Higher-order Interpretations and Program Complexity

Patrick Baillot[a], Ugo Dal Lago[b]

[a] *Laboratoire d'Informatique du Parallélisme, Université de Lyon, CNRS, Ecole Normale Supérieure de Lyon, INRIA, Université Claude Bernard Lyon 1*
[b] *Università di Bologna & INRIA*

Abstract

Polynomial interpretations and their generalizations like quasi-interpretations have been used in the setting of first-order functional languages to design criteria ensuring statically some complexity bounds on programs [10]. This fits in the area of implicit computational complexity, which aims at giving machine-free characterizations of complexity classes. In this paper, we extend this approach to the higher-order setting. For that we consider simply-typed term rewriting systems [35], we define higher-order polynomial interpretations for them, and we give a criterion ensuring that a program can be executed in polynomial time. In order to obtain a criterion flexible enough to validate interesting programs using higher-order primitives, we introduce a notion of polynomial quasi-interpretations, coupled with a simple termination criterion based on linear types and path-like orders.

Keywords: implicit computational complexity, term rewriting systems, type systems, lambda-calculus

1. Introduction

The problem of statically analyzing the performance of programs can be attacked in many different ways. One of them consists in verifying *complexity* properties early in the development cycle, when programs are still expressed in high-level languages, like functional or object-oriented idioms. And in this scenario, results from an area known as *implicit* computational complexity (ICC in the following) can be useful: often, they consist in characterizations of complexity classes in terms of paradigmatic programming languages (recursion schemes

URL: `patrick.baillot@ens-lyon.fr` (Patrick Baillot), `dallago@cs.unibo.it` (Ugo Dal Lago)

This work has been partially supported by ANR Project ELICA ANR-14-CE25-0005 and by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

[30, 8], λ-calculus [31], term rewriting systems [10], etc.) or logical systems (proof-nets, natural deduction, etc.), from which static analysis methodologies can be distilled. Examples are type systems, path-orderings and variations on the interpretation method. The challenge here is defining ICC systems which are not only simple, but also *intensionally* powerful: many natural programs among those with bounded complexity should be recognized as such by the ICC system, i.e., should actually be programs *of* the system.

One of the most fertile direction in ICC is indeed the one in which programs are term rewriting systems (TRS in the following) [10, 11], whose complexity can be kept under control by way of variations of the powerful techniques developed to check termination of TRSs, namely path orderings [20], dependency pairs [33] and the interpretation method [29]. Many different complexity classes have been characterized this way, from polynomial time to polynomial space, to exponential time to logarithmic space. And remarkably, many of the introduced characterizations are intensionally powerful, in particular when the interpretation method is relaxed and coupled with recursive path orderings, like in quasi-interpretations [11]. These techniques can be fruitfully combined into concrete tools (see, e.g., [4])

The cited results indeed represent the state-of-the art in complexity analysis for *first-order* functional programs, i.e. when functions are *not* first-class citizens. If the class of programs of interest includes higher-order functional programs, the techniques above can only be applied when programs are either defunctionalized or somehow put in first-order form, for example by applying a translation scheme due to the second author and Simone Martini [19]. However, it seems difficult to ensure in that case that the target first-order programs satisfy termination criteria such as those used in [11], although some promising results in this direction have been recently obtained [3]. The article [13] proposed to get around this problem by considering a notion of *hierarchical union* of TRSs, and showed that this technique allows to handle some examples of higher-order programs. This approach is interesting but it is not easy to assess its generality, besides particular examples. In the present work we want to switch to a higher-order interpretations setting, in order to provide a more abstract account of such situations.

We thus propose to generalize TRS techniques to systems of higher-order rewriting, which come in many different flavours [26, 28, 35]. The majority of the introduced higher-order generalizations of rewriting are quite powerful but also complex from a computational point of view, being conceived to model not only programs but also proofs involving quantifiers. As an example, even computing the reduct of a term according to a reduction rule can in some cases be undecidable. Higher-order generalizations of TRS techniques [27, 34], in turn, reflect the complexity of the languages on top of which they are defined. Summing up, devising ICC systems this way seems quite hard.

In this paper, we consider one of the simplest higher-order generalizations of TRSs, namely Yamada's simply-typed term rewriting systems [35] (STTRSs in the following), we define a system of higher-order polynomial interpretations [34] for them and prove that, following [10], this allows to exactly characterize the

class of polynomial time computable functions. We show, however, that this way the class of (higher-order) programs which can be given a polynomial interpretation does not include interesting and natural examples, like `foldr`, and that this problem can be overcome by switching to another technique, designed along the lines of quasi-interpretations [11]. This is the subject of sections 4 and 5, which also show how non-trivial examples can be proved polytime this way.

Another problem we address in this paper is related to the expressive power of simply-typed term rewriting systems. Despite their simplicity, simply-typed term rewriting systems subsume the simply-typed λ-calculus and extensions of it with full recursion, like PCF. This can be proved following [19] and is the subject of Section 3.

A preliminary version of this work appeared as a conference paper in [6]. Compared to this short version, the main contributions of the present paper are the following ones:

- detailed proofs of the results (several of them had to be omitted or only sketched in [6] because of space constraints);
- description of translations of typed λ-calculus and PCF into STTRS (Section 3);
- more examples of programs to which one can apply the complexity criterion of higher-order quasi-interpretations (Section 5.6);
- discussion of embeddings of other ICC systems into our setting (Section 6).

2. Simply-Typed Term Rewriting Systems

2.1. Basic Definitions and Notation

We recall here the definition of a simply-typed term rewriting systems (STTRS), following [35, 2]. We will actually consider a subclass of STTRSs, basically the one of those STTRSs whose rules' left hand side consists of a function symbol applied to a sequence of *patterns*. For first-order rewriting systems this corresponds to the notion of a *constructor rewriting system*.

We consider a denumerable set of base types, which we call *data-types*, that we denote with metavariables like D And E. *Types* are defined by the following grammar:

$$A, A_i ::= D \quad \Big| \quad A_1 \times \cdots \times A_n \to A.$$

A *functional type* is a type which contains an occurrence of $\to$. Some examples of base types are the type W_n of strings over an alphabet of n symbols, and the type NAT of tally integers.

