Quasi-interpretation Synthesis by Decomposition : An application to higher-order programs

Guillaume Bonfante, Jean-Yves Marion, Romain Péchoux

Modularity of the Quasi-interpretations Synthesis and an Application to Higher-Order Programs

Guillaume Bonfante, Jean-Yves Marion and Romain Péchoux

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

Abstract. Quasi-interpretations have shown their interest to deal with resource analysis of first order functional programs. There are at least two reasons to study the question of modularity of quasi-interpretations. Firstly, modularity allows to decrease the complexity of the quasi-interpretation search algorithms. Secondly, modularity allows to increase the intentionality of the quasi-interpretation method, that is the number of captured programs. In particular, we take advantage of modularity conditions to extend smoothly quasi-interpretations to higher order programs. In this paper, we study the modularity of quasi-interpretations through the notions of constructor-sharing and hierarchical unions. We show that in the case of constructor-sharing and hierarchical unions, the existence of quasi-interpretations is no longer a modular property. However, we can still certify the complexity of programs.

1 Introduction

1.1 Certifying resources by Quasi-interpretations

The control of resource (i.e. memory/space or time) is a fundamental issue for critical systems. The present work, which considers more specifically the static analysis of first order functional programs, is a contribution to that field and, in particular, to the field of *implicit computational complexity (ICC)*.

We briefly review four distinct approaches to that issue of the *(ICC)* community. The first one deals with linear type disciplines in order to restrict computational time and began with the seminal work of Girard [18] which defined Light Linear Logic. The interested reader should consult the recent results of Baillot-Terui [5], Lafont [28] and Coppola-Ronchi [13]. The second approach is due to Hofmann [21, 22, 4], who introduced a resource atomic type, into linear type systems, for higher-order functional programming. The third one considers imperative programming languages and is developed by Jones-Kristianssen, Niggl [35], Marion-Moyen [34, 31]. The fourth approach is the one on which we focus in this paper. It concerns term rewriting systems and quasi-interpretations. A quasi-interpretation of a first order functional program provides an upper bound

on the size of any value computed. The paper [7] is a comprehensive introduction to quasi-interpretations. Combined with recursive path orderings, it allows characterizing complexity classes such as the set of polynomial time functions or yet the set of polynomial space functions. The main features of quasi-interpretations (abbreviated QI) are the following.

1. QI analysis includes a broad class of algorithms, even some that have an exponentially length derivation but denote a polynomial time computable function using dynamic programming techniques. See [30, 6, 8]
2. Resource verification of bytecode programs obtained by compiling first order functional and reactive programs which admit quasi-interpretations. See for example [2, 3, 14].
3. In [9], the synthesis of QI was shown to be decidable in exponential time for polynomial quasi-interpretations of bounded degree over real numbers.

QI provide a methodology in order to determine and control the involved resources in a computation. From a developer perspective, a virtual machine runs and manages codes. We attach to each program a QI. As programs are loaded for execution, each program has its own QI. So the virtual machine, which executes a program, knows an upper bound on necessary resources and may predict and be aware of, say, the size of stack frames, which could be of great importance for execution efficiency.

1.2 Improving QI Synthesis

The problem of the QI synthesis, which was first studied by Amadio in [1], consists in finding a QI for a given program. In a perspective of automatic analysis of the complexity of programs, such a problematic is fundamental. The synthesis of QI is a very tricky problem which is undecidable in general. Using Tarski's result showing that the theory of real numbers with equality, comparison, sum and product is decidable, we have shown in [9] that the QI synthesis for a program having n variables, has a time complexity in 2^{n^α}, for some constant α, as long as we take polynomial quasi-interpretations with degree smaller than an arbitrarily fixed constant. On one hand we have a very general procedure, on the other hand the procedure has a high cost.

The question of modularity of QI is central as long as one considers a divide and conquer strategy to find QI. Take a program and divide it into k subprograms having n_j variables for j from 1 to k. Then the complexity of the QI synthesis decreases from $2^{(\sum_{j=1}^{k} n_j)^\alpha}$ to $\sum_{j=1}^{k} 2^{n_j^\alpha}$, for some constant α. Such results could allow the improvement of a software called CROCUS that we are currently developing and which finds QI using some heuristics.

1.3 A modular approach to program resource analysis

The study of modularity is nowadays a classical approach to solve problems, like confluence or termination, by dividing the problem into smaller parts. The reader should consult [26, 20, 39, 38, 33, 27] to read an overview on this subject.

In this context, we first consider the *constructor-sharing case* in which functions defined by programs share constructors. We show that QI are not modular, except in the special case of disjoint union, but we can use QI in order to predict resource bounds. The consequence is that we are able to analyse the complexity of more programs. Indeed, suppose that a program does not admit a QI. A strategy is to cut it into two programs, which just have some constructors in common. Then, we can try to find QI of both sub-programs. We demonstrate that in this case we can still determine an upper bound on the amount of resource involved in a computation. Another consequence is a new characterization of the sets of polynomial time functions and of polynomial space functions. The second case is the *hierarchical union*, where constructors of one program are defined function symbols of another program. Again the fact of admitting a QI is not modular. We use the work of Dershowitz [17] introducing some conditions for ensuring the modularity of completeness for hierarchical unions and put some syntactic restriction on the programs considered. Again we are able, under these conditions, to analyse resource bounds. These results allow to divide programs in a more flexible fashion. Lastly, the hierarchical union of two programs can be considered as a way to deal with higher-order programs. Up to now, the QI method only applies to first order functional programs. A way to deal with higher-order programs is to transform a higher-order definition into a hierarchical union of programs using higher-order removal methods. Then, modularity is a natural way to ensure resource complexity.

2 First order functional programming

2.1 Syntax of first order programs

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

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

where $x \in \mathcal{X}$, $\mathtt{f} \in \mathcal{F}$, and $\mathbf{c} \in \mathcal{C}$.

The set of rules induces a rewriting relation $\to$. The relation $\xrightarrow{*}$ is the reflexive and transitive closure of $\to$. Throughout, we consider only orthogonal programs, that is, rule patterns are disjoint and linear. So each program is confluent [23].

The domain of computation of a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$ is the constructor algebra $\mathcal{T}(\mathcal{C})$. For each function symbol $\mathtt{f} \in \mathcal{F}$, the program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$ computes a partial function $[\![\mathtt{f}]\!] : \mathcal{T}(\mathcal{C})^n \to \mathcal{T}(\mathcal{C})$ defined by: $[\![\mathtt{f}]\!](v_1, \cdots, v_n) = w$ iff $\mathtt{f}(v_1, \cdots, v_n) \xrightarrow{*} w$ for all $v_i \in \mathcal{T}(\mathcal{C})$ and w is in $\mathcal{T}(\mathcal{C})$. Otherwise $[\![\mathtt{f}]\!](v_1, \cdots, v_n)$ is undefined. A substitution σ is a mapping from variables to terms and a ground

substitution is one which ranges over values of $\mathcal{T}(\mathcal{C})$. Observe that values are normal forms for the program. By extension, given a term t and a ground substitution σ, $[\![t\sigma]\!] = w$ if $t\sigma \xrightarrow{*} w$ and w is in $\mathcal{T}(\mathcal{C})$.

The size $|t|$ of a term t is defined to be the number of symbols of arity strictly greater than 0 occurring in it.

2.2 Recursive Path orderings

Given a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$, we define a *precedence* $\geq_{\mathcal{F}}$ on function symbols and its transitive closure that we also note $\geq_{\mathcal{F}}$ by $\mathtt{f} \geq_{\mathcal{F}} \mathtt{g}$ if there is a rule of the shape $\mathtt{f}(p_1, \cdots, p_n) \to \mathsf{C}[\mathtt{g}(e_1, \cdots, e_m)]$ with $\mathsf{C}[\diamond]$ a context and $e_1, \cdots, e_m$ terms. $\mathtt{f} \approx_{\mathcal{F}} \mathtt{g}$ iff $\mathtt{f} \geq_{\mathcal{F}} \mathtt{g}$ and $\mathtt{g} \geq_{\mathcal{F}} \mathtt{f}$. $\mathtt{f} >_{\mathcal{F}} \mathtt{g}$ 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 $st(\mathtt{f})$ in $\{p, l\}$ and which satisfies if $\mathtt{f} \approx_{\mathcal{F}} \mathtt{g}$ then $st(\mathtt{f}) = st(\mathtt{g})$. The status indicates how to compare recursive calls. When $st(\mathtt{f}) = p$, the status of $\mathtt{f}$ is said to be product. In that case, the arguments are compared with the product extension of $\prec_{rpo}$. Otherwise, the status is said to be lexicographic.

Definition 1. *The product extension $\prec^p$ and the lexicographic extension $\prec^l$ of $\prec$ over sequences are defined by:*

- $(m_1, \cdots, m_k) \prec^p (n_1, \cdots, n_k)$ *if and only if (i)* $\forall i \leq k, m_i \preceq n_i$ *and (ii)* $\exists j \leq k$ *such that* $m_j \prec n_j$.
- $(m_1, \cdots, m_k) \prec^l (n_1, \cdots, n_l)$ *if and only if* $\exists j$ *such that* $\forall i < j,\ m_i \preceq n_i$ *and* $m_j \prec n_j$

Definition 2 ([16],[25]). *Given a precedence $\preceq_{\mathcal{F}}$ and a status st, we define the recursive path ordering $\prec_{rpo}$ as follows:*

$$\frac{u \preceq_{rpo} t_i}{u \prec_{rpo} f(\ldots, t_i, \ldots)}\, f \in \mathcal{F} \bigcup \mathcal{C} \qquad \frac{\forall i\ u_i \prec_{rpo} \mathtt{f}(t_1, \cdots, t_n) \qquad g \prec_{\mathcal{F}} \mathtt{f}}{g(u_1, \cdots, u_m) \prec_{rpo} \mathtt{f}(t_1, \cdots, t_n)}\, g \in \mathcal{F} \bigcup \mathcal{C}$$

$$\frac{(u_1, \cdots, u_n) \prec_{rpo}^{st(\mathtt{f})} (t_1, \cdots, t_n) \qquad \mathtt{f} \approx_{\mathcal{F}} \mathtt{g} \qquad \forall i\ u_i \prec_{rpo} \mathtt{f}(t_1, \cdots, t_n)}{\mathtt{g}(u_1, \cdots, u_n) \prec_{rpo} \mathtt{f}(t_1, \cdots, t_n)}$$

A program is ordered by $\prec_{rpo}$ if there are a precedence $\preceq_{\mathcal{F}}$ and a status st such that for each rule $l \to r$, the inequality $r \prec_{rpo} l$ holds.

