbf{ff}\\ \mathtt{not}(\mathbf{ff}) &\to \mathbf{tt}\\ \mathtt{equal}(\mathbf{0},\mathbf{0}) &\to \mathbf{tt}\\ \mathtt{equal}(\mathbf{S}(x),\mathbf{0}) &\to \mathbf{ff}\\ \mathtt{or}(\mathbf{tt},x) &\to \mathbf{tt}\\ \mathtt{or}(\mathbf{ff},x) &\to x\\ \mathtt{equal}(\mathbf{0},\mathbf{S}(y)) &\to \mathbf{ff}\\ \mathtt{equal}(\mathbf{S}(x),\mathbf{S}(y)) &\to \mathtt{equal}(x,y)\\ \mathtt{in}(x,\mathbf{nil}) &\to \mathbf{ff}\\ \mathtt{in}(x,\mathbf{c}(a,l)) &\to \mathtt{or}(\mathtt{equal}(x,a),\mathtt{in}(x,l))\\ \mathtt{ver}(\mathbf{Var}(x),t) &\to \mathtt{in}(x,t)\\ \mathtt{ver}(\mathbf{Exists}(n,l),t) &\to \mathtt{or}(\mathtt{ver}(l,\mathbf{c}(n,t)),\mathtt{ver}(l,t))\\ \mathtt{ver}(\mathbf{Not}(l),t) &\to \mathtt{not}(\mathtt{ver}(l,t))\\ \mathtt{ver}(\mathbf{Or}(l_1,l_2),t) &\to \mathtt{or}(\mathtt{ver}(l_1,t),\mathtt{ver}(l_2,t))\\ \mathtt{qbf}(l) &\to \mathtt{ver}(l,\mathbf{nil})\end{aligned}$$

where booleans are encoded by the constructor symbols $\{\mathbf{tt},\mathbf{ff}\}$, formula are built using $\{\mathbf{Var},\mathbf{Not},\mathbf{Or},\mathbf{Exists}\}$ and variables are ranging over unary integers using the constructor symbols $\{\mathbf{S},\mathbf{0}\}$. We show that the program belongs to $DP(\geq^{(\!|-|\!)}, >_1^{(\!|-|\!)}) \cap BRC((\!|-|\!)) \subset DP(\text{SPACE})$ by taking $(\!|\mathbf{nil}|\!) = (\!|\mathbf{ff}|\!) = (\!|\mathbf{tt}|\!) = (\!|\mathbf{0}|\!) = (\!|\mathtt{not}|\!)(X) = 0$, $(\!|\mathbf{S}|\!)(X) = (\!|\mathbf{Var}|\!)(X) = (\!|\mathbf{Not}|\!)(X) = X + 1$, $(\!|\mathbf{c}|\!)(X,Y) = (\!|\mathbf{Or}|\!)(X,Y) = X + Y + 1$, $(\!|\mathbf{Exists}|\!)(X,Y) = X + Y + 2$, $(\!|\mathtt{equal}|\!)(X,Y) = (\!|\mathtt{qbf}|\!)(X) = X$, $(\!|\mathtt{ver}|\!)(X,Y) = X + Y$ and $(\!|\mathtt{or}|\!)(X,Y) = (\!|\mathtt{in}|\!)(X,Y) = Y$.

5.2 A criterion for FPTIME

First, we define the notion of neighborhood which allows us to control the recursive calls corresponding to the same recursive rule together:

definition 22 (Neighborhood)**.** *Given a rule $l \to r$ of the program, we define its neighborhood $N(l \to r)$ by:*

$N(l \to r)$=$\{(l,t)$, *such that* (l,t) *is involved in a cycle of the SDP graph and there is a context* $\mathsf{C}[\Box]$ *such that* $r = \mathsf{C}[t]$ $\}$

example 15. *In the program of example 1, we have:*

$$N(\mathtt{minus}(\mathbf{S}(u),\mathbf{S}(v)) \to \mathtt{minus}(u,v))$$
$$= \{(\mathtt{minus}(\mathbf{S}(u),\mathbf{S}(v)), \mathtt{minus}(u,v))\}$$
$$N(\mathtt{quo}(\mathbf{S}(y),\mathbf{S}(z)) \to \mathbf{S}(\mathtt{quo}(\mathtt{minus}(y,z),\mathbf{S}(z))))$$
$$= \{(\mathtt{quo}(\mathbf{S}(y),\mathbf{S}(z)), \mathtt{quo}(\mathtt{minus}(y,z),\mathbf{S}(z))\}$$