We denote by $\mathcal{F}$ the set of *function symbols* (or just *functions*), $\mathcal{C}$ the set of *constructors* and $\mathcal{X}$ the set of *variables*. Constructors $\mathbf{c} \in \mathcal{C}$ have a type of the form $D_1 \times \cdots \times D_n \to D$, for $n \geq 0$. For instance W_n has constructors **empty** of type W_n and $\mathbf{c}_1, \ldots, \mathbf{c}_n$ of type $W_n \to W_n$. Functions $\mathtt{f} \in \mathcal{F}$, on the other hand, can themselves have any functional type. Variables $x \in \mathcal{X}$ can have any type. *Terms* are typed and defined by the following grammar:

$$t, t_i := x^A \quad \Big| \quad \mathbf{c}^A \quad \Big| \quad \mathtt{f}^A \quad \Big| \quad (t^{A_1 \times \cdots \times A_n \to A}\ t_1^{A_1} \ldots t_n^{A_n})^A,$$

where $x^A \in \mathcal{X}$, $\mathbf{c}^A \in \mathcal{C}$, $\mathtt{f}^A \in \mathcal{F}$. We denote by $\mathcal{T}$ the set of all terms. Observe how application is primitive and is in general treated differently from other function symbols. This is what makes STTRSs different from ordinary TRSs. $FV(t)$ is the set of variables occurring in t. t is closed iff $FV(t) = \emptyset$. STTRS, in other words, can be seen as a generalized, typed form of combinatory logic.

To simplify the writing of terms, we often elide their type. We will also write $(t\ \overline{s})$ for $(t\ s_1 \dots s_n)$. Therefore any term t is of the form $(\dots((\alpha\ \overline{s_1})\ \overline{s_2})\dots\overline{s_k})$ where $k \geq 0$ and $\alpha \in \mathcal{X} \cup \mathcal{C} \cup \mathcal{F}$. We will also use the following convention: any term t of the form $(\dots((s\ \overline{s_1})\ \overline{s_2})\dots\overline{s_k})$ will be written $(\!(s\ \overline{s_1}\ \dots\overline{s_k})\!)$ or $(\!(s\ s_{11}\ \dots s_{1n_1}\dots s_{k1}\dots s_{kn_k})\!)$. Observe however that, e.g., if r has type $A_1 \times A_2 \to (B_1 \times B_2 \to B)$, t_i has type A_i for $i \in \{1,2\}$, s_i has type B_i for $i \in \{1,2\}$, then both $(r\ \overline{t})$ and $((r\ \overline{t})\ \overline{s})$ are well-typed (with type $B_1 \times B_2 \to B$ and B, respectively), but $(r\ t_1)$ and $((r\ \overline{t})\ s_1)$ are *not* well-typed. We define the size $|t|$ of a term t as the number of symbols (elements of $\mathcal{F} \cup \mathcal{C} \cup \mathcal{X}$) it contains. We denote $t\{x/s\}$ the substitution of term s for x in t.

A *pattern* is a term generated by the following grammar:

$$p, p_i := x^A \ \Big|\ (\mathbf{c}^{D_1 \times \dots \times D_n \to D}\ p_1^{D_1} \dots p_n^{D_n}).$$

$\mathcal{P}$ is the set of all patterns. Observe that patterns of functional type are necessarily variables. We consider *rewriting rules* in the form $l \to r$, where:

1. l and r are terms of the same type A, $FV(r) \subseteq FV(l)$, and any variable occurs at most once in l;
2. l must have the form $(\!(\mathtt{f}\ p_1 \dots p_k)\!)$ where each p_i for $i \in \{1, \dots, k\}$ is a pattern. The rule is said to be a rule *defining* $\mathtt{f}$, while $k \in \mathbb{N}$ is the *arity* of the rule. Notice that the arity of a rule is univocally defined.

Now, a *simply-typed term rewriting system* (STTRS in the following) is a set R of non-overlapping rewriting rules such that for every function symbol $\mathtt{f}$, every rule defining $\mathtt{f}$ has the same arity, which is said to be the arity *of* $\mathtt{f}$. A *program* $\mathtt{P} = (\mathtt{f}, R)$ is given by a STTRS R and a chosen function symbol $\mathtt{f} \in \mathcal{F}$.

In the next section, a notion of reduction will be given which crucially relies on the concept of a value. More specifically, only values will be passed as arguments to functions. Formally, we say that a term is a *value* if either:

1. it has a type D and is in the form $(\mathbf{c}\ v_1 \dots v_n)$, where $v_1, \dots, v_n$ are themselves values;
2. or it has functional type A and is of the form $(\!(\mathtt{f}\ v_1\ \dots\ v_n)\!)$, where the terms in $v_1, \dots\ v_n$ are themselves values and n is *strictly* smaller than the arity of $\mathtt{f}$.

Condition 2 is reminiscent of the λ-calculus, where an abstraction is a value. Please observe that values are always closed, i.e., they cannot contain variables from $\mathcal{X}$. We denote values as v, u and the set of all values as $\mathcal{V}$.

2.2. Dynamics

The evaluation of terms will be formalized by a rewriting relation. Before giving it, we need to introduce notions of substitution and unification. A substitution σ is a map from variables to values with a finite domain, and such that

$\sigma(x^A)$ has type A. A substitution σ is extended in the natural way to a function from $\mathcal{T}$ to itself, that we shall also write σ. The image of a term t under the substitution σ is denoted $t\sigma$. *Contexts* are defined as terms but with the proviso that they contain exactly one occurrence of a special constant $\bullet^A$ (*hole*) having type A. They are denoted with metavariables like $\mathcal{C}$ or $\mathcal{D}$. If $\mathcal{C}$ is a context with hole $\bullet^A$, and t is a term of type A, then $\mathcal{C}\{t\}$ is the term obtained from $\mathcal{C}$ by replacing the occurrence of $\bullet^A$ by t. Consider a STTRS R. We say that t reduces to s in call-by-value, denoted as $t \to_R s$, if there exist a rule $l \to r$ of R, a context $\mathcal{C}$ and a substitution σ such that $l\sigma$ is a closed term, $t = \mathcal{C}\{l\sigma\}$ and $s = \mathcal{C}\{r\sigma\}$. When there is no ambiguity on R, we simply write $\to$ instead of $\to_R$. Since we are considering non-overlapping systems and a call-by-value reduction, we have the following property:

Proposition 1 (Strong Confluence). *Let R be a STTRS and suppose we have $t \to_R s$ and $t \to_R r$. Then either $s = r$ or there exists q such that $s \to_R q$ and $r \to_R q$.*

Please notice that one of the advantages of STTRSs over similar formalisms (like HORS [26] and CRS [28]) is precisely the simplicity of the underlying unification mechanism, which does not involve any notion of binding and is thus computationally simpler than higher-order matching. There is a price to pay in terms of expressivity, obviously. In the next section, however, we show how STTRSs are expressive enough to capture standard typed λ-calculi.