A program which is ordered by $\prec_{rpo}$ terminates and one of the key point of our work (see [29] and [32] for dependency pairs) is to use termination proof as a mask to capture algorithmic patterns.

3 Quasi-interpretations

3.1 Definition of Quasi-interpretations

An assignment of a symbol $b \in \mathcal{F} \bigcup \mathcal{C}$ of arity n is a function $(\!|b|\!) : (\mathbb{R}^+)^n \to \mathbb{R}^+$.

An assignment satisfies the *subterm property* if for any $i = 1, n$ and any $X_1, \cdots, X_n$ in $\mathbb{R}^+$, we have

$$(\!|b|\!)(X_1, \cdots, X_n) \geq X_i$$

An assignment is *weakly monotone* 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 $X_1, \cdots, X_n, Y_1, \cdots, Y_n$ of $\mathbb{R}$ with $X_i \leq Y_i$, we have $(\!|b|\!)(X_1, \cdots, X_n) \leq (\!|b|\!)(Y_1, \cdots, Y_n)$.

We extend assignment $(\!|-|\!)$ to terms canonically. Given a term t with m variables, the assignment $(\!|t|\!)$ is a function $(\mathbb{R}^+)^m \to \mathbb{R}^+$ defined by the rules:

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

where X is a fresh variable ranging over reals.

Definition 3 (Quasi-interpretation). *A quasi-interpretation $(\!|-|\!)$ of a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$ is a weakly monotonic assignment satisfying the subterm property such that for each rule $l \to r \in \mathcal{R}$, and for every constructor substitution σ*

$$(\!|l\sigma|\!) \geq (\!|r\sigma|\!)$$

3.2 Polynomial and Additive Quasi-Interpretations

Definition 4. *Given a semi-ring $\mathbb{K}$, let **Max-Poly**$\{\mathbb{K}\}$ be the set of functions defined to be constant functions over $\mathbb{K}$, projections,* max, $+$, $\times$ *and closed by composition. An assignment $(\!|-|\!)$ is said to be polynomial if for each symbol $b \in \mathcal{F} \bigcup \mathcal{C}$, $(\!|b|\!)$ is a function in **Max-Poly**$\{\mathbb{R}^+\}$. A quasi-interpretation $(\!|-|\!)$ is polynomial if the assignment $(\!|-|\!)$ is polynomial.*

Now, say that an assignment of a symbol b of arity $n > 0$ is additive if

$$(\!|b|\!)(X_1, \cdots, X_n) = \sum_{i=1}^{n} X_i + \alpha \text{ with } \alpha \geq 1$$

An assignment $(\!|-|\!)$ of a program $\mathtt{p}$ is additive if $(\!|-|\!)$ is polynomial and each constructor symbol of $\mathtt{p}$ has an additive assignment. *A program is additive if it admits a quasi-interpretation which is an additive assignment.*

Example 1. Consider the program which computes the logarithm function and described by the following rules:

$$\begin{array}{rl|rl} \mathtt{log}(\mathbf{0}) & \to \mathbf{0} & \mathtt{half}(\mathbf{0}) & \to \mathbf{0} \\ \mathtt{log}(\mathbf{S}(\mathbf{0})) & \to \mathbf{0} & \mathtt{half}(\mathbf{S}(\mathbf{0})) & \to \mathbf{0} \\ \mathtt{log}(\mathbf{S}(\mathbf{S}(y))) & \to \mathbf{S}(\mathtt{log}(\mathbf{S}(\mathtt{half}(y)))) & \mathtt{half}(\mathbf{S}(\mathbf{S}(y))) & \to \mathbf{S}(\mathtt{half}(y)) \end{array}$$

It admits the following additive quasi-interpretation:

$$\begin{aligned}(\!|\mathbf{0}|\!) &= 0\\ (\!|\mathbf{S}|\!)(X) &= X+1\\ (\!|\mathtt{log}|\!)(X) = (\!|\mathtt{half}|\!)(X) &= X\end{aligned}$$

3.3 Key properties

Proposition 1. *Assume that $(\!|-|\!)$ is an additive quasi-interpretation of a program $\mathtt{p}$.*

1. *For any terms u and v such that $u \xrightarrow{*} v$, we have $(\!|u|\!) \geq (\!|v|\!)$*
2. *There is a constant k such that $|t| \leq (\!|t|\!) \leq k \times |t|$.*
3. *For any term u and any constructor term $t \in \mathcal{T}(\mathcal{C})$, if $u \xrightarrow{*} t$, we have $|t| \leq (\!|u|\!)$.*

Proof.

1. A context is a particular term that we write $\mathsf{C}[\diamond]$ where $\diamond$ is a new variable. The substitution of $\diamond$ in $\mathsf{C}[\diamond]$ by a term t is noted $\mathsf{C}[t]$.
   The proof goes by induction on the derivation length n. For this, suppose that $u = u_0 \to \ldots \to u_n = v$. If $n = 0$ then the result is immediate. Otherwise $n > 0$ and in this case, there is a rule $\mathtt{f}(p_1, \cdots, p_n) \to t$ and a constructor substitution σ such that $u_0 = \mathsf{C}[\mathtt{f}(p_1, \cdots, p_n)\sigma]$ and $u_1 = \mathsf{C}[t\sigma]$. Since $(\!|-|\!)$ is a quasi-interpretation, we have $(\!|t\sigma|\!) \leq (\!|\mathtt{f}(p_1, \cdots, p_n)\sigma|\!)$. The weak monotonicity property (1) implies that $(\!|\mathsf{C}[t\sigma]|\!) \leq (\!|\mathsf{C}[\mathtt{f}(p_1, \cdots, p_n)\sigma]|\!)$. We conclude by induction hypothesis.
2. The second point is poved by induction on the size of a term t.
3. The last one is a consequence of the the two first assertions.

□

From the above results, we deduce a polynomial upper bound on the size of the values computed by function symbols of an additive program.

Lemma 1 (Fundamental Lemma). *Assume that $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$ is a program admitting an additive quasi-interpretation $(\!|-|\!)$. There is a polynomial P such that for any term t which has n variables $x_1, \cdots, x_n$ and for any ground substitution σ such that $x_i\sigma = v_i$, we have*

$$|[\![t\sigma]\!]| \leq P^{|t|}(\max_{i=1..n} |v_i|)$$

where $P^1(X) = P(X)$ and $P^{k+1}(X) = P(P^k(X))$.

A proof of the Lemma is written in [30]. This result is a consequence of the combination of the above proposition. Notice that the complexity bound just depends on the inputs and not on the term t which is of fixed size.

Theorem 1 ([30]). *The set of functions computed by additive programs ordered by $\prec_{rpo}$ where each function symbol has a product status is exactly the set of functions computable in polynomial time.*

The proof is fully written in [30, 7]. It relies on the fundamental Lemma, combined with a memoization technique á la Jones [24] using a call-by-value with cache for the functions computable in polynomial time, which comes from the seminal work of Cook [12].

Theorem 2 ([6]). *The set of functions computed by additive programs ordered by $\prec_{rpo}$ is exactly the set of functions computable in polynomial space.*

4 Constructor-sharing and Disjoint unions

Two programs $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle$ and $\langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ are constructor-sharing if

$$\mathcal{F}_1 \cap \mathcal{F}_2 = \mathcal{F}_1 \cap \mathcal{C}_2 = \mathcal{F}_2 \cap \mathcal{C}_1 = \emptyset$$

In other words, two programs are constructor-sharing if their only shared symbols are constructor symbols. The constructor-sharing union of two programs $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle$ and $\langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ is defined as the program

$$\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle \bigsqcup \langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle = \langle \mathcal{X}_1 \cup \mathcal{X}_2, \mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{R}_1 \cup \mathcal{R}_2 \rangle$$

Notice that the semantics for constructor-sharing union is defined since the confluence is modular for constructor sharing unions of left-linear systems [36].

Two programs $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle$ and $\langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ are disjoint if they are constructor-sharing and $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$. The disjoint union of two disjoint programs $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle$ and $\langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ is a special case of constructor-sharing union noted $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle \biguplus \langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$.

In [27], Kurihara and Ohuchi show that simple termination is a modular property of constructor-sharing programs. As a consequence:

Proposition 2 (Modularity of $\prec_{rpo}$). *Assume that $\mathtt{p}_1$ and $\mathtt{p}_2$ are two programs ordered by $\prec_{rpo}$ then $\mathtt{p}_1 \bigsqcup \mathtt{p}_2$ is also ordered by $\prec_{rpo}$, with the same status.*

It is easy to establish that QIs are modular for disjoint union:

Proposition 3. *The property of having an additive quasi-interpretation is modular w.r.t disjoint union. In other words, given two disjoint programs $\mathtt{p}_1$ and $\mathtt{p}_2$, the programs $\mathtt{p}_1$ and $\mathtt{p}_2$ admit a quasi-interpretation iff $\mathtt{p}_1 \biguplus \mathtt{p}_2$ admits a quasi-interpretation.*

Proof. Given two programs $\mathtt{p}_1$ and $\mathtt{p}_2$, if $\mathtt{p}_1 \biguplus \mathtt{p}_2$ has a quasi-interpretation $(\!|-|\!)$, then $(\!|-|\!)$ is a quasi-interpretation of $\mathtt{p}_1$ and $\mathtt{p}_2$. Conversely, suppose that $\mathtt{p}_1$ and $\mathtt{p}_2$ have respective QIs $(\!|-|\!)_1$ and $(\!|-|\!)_2$ and define $(\!|-|\!)$ by $\forall b \in \mathcal{C}_i \cup \mathcal{F}_i$, $(\!|b|\!) = (\!|b|\!)_i$. It is a routine to check that $(\!|-|\!)$ is a QI for $\mathtt{p}_1 \biguplus \mathtt{p}_2$. □