example 16. *In the program of example 9, we have:*

$$N(\max(\mathbf{S}(x),\mathbf{S}(y)) \to \mathbf{S}(\max(x,y)))$$
$$= \{(\max(\mathbf{S}(x),\mathbf{S}(y)), \max(x,y))\}$$
$$N(\mathtt{lcs}(\mathbf{i}(x),\mathbf{i}(y)) \to \mathbf{S}(\mathtt{lcs}(x,y)))$$
$$= \{(\mathtt{lcs}(\mathbf{i}(x),\mathbf{i}(y)), \mathtt{lcs}(x,y))\}$$
$$N(\mathtt{lcs}(\mathbf{i}(x),\mathbf{j}(y)) \to \max(\mathtt{lcs}(\mathbf{i}(x),y), \mathtt{lcs}(x,\mathbf{j}(y)))$$
$$= \{(\mathtt{lcs}(\mathbf{i}(x),\mathbf{j}(y)), \mathtt{lcs}(\mathbf{i}(x),y)), (\mathtt{lcs}(\mathbf{i}(x),\mathbf{j}(y)), \mathtt{lcs}(x,\mathbf{j}(y)))\}$$

definition 23. *A program* $\mathtt{p}$ *belongs to* $DP(\text{TIME})$ *if there is an additive, monotonic and max-polynomial assignment* $(\!|-|\!)$ *and a constant* $\delta > 0$ *such that:*

- $\mathtt{p} \in DP(\geq^{(\!|-|\!)}, >_\delta^{(\!|-|\!)}) \cap BRC((\!|-|\!))$
- *and, for each neighborhood* $\{(l,t_1),\ldots,(l,t_n)\}$*:*

$$(\!|l|\!) >_\delta \sum_{i=1}^{n} (\!|t_i|\!)$$

example 17. *The programs of examples 1 and 10 are in* $DP(\text{TIME})$ *since the conditions on neighborhood add no further constraint. Programs of examples 9 and 14 are not in* $DP(\text{TIME})$*. This result is natural for example 14, which is a complete problem for polynomial space. However, for example 9, it means that the constraints on* $DP(\text{TIME})$ *exclude the algorithms with exponential derivation length even if they are computable in polynomial time using dynamic programming techniques. It seems to be a drawback, but we manage to capture non subterm upper bounds at this cost.*

remark 4. *Notice that we have the following inclusion:*

$$DP(\text{TIME}) \subset DP(\text{SPACE})$$

5.3 Characterizations of FPTIME and FPSPACE

If we consider programs in $DP(\text{SPACE})$, the size of every computed value is polynomially bounded by the input size:

lemma 6. *Given a program* $\mathtt{p} \in DP(\text{SPACE})$ *and a function symbol* $\mathtt{f}$*, there is a polynomial* P *such that for every state* η *of a call-tree of root* $\langle \mathtt{f}, v_1, \cdots, v_n\rangle$*, we have* $|\eta| \leq P(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|)$*.*

Proof. Consider a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}\rangle \in DP(\text{SPACE})$. There is a constant $\delta > 0$ such that $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}\rangle \in DP(\geq^{(\!|-|\!)}, >_\delta^{(\!|-|\!)}) \cap BRC((\!|-|\!))$. Given a transition $\langle \mathtt{f}, v_1, \cdots, v_n\rangle \rightsquigarrow \langle \mathtt{g}, s_1, \cdots, s_m\rangle$, suppose that there are a rule $\mathtt{f}(p_1,\cdots,p_n) \to \mathsf{C}[\mathtt{g}(e_1,\cdots,e_m)]$ and a substitution σ such that $p_i\sigma = v_i$ and $[\![e_j\sigma]\!] = s_j$.