3. Typed λ-calculi as STTRSs

The goal of this Section is to illustrate the fact that the choice of the STTRS framework as higher-order calculus is not *too* restrictive: indeed we will show that we can simulate in it PCF equipped with *weak reduction* (i.e. where one does not reduce under the scope of abstractions). This is achieved using ideas developed for encodings of the λ-calculus into first-order rewriting systems [19].

3.1. A Few Words About PCF

We assume a total order $\leq$ on the set $\mathcal{X}$ of variables. *PCF types* are defined as follows:

$$A, B ::= NAT \;\Big|\; A \to A.$$

PCF terms, on the other hand, are defined as follows:

$$M, L ::= x^A \;\Big|\; (\lambda x^A.M^B)^{A\to B} \;\Big|\; (M^{A\to B}L^A)^B \;\Big|\; (\mathit{fix}\ M^{(A\to B)\to A\to B})^{A\to B} \;\Big|$$
$$\mathbf{n}^{NAT} \;\Big|\; \mathtt{succ}^{NAT\to NAT} \;\Big|\; \mathtt{pred}^{NAT\to NAT} \;\Big|\; (\mathit{ifz}\ M^A\ L^A)^{NAT\to A},$$

where $\mathbf{n}$ ranges over the natural numbers, and x ranges over $\mathcal{X}$. We omit types in terms whenever this does not cause ambiguity. A *PCF value* is any term different from an application. PCF values are indicated with metavariables like

V. A call-by-value operational semantics for PCF can be expressed by way of some standard reduction rules:

$$\begin{aligned}
(\lambda x.M)V &\longrightarrow M\{V/x\};\\
(\mathit{fix}\ M)V &\longrightarrow M((\mathit{fix}\ M)V);\\
\mathtt{succ}\ \mathbf{n} &\longrightarrow \mathbf{n+1};\\
\mathtt{pred}\ \mathbf{0} &\longrightarrow \mathbf{0};\\
\mathtt{pred}\ \mathbf{n+1} &\longrightarrow \mathbf{n};\\
(\mathit{ifz}\ M\ L)\ \mathbf{0} &\longrightarrow M;\\
(\mathit{ifz}\ M\ L)\ \mathbf{n+1} &\longrightarrow L.
\end{aligned}$$

The reduction rules above can be propagated to any applicative context by the rules below, thereby defining a weak call-by-value semantics:

$$\frac{M \longrightarrow L}{MP \longrightarrow LP} \qquad \frac{M \longrightarrow L}{PM \longrightarrow PL}$$

Example 1. *As an example of a PCF program, consider the following term of type $NAT \to NAT$ computing the factorial of any natural number:*

$$FACT = (\mathit{fix}\ (\lambda f.\lambda y.(\mathit{ifz}\ \mathbf{1}\ (PROD\ (f\ (\mathtt{pred}\ y))\ y))\ y)),$$

where PROD is another program of type $NAT \to NAT \to NAT$ computing the product of two natural numbers. Now, one easily verifies that:

$$\begin{aligned}
FACT\ \mathbf{2} &\to^3 (\mathit{ifz}\ \mathbf{1}\ (PROD\ (FACT\ (\mathtt{pred}\ \mathbf{2}))\ \mathbf{2}))\ \mathbf{2}\\
&\to PROD\ (FACT\ (\mathtt{pred}\ \mathbf{2}))\ \mathbf{2}\\
&\to^3 PROD\ ((\mathit{ifz}\ \mathbf{1}\ (PROD\ (FACT\ (\mathtt{pred}\ \mathbf{1}))\ \mathbf{1}))\ \mathbf{1})\ \mathbf{2}\\
&\to PROD\ (PROD\ (FACT\ (\mathtt{pred}\ \mathbf{1}))\ \mathbf{1})\ \mathbf{2}\\
&\to^3 PROD\ (PROD\ ((\mathit{ifz}\ \mathbf{1}\ (PROD\ (FACT\ (\mathtt{pred}\ \mathbf{0}))\ \mathbf{0}))\ \mathbf{0})\ \mathbf{1})\ \mathbf{2}\\
&\to PROD\ (PROD\ \mathbf{1}\ \mathbf{1})\ \mathbf{2}\\
&\to^* PROD\ \mathbf{1}\ \mathbf{2}\\
&\to^* \mathbf{2}.
\end{aligned}$$

3.2. PCF as a STTRS

PCF can be turned into a STTRS R_{PCF} with infinitely many function symbols. First, for each term M of type B, with free variables $x_1 \le x_2 \le \ldots \le x_n$ of types $A_1, \ldots, A_n$, and possibly x of type A, we introduce a function symbol $\mathtt{abs}_{M,x}$ of $\mathcal{F}$, with type $A_1 \times \cdots \times A_n \to (A \to B)$. Please notice that in $\mathtt{abs}_{M,x}$, as it is a *function symbol*, no variable occurs free, even if M can of course be open. Then, for each pair of terms M, L of type B, both with free variables

among $x_1 \leq x_2 \leq \ldots \leq x_n$ of types $A_1, \ldots, A_n$, we introduce a function $\mathtt{ifz}_{M,L}$ of $\mathcal{F}$, with type $A_1 \times \cdots \times A_n \to (NAT \to B)$. Similarly, for every M of type $(A \to B) \to A \to B$ with free variables $x_1 \leq \ldots \leq x_n$ of types $A_1, \ldots, A_n$, we need a function symbol $\mathtt{fix}_M$, with type $A_1 \times \cdots \times A_n \to (A \to B)$ Finally, we need function symbols for $\mathtt{succ}$, $\mathtt{pred}$. Now, the translation $\langle M \rangle$ is defined by induction on M:

$$\begin{aligned}
\langle x \rangle &= x;\\
\langle M\ L \rangle &= (\langle M \rangle\ \langle L \rangle);\\
\langle \lambda x.M \rangle &= (\mathtt{abs}_{M,x}\ x_1 \ldots x_n),\\
&\qquad \text{if } FV(M) - \{x\} = x_1 \leq \ldots \leq x_n;\\
\langle \mathit{fix}\ M \rangle &= (\mathtt{fix}_M\ x_1 \ldots x_n),\\
&\qquad \text{if } FV(M) = x_1 \leq \ldots \leq x_n;\\
\langle \mathbf{n} \rangle &= (\underbrace{\mathbf{s}\ (\mathbf{s}\ (\ldots (\mathbf{s}}_{n \text{ times}}\ \mathbf{0}) \ldots)))\\
\langle \mathtt{succ} \rangle &= \mathtt{succ};\\
\langle \mathtt{pred} \rangle &= \mathtt{pred};\\
\langle \mathit{ifz}\ M\ L \rangle &= (\mathtt{ifz}_{M,L}\ x_1 \ldots x_n),\\
&\qquad \text{if } FV(M) \cup FV(L) = x_1 \leq \ldots \leq x_n.
\end{aligned}$$