However we are showing a negative result for the modularity of quasi-interpretations in the case of constructor-sharing union:

Proposition 4. *The property of having an additive quasi-interpretation is not a modular property w.r.t. constructor-sharing union.*

Proof. We exhibit a counter-example. We consider two programs $\mathtt{p}_0$ and $\mathtt{p}_1$. Both constructor sets $\mathcal{C}_0$ and $\mathcal{C}_1$ are taken to be $\{\mathbf{a}, \mathbf{b}\}$, $\mathcal{F}_0 = \{\mathtt{f}_0\}$ and $\mathcal{F}_1 = \{\mathtt{f}_1\}$. Rules are defined respectively by:

$$\begin{array}{l|l} \mathtt{f}_0(\mathbf{a}(x)) \to \mathtt{f}_0(\mathtt{f}_0(x)) & \mathtt{f}_1(\mathbf{b}(x)) \to \mathtt{f}_1(\mathtt{f}_1(x)) \\ \mathtt{f}_0(\mathbf{b}(x)) \to \mathbf{a}(\mathbf{a}(\mathtt{f}_0(x))) & \mathtt{f}_1(\mathbf{a}(x)) \to \mathbf{b}(\mathbf{b}(\mathtt{f}_1(x))) \end{array}$$

$\mathtt{p}_0$ and $\mathtt{p}_1$ admit the respective additive quasi-interpretations $(\!|-|\!)_0$ and $(\!|-|\!)_1$ defined by:

$$\begin{array}{l|l} (\!|\mathbf{a}|\!)_0(X) = X + 1 & (\!|\mathbf{a}|\!)_1(X) = X + 2 \\ (\!|\mathbf{b}|\!)_0(X) = X + 2 & (\!|\mathbf{b}|\!)_1(X) = X + 1 \\ (\!|\mathtt{f}_0|\!)_0(X) = X & (\!|\mathtt{f}_1|\!)_1(X) = X \end{array}$$

Ad absurdum, we prove that $\mathtt{p}_0 \bigsqcup \mathtt{p}_1$ admits no additive quasi-interpretation. Suppose that it admits the additive quasi-interpretation $(\!|-|\!)$. Since $(\!|-|\!)$ is additive, let $(\!|\mathbf{a}|\!)(X) = X + k_{\mathbf{a}}$ and $(\!|\mathbf{b}|\!)(X) = X + k_{\mathbf{b}}$, with $k_{\mathbf{a}}, k_{\mathbf{b}} \geq 1$. For the simplicity of the proof, suppose that the polynomial $(\!|\mathtt{f}_0|\!)$ can be written without max operation. For the first rule of $\mathtt{p}_0$, $(\!|\mathtt{f}_0|\!)$ has to verify the following inequality:

$$(\!|\mathtt{f}_0|\!)(X + k_{\mathbf{a}}) \geq (\!|\mathtt{f}_0|\!)((\!|\mathtt{f}_0|\!)(X))$$

Now, write $(\!|\mathtt{f}_0|\!)(X) = \alpha X^d + Q(X)$, where Q is a polynomial of degree strictly smaller than d. Observe that $(\!|\mathtt{f}_0|\!)(X + k_{\mathbf{a}})$ is of the shape $\alpha X^d + R(X)$, where R is a polynomial of degree strictly smaller than d, and that $(\!|\mathtt{f}_0|\!)((\!|\mathtt{f}_0|\!)(X))$ is of the shape $\alpha^2 X^{d^2} + S(X)$, where S is a polynomial of degree strictly smaller than d^2. For X large enough, the inequality above yields the following inequalities $d \geq d^2$ which gives $d = 1$. So, we can compare leading coefficient, $\alpha \geq \alpha^2$. So that, $\alpha = 1$ and, in conclusion, $(\!|\mathtt{f}_0|\!)(X) = X + k$. By symmetry, the same result holds for $(\!|\mathtt{f}_1|\!)(X) = X + k'$.

Now the last two rules imply the inequalities:

$$\begin{aligned} k_{\mathbf{b}} + k &\geq 2k_{\mathbf{a}} + k \\ k_{\mathbf{a}} + k' &\geq 2k_{\mathbf{b}} + k' \end{aligned}$$

Consequently, $k_{\mathbf{a}} = k_{\mathbf{b}} = 0$, which is a contradiction with the requirement that $k_{\mathbf{a}}, k_{\mathbf{b}} \geq 1$. □

We give an example of $\prec_{rpo}$ ordered programs with non modular QIs:

Example 2 (code and decode). Consider the program $\mathtt{p} = \mathtt{p}_1 \bigsqcup \mathtt{p}_2$ obtained by constructor sharing union of the two respective programs $\mathtt{p}_1$ and $\mathtt{p}_2$:

$$\begin{aligned}
\mathtt{code_1}(\mathbf{0}(x)) &\to \mathbf{1}(\mathtt{code_1}(x))\\
\mathtt{code_1}(\mathbf{nil}) &\to \mathbf{nil}\\
\mathtt{shuffle_1}(\mathbf{1}(x), \mathbf{1}(y)) &\to \mathbf{1}(\mathbf{1}(\mathtt{shuffle_1}(x, y)))\\
\mathtt{shuffle_1}(\mathbf{1}(x), \mathbf{0}(y)) &\to \mathbf{1}(\mathtt{shuffle_1}(x, \mathtt{code_1}(\mathbf{0}(y))))\\
\mathtt{shuffle_1}(\mathbf{nil}, y) &\to \mathtt{code_1}(y)
\end{aligned}$$

$$\begin{aligned}
\mathtt{code_0}(\mathbf{1}(x)) &\to \mathtt{f}(\mathbf{0}(\mathbf{0}(\mathtt{code_0}(x))))\\
\mathtt{code_0}(\mathbf{nil}) &\to \mathbf{nil}\\
\mathtt{f}(\mathbf{0}(\mathbf{0}(x))) &\to \mathbf{0}(x)\\
\mathtt{shuffle_0}(\mathbf{0}(x), \mathbf{0}(y)) &\to \mathbf{0}(\mathbf{0}(\mathtt{shuffle_0}(x, y)))\\
\mathtt{shuffle_0}(\mathbf{0}(x), \mathbf{1}(y)) &\to \mathbf{0}(\mathtt{shuffle_0}(x, \mathtt{code_0}(\mathbf{1}(y))))\\
\mathtt{shuffle_0}(\mathbf{nil}, y) &\to \mathtt{code_0}(y)
\end{aligned}$$

The programs $\mathtt{p}_1$ and $\mathtt{p}_2$ admit the following respective QIs $(\!|-|\!)_1$ and $(\!|-|\!)_2$:

$$\begin{aligned}
(\!|\mathbf{nil}|\!)_1 &= 0\\
(\!|\mathbf{0}|\!)_1(X) &= X + 1\\
(\!|\mathbf{1}|\!)_1(X) &= X + 1\\
(\!|\mathtt{code_1}|\!)_1(X) &= X\\
(\!|\mathtt{shuffle_1}|\!)_1(X, Y) &= X + Y
\end{aligned}
\qquad
\begin{aligned}
(\!|\mathbf{nil}|\!)_2 &= 0\\
(\!|\mathbf{0}|\!)_2(X) &= X + 1\\
(\!|\mathbf{1}|\!)_2(X) &= X + 2\\
(\!|\mathtt{f}|\!)_2(X) &= X\\
(\!|\mathtt{code_0}|\!)_2(X) &= X\\
(\!|\mathtt{shuffle_0}|\!)_2(X, Y) &= X + Y
\end{aligned}$$

Ad absurdum, suppose that $\mathtt{p}_1 \bigsqcup \mathtt{p}_2$ has a QI $(\!|-|\!)$ and that $(\!|\mathbf{1}|\!)(X) = X + k_{\mathbf{1}}$ and $(\!|\mathbf{0}|\!)(X) = X + k_{\mathbf{0}}$, with $k_{\mathbf{1}}, k_{\mathbf{0}} \geq 1$. The rules for $\mathtt{shuffle_0}$ and $\mathtt{shuffle_1}$

$$\mathtt{shuffle_i}(\mathbf{i}(x), \mathbf{j}(y)) \to \mathbf{i}(\mathtt{shuffle_i}(x, \mathtt{code_i}(\mathbf{j}(y))))\text{with } \mathbf{i}, \mathbf{j} \in \{\mathbf{0}, \mathbf{1}\},\ \mathbf{i} + \mathbf{j} = \mathbf{1}$$

force the QI of $\mathtt{code_0}$ and respectively $\mathtt{code_1}$ to be at most $X + d$ for some constant d. Indeed, by Definition of QIs, we have:

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

Consequently, the above program cannot have any polynomial QI $(\!|\mathtt{shuffle_i}|\!)$ if $(\!|\mathtt{code_i}|\!)(X)$ is greater than $X + d$, for some constant d. The first rules of $\mathtt{p}_1$ and $\mathtt{p}_2$ give the following inequalities:

$$\begin{aligned}
(\!|\mathtt{code_1}(\mathbf{0}(x))|\!) = X + d + k_{\mathbf{0}} &\geq X + d + k_{\mathbf{1}} = (\!|\mathbf{1}(\mathtt{code_1}(x))|\!) && \text{For } \mathtt{p}_1\\
(\!|\mathtt{code_0}(\mathbf{1}(x))|\!) = X + d + k_{\mathbf{1}} &\geq (\!|\mathtt{f}(\mathbf{0}(\mathbf{0}(\mathtt{code_0}(x))))|\!) && \text{For } \mathtt{p}_2\\
&\geq X + d + 2 \times k_{\mathbf{0}} && \text{Subterm prop.}
\end{aligned}$$

Combining the two inequalities, we obtain that $k_{\mathbf{0}} \geq 2 \times k_{\mathbf{0}}$ which is not compatible with the requirement that $k_{\mathbf{0}} \geq 1$, so that $\mathtt{p}$ has no QI. Finally, notice that both programs are $\prec_{rpo}$ terminating with lexicographic status.