- If $(\mathtt{f}(p_1,\cdots,p_n), \mathtt{g}(e_1,\cdots,e_m))$ is involved in a cycle, since $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}\rangle \in BRC((\!|-|\!))$, we have

$$\sum_{j\in\{1,\ldots,n\}} (\!|p_j\sigma|\!) \geq \sum_{i\in\{1,\ldots,m\}} (\!|e_i\sigma|\!)$$

Taking α to be the constant of lemma 1, we obtain:

$$\alpha \times |\langle \mathtt{f}, v_1, \cdots, v_n\rangle| \geq \sum_{j\in\{1,\ldots,n\}} (\!|p_j\sigma|\!)$$

(By lemma 1)

$$\geq \sum_{j\in\{1,\ldots,m\}} (\!|e_j\sigma|\!)$$

(Since $\mathtt{p} \in BRC((\!|-|\!))$)

$$\geq \sum_{j\in\{1,\ldots,m\}} (\!|s_j|\!)$$

(By proposition 1)

$$\geq \sum_{j\in\{1,\ldots,m\}} |s_j| = |\langle \mathtt{g}, s_1, \cdots, s_m\rangle|$$

(By lemma 1)

Moreover, we can show by induction on the length of the path that for every state $\langle \mathtt{h}, v'_1, \cdots, v'_p\rangle$ of a path $\langle \mathtt{f}, v_1, \cdots, v_n\rangle \overset{+}{\rightsquigarrow} \langle \mathtt{f}, u_1, \cdots, u_n\rangle$, we have:

$$|\langle \mathtt{h}, v'_1, \cdots, v'_p\rangle| \leq \alpha \times |\langle \mathtt{f}, v_1, \cdots, v_n\rangle|$$

- Otherwise, $(\mathtt{f}(p_1,\cdots,p_n), \mathtt{g}(e_1,\cdots,e_m))$ is not involved in a cycle. Define Q by $Q(X) = \max_{b\in\mathcal{C}\cup\mathcal{F}}(\!|b|\!)(\alpha\times X,\ldots,\alpha\times X)$, with α the constant of lemma 1. Suppose that the size of $e_j = b(d_1,\cdots,d_m)$ is k and, by induction hypothesis, that for every term d in e_j of size strictly smaller than k, we have

$$|[\![d\sigma]\!]| \leq Q^k(\max_{i\in\{1,\ldots,n\}} |p_i\sigma|)\ (I.H.)$$

We obtain:

$$|[\![b(d_1,\cdots,d_m)\sigma]\!]| \leq (\!|b|\!)((\!|d_1\sigma|\!),\ldots,(\!|d_m\sigma|\!))$$
(By lemma 1 and cbv)
$$\leq (\!|b|\!)((\!|[\![d_1\sigma]\!]|\!),\ldots,(\!|[\![d_m\sigma]\!]|\!))$$
(By proposition 1)
$$\leq (\!|b|\!)(\alpha\times|[\![d_1\sigma]\!]|,\ldots,\alpha\times|[\![d_1\sigma]\!]|)$$
(By lemma 1)
$$\leq (\!|b|\!)(..\alpha\times Q^k(\max_{i\in\{1,\ldots,n\}} |p_i\sigma|)..)$$
(By I.H.)
$$\leq Q^{k+1}(\max_{i\in\{1,\ldots,n\}} |p_i\sigma|)$$
(By definition of Q)

and we obtain that

$$|\langle \mathtt{g}, s_1, \cdots, s_m\rangle| \leq l \times Q^k(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|)$$

if k is the maximal size of a term in the program and l the maximal arity of a symbol. We define P' by $P'(X) = \max(l\times Q^k(X), \alpha\times X)$ and we obtain that $P'(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|) \geq |\langle \mathtt{g}, s_1, \cdots, s_m\rangle|$. The number of such transitions is bounded by a constant γ (which depends on the size of the program) and consequently, $R(X) = P'^\gamma(X)$ provides an upper bound on the size of every state.