A converse translation $[\cdot]$ can be easily defined. As an example, the following equations hold:

$$\begin{aligned}
[(\mathtt{abs}_{M,x}\ t_1 \ldots t_n)] &= (\lambda x.M)\{[t_1]/x_1, \ldots, [t_n]/x_n\};\\
[(\mathtt{fix}_L\ t_1 \ldots t_m)] &= (\mathit{fix}\ L)\{[t_1]/y_1, \ldots, [t_m]/y_m\};\\
[(\mathtt{ifz}_{P,Q}\ t_1 \ldots t_p)] &= (\mathit{ifz}\ P\ Q)\{[t_1]/z_1, \ldots, [t_p]/z_p\};
\end{aligned}$$

where $x_1 \leq \ldots \leq x_n$ are the free variables of $\lambda x.M$, $y_1 \leq \ldots \leq y_m$ are the free variables of L and $z_1 \leq \ldots \leq z_p$ are the free variables of P and Q. Please observe that defining $[\cdot]$ requires a bit of care: the way an application $(t\ s_1\ \ldots\ s_n)$ is translated back into a PCF term depends on the nature of t.

Lemma 2. *If M is a PCF term of type A, then $\langle M \rangle$ is a well-typed term, of type A.*

Proof. An easy induction on M. □

Reduction rules of the STTRS R_{PCF} are the following:

$$\begin{aligned}
((\mathtt{abs}_{M,x}\ x_1 \dots x_n)\ x) &\to \langle M\rangle;\\
((\mathtt{fix}_M\ x_1 \dots x_n)\ x) &\to ((\langle M\rangle\ (\mathtt{fix}_M\ x_1 \dots x_n))\ x);\\
(\mathtt{succ}\ x) &\to (\mathbf{s}\ x);\\
(\mathtt{pred}\ \mathbf{0}) &\to \mathbf{0};\\
(\mathtt{pred}\ (\mathbf{s}\ x)) &\to x;\\
((\mathtt{ifz}_{M,L}\ x_1 \dots x_n)\ \mathbf{0}) &\to \langle M\rangle;\\
((\mathtt{ifz}_{M,L}\ x_1 \dots x_n)\ (\mathbf{s}\ x)) &\to \langle L\rangle.
\end{aligned}$$

Example 2. *Let us reconsider the term FACT from Example 1. Recall that*

$$FACT = (\mathit{fix}\ (\lambda x.\lambda y.(\mathit{ifz}\ \mathbf{1}\ (\mathit{PROD}\ (x\ (\mathtt{pred}\ y))\ y))\ y)).$$

Now, define the following shortcuts:

$$\begin{aligned}
M &= \lambda x.\lambda y.(\mathit{ifz}\ \mathbf{1}\ (\mathit{PROD}\ (x\ (\mathtt{pred}\ y))\ y))\ y;\\
L &= \lambda y.(\mathit{ifz}\ \mathbf{1}\ (\mathit{PROD}\ (x\ (\mathtt{pred}\ y))\ y))\ y;\\
P &= (\mathit{ifz}\ \mathbf{1}\ (\mathit{PROD}\ (x\ (\mathtt{pred}\ y))\ y))\ y;\\
Q &= \mathit{PROD}\ (x\ (\mathtt{pred}\ y))\ y;
\end{aligned}$$

Among the rules of R_{PCF}, one has, for example,

$$\begin{aligned}
(\mathtt{fix}_M\ x) &\to ((\langle M\rangle\ \mathtt{fix}_M)\ x);\\
(\mathtt{abs}_{L,x}\ x) &\to \langle L\rangle;
\end{aligned}$$

and similar rules for $\mathtt{abs}_{P,y}$ and $\mathtt{ifz}_{\mathbf{1},Q}$. It is then easy to verify that, as an example,

$$\begin{aligned}
\langle \mathit{FACT}\ \mathbf{2}\rangle = (\mathtt{fix}_M\ \langle\mathbf{2}\rangle) &\to ((\langle M\rangle\ \mathtt{fix}_M)\langle\mathbf{2}\rangle) = ((\mathtt{abs}_{L,x}\ \mathtt{fix}_M)\langle\mathbf{2}\rangle)\\
&\to ((\langle L\rangle\{\mathtt{fix}_M/x\})\langle\mathbf{2}\rangle) = ((\mathtt{abs}_{P,y}\ \mathtt{fix}_M)\ \langle\mathbf{2}\rangle)\\
&\to ((\langle P\rangle\{\mathtt{fix}_M/x, \langle\mathbf{2}\rangle/y\} = ((\mathtt{ifz}_{\mathbf{1},Q}\ \mathtt{fix}_M\ \langle\mathbf{2}\rangle)\ \langle\mathbf{2}\rangle)\\
&\to ((\langle Q\rangle\{\mathtt{fix}_M/x, \langle\mathbf{2}\rangle/y\} = ((\langle \mathit{PROD}\rangle\ (\mathtt{fix}_M\ (\mathtt{pred}\ \langle\mathbf{2}\rangle)))\ \langle\mathbf{2}\rangle)\\
&\to ((\langle \mathit{PROD}\rangle\ (\mathtt{fix}_M\ \langle\mathbf{1}\rangle))\ \langle\mathbf{2}\rangle) = ((\langle \mathit{PROD}\rangle\ \langle \mathit{FACT}\ \mathbf{1}\rangle)\ \langle\mathbf{2}\rangle).
\end{aligned}$$

A term of R_{PCF} is said to be *canonical* if it is either a first-order value or a variable or if it is in the form $((\mathtt{f}\ t_1 \dots t_n\ s_1 \dots s_m))$, where $n, m \geq 0$, $n+1$ is the arity of $\mathtt{f}$, $t_1, \dots, t_n$ are values and $s_1, \dots, s_m$ are themselves canonical. The following are technical intermediate results towards Theorem 8:

Lemma 3. *For every PCF term M, $[\langle M\rangle] = M$.*

Proof. By induction on the structure of M. Some interesting cases:

- If $M = \mathit{ifz}\ L\ P$, then

$$\begin{aligned}
[\langle \mathit{ifz}\ L\ P\rangle] &= [(\mathtt{ifz}_{L,P}\ x_1 \dots x_n)]\\
&= (\mathit{ifz}\ L\ P)\{x_1/x_1, \dots, x_n/x_n\} = \mathit{ifz}\ M\ L.
\end{aligned}$$

- If $M = \lambda x.L$, then

$$\begin{aligned}[\langle \lambda x.L \rangle] &= [(\mathtt{abs}_{L,x}\ x_1 \dots x_n)] \\ &= (\lambda x.L)\{x_1/x_1, \dots, x_n/x_n\} = \lambda x.L.\end{aligned}$$

This concludes the proof. □

Lemma 4. *For every closed PCF term M, $\langle M \rangle$ is canonical. Moreover, if t is canonical and $t \to s$, then s is canonical.*

Proof. The fact that $\langle M \rangle$ is always canonical can be proved by induction on the structure of M. Some interesting cases:

- If M is either `succ` or `pred` or $\mathbf{n}$, then $\langle M \rangle$ is a value, and values are always canonical.
- If M is *ifz L P*, then $t = \langle M \rangle$ is in the form $(\mathtt{ifz}_{L,P}\ x_1 \dots x_n)$ and the arity of $\mathtt{ifz}_{L,P}$ is $n+1$. As a consequence, t is canonical.
- If M is $L\ P$, then $t = \langle M \rangle$ is $(\langle L \rangle\ \langle P \rangle)$. By induction hypothesis, $s = \langle L \rangle$ and $r = \langle P \rangle$ are canonical. Moreover, s is of functional type, thus in the form $((\mathtt{f}\ t_1 \dots t_n\ s_1 \dots s_m))$, where $n, m \geq 0$, $n+1$ is the arity of $\mathtt{f}$, $t_1, \dots, t_n$ are values and $s_1, \dots, s_m$ are themselves canonical. Observe, however, that t can be written as

$$((\mathtt{f}\ t_1 \dots t_n\ s_1 \dots s_m\ r)),$$

  which is itself canonical, because r is canonical.

The fact that canonicity is preserved by reduction is a consequence of the adoption of a call-by-value notion of reduction: it is routine to prove that substitution of values for variables in $\langle M \rangle$ ends up in a canonical term. □

Lemma 5. *A closed canonical term t is a normal form iff $[t]$ is a normal form.*

Proof. Again, a simple induction of t, exploiting the following fact: for both PCF and R_{PCF}, the set of closed normal forms coincides with the set of values.

- If t is a first-order value, then it is in normal form and, moreover, also $[t]$ is in normal form.
- t cannot be a variable.
- Now, suppose that t is in the form $((\mathtt{f}\ t_1 \dots t_n\ s_1 \dots s_m))$, where $n, m \geq 0$, $n+1$ is the arity of $\mathtt{f}$, $t_1, \dots, t_n$ are values and and $s_1, \dots, s_m$ are themselves canonical. By some case analysis on $\mathtt{f}$, one can easily reach the thesis.

This concludes the proof. □

Lemma 6. *If t is canonical and $t \to s$, then $[t] \to [s]$.*

Proof. Without any loss of generality, we can assume that t is the redex fired to get s. Thus, the thesis is reached once a simple substitution lemma is proved: if r and q are both canonical, then $[r\{x/q\}] = [r]\{x/[q]\}$. This is an easy induction on r. □

Lemma 7. *If $M \longrightarrow L$, t is canonical and $[t] = M$, then $t \to s$, where $[s] = L$.*

Proof. Again, without any loss of generality we can assume that M is the redex fired to get L. Then one can proceed by some case analysis on M, and easily conclude. □

Theorem 8 (Term Reducibility). *Let M be a closed PCF term. The following two conditions are equivalent:*
1. $M \longrightarrow^* L$ where L is in normal form;
2. $\langle M \rangle \to^* t$ where $[t] = L$ and t is in normal form.

Proof. Suppose $M \longrightarrow^n L$, where L is in normal form. Then, by applying Lemma 7, we obtain a term t such that $\langle M \rangle \to^n t$ and $[t] = L$. By Lemma 4, t is canonical and, by Lemma 5, it is in normal form. Now, suppose $\langle M \rangle \to^n t$ where $[t] = L$ and t is in normal form. By applying n times Lemma 6, we obtain $[\langle M \rangle] \longrightarrow^n [t] = L$. But $[\langle M \rangle] = M$ by Lemma 3 and L is a normal form by Lemma 5, since $\langle M \rangle$ and t are canonical by Lemma 4. □

It is also instructive to consider a source language which is less expressive than PCF, namely Gödel's T. For that, remove in the language the unary construct *fix*, and replace it with a binary construct

$$(rec\ M^A\ L^{NAT \to A \to A})^{NAT \to A},$$

obeying the following reduction rules:

$$(rec\ M\ L)\ \mathbf{0} \longrightarrow M;$$
$$(rec\ M\ L)\ \mathbf{n} + \mathbf{1} \longrightarrow L\ \mathbf{n}\ ((rec\ M\ L)\ \mathbf{n}).$$

We translate it to a STTRS R_{T} similar to R_{PCF}: one adds function symbols $\mathtt{rec}_{M,L}$, each with the following rules:

$$((\mathtt{rec}_{M,L}\ x_1 \dots x_n)\ \mathbf{0}) \to \langle M \rangle;$$
$$((\mathtt{rec}_{M,L}\ x_1 \dots x_n)\ (\mathbf{s}\ y)) \to ((\langle L \rangle\ y)((\mathtt{rec}_{M,L}\ x_1 \dots x_n)\ y));$$

where, as usual, $x_1 \dots x_n$ are the variables occurring free in either M or L. This encoding of system T then enjoys properties similar to the one of PCF.