At first glance, this negative result seems to be the dead end in our will to split programs into sub-programs in order to obtain complexity bound certificates. However, and this is really surprising, even if the constructor-sharing union does not admit a quasi-interpretation, the complexity bounds remain correct:

Proposition 5. *Given $\mathtt{p}_1 = \langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle$ and $\mathtt{p}_2 = \langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$, two programs having an additive quasi-interpretation. Then, the Fundamental Lemma holds:*

There is a polynomial P such that for any term t of $\mathtt{p}_1 \bigsqcup \mathtt{p}_2$ which has n variables $x_1, \cdots, x_n$, and for any ground substitution σ such that $x_i\sigma = v_i$:

$$|[\![t\sigma]\!]| \leq P^{|t|}(\max_{i=1..n} |v_i|)$$

Proof. The Fundamental Lemma holds for both $\mathtt{p}_1$ and $\mathtt{p}_2$ with respective polynomials P_1 and P_2, take $P = \max(P_1, P_2)$. Consider any values $v_1, \cdots, v_n \in \mathcal{T}(\mathcal{C}_1 \cup \mathcal{C}_2)$. Observe that the evaluation of $\mathtt{f}(v_1, \cdots, v_n) \xrightarrow{*} v$, for some $\mathtt{f} \in \mathcal{F}_j$, is performed using only rules of the program $\langle \mathcal{X}_j, \mathcal{C}_j, \mathcal{F}_j, \mathcal{R}_j \rangle$. As a consequence, the Fundamental Lemma 1 holds for such a term and we have $|v| \leq P(\max_{i=1..n} |v_i|)$. We end the proof by induction on the structure of the term t. The case $t = x$ is trivial. For $t = \mathtt{g}(t_1, \cdots, t_n)$, use the remark above together with the fact that the composition of polynomials is polynomially bounded. □

Together with the fact that $\prec_{rpo}$ is modular, Proposition 5 implies:

Theorem 3 (TIME and SPACE for constructor-sharing union).

- *The set of functions computed by constructor-sharing union of additive programs ordered by $\prec_{rpo}$ where each function symbol has a product status is exactly the set of functions computable in polynomial time.*
- *The set of functions computed by constructor-sharing union of additive programs ordered by $\prec_{rpo}$ is exactly the set of functions computable in polynomial space.*

In order to certify program, the modular approach of QI has two advantages. First, this allows to analyse more programs as the counter-example built for Proposition 3's proof shows it. In fact, there are several meaningful examples based on coding/encoding procedures which are now captured, but which were not previously, by dividing the QI analyzing on subprograms of the original one. So, the time/space characterization that we have established, is intentionally more powerful than the previous ones. Second, it gives rise to an interesting strategy for synthetizing quasi-interpretations, which consists in dividing a program into two sub-programs having disjoint sets of function symbols, and iterating this division as much as possible.

5 Hierarchical union

Two programs $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle$ and $\langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ are hierarchical if

$$\mathcal{F}_1 \cap \mathcal{F}_2 = \mathcal{F}_2 \cap \mathcal{C}_1 = \emptyset \text{ and } \mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset \text{ and } \mathcal{F}_1 \cap \mathcal{C}_2 \neq \emptyset$$

where symbols of $\mathcal{F}_1$ do not appear in patterns of $\mathcal{R}_2$. Their hierarchical union is defined as the program:

$$\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle \ll \langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle = \langle \mathcal{X}_1 \cup \mathcal{X}_2, \mathcal{C}_1 \cup \mathcal{C}_2 - \mathcal{F}_1, \mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{R}_1 \cup \mathcal{R}_2 \rangle$$

Notice that the hierarchical union is no longer a commutative operation in contrast to constructor-sharing union. Indeed, the program $\langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ is calling function symbols of the program $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle$ and the converse does not hold. In other words, the hierarchical union of programs corresponds to a program which can load and execute libraries.

The hypothesis that patterns in $\mathcal{R}_2$ are over $\mathcal{C}_2 - \mathcal{F}_1$ symbols entails that there is no critical pair. Consequently,confluence is a modular property of hierarchical union and the semantics is well defined.

Proposition 6 (Modularity of $\prec_{rpo}$). *Assume that $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle$ is a program ordered by $\prec_{rpo}$ with a status function st_1 and a precedence $\preceq_{\mathcal{F}_1}$ and $\langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ is a program ordered by $\prec_{rpo}$ with a status function st_2 and a precedence $\preceq_{\mathcal{F}_2}$, then $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle \ll \langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ is ordered by $\prec_{rpo}$ with a status function st and a precedence $\preceq_{\mathcal{F}_1 \cup \mathcal{F}_2}$ defined as follows:*

$$\begin{array}{lll} st(f) = st_i(f) & f \in \mathcal{F}_i & i \in \{1,2\} \\ g \preceq_{\mathcal{F}_1 \cup \mathcal{F}_2} f & \text{if } g \preceq_{\mathcal{F}_i} f & i \in \{1,2\} \\ g \prec_{\mathcal{F}_1 \cup \mathcal{F}_2} f & \text{if } f \in \mathcal{F}_2 \text{ and } g \in \mathcal{F}_1 \cap \mathcal{C}_2 & \end{array}$$

Since the constructor-sharing union is a particular case of hierarchical union (i.e. taking $\mathcal{F}_1 \cap \mathcal{C}_2 = \emptyset$), the following holds:

Proposition 7. *The property of having an additive quasi-interpretation is not a modular property w.r.t. hierarchical union.*

Moreover contrarily to what happened with constructor-sharing union, the Fundamental Lemma does not hold. That is why we separate both cases. Here is a counter-example:

Example 3. The programs $\mathtt{p}_1$ and $\mathtt{p}_2$ are given by the rules:

$$\begin{array}{c|c} \mathtt{d}(\mathbf{S}(x)) \to \mathbf{S}(\mathbf{S}(\mathtt{d}(x))) & \exp(\mathbf{S}(x)) \to \mathtt{d}(\exp(x)) \\ \mathtt{d}(\mathbf{0}) \to \mathbf{0} & \exp(\mathbf{0}) \to \mathbf{S}(\mathbf{0}) \end{array}$$

$\mathtt{p}_1$ and $\mathtt{p}_2$ are ordered by $\prec_{rpo}$ with product status and admit the following additive quasi-interpretations:

$$\begin{array}{c|c} (\!|\mathbf{0}|\!)_1 = 0 & (\!|\mathbf{0}|\!)_2 = 0 \\ (\!|\mathbf{S}|\!)_1(X) = X + 1 & (\!|\mathbf{S}|\!)_2(X) = (\!|\mathtt{d}|\!)_2(X) = X + 1 \\ (\!|\mathtt{d}|\!)_1(X) = 2 \times X & (\!|\exp|\!)_2(X) = X \end{array}$$

$\mathtt{d}$ can be viewed as a constructor symbol whose quasi-interpretation is additive in $\mathtt{p}_2$ whereas it is a function symbol whose quasi-interpretation is affine in $\mathtt{p}_1$. The exponential comes from the distinct kinds of polynomial allowed for $\mathtt{d}$.

From now on, some restrictions which allow to preserve the Fundamental Lemma are established. In order to avoid the previous counter-example, we put restriction on the shape of the polynomials allowed for the quasi-interpretations of the shared symbols in a criteria called Kind preserving.

For that purpose, first define an **honest polynomial** to be a polynomial whose coefficients are greater than 1. By extension, define the QI $(\!|-|\!)$ to be honest if $(\!|b|\!)$ is honest for every symbol b. Honest polynomials are very common in practice because of the subterm property.

Given n variables $X_1, \cdots, X_n$ and n natural numbers $a_1, \cdots, a_n$, define a monomial m to be a polynomial of one term, of the shape $m(X_1, \cdots, X_n) = X_1^{a_1} \times \ldots \times X_n^{a_n}$ where some $a_j \neq 0$. Given a monomial m and a polynomial P, define $m \sqsubseteq P$ iff $P = \sum_{j=1}^{n} \alpha_j \times m_j$, with α_j constants and m_j pairwise distinct monomials, and there is $i \in \{1, n\}$ s.t. $m_i = m$ and $\alpha_i \neq 0$. The coefficient α_i, also noted $\mathtt{coef}_P(m)$, is defined to be the multiplicative coefficient associated to m in P.

Definition 5 (Kind preserving). *Assume that $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle \ll \langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ is the hierarchical union of two programs with respective polynomial QIs $(\!|-|\!)_1$ and $(\!|-|\!)_2$. We say that $(\!|-|\!)_1$ and $(\!|-|\!)_2$ are Kind preserving if $\forall b \in \mathcal{C}_2 \cap \mathcal{F}_1$:*

1. *$(\!|b|\!)_1$ and $(\!|b|\!)_2$ are honest polynomials*
2. *$\forall m$, $m \sqsubseteq (\!|b|\!)_1 \Leftrightarrow m \sqsubseteq (\!|b|\!)_2$*
3. *$\forall m$, $\mathtt{coef}_{(\!|b|\!)_2}(m) = 1 \Leftrightarrow \mathtt{coef}_{(\!|b|\!)_1}(m) = 1$*

*Two Kind preserving QIs $(\!|-|\!)_1$ and $(\!|-|\!)_2$ are called **additive Kind preserving** if the following conditions are satisfied:*

- *$(\!|b|\!)_1$ is additive for every $b \in \mathcal{C}_1$,*
- *$(\!|b|\!)_2$ is additive for every $b \in \mathcal{C}_2 - \mathcal{F}_1$.*

Notice that the QIs $(\!|-|\!)_1$ and $(\!|-|\!)_2$ of example 3 are not additive Kind preserving because of the symbol $\mathtt{d}$ which admits" $(\!|\mathtt{d}|\!)_1(X) = 2 \times X$ and $(\!|\mathtt{d}|\!)_2(X) = X + 1$.