□

We can extend this result to the paths of every call-tree when considering programs of $\mathtt{p} \in DP(\geq^{(\!|-|\!)}, >_\delta^{(\!|-|\!)})$:

lemma 7. *Given a program* $\mathtt{p} \in DP(\geq^{(\!|-|\!)}, >_\delta^{(\!|-|\!)})$*, for some constant* $\delta > 0$ *and some additive, max-polynomial and monotonic assignment* $(\!|-|\!)$*, and a call-tree corresponding to one execution of this program and containing a path of the shape* $\langle \mathtt{f}, v_1, \cdots, v_n\rangle \overset{+}{\rightsquigarrow} \langle \mathtt{f}, u_1, \cdots, u_n\rangle$*, then:*

$$\mathtt{f}(v_1,\cdots,v_n) >_\delta^{(\!|-|\!)} \mathtt{f}(u_1,\cdots,u_n)$$

Proof. By definitions 2 and 9, there are a transition of the shape $\langle \mathtt{h}, t_1, \cdots, t_l\rangle \rightsquigarrow \langle \mathtt{g}, s_1, \cdots, s_m\rangle$ in the path, a rule of the shape $\mathtt{h}(p_1, \cdots, p_l) \to \mathsf{C}[\mathtt{g}(e_1, \cdots, e_m)]$ and a substitution σ such that $p_i\sigma = t_i$ and $[\![e_j\sigma]\!] = s_j$ and $(\!|\mathtt{h}(p_1, \cdots, p_l)|\!) >_\delta (\!|\mathtt{g}(e_1, \cdots, e_m)|\!)$. Consequently, we obtain:

$$\begin{aligned}
(\!|\mathtt{h}(t_1, \cdots, t_l)|\!) &= (\!|\mathtt{h}(p_1, \cdots, p_l)\sigma|\!) \\
&>_\delta (\!|\mathtt{g}(e_1, \cdots, e_m)\sigma|\!) && \geq \text{is stable} \\
&= (\!|\mathtt{g}|\!)((\!|e_1\sigma|\!), \ldots, (\!|e_m\sigma|\!)) && \text{By definition 15} \\
&\geq (\!|\mathtt{g}|\!)((\!|[\![e_1\sigma]\!]|\!), .., (\!|[\![e_m\sigma]\!]|\!)) && \text{By proposition 1} \\
&= (\!|\mathtt{g}(s_1, \cdots, s_m)|\!) && \text{By definition 15}
\end{aligned}$$

By definition 9, we know that for every transition of the shape $\langle \mathtt{h}, t_1, \cdots, t_l\rangle \rightsquigarrow \langle \mathtt{g}, s_1, \cdots, s_m\rangle$ in the path we have

$$(\!|\mathtt{h}(t_1, \cdots, t_l)|\!) \geq (\!|\mathtt{g}(s_1, \cdots, s_m)|\!)$$

(Just replace $>_\delta$ by $\geq$ in the above inequalities).

We have shown that there are two states $\langle \mathtt{h}, v'_1, \cdots, v'_l\rangle$ and $\langle \mathtt{g}, s_1, \cdots, s_m\rangle$ in the path $\langle \mathtt{f}, v_1, \cdots, v_n\rangle \overset{*}{\rightsquigarrow} \langle \mathtt{h}, v'_1, \cdots, v'_l\rangle \rightsquigarrow \langle \mathtt{g}, s_1, \cdots, s_m\rangle \overset{*}{\rightsquigarrow} \langle \mathtt{f}, u_1, \cdots, u_n\rangle$ such that the following inequalities hold:

$$\begin{aligned}
(\!|\mathtt{f}(v_1, \cdots, v_n)|\!) &\geq \ldots \\
&\geq (\!|\mathtt{h}(v'_1, \cdots, v'_l)|\!) \\
&>_\delta (\!|\mathtt{g}(s_1, \cdots, s_m)|\!) \\
&\geq \ldots \\
&\geq (\!|\mathtt{f}(u_1, \cdots, u_n)|\!)
\end{aligned}$$

Since $(\geq, >_\delta)$ is a strong reduction pair, by proposition 2, the inequalities collapse into $(\!|\mathtt{f}(v_1, \cdots, v_n)|\!) >_\delta (\!|\mathtt{f}(u_1, \cdots, u_n)|\!)$. □

lemma 8. *Given a program $\mathtt{p} \in DP(\text{SPACE})$, for some constant $\delta > 0$ and some additive, max-polynomial and monotonic assignment $(\!|-|\!)$, then there is a polynomial P such that for every path B of a call-tree of root $\langle \mathtt{f}, v_1, \cdots, v_n\rangle$ corresponding to one execution of this program, we have:*