Remark. The encodings of PCF and Gödel's T we have just described give rise to infinite STTRSs, called $\langle \mathsf{PCF} \rangle$ and $\langle \mathsf{T} \rangle$ respectively. On the other hand, if we restrict our attention to, e.g., one T program M of type $NAT \to NAT$, one can prove that only finitely many symbols among the ones in $\langle \mathsf{T} \rangle$ are sufficient not only to *encode* M and all possible inputs for it, but also to *evaluate* them. Of course, one cannot hope to catch all those fragments of $\langle \mathsf{T} \rangle$ if his or her aim is to be sound for polynomial time computability. But, as we will discuss in Section 5.1, our termination criterion is powerful enough to catch (the encoding of) many T terms. □

4. Higher-Order Polynomial Interpretations

We want to demonstrate how first-order rewriting-based techniques for ICC can be adapted to the higher-order setting. Our goal is to devise criteria ensuring complexity bounds on programs of *first-order type* possibly containing subprograms of *higher-order types*. A typical application will be to find out under which conditions a higher-order functional program such as *e.g.* `map`, `iteration` or `foldr`, fed with a (first-order) polynomial time program produces a polynomial time program.

As a first illustrative step we consider the approach based on polynomial interpretations from [10], which offers the advantage of simplicity. We thus build a theory of *higher-order polynomial interpretations* for STTRSs. It starts from a particular concrete instantiation of the methodology proposed in [35] for proving termination by interpretation, on which we prove additional properties in order to obtain polynomial time complexity bounds.

Higher-order polynomials (HOPs) take the form of terms in a typed λ-calculus whose only base type is that of natural numbers. To each of those terms can be assigned a strictly increasing function in a category $\mathbb{FSPOS}$ with products and functions. So, the whole process can be summarized by the following diagram:

$$\text{STTRSs} \xrightarrow{[\cdot]} \text{HOPs} \xrightarrow{[\![\cdot]\!]} \mathbb{FSPOS}$$

4.1. Higher-Order Polynomials (HOPs)

Let us consider types built from a base type $\mathbf{N}$:

$$A, B ::= \mathbf{N} \;\Big|\; A \to A.$$

$A^n \to B$ stands for the type

$$\underbrace{A \to \ldots \to A}_{n \text{ times}} \to B.$$

Let C_P be the following set of constants:

$$C_P = \{+ : \mathbf{N}^2 \to \mathbf{N}, \times : \mathbf{N}^2 \to \mathbf{N}\} \cup \{n : \mathbf{N} \mid\ n \in \mathbb{N}^\star\}.$$

Observe that in C_P we have constants of type $\mathbf{N}$ only for *strictly* positive integers. We consider the following grammar of Church-typed terms:

$$M, P := x^A \;\Big|\; \mathbf{c}^A \mid (M^{A\to B} P^A)^B \mid (\lambda x^A.M^B)^{A\to B},$$

where $\mathbf{c}^A \in C_P$ and *in* $(\lambda x^A.M^B)$ *we require that* x *occurs free in* M. A *higher-order polynomial* (HOP) is a term of this grammar which is in β-normal form. We use an infix notation for $+$ and $\times$.

HOP contexts (or simply contexts) are defined as HOPs but with the proviso that they must contain exactly one occurrence of a special constant $\bullet^A$ (the

$$
\begin{aligned}
[\![n]\!] &= n;\\
[\![+]\!](a,b) &= a+b;\\
[\![\times]\!](a,b) &= a\times b;\\
([\![(\lambda x^A.M^B)^{A\to B}]\!](a_1,\ldots,a_n))(a) &= [\![M^B]\!](a_1,\ldots,a_n,a);\\
[\![(M^{A\to B}P^A)^B]\!](a_1,\ldots,a_n) &= ([\![(M^{A\to B}]\!](a_1,\ldots,a_n))([\![P^A]\!](a_1,\ldots,a_n)).
\end{aligned}
$$

Figure 1: Set-theoretic Interpretation of HOPs.

hole), for a type A. They are indicated with metavariables like $\mathcal{C}$, $\mathcal{D}\ldots$ If $\mathcal{C}$ is a HOP context with hole $\bullet^A$, and M is a HOP of type A, then $\mathcal{C}\{M\}$ is the HOP obtained from $\mathcal{C}$ by replacing the occurrence of $\bullet^A$ by M and reducing to the β normal form.

We consider the usual set-theoretic interpretation of types and terms, denoted as $[\![A]\!]$ and $[\![M]\!]$, respectively: if M has type A and $FV(M)=\{x_1^{A_1},\ldots,x_n^{A_n}\}$, then $[\![M]\!]$ is a function from $[\![A_1]\!]\times\ldots\times[\![A_n]\!]$ to $[\![A]\!]$; the definition is given in Figure 1. We denote by $\equiv$ the equivalence relation which identifies terms denoting the same function, e.g. we have: $\lambda x.(2\times((3+x)+y))\equiv\lambda x.(6+(2\times x+2\times y))$. We will omit bracketing when possible thanks to associativity of $+$ and $\times$ modulo $\equiv$, e.g. write $\lambda x.(2\times(3+x+y)\equiv\lambda x.(2\times((3+x)+y))$.

Noticeably, even if HOPs can be built using higher-order functions, the first order fragment only contains polynomials:

Lemma 9. *If M is a HOP of type $\mathbf{N}^n\to\mathbf{N}$ and such that $FV(M)=\{y_1:\mathbf{N},\ldots,y_k:\mathbf{N}\}$, then the function $[\![M]\!]$ is bounded by a polynomial function.*

Proof. We prove by induction on M that $[\![M]\!]$ *is* a polynomial function:

- if $M=x$ or $M\in\{+,\times\}$ or $M=m$: the result is trivial;
- if $M=\lambda x^A.P^B$: then we have $A=\mathbf{N}$ and the type B is of the form $\mathbf{N}^{n-1}\to\mathbf{N}$, so by i.h. on P the property is true for P, hence for M;
- otherwise M is an application. As M is in β-normal form, there exists $k\geq 0$ such that:
  - M is of the form $M=P\ M_1\ \ldots M_k$;
  - and $P=x^A,+,\times$ or m.

  Now, if $P\{+,\times\}$, then for any $1\leq i\leq k$ we have that M_i is of type $\mathbf{N}$, so by induction hypothesis on M_i it satisfies the property, therefore M represents a polynomial function. If $P=x^A$, then as x^A is free in M, by assumption we know that $A=\mathbf{N}$, thus $k=0$ and the property is valid. Similarly if $P=m$.

This concludes the proof. □

A *HOP substitution* θ is a map from variables to HOPs, with finite domain. We will simply speak of *substitution* if there is no ambiguity. For any HOP M and HOP substitution θ, $M\theta$ is the HOP defined in the expected way.