Consequently, an interesting restriction for preserving the Fundamental Lemma might be to force the quasi-interpretations of a hierarchical union to be additive Kind preserving. However, this restriction is not enough as illustrated by the following program:

Example 4. Consider the following respective programs $\mathtt{p}_1$ (on the left) and $\mathtt{p}_2$:

$$\mathtt{g}(t) \to \mathbf{S}(\mathbf{S}(t)) \qquad \left| \qquad \begin{array}{l} \mathtt{f}(\mathbf{S}(x), \mathbf{0}, t) \to \mathtt{f}(x, t, t) \\ \mathtt{f}(x, \mathbf{S}(z), t) \to \mathtt{f}(x, z, \mathtt{g}(t)) \\ \mathtt{f}(\mathbf{0}, \mathbf{0}, t) \to t \end{array} \right.$$

Their hierarchical union $\mathtt{p}_1 \ll \mathtt{p}_2$ computes an exponential function. Using the notation $\underline{n}$ for $\underbrace{\mathbf{S}(\ldots \mathbf{S}(\mathbf{0}) \ldots)}_{n \text{ times } \mathbf{S}}$, we have $[\![\mathtt{f}]\!](\underline{n}, \underline{m}, \underline{p}) = \underline{3^n \times (2 \times m + p)}$.

$\mathtt{p}_1$ and $\mathtt{p}_2$ are ordered by $\prec_{rpo}$ with lexicographic status and admit the following additive Kind preserving quasi-interpretations:

$$\begin{array}{c|c} (\!|\mathbf{S}|\!)_1(X) = X+1 & (\!|\mathbf{0}|\!)_2 = 0 \\ (\!|\mathtt{g}|\!)_1(X) = X+2 & (\!|\mathbf{S}|\!)_2(X) = (\!|\mathtt{g}|\!)_2(X) = X+1 \\ & (\!|\mathtt{f}|\!)_2(X,Y,Z) = \max(X,Y,Z) \end{array}$$

The problem of the above counter-example comes directly from the fact that the number of alternations between rules of both programs used during the evaluation depends on the inputs. A way to deal with Kind preserving QIs is to bound the number of alternations by some constant. For that purpose, we also put some syntactic restrictions over the considered programs, considering a notion of flat programs introduced by Dershowitz in [17], where it was used in order to ensure modularity of completeness of hierarchical unions.

Definition 6 (Flat program). *A term is flat if it has no nesting of function symbols. In other words, a flat term is a term without composition of function symbols. By extension, a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$ is flat if, for every rule $\mathtt{f}(p_1, \cdots, p_n) \to e$ of $\mathcal{R}$, e is a flat term.*

Definition 7 (Stratified union). *The hierarchical union $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle \ll \langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ is called stratified union if*

- *for all rule $\mathtt{f}(p_1, \cdots, p_n) \to e$ in $\mathcal{R}_2$, we have: For each $\mathtt{g}(e_1, \cdots, e_n)$ subterm of e such that $\mathtt{g} \approx_{\mathcal{F}_2} \mathtt{f}$, no (shared) function symbols of $\mathcal{C}_2 \cap \mathcal{F}_1$ occurs in the arguments $e_1, \cdots, e_n$ of $\mathtt{g}$.*
- *The program $\mathtt{p}_2$ is flat*

Given the hierarchical union $\langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle \ll \langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ of two programs and a term t, we say that that t is evaluated using k alternations between rules of $\mathcal{R}_1$ and $\mathcal{R}_2$ if there are some terms $u_1, \cdots, u_k, v_1, \cdots, v_k$ s.t.

$$t \xrightarrow{*}_2 u_1 \xrightarrow{*}_1 v_1 \ldots \xrightarrow{*}_2 u_k \xrightarrow{*}_1 v_k$$

where v_k is a normal form and $\xrightarrow{*}_i$ denotes a sequence of rewriting rules in $\mathcal{R}_i$.

We first establish a Lemma defining a particular evaluation strategy for our programs:

Lemma 2. *Given the stratified union $p_1 \ll \mathtt{p}_2$ of the program $\mathtt{p}_1 = \langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle$ and the flat program $\mathtt{p}_2 = \langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$, for every function symbol $\mathtt{f}$ of arity n and every values $v_1, \cdots, v_n$, $\mathtt{f}(v_1, \cdots, v_n)$ can be evaluated using a constant number of alternations between rules of $\mathcal{R}_1$ and rules of $\mathcal{R}_2$.*

Proof. We are going to prove the Lemma using a particular rewrite strategy. Since confluence is a modular property of hierarchical union, as mentioned at the beginning of Section 5, we can evaluate the program in such a way.

First, we define a rank function from function symbols $\mathcal{F}$ to natural numbers $\mathbb{N}$ which satisfies

$$\begin{aligned}
&\mathrm{rk}_{\mathcal{F}}(\mathtt{f}) = 0 && \text{if } \nexists \mathtt{g} \in \mathcal{F}, \text{ s.t. } \mathtt{f} >_{\mathcal{F}} \mathtt{g}\\
&\mathrm{rk}_{\mathcal{F}}(\mathtt{f}) = \max(\mathrm{rk}_{\mathcal{F}}(\mathtt{g})) + 1 && \text{if } \forall \mathtt{g} \in \mathcal{F} \text{ s.t. } \mathtt{f} >_{\mathcal{F}} \mathtt{g}\\
&\mathrm{rk}_{\mathcal{F}}(\mathtt{f}) = \mathrm{rk}_{\mathcal{F}}(\mathtt{g}) && \text{if } \mathtt{f} \approx_{\mathcal{F}} \mathtt{g}
\end{aligned}$$

The proof is by induction on the rank $\mathrm{rk}_{\mathcal{F}_2}$:

- If $\mathtt{f} \in \mathcal{F}_1$ then the result is obvious and without alternation.
- Now suppose that $\mathtt{f} \in \mathcal{F}_2$, we are going to show by induction that a function symbol $\mathtt{f}$ in $\mathcal{F}_2$ of rank k can be evaluated using at most $k+1$ alternations:
  - If $\mathrm{rk}_{\mathcal{F}_2}(\mathtt{f}) = 0$ then every function symbol $\mathtt{g}$ appearing in the right hand side of a rule defining the function symbol $\mathtt{f}$ is an equivalent fonction symbol for the precedence $\geq_{\mathcal{F}_2}$. Consequently, by definition of stratified union, the evaluation of $\mathtt{f}$ can be performed, using only rules of $\mathcal{R}_2$. We first evaluate all these recursive calls. At the end either an error occurs or we obtain a value in $\mathcal{T}(\mathcal{C}_1 \cup \mathcal{C}_2)$. Now, since there are no longer function symbols in $\mathcal{F}_2$, we finish the evaluation with the function symbols of $\mathcal{F}_1$, applying only rules in $\mathcal{R}_1$. Finally, we have performed the evaluation using only one alternation.
  - Now suppose that for $\mathrm{rk}_{\mathcal{F}_2}(\mathtt{g}) \leq n$, the evaluation of $\mathtt{g}(u_1, \cdots, u_k)$ can be performed using at most $n+1$ alternations and take $\mathtt{f}$ s.t. $\mathrm{rk}_{\mathcal{F}_2}(\mathtt{f}) = n+1$. Since we consider a stratified union, every recursive call of the shape $\mathtt{h}(v_1, \cdots, v_m)$ with $v_1, \cdots, v_m$ values and $\mathtt{h} \approx_{\mathcal{F}_2} \mathtt{f}$ can be performed using only rules in $\mathcal{R}_2$. The flat condition ensures that function symbols of $\mathcal{F}_2$ are not composed in the recursive calls. Indeed a recursive composition of function symbols in $\mathcal{F}_2$ can lead to an unbounded number of alternations. Consequently, we can evaluate every recursive call of a function symbol $\mathtt{h}$ s.t. $\mathtt{h} \approx_{\mathcal{F}_2} \mathtt{f}$ using only rules in $\mathcal{R}_2$. Now it remains to evaluate the function symbols of rank strictly smaller than $n+1$. For that purpose, we evaluate their arguments first. Since the program is flat, this evaluation is done using only rules of $\mathcal{R}_1$, so that we have a first alternation. Now we apply the induction hypothesis, adding $n+1$ more alternations, evaluating all these symbols in parallel which is possible since there is no composition, and eliminating all the remaining function symbols in $\mathcal{F}_2$. It only remains function symbols in $\mathcal{F}_1$ that we evaluate using only rules in $\mathcal{R}_1$. Notice that this last evaluation does not add any alternation. Finally, we have evaluated a function symbol of rank $n+1$ using at most $n+2$ alternations of the rules in $\mathtt{p}_1$ and $\mathtt{p}_2$.

It remains to see that the rank of a function symbol is bounded by the size of the program and we obtain the required result. □

Definition 8 (Extension). *Given the hierarchical union of two programs $\mathtt{p}_1$ and $\mathtt{p}_2$ having respective QIs $(\!|-|\!)_1$ and $(\!|-|\!)_2$, define the extensions of the quasi-interpretation $(\!|-|\!)_1$ (resp. $(\!|-|\!)_2$), that we also note $(\!|-|\!)_1$ (resp. $(\!|-|\!)_2$), by the following rules $\forall b \in \mathcal{C}_2 \cup \mathcal{F}_2 \backslash (\mathcal{C}_1 \cup \mathcal{F}_1)$ (resp. $\mathcal{C}_1 \cup \mathcal{F}_1 \backslash (\mathcal{C}_2 \cup \mathcal{F}_2)$) $(\!|b|\!)_1 =_{def} (\!|b|\!)_2$ (resp. $(\!|b|\!)_2 =_{def} (\!|b|\!)_1$).*

The extensions of $(\!|-|\!)_1$ *and* $(\!|-|\!)_2$ *are defined over all terms of* $\mathtt{p}_1 \ll \mathtt{p}_2$*. Notice that these extensions preserve the fact that* $(\!|-|\!)_1$ *and* $(\!|-|\!)_2$ *are Kind preserving QIs.*

Lemma 3. *Given the hierarchical union of two programs* $\mathtt{p}_1 \ll \mathtt{p}_2$ *having Kind preserving QIs* $(\!|-|\!)_1$ *and* $(\!|-|\!)_2$*, then there exist two polynomials* P *and* Q *s.t. for every term* w *of* $\mathtt{p}_1 \ll \mathtt{p}_2$*, the extensions of* $(\!|-|\!)_1$ *and* $(\!|-|\!)_2$ *satisfy:*