$$|B| \leq P(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|)$$

Proof. Take a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}\rangle \in DP(\text{SPACE})$. There is a constant $\delta > 0$ such that $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R}\rangle \in DP(\geq^{(\!|-|\!)}, >_\delta^{(\!|-|\!)}) \cap BRC((\!|-|\!))$. By lemma 6, there is a polynomial P' such that, for every state η of a path B, we have $|\eta| \leq P'(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|)$. We start by showing that for every path of the shape $B = \langle \mathtt{g}, s_1, \cdots, s_m\rangle \overset{+}{\rightsquigarrow} \langle \mathtt{g}, s'_1, \cdots, s'_m\rangle$, we have $\mathtt{lg}(B) \leq \gamma \times (\!|\mathtt{g}(s_1, \cdots, s_m)|\!)$.

By lemma 7, we know that two successive calls of the same function symbol correspond to a strict decrease:

$$(\!|\mathtt{g}(s_1, \cdots, s_m)|\!) >_\delta (\!|\mathtt{g}(s'_1, \cdots, s'_m)|\!)$$

It means that $(\!|\mathtt{g}(s_1, \cdots, s_m)|\!) \geq \delta + (\!|\mathtt{g}(s'_1, \cdots, s'_m)|\!)$, with $\delta > 0$. Consequently, if we have a path of the shape:

$$\langle \mathtt{g}, s_1, \cdots, s_m\rangle \overset{+}{\rightsquigarrow} \langle \mathtt{g}, s^1_1, \cdots, s^1_m\rangle \overset{+}{\rightsquigarrow} \ldots \overset{+}{\rightsquigarrow} \langle \mathtt{g}, s^i_1, \cdots, s^i_m\rangle$$

then we have:

$$(\!|\mathtt{g}(s_1, \cdots, s_m)|\!) \geq i \times \delta + (\!|\mathtt{g}(s^i_1, \cdots, s^i_m)|\!)$$

and there are at most $(\!|\mathtt{g}(s_1, \cdots, s_m)|\!)/\delta$ states corresponding to the function symbol g in the depth of the call-tree. Since the maximal number of dependency pairs involved in a cycle of the SDP graph is bounded by a constant β, which only depends on p, we obtain that each path $\langle \mathtt{g}, s^i_1, \cdots, s^i_m\rangle \overset{+}{\rightsquigarrow} \langle \mathtt{g}, s^{i+1}_1, \cdots, s^{i+1}_m\rangle$ corresponding to a cycle in the SDP graph has a length bounded by β. Finally, we obtain that the path has a length bounded by $\beta/\delta \times (\!|\mathtt{g}(s_1, \cdots, s_m)|\!)$ and we set $\gamma = \beta/\delta$.

Combining these two results, we obtain, by monotonicity of $(\!|-|\!)$, that:

$$\mathtt{lg}(B) \leq \gamma \times (\!|\mathtt{g}|\!)(P'(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|), .., P'(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|))$$

As a result, every path of the call three has a length bounded by $R(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|)$, where R is defined by $R(X) = \#\mathcal{F} \times \gamma \times \max_{\mathtt{h} \in \mathcal{F}} (\!|\mathtt{h}|\!)(P'(X), \ldots, P'(X))$.

Since R is in **Max-Poly** $\{\mathbb{R}_{\geq 0}\}$, there is a polynomial P such that $\forall X,\ P(X) \geq R(X)$. □

lemma 9. *Given a program $\mathtt{p} \in DP(\text{TIME})$ then there is a polynomial P such that for every call-tree T of root $\langle \mathtt{f}, v_1, \cdots, v_n\rangle$ corresponding to one execution of this program, we have:*

$$|T| \leq P(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|)$$

Proof. Since $DP(\text{TIME}) \subset DP(\text{SPACE})$, we can apply lemma 6: There is a polynomial R such that for every state $\langle \mathtt{g}, s_1, \cdots, s_m\rangle$ of a call-tree of root $\langle \mathtt{f}, v_1, \cdots, v_n\rangle$, we have:

$$|\langle \mathtt{g}, s_1, \cdots, s_m\rangle| \leq R(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|)$$