4.2. Semantic Interpretation

Now, we consider a subcategory $\mathbb{FSPOS}$ of the category $\mathbb{SPOS}$ of strict partial orders as objects and *strictly increasing* total functions as morphisms. Objects of $\mathbb{FSPOS}$ are freely generated as follows:

- $\mathcal{N}$ is the domain of *strictly positive* integers, equipped with the natural strict order $\prec_{\mathcal{N}}$,
- 1 is the trivial order with one point;
- if σ, τ are objects, then $\sigma \times \tau$ is obtained by the product ordering;
- $\sigma \to \tau$ is the set of *strictly increasing* total functions from σ to τ, equipped with the following strict order: $f \prec_{\sigma\to\tau} g$ if for any a of σ we have $f(a) \prec_\tau g(a)$. We denote as $h \circ g$ the composition the g and h, defined as usual by $(h \circ g)(a) = h(g(a))$.

We denote by $\preceq_\sigma$ the reflexive closure of $\prec_\sigma$. About the product construction note that we have: $(a, b) \prec_{\sigma\times\tau} (c, d)$ iff $a \prec_\sigma c$ and $b \preceq_\tau d$, or $a \preceq_\sigma c$ and $b \prec_\tau d$. Actually we will also need to compare the semantics of terms which do not have the same free variables. For that we stipulate that if $f \in \sigma_1 \times \ldots \times \sigma_n \to \tau$, $g \in \sigma_1 \times \ldots \times \sigma_m \to \tau$ and $n \leq m$, then: $f \prec g$ if $\forall a_1 \in \sigma_1, \ldots, \forall a_m \in \sigma_m$, $f(a_1, \ldots, a_n) \prec_\tau g(a_1, \ldots, a_m)$.

The category $\mathbb{FSPOS}$ is a subcategory of $\mathbb{SET}$ with all the necessary structure to interpret types and terms. $[\![A]\!]_\prec$ denotes the semantics of A as an object of $\mathbb{FSPOS}$: we choose to set $[\![\mathbf{N}]\!]_\prec = \mathcal{N}$, while $[\![A_1 \times \ldots \times A_n \to A]\!]_\prec$ is $[\![A_1]\!]_\prec \times \ldots \times [\![A_n]\!]_\prec \to [\![A]\!]_\prec$. Observe that $[\![A]\!]_\prec$ is a subset of $[\![A]\!]$.

Now we want to show that the set-theoretic interpretation $[\![M]\!]$ of HOPs that we have given before induces an interpretation in $\mathbb{FSPOS}$, that we shall denote as $[\![M]\!]_\prec$. For that we prove the following property:

Proposition 10. *Let M be a HOP of type A and $x_1^{A_1}, \ldots, x_n^{A_n}$ be its free variables. Then for any $a \in [\![A_1]\!]_\prec \times \ldots \times [\![A_n]\!]_\prec$ we have that $[\![M]\!](a) \in [\![A]\!]_\prec$. We thus indicate as $[\![M]\!]_\prec$ the restriction of $[\![M]\!]$ to $[\![A_1]\!]_\prec \times \ldots \times [\![A_n]\!]_\prec$.*

Proof. We will prove by induction on M^A the following statements, where $x_1^{A_1}, \ldots, x_n^{A_n}$ are the free variables of M^A:

1. if $e \in [\![A_1]\!]_\prec \times \ldots \times [\![A_n]\!]_\prec$ then $[\![M]\!](e) \in [\![A]\!]_\prec$;
2. if $e, f \in [\![A_1]\!]_\prec \times \ldots \times [\![A_n]\!]_\prec$ and $e \prec f$ then $[\![M]\!](e) \prec [\![M]\!](f)$.

When 1. and 2. are satisfied we denote as $[\![M]\!]_\prec$ the induced interpretation of M in $[\![A_1]\!]_\prec \times \ldots \times [\![A_n]\!]_\prec \to [\![A]\!]_\prec$. Let us proceed with the proof:

- $M = x^A$: trivial;
- $M = c^A \in C_P$: this holds because as we have interpreted $\mathbf{N}$ as $\mathbb{N}^\star$ the terms $+$ and $\times$ denote strictly increasing functions (note that it would not have been the case for $\times$ if we had interpreted $\mathbf{N}$ as $\mathbb{N}$);
- $M = \lambda x^B.P^C$: by definition we know that x is free in P. As $x_1^{A_1}, \ldots, x_n^{A_n}$ denote the free variables of M let us write $A_{n+1} = B$ and $x_{n+1}^{A_{n+1}} = x^B$. So $x_1^{A_1}, \ldots, x_{n+1}^{A_{n+1}}$ are the free variables of P. By i.h. we know that P satisfies 1. and 2., and $[\![P]\!]_\prec \in [\![A_1 \times \ldots \times A_{n+1} \to C]\!]_\prec$. For $(e_1, \ldots, e_n)$ in $[\![A_1 \times \ldots \times A_n]\!]_\prec$ we then consider the map f from $[\![A_{n+1}]\!]_\prec$ to $[\![C]\!]_\prec$ defined by $f(e_{n+1}) = [\![P]\!]_\prec(e_1, \ldots, e_{n+1})$. As $[\![P]\!]_\prec \in [\![A_1 \times \cdots \times A_{n+1} \to C]\!]_\prec$

we have that $f \in [\![A_{n+1} \to C]\!]_\prec$, so M satisfies 1. Suppose now that $g \in [\![A_{n+1} \to C]\!]_\prec$ is defined in a similar way from $(h_1, \ldots, h_n) \in [\![A_1 \times \ldots \times A_n]\!]_\prec$ such that $(e_1, \ldots, e_n) \prec (h_1, \ldots, h_n)$. Then we have that if $e_{n+1} \in [\![A_{n+1}]\!]_\prec$, then $(e_1, \ldots, e_n, e_{n+1}) \prec (h_1, \ldots, h_n, e_{n+1})$, thus $[\![P]\!]_\prec(e_1, \ldots, e_n, e_{n+1}) \prec [\![P]\!]_\prec(h_1, \ldots, h_n, e_{n+1})$. This shows that $f \prec g$. Therefore M satisfies 2.