- $(\!|w|\!)_1 \leq P((\!|w|\!)_2)$
- $(\!|w|\!)_2 \leq Q((\!|w|\!)_1)$

Proof. We exhibit the polynomial P, the result follows by symmetry of the Kind preserving condition for the extensions of $(\!|-|\!)_1$ and $(\!|-|\!)_2$. Define α to be the smallest multiplicative coefficient strictly greater than 1 of the polynomials $(\!|b|\!)_2$ (if there is no such a coefficient, then Definition 5 implies that the quasi-interpretations are similar, as in previous section) and β to be the greatest multiplicative and additive coefficient of the polynomials $(\!|b|\!)_1$, for every symbol b in $\mathcal{T}(\mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{X}_1 \cup \mathcal{X}_2)$. Now we define four new assignments $(\!|-|\!)_\alpha$, $(\!|-|\!)_{\alpha=1}$, $(\!|-|\!)_\beta$ and $(\!|-|\!)_{\beta=1}$:

- $(\!|-|\!)_\alpha$ is defined from $(\!|-|\!)_2$ by replacing every multiplicative coefficient distinct from 1 by α and every additive coefficient by 1.
- $(\!|-|\!)_{\alpha=1}$ is defined from $(\!|-|\!)_\alpha$ by replacing every multiplicative coefficient distinct from 1 by 1.
- $(\!|-|\!)_\beta$ is defined from $(\!|-|\!)_1$ by replacing every multiplicative and additive coefficient distinct from 1 by β.
- $(\!|-|\!)_{\beta=1}$ is defined from $(\!|-|\!)_\beta$ by replacing every multiplicative and additive coefficient distinct from 1 by 1.

Intuitively, $(\!|-|\!)_\alpha$ and $(\!|-|\!)_\beta$ represent respectively a lower bound on $(\!|-|\!)_2$ and an upper bound on $(\!|-|\!)_1$. We can show by structural induction that for any ground term w, we have:

$$(\!|w|\!)_\alpha \leq (\!|w|\!)_2 \qquad \text{By Definition of } (\!|-|\!)_\alpha \qquad (1)$$

$$(\!|w|\!)_1 \leq (\!|w|\!)_\beta \qquad \text{By Definition of } (\!|-|\!)_\beta \qquad (2)$$

$$(\!|w|\!)_{\beta=1} = (\!|w|\!)_{\alpha=1} \qquad \text{By Condition 2 of Definition 5} \qquad (3)$$

Now, consider α (repsectively β) as a variable. $(\!|w|\!)_\alpha$ (resp. $(\!|w|\!)_\beta$) can be seen as a polynomial in α (resp. β) (Even if the degree of the polynomial can depend on the size of the term w). Now suppose that $(\!|w|\!)_\beta$ is a polynomial of degree d in β. Since $\beta \geq 1$ by definition of QI, for every $k \leq d$, $\beta^k \leq \beta^d$. Write

$(\!|w|\!)_\beta = \sum_{i=1}^{d} \gamma_i \beta^i$, for some constants γ_i. We have:

$$
\begin{aligned}
(\!|w|\!)_\beta &= \sum_{i=1}^{d} \gamma_i \beta^i && && (4)\\
&\leq (\sum_{i=1}^{d} \gamma_i) \times \beta^d && \text{Since } \beta \geq 1 && (5)\\
&= (\!|w|\!)_{\beta=1} \times \beta^d && \text{By Definition of } (\!|-|\!)_{\beta=1} && (6)\\
&= (\!|w|\!)_{\alpha=1} \times \beta^d && \text{By Inequality (3)} && (7)
\end{aligned}
$$

Define p to be the highest degree of the polynomials $(\!|b|\!)_1$ and e be the degree of $(\!|w|\!)_\alpha$ in α. We are going to show by induction on the structure of the term w that e ,d and p are linked throught the following inequality $d \leq p \times e + 1$:

- if w is a constant symbol (i.e. of arity 0), $(\!|w|\!)_\alpha = (\!|w|\!)_\beta = 0$. Consequently, $d = e = 0$ and the inequality is satisfied.
- Now suppose that $w = h(t_1, \cdots, t_n)$ with d_j the degree of $(\!|t_j|\!)_\beta$, and e_j the degree of $(\!|t_j|\!)_\alpha$. By induction hypothesis, we have $d_j \leq p \times e_j + 1$. Now suppose that $(\!|h|\!)_\beta(X_1, \cdots, X_n) = \sum_{j_1 \leq d_1 \ldots j_n \leq d_n} \beta \times X_1^{j_1} \ldots X_n^{j_n}$. Let $(i_1, \cdots, i_n)$ be the the indices in the polynomial $(\!|h|\!)_\beta$ where the degree $d = \sum_{j=1}^{n} d_j i_j + 1$ is reached.

$$
\begin{aligned}
d &\leq \sum_{j=1}^{n} (p \times e_j + 1) i_j + 1 = p \sum_{j=1}^{n} e_j \times i_j + \sum_{j=1}^{n} i_j + 1 && \text{By I.H.}\\
&\leq p \sum_{j=1}^{n} e_j \times i_j + p + 1 = p \times (\sum_{j=1}^{n} e_j \times i_j + 1) + 1 && \text{By definition of } p\\
&\leq p \times e + 1 && \text{By definition of } e
\end{aligned}
$$

Taking this result into account in inequality (7), we obtain:

$$(\!|w|\!)_\beta \leq (\!|w|\!)_{\alpha=1} \times \beta^{p \times e + 1} = (\!|w|\!)_{\alpha=1} \times \beta (\beta^p)^e \tag{8}$$

We take z such that $\alpha^z \geq \beta^p$ and define the polynomial P by $P(X) = \beta \times X^{z+1}$. Notice that such a z exists since $\alpha > 1$ and that the polynomial P does not depend on the term w but depends on the coefficients of the QIs.

It remains to check that $(\!|w|\!)_1 \leq P((\!|w|\!)_2)$:

$$
\begin{aligned}
(\!|w|\!)_1 &\leq (\!|w|\!)_{\alpha=1} \times \beta(\beta^p)^e && \text{By (8)}\\
&\leq (\!|w|\!)_\alpha \times \beta(\beta^p)^e && \text{Since } \alpha > 1\\
&\leq (\!|w|\!)_\alpha \times \beta(\alpha^z)^e = (\!|w|\!)_\alpha \times \beta(\alpha^e)^z && \text{By definition of } z\\
&\leq (\!|w|\!)_\alpha \times \beta((\!|w|\!)_\alpha)^z) = P((\!|w|\!)_\alpha) && \text{Since } \alpha^e \leq (\!|w|\!)_\alpha\\
&\leq P((\!|w|\!)_2) && \text{By (1)}
\end{aligned}
$$

□

Proposition 8. *Given the stratified union* $\mathtt{p}_1 \ll \mathtt{p}_2$ *of two programs* $\mathtt{p}_1 = \langle \mathcal{X}_1, \mathcal{C}_1, \mathcal{F}_1, \mathcal{R}_1 \rangle$ *and* $\mathtt{p}_2 = \langle \mathcal{X}_2, \mathcal{C}_2, \mathcal{F}_2, \mathcal{R}_2 \rangle$ *having additive Kind preserving quasi-interpretations* $(\!|-|\!)_1$ *and* $(\!|-|\!)_2$*, then, the Fundamental Lemma holds:*

There is a polynomial P *such that for any term* t *of* $\mathtt{p}_1 \ll \mathtt{p}_2$ *which has* n *variables* $x_1, \cdots, x_n$*, and for any ground substitution* σ *such that* $x_i\sigma = v_i$*:*

$$|[\![t\sigma]\!]| \leq P^{|t|}(\max_{i=1..n} |v_i|)$$

Proof. The proof is by structural induction on a term t.

(i) Consider the base case where $t = \mathtt{f}(x_1, \ldots, x_k)$ for some variables x_i. We use the strategy of evaluation described in Lemma 2. So, the term is computed within ℓ alternation: $\mathtt{f}(v_1, \cdots, v_n) = w_0 \xrightarrow{*}_2 u_1 \xrightarrow{*}_1 w_1 \ldots \xrightarrow{*}_2 u_l \xrightarrow{*}_1 w_\ell$, with ℓ bounded by the rank.
We are going to show by iduction on $m \leq l$ that there is a polynomial R such that $(\!|w_m|\!)_1 \leq R^m(P((\!|w_0|\!)_2))$ (I.H.).
The base case $m = 0$ is trivial. Define $R = Q \circ P$ with P and Q the polynomials of Lemma 3. We have:

$$\begin{array}{lll}
(\!|w_{m+1}|\!)_2 & \leq Q((\!|w_{m+1}|\!)_1) & \text{By Lemma 3} \\
& \leq Q((\!|u_{m+1}|\!)_1) & \text{Proposition 1} \\
& \leq Q(P((\!|u_{m+1}|\!)_2)) & \text{By Lemma 3} \\
& \leq Q(P((\!|w_m|\!)_2)) & \text{Proposition 1} \\
& \leq R(R^m(P((\!|w_0|\!)_2))) & \text{By I.H.} \\
& = R^{m+1}(P((\!|w_0|\!)_2)) &
\end{array}$$

Applying Fundamental Lemma to $\mathtt{p}_2$, we get a polynomial S. Then, $(\!|w_0|\!)_2 \leq S(\max_{i=1..n} |v_i|)$.So,

$$|w_\ell| \leq (\!|w_\ell|\!)_2 \leq R^\ell(P(S(\max_{i=1..n} |v_i|)))$$

We conclude taking $P(X) = R^\ell \circ P \circ S$.

(ii) The induction steps follows by composition of the polynomials.