If the state $\langle \mathtt{g}, s_1, \cdots, s_m\rangle$ corresponds to a recursive call, there is a neighborhood of the shape $\{(\mathtt{g}(p_1, \cdots, p_m), \mathtt{f}_1(\ldots, e_1, \ldots))\ldots (\mathtt{g}(p_1, \cdots, p_m), \mathtt{f}_n(\ldots, e_n, \ldots))\}$ corresponding to a rule of the shape $\mathtt{g}(p_1, \cdots, p_m) \to e$ and there is a substitution σ such that $p_i\sigma = s_i$ and we have:

$$\begin{aligned}
(\!|\mathtt{g}(s_1, \cdots, s_m)|\!) &>_\delta \sum_{i=1}^{n} (\!|\mathtt{f}_i(\ldots, e_i\sigma, \ldots)|\!) \\
&\text{(By definition 23)} \\
&\geq \delta + \sum_{i=1}^{n} (\!|\mathtt{f}_i|\!)(\ldots, (\!|e_i\sigma|\!), \ldots) \\
&\text{(By definition of } (\!|-|\!)) \\
&\geq \delta + \sum_{i=1}^{n} (\!|\mathtt{f}_i(\ldots, [\![e_i\sigma]\!], \ldots)|\!) \\
&\text{(By proposition 1 and monotonicity)}
\end{aligned}$$

Consequently, we can apply at most $1/\delta \times (\!|\mathtt{g}(s_1, \cdots, s_m)|\!)$ times this recursive rule and there is a constant β such that the number of states in T corresponding to an application of this recursive rule is bounded by $\beta/\delta \times (\!|\mathtt{g}|\!)(\alpha \times R(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|), \ldots, \alpha \times R(|\langle \mathtt{f}, v_1, \cdots, v_n\rangle|))$, with α the constant of lemma 1. It remains to apply the same reasoning for every recursive rule. Just notice that non recursive rules only add a constant number of new states in the call-tree. Since both the number of states and the size of each state in the call-tree are polynomially bounded by the input size, we obtain the required result. □

theorem 3. *The set of functions computed by programs of:*

1. $DP(\text{TIME})$ *is exactly the set of functions* FPTIME
2. $DP(\text{SPACE})$ *is exactly the set of functions* FPSPACE

Proof. (1) By Lemma 9, we know that the size of every call-tree is polynomially bounded by the input. As a consequence, every program can be simulated by a Turing Machine in polynomial time. Conversely, every polynomial and additive interpretation of (Bonfante et al. 2001) is in $DP(\text{TIME})$. (2) By lemma 8, we know that the size of each path of the call-tree is polynomially bounded by the input size. Evaluating the program in the depth of the call-tree, we obtain that the set of functions computed by programs which admit a polynomial space interpretation is included in PSPACE. Conversely, we use the proof of lemma 60 in (Bonfante et al. 2007). □

6. Conclusion

The characterizations using quasi-interpretations and RPO presented in section 4 strictly generalize the ones of (Bonfante et al. 2007), as illustrated by example 8, still allowing to capture programs having an exponential derivation length but being computable in polynomial time using memoization techniques. Consequently, these results confer a new approach with the notion of reduction pair and ameliorate the intensionality of the QI method.

On the other hand, the second approach of section 5 uses only polynomial assignments. It allows to obtain non subterm assignments and increases consequently the intensionality of our study (even if the programs having an exponential derivation length are not captured in $DP(\text{TIME})$). This is a real improvement in the study of semantics interpretations: Indeed, we are now able to take the interpretation of the symbol `minus` to be $(\!|\mathtt{minus}|\!)(X,Y) = X$ (instead of $\max(X,Y)$). The closer interpretations are from the size of the computed functions, the better intensionality our study has. We claim that the following example which computes the logarithm has no quasi-interpretation:

$$\begin{aligned}
\mathtt{half}(\mathbf{0}) &\to \mathbf{0}\\
\mathtt{half}(\mathbf{S}(\mathbf{0})) &\to \mathbf{0}\\
\mathtt{half}(\mathbf{S}(\mathbf{S}(x))) &\to \mathbf{S}(\mathtt{half}(x))\\
\mathtt{log}(\mathbf{S}(\mathbf{0})) &\to \mathbf{0}\\
\mathtt{log}(\mathbf{S}(x)) &\to \mathbf{S}(\mathtt{log}(\mathtt{half}(\mathbf{S}(x))))
\end{aligned}$$

because of the last rule. However, it is in $DP(\text{TIME})$ by taking $(\!|\mathbf{0}|\!) = 0$, $(\!|\mathbf{S}|\!)(X) = X+1$, $(\!|\mathtt{half}|\!)(X) = X/2$ and $(\!|\mathtt{log}|\!)(X) = 2 \times X$.