- $M = P^{B \to A}\ L^B$: this is the crucial case. By i.h. $[\![P]\!]_\prec$ and $[\![L]\!]_\prec$ have been defined. Denote by $x_1^{A_1}, \ldots, x_n^{A_n}$ the free variables of M. Take $e = (e_1, \ldots, e_n) \in [\![A_1 \times \ldots \times A_n]\!]_\prec$. By abuse of notation we will simply write $[\![P]\!]_\prec(e)$ instead of $[\![P]\!]_\prec(e_{i_1}, \ldots, e_{i_k})$ where $x_{i_1}, \ldots, x_{i_k}$ are the free variables of P. Similarly for $[\![L]\!]_\prec(e)$. Now we define f by $f = [\![P]\!]_\prec(e)([\![L]\!]_\prec(e))$. We know that $f \in [\![A]\!]_\prec$ and by i.h. we have that $f = [\![P]\!](e)([\![L]\!](e)) = [\![M]\!](e)$. So M satisfies 1. Let us now consider condition 2. If $n = 0$ it is trivial, so let us assume $n \geq 1$. Take e, g two elements of $[\![A_1 \times \ldots \times A_n]\!]_\prec$ with $e \prec g$. Let $f = [\![P]\!]_\prec(e)([\![L]\!]_\prec(e))$ and $h = [\![P]\!]_\prec(g)([\![L]\!]_\prec(g))$. Let us distinguish two subcases:
  - if there exists i in $\{1, \ldots, n\}$ such that $e_i \prec g_i$ and $x_i \in FV(P)$: then by i.h. on P we have $[\![P]\!]_\prec(e) \prec_{B \to A} [\![P]\!]_\prec(g)$; moreover by i.h. we have: $[\![L]\!]_\prec(e) \preceq_B [\![L]\!]_\prec(g)$ (note the non-strict ordering). From these two inequalities, by definition of $\prec_{B \to A}$ we can deduce: $[\![P]\!]_\prec(e)([\![L]\!]_\prec(e)) \prec_A [\![P]\!]_\prec(g)([\![L]\!]_\prec(g))$.
  - Otherwise: we know that there exists i in $\{1, \ldots, n\}$ such that $e_i \prec g_i$, and x_i is free in M so it must be free in L. So by i.h. on L we have $[\![L]\!]_\prec(e) \prec_B [\![L]\!]_\prec(g)$; besides we know that $[\![P]\!]_\prec(e) = [\![P]\!]_\prec(g)$. We know that $[\![P]\!]_\prec(e)$ belongs to $[\![B \to A]\!]_\prec$ so it is strictly increasing, so we deduce that: $[\![P]\!]_\prec(e)([\![L]\!]_\prec(e)) \prec_A [\![P]\!]_\prec(g)([\![L]\!]_\prec(g))$.

  So in both subcases we have concluded that $f \prec h$, which completes the proof of 2. and thus of the claim.

This concludes the proof. □

Lemma 11. *Let M be a HOP and θ a HOP substitution with $FV(M) = \{x_1^{A_1}, \ldots, x_n^{A_n}\}$ and $FV(M\theta) = \{y_1^{B_1}, \ldots, y_m^{B_m}\}$. Then, for every $i \in \{1, \ldots, m\}$ the function $f_i = [\![\theta(x_i)]\!]_\prec$ is a strictly increasing map. Moreover, we have $[\![M\theta]\!]_\prec = [\![M]\!]_\prec \circ (f_1, \ldots, f_n)$.*

Proof. For $i \in \{1, \ldots, m\}$ we have $f_i = [\![\theta(x_i)]\!]_\prec$. We know that $\theta(x_i)$ is a HOP, and $FV(\theta(x_i)) \subseteq FV(M\theta)$. Let $= \{y_{k_1}^{B_{k_1}}, \ldots, y_{k_i}^{B_{k_i}}\}$ be the free variables of $\theta(x_i)$. Then f_i belongs to $[\![B_{k_1}]\!]_\prec \times \ldots \times [\![B_{k_i}]\!]_\prec \to [\![A_i]\!]_\prec$. We can then prove the second statement by induction on M. As an illustration let us just examine here the base cases:

- The case where $M = c^A$ is trivial because $n = 0$ and $M\theta = M$.
- In the case where M is a variable we have $n = 1$ and $M = x_1^{A_1}$. Then $[\![M\theta]\!]_\prec = [\![\theta(x_1)]\!]_\prec = f_1 = id_{A_1} \circ f_1$.

This concludes the proof. □

Applying the same substitution to two HOPs having the same type preserve the properties of the underlying interpretation:

Lemma 12. *If M^A, P^A are HOPs such that $FV(M) \subseteq FV(P)$, θ is a HOP substitution and if $[\![M]\!]_\prec \prec [\![P]\!]_\prec$, then $[\![M\theta]\!]_\prec \prec [\![P\theta]\!]_\prec$.*

Proof. By Lemma 11 we have:

$$[\![M\theta]\!]_\prec = [\![M]\!]_\prec \circ (f_1, \ldots, f_m);$$
$$[\![P\theta]\!]_\prec = [\![P]\!]_\prec \circ (f_1, \ldots, f_n);$$

where $m \leq n$. Moreover $FV(M\theta) \subseteq FV(P\theta)$. We thus have:

$$[\![M\theta]\!]_\prec \in \sigma_1 \times \ldots \sigma_k \to \sigma;$$
$$[\![P\theta]\!]_\prec \in \sigma_1 \times \ldots \sigma_l \to \sigma;$$

where $k \leq l$ and $\sigma = [\![A]\!]_\prec$. Take now $a \in \sigma_1 \times \ldots \sigma_l$ and let $b = (a_1, \ldots, a_k)$. By what precedes we know that for $1 \leq i \leq m$, $f_i(a)$ only depends on b, hence we also write it as $f_i(b)$. Now we have that:

$$\begin{aligned}
[\![M\theta]\!]_\prec(b) &= [\![M]\!]_\prec(f_1(b), \ldots, f_m(b)) \\
&\prec [\![P]\!]_\prec(f_1(a), \ldots, f_n(a)), \text{ because } [\![M]\!]_\prec \prec [\![P]\!]_\prec, \\
&= [\![P\theta]\!]_\prec(a).
\end{aligned}$$

We can thus conclude that $[\![M\theta]\!]_\prec \prec [\![P\theta]\!]_\prec$. □

4.3. Assignments and Polynomial Interpretations

We consider $\mathcal{X}$, $\mathcal{C}$ and $\mathcal{F}$ as in Section 2. To each variable x^A we associate a variable $\underline{x}^{\underline{A}}$ where $\underline{A}$ is obtained from A by replacing each occurrence of base type by the base type $\mathbf{N}$ and by curryfication. An *assignment* $[\,\cdot\,]$ is a map from $\mathcal{C} \cup \mathcal{F}$ to HOPs such that if $f \in \mathcal{C} \cup \mathcal{F}$ has type A, $[f]$ is a closed HOP of type $\underline{A}$. Now, for $t \in \mathcal{T}$ of type A, we define an HOP $[t]$ of type