And we obtain the required result. □

Example 5. Consider the following programs $\mathtt{p}_1$ and $\mathtt{p}_2$:

$$\begin{array}{r|l}
\mathtt{d}(\mathbf{S}(x)) \to \mathbf{S}(\mathbf{S}(\mathtt{d}(x))) & \mathtt{sq}(\mathbf{S}(x)) \to \mathbf{S}(\mathtt{add}(\mathtt{sq}(x), \mathtt{d}(x))) \\
\mathtt{d}(\mathbf{0}) \to \mathbf{0} & \mathtt{sq}(\mathbf{0}) \to \mathbf{0} \\
\hline
\mathtt{add}(\mathbf{S}(x), y) \to \mathbf{S}(\mathtt{add}(x, y)) & \\
\mathtt{add}(\mathbf{0}, y) \to y &
\end{array}$$

Their hierarchical union $\mathtt{p}_1 \ll \mathtt{p}_2 = \langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{R} \rangle$ computes the square of a unary number given as input. For the precedence $\geq_\mathcal{F}$,we have $\mathtt{sq} >_\mathcal{F} \{\mathtt{add}, \mathtt{d}\}$. Moreover the program $\mathtt{p}_2$ is flat since there is no composition of function symbols

in its rules. Consequently, $\mathtt{p}_1 \ll \mathtt{p}_2$ is a stratified union, since the argument of the recursive call $\mathtt{sq}(x)$ is a variable. Both $\mathtt{p}_1$ and $\mathtt{p}_2$ are ordered by $\prec_{rpo}$ with product status. Define the following quasi-interpretations $(\!|-|\!)_1$ and $(\!|-|\!)_2$ by:

$$\begin{array}{c|c} (\!|\mathbf{0}|\!)_1 = 0 & (\!|\mathbf{0}|\!)_2 = 0 \\ (\!|\mathbf{S}|\!)_1(X) = X + 1 & (\!|\mathbf{S}|\!)_2(X) = X + 1 \\ (\!|\mathtt{d}|\!)_1(X) = 3 \times X & (\!|\mathtt{d}|\!)_2(X) = 2 \times X \\ (\!|\mathtt{add}|\!)_1(X,Y) = X + Y & (\!|\mathtt{add}|\!)_2(X,Y) = X + Y + 1 \\ & (\!|\mathtt{sq}|\!)_2(X) = 2 \times X^2 \end{array}$$

$(\!|-|\!)_1$ and $(\!|-|\!)_2$ are additive Kind preserving QIs, so that the program $\mathtt{p}_1 \ll \mathtt{p}_2$ computes values whose size is polynomially bounded by the inputs size. (Both QI of $\mathtt{d}$ illustrate the fact that they can be equal up to a multiplivative constant if the coefficient is > 1.)

Moreover the program division can be iterated on $\mathtt{p}_1$ by separating the rules for function symbols $\mathtt{add}$ and $\mathtt{d}$, thus obtaining a constructor-sharing union.

Theorem 4 (TIME and SPACE for hierarchical union of Kind preserving QIs). *The set of functions computed by a hierarchical union of two programs $\mathtt{p}_1$ and $\mathtt{p}_2$ such that*

1. *$\mathtt{p}_1 \ll \mathtt{p}_2$ is a stratified union,*
2. *$\mathtt{p}_1$ and $\mathtt{p}_2$ admit the respective additive Kind preserving QIs $(\!|-|\!)_1$ and $(\!|-|\!)_2$,*
3. *$\mathtt{p}_1$ and $\mathtt{p}_2$ are ordered by $\prec_{rpo}$ and each function symbol has a product status,*

is exactly the set of functions computable in polynomial time.

Moreover, if condition (3) is replaced by: $\mathtt{p}_1$ and $\mathtt{p}_2$ are ordered by $\prec_{rpo}$ then we characterize exactly the class of polynomial space functions.

Proof. Again, this result is a consequence of the fact that we have the Fundamental Lemma in Proposition 8 and the $\prec_{rpo}$ ordering with product status, still using a call-by-value with cache. □

6 Application to higher-order programs

Resource control of higher-order programs by QI is not straight forward, because we should deal at first glance with higher-order assignments. However, higher-order mechanisms can be reduced to an equivalent first order functional program by *defunctionalization*, which was introduced by Reynolds [37]. Defunctionalization consists in two steps. First, a new constructor symbol is substituted to every higher-order function declaration. Second, each function application is eliminated by introducing a new function symbol for application. We refer to [15] which investigates works related to Defunctionalization and Continuation-Passing Style and gives a lot of references. Other higher-order removal techniques are treated, among others, by Wadler [40], Goguen [19] and Chin and Darlington [11].

This part is an application to modular approach to get QI as it is described in the previous Section. So, we are not formalizing higher-order programs but we rather illustrate the key concepts

Example 6. Suppose that $\mathtt{g}$ is defined by a program $\mathtt{q}_3$. Consider the following higher-order program $\mathtt{p}$.

$$\begin{aligned}
\mathtt{fold}(\lambda x.\mathtt{f}(x), \mathbf{nil}) &\to \mathbf{0}\\
\mathtt{fold}(\lambda x.\mathtt{f}(x), \mathbf{c}(x,l)) &\to \mathtt{f}(\mathtt{fold}(\lambda x.\mathtt{f}(x), l))\\
\mathtt{h}(l) &\to \mathtt{fold}(\lambda x.\mathtt{g}(x), l) \qquad \text{for } \mathtt{g} \text{ dfn in } \mathtt{q}_3
\end{aligned}$$

$\mathtt{h}(l)$ iterates g such that $[\![\mathtt{h}]\!](l) = [\![\mathtt{g}]\!]^n(0)$ where n is the number of elements in the list l. From $\mathtt{p}$, we obtain $\hat{\mathtt{p}}$ by defunctionalization:

$$\mathtt{q}_1 = \begin{cases} \hat{\mathtt{fold}}(\mathbf{nil}) \to \mathbf{0} \\ \hat{\mathtt{fold}}(\mathbf{c}(x,l)) \to \mathtt{app}(\mathbf{c}_0, \hat{\mathtt{fold}}(l)) & \mathbf{c}_0 \text{ is a new constructor} \\ \hat{\mathtt{h}}(l) \to \hat{\mathtt{fold}}(l) \end{cases}$$

$$\mathtt{q}_2 = \{\, \mathtt{app}(\mathbf{c}_0, x) \to \mathtt{g}(x)$$

We are now able to use QI to higher-order programs by considering their first-order transformations.

Theorem 5 (TIME and SPACE for higher-order programs).

- *The set of functions computed by a higher-order program $\mathtt{p}$ such that the defunctionalization $\hat{\mathtt{p}}$ of $\mathtt{p}$ admits an additive quasi-interpretation and is ordered by $\prec_{rpo}$ in which each function symbol has a product status is exactly the set of functions computable in polynomial time.*
- *The set of functions computed by a higher-order program $\mathtt{p}$ such that the defunctionalization $\hat{\mathtt{p}}$ of $\mathtt{p}$ admits an additive quasi-interpretation and is ordered by $\prec_{rpo}$ is exactly the set of functions computable in polynomial space.*

In fact, the above example illustrates the fact that a defunctionalized program $\hat{p}$ is divided into three parts: the programs $\mathtt{q}_1$ and $\mathtt{q}_2$ above and a program $\mathtt{q}_3$ which computes $\mathtt{g}$. Notice that the hierarchical union of $\mathtt{q}_2 \ll \mathtt{q}_1$ is stratified. Moreover they admit the following additive Kind preserving QIs:

$$\begin{array}{rl|rl}
(\!|\hat{\mathtt{fold}}|\!)_1(X) &= (\!|\mathtt{h}|\!)_1(X) = X & (\!|\mathtt{g}|\!)_2(X) &= X+1\\
(\!|\mathbf{c}|\!)_1(X,Y) &= X+Y+1 & (\!|\mathbf{c}_0|\!)_2 &= 0\\
(\!|\mathbf{0}|\!)_1 &= (\!|\mathbf{c}_0|\!)_1 = (\!|\mathbf{nil}|\!)_1 = 0 & (\!|\mathtt{app}|\!)_2(X,Y) &= X+Y+1\\
(\!|\mathtt{app}|\!)_1(X,Y) &= X+Y+1 & &
\end{array}$$

Now, the results on modularity that we have previously established, allow us to give a sufficient condition on the QI of $\mathtt{g}$ defined in $\mathtt{q}_3$, in order to guarantee that the computation remains polynomially bounded. Indeed, Proposition 8 implies that $(\!|\mathtt{g}|\!)_3$ should be Kind preserving. That is, $(\!|\mathtt{g}|\!)_3(X) = (\!|\mathtt{g}|\!)_2(X)+\alpha = X+\alpha+1$, where α is some constant. Notice that $(\!|\mathtt{g}|\!)_2$ is forced by $(\!|\mathtt{app}|\!)_2$, and on the other hand $(\!|\mathtt{app}|\!)_2$ is forced by $(\!|\mathtt{app}|\!)_1$.

Example 7. Consider the following program $\mathtt{p}$, which visits a list l in continuation passing style.

$$\begin{aligned}
\mathtt{visit}(\mathbf{nil}, \lambda x.\mathtt{f}(x), y) &\to \mathtt{f}(y)\\
\mathtt{visit}(\mathbf{c}(x,l), \lambda x.\mathtt{f}(x), y) &\to \mathtt{visit}(l, \lambda x.\mathtt{g}_1(\mathtt{f}(x)), y)\\
\mathtt{h}(l) &\to \mathtt{visit}(l, \lambda x.\mathtt{g}_0(x), 0)
\end{aligned}$$

where $\mathtt{g}_0$ and $\mathtt{g}_1$ are defined by some program $\mathtt{q}_3$ which admits a additive QI $(\!|\_|\!)_3$. We have $[\![\mathtt{h}]\!](l) = [\![\mathtt{g}_1]\!]^n([\![\mathtt{g}_0]\!](0))$ where n is the number of elements in the list l. We obtain $\hat{\mathtt{p}}$

$$\mathtt{q}_1 = \begin{cases} \hat{\mathtt{visit}}(\mathbf{nil}, k, y) \to \mathtt{app}(k, y) \\ \hat{\mathtt{visit}}(\mathbf{c}(x, l), k, y) \to \hat{\mathtt{visit}}(l, \mathbf{c}_1(k), y) \\ \mathtt{h}(l) \to \hat{\mathtt{visit}}(l, \mathbf{c}_0, 0) \qquad \mathbf{c}_0 \text{ and } \mathbf{c}_1 \text{ are new const.} \end{cases}$$

$$\mathtt{q}_2 = \begin{cases} \mathtt{app}(\mathbf{c}_0, x) \to \mathtt{g}_0(x) \\ \mathtt{app}(\mathbf{c}_1(k), x) \to \mathtt{g}_1(\mathtt{app}(k, x)) \end{cases}$$