Moreover, this last section also allows to capture programs having a quasi-interpretation but which are not terminating under RPO. Indeed, all the program including a rule of the shape $\mathtt{f}(x) \to \mathtt{f}(\mathtt{g}(x))$, with `g` a non size increasing function, will admit a quasi-interpretation (just take $(\!|\mathtt{f}|\!)(X) = (\!|\mathtt{g}|\!)(X) = X$). However, since $x \succ_{rpo} \mathtt{g}(x)$ does not hold, they are not terminating by RPO.

To conclude, the non-subterm assignments studied in the criterion for $DP(\text{SPACE})$ are inspired by the assignments of (Lucas 2005) and correspond to the notion of sup-interpretation introduced in (Marion and Péchoux 2006). Since the criteria using DP is decidable and so are the inequalities of $DP(\text{SPACE})$ using polynomial assignments of bounded degree (as demonstrated in (Bonfante et al. 2005)), we obtain a decidable criterion to synthesize non-subterm assignments, which is a new result.

References

E. Albert, P. Arenas, S. Genaim, and G. Puebla. Automatic Inference of Upper Bounds for Recurrence Relations in Cost Analysis. *Proc. of Static Analysis Symposium (SAS)*, a. To appear.

E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. Cost analysis of java bytecode. *16th European Symposium on Programming, ESOP*, 7:157–172, b.

T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

P. Baillot and V. Mogbil. Soft lambda-calculus: a language for polynomial time computation. In *FOSSACS 2004*, volume 2987 of *LNCS*, pages 27–41. Springer, 2004.

G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Algorithms with polynomial interpretation termination proof. *Journal of Functional Programming*, 11(1):33–53, 2001.

G. Bonfante, J.-Y. Marion, J.-Y. Moyen, and R. Péchoux. Synthesis of quasi-interpretations. *LCC2005, LICS affiliated Workshop*, 2005. `http://hal.inria.fr/`.

G. Bonfante, J.Y. Marion, and J.Y. Moyen. Quasi-interpretations, a way to control resources. *Accepted to Theoretical Computer Science*, 2007. `http://www.loria.fr/~marionjy/Research/Publications/Articles/TCS.pdf`.

N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.

M. Gaboardi and S. Ronchi Della Rocca. A soft type assignment system for λ-calculus. *CSL 2007*, 4646:253–267, 2007.

M. Gaboardi, S.R. Della Rocca, and J.Y. Marion. A Logical Account of PSPACE. In *POPL*, 2008.

J.-Y. Girard. Light linear logic. *Information and Computation*, 143(2): 175–204, 1998.

M. Hofmann. The strength of Non-Size Increasing computation. In *POPL*, pages 260–269, 2002.

G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

S. Kamin and J-J Lévy. Attempts for generalising the recursive path orderings. Technical report, University of Illinois, Urbana, 1980.

L. Kristiansen and N.D. Jones. The flow of data and the complexity of algorithms. In *CIE*, volume 3526, pages 263–274. Springer, 2005.

K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. *PPDP*, 1702:48–62, 1999.

Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318:163–180, 2004.

D.S. Lankford. On proving term rewriting systems are noetherien. Technical report, 1979.

S. Lucas. Polynomials over the reals in proofs of termination: from theory to practice. *RAIRO Theoretical Informatics and Applications*, 39(3):547–586, 2005.

Z. Manna and S. Ness. On the termination of Markov algorithms. In *Third hawaii international conference on system science*, pages 789–792, 1970.

J.-Y. Marion and R. Péchoux. Resource analysis by sup-interpretation. In *FLOPS*, volume 3945 of *LNCS*, pages 163–176, 2006.

K.-H. Niggl and H. Wunderlich. Certifying polynomial time and linear/polynomial space for imperative programs. *SIAM Journal on Computing*, 35(5):1122–1147, 2006.

A. Tarski. *A Decision Method for Elementary Algebra and Geometry, 2nd ed.* University of California Press, 1951.