The hierarchical union $\mathtt{q}_2 \ll \mathtt{q}_1$ is stratified and admit the following additive Kind preserving QIs:

$$\begin{array}{rl|rl} (\!|\hat{\mathtt{visit}}|\!)_1(X) = (\!|\mathtt{h}|\!)_1(X) = X + 1 & & (\!|\mathtt{g}_1|\!)_2(X) = (\!|\mathtt{g}_0|\!)_2(X) = X + 1 \\ (\!|\mathbf{c}_1|\!)_1(X, Y) = (\!|\mathbf{c}|\!)_1(X, Y) = X + Y + 1 & & (\!|\mathbf{c}_1|\!)_2(X) = X + 1 \\ (\!|\mathbf{0}|\!)_1 = (\!|\mathbf{c}_0|\!)_1 = (\!|\mathbf{nil}|\!)_1 = 0 & & (\!|\mathbf{0}|\!)_2 = (\!|\mathbf{c}_0|\!)_2 = 0 \\ (\!|\mathtt{app}|\!)_1(X, Y) = X + Y + 1 & & (\!|\mathtt{app}|\!)_2(X, Y) = X + Y + 1 \end{array}$$

Now, suppose that we have two QI $(\!|\mathtt{g}_0|\!)_3$ and $(\!|\mathtt{g}_1|\!)_3$, which are two resource certificates for $\mathtt{g}_0$ and $\mathtt{g}_1$ wrt $\mathtt{q}_3$. Proposition 8 states that we are sure to remain polynomial if the QI of $\mathtt{g}_0$ and $\mathtt{g}_1$ are kind preserving. In other words, $(\!|\mathtt{g}_0|\!)_3(X) = X + \alpha$ and $(\!|\mathtt{g}_1|\!)_3(X) = X + \beta$ for some constants α and β.

So, a modular approach is a way to predict safely and efficiently if we can apply a function in an higher-order computational mechanisms.

Finally, we state the following characterizations:

Theorem 6 (Modularity and higher-order programs). *The set of functions computed by a higher-order programs $\mathtt{p}$ such that the defunctionalization $\hat{\mathtt{p}}$ is defined by hierarchical union $\mathtt{p}_1 \ll p_2$ of two programs $\mathtt{p}_1$ and $\mathtt{p}_2$ satisfying:*

1. *$\mathtt{p}_1 \ll \mathtt{p}_2$ is a stratified union,*
2. *$\mathtt{p}_1$ and $\mathtt{p}_2$ admit the respective additive Kind preserving QIs $(\!|-|\!)_1$ and $(\!|-|\!)_2$,*
3. *$\mathtt{p}_1$ and $\mathtt{p}_2$ are ordered by $\prec_{rpo}$ and each function symbol has a product status,*

is exactly the set of functions computable in polynomial time.

Moreover, if condition (3) is replaced by: $\mathtt{p}_1$ and $\mathtt{p}_2$ are ordered by $\prec_{rpo}$ then we characterize exactly the class of polynomial space functions.

7 Conclusion

One goal of this paper was to solve the problem of resource certificates, when they come attached to some programs. The question is how to recombine those certificates when programs are loaded. So we provide sufficient conditions from which a resource bound is guaranteed. In particular, we have shown that this approach was relevant for defunctionalized higher-order programs. We think that

another interesting direction is to consider mobile code in a functional programming setting, which could be a fragment of Boudol's ULM [10].

Acknowledgments. The authors would like to thank C. Kirchner, for introducing the issue of Modularity and O. Danvy, for giving helpful comments on Defunctionalization, and the support of CRISS project.

References

1. R. Amadio. Max-plus quasi-interpretations. In Martin Hofmann, editor, *Typed Lambda Calculi and Applications, 6th International Conference, TLCA 2003, Valencia, Spain, June 10-12, 2003, Proceedings*, volume 2701 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2003.
2. R. Amadio, S. Coupet-Grimal, S. Dal-Zilio, and L. Jakubiec. A functional scenario for bytecode verification of resource bounds. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic, 18th International Workshop, CSL 13th Annual Conference of the EACSL, Karpacz, Poland*, volume 3210 of *Lecture Notes in Computer Science*, pages 265–279. Springer, 2004.
3. R. Amadio and S. Dal-Zilio. Resource control for synchronous cooperative threads. In *Concur*, pages 68–82, 2004.
4. D. Aspinall and A. Compagnoni. Heap Bounded Assembly Language. *Journal of Automated Reasoning (Special Issue on Proof-Carrying Code)*, 31:261–302, 2003.
5. P. Baillot and K. Terui. A feasible algorithm for typing in elementary affine logic. In Springer, editor, *TLCA*, volume 3461 of *Lecture Notes in Computer Science*, pages 55–70, 2005.
6. G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. On lexicographic termination ordering with space bound certifications. In *PSI 2001, Akademgorodok, Novosibirsk, Russia, Ershov Memorial Conference*, volume 2244 of *Lecture Notes in Computer Science*. Springer, Jul 2001.
7. G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-interpretation a way to control resources. *Submitted to Theoretical Computer Science*, 2005. `http://www.loria.fr/~marionjy`.
8. G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-interpretations and small space bounds. In Jürgen Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2005.
9. G. Bonfante, J.-Y. Marion, J.-Y. Moyen, and R. Péchoux. Synthesis of quasi-interpretations. *Workshop on Logic and Complexity in Computer Science, LCC2005, Chicago*, 2005. `http://www.loria/~pechoux`.
10. G. Boudol. Ulm, a core programming model for global computing. In *ESOP conference*, number 2986 in Lecture Notes in Computer Science, pages 234–248, 2004.
11. W. N. Chin and J. Darlington. A higher-order removal method. *Lisp Symb. Comput.*, 9(4):287–322, 1996.
12. S. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18(1):4–18, January 1971.
13. P. Coppola and S. Ronchi Della Rocca. Principal typing for lambda calculus in elementary affine logic. *Fundamenta Informaticae*, 65(1-2):87–112, 2005.

14. S. Dal-Zilio and R. Gascon. Resource bound certification for a tail-recursive virtual machine. In *APLAS 2005 – 3rd Asian Symposium on Programming Languages and Systems*, volume 3780 of *Lecture Notes in Computer Science*, pages 247–263. Springer-Verlag, November 2005.
15. O. Danvy. An analytical approach to programs as data objects, 2006. Doctor Scientarum degree in Computer Science. BRICS. Departement of Computer Science. University of Aarhus.
16. N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.
17. N. Dershowitz. Hierachical termination. conditional and typed rewriting systems, 4th international workshop, ctrs-94, jerusalem, israel, july 13-15, 1994, proceedings. In *CTRS*, volume 968 of *Lecture Notes in Computer Science*. Springer, 1995.
18. J.-Y. Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998. prsent à LCC'94, LNCS 960.
19. J. A. Goguen. Higher-order functions considered unnecessary for higher-order programming. In D. A. Turner, editor, *Research Topics in Functional Programming*, pages 309–351. Addison-Welsey, 1990.
20. B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. In *In Proceedings of the Third International Conference on Algebraic and Logic Programming*, volume 632, pages 53–68. Berlin: Springer Verlag, 1992.
21. M. Hofmann. A type system for bounded space and functional in-place update. *Nordic Journal of Computing*, 7(4):258–289, 2000.
22. M. Hofmann. The strength of Non-Size Increasing computation. In *Proceedings of POPL'02*, pages 260–269, 2002.
23. G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.
24. N. D. Jones. *Computability and complexity, from a programming perspective*. MIT press, 1997.
25. S. Kamin and J-J Lévy. Attempts for generalising the recursive path orderings. Technical report, Univerity of Illinois, Urbana, 1980. Unpublished note. Accessible on http://perso.ens-lyon.fr/pierre.lescanne/not_accessible.html.
26. J.W. Klop. Term rewriting systems. In D. Gabbay S. Abramsky and T. Maibaum, editors, *Handbook of logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, 1992.
27. M. Kurihara and A. Ohuchi. Modularity of simple termination of term rewriting systems with shared constructors. *Theoretical Computer Science*, 103:273–282, 1992.
28. Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318:163–180, 2004.
29. J.-Y. Marion. Analysing the implicit complexity of programs. *Information and Computation*, 183:2–18, 2003.
30. J.-Y. Marion and J.-Y. Moyen. Efficient first order functional program interpreter with time bound certifications. In Michel Parigot and Andrei Voronkov, editors, *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France*, volume 1955 of *Lecture Notes in Computer Science*, pages 25–42. Springer, Nov 2000.
31. J.-Y. Marion and J.-Y. Moyen. Heap analysis for assembly programs. Technical report, Loria, 2006.
32. J.-Y. Marion and R. Pechoux. Resource analysis by sup-interpretation. In M. Hagiya and P. Wadler, editors, *Functional and Logic Programming: 8th In-*

ternational Symposium, FLOPS 2006, volume 3945 of *Lecture Notes in Computer Science*, pages 163–176, 2006.
33. A. Middeldorp. *Modular properties of term rewriting Systems*. PhD thesis, Vrije Universiteit te Amsterdam, 1990.
34. J.-Y. Moyen. *Analyse de la complexité et transformation de programmes*. Thèse d'université, Nancy 2, Dec 2003.
35. K.-H. Niggl and H. Wunderlich. Certifying polynomial time and linear/polynomial space for imperative programs. *SIAM Journal on Computing*. to appear.
36. M.R.K. Krishna Rao. Modular proofs of completeness of hierarchical term rewriting systems. *Theoretical Computer Science*, 151:487–512, 1995.
37. J. C. Reynolds. Definitional interpreters for higher-order programming languages. In *ACM '72: Proceedings of the ACM annual conference*, pages 717–740. ACM Press, 1972.
38. Y. Toyama. Counterexamples for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.
39. Y. Toyama. On the church-rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.
40. P. Wadler. Deforestation: Transforming programs to eliminate trees. In *ESOP '88. European Symposium on Programming, Nancy, France, 1988 (Lecture Notes in Computer Science, vol. 300)*, pages 344–358. Berlin: Springer-Verlag, 1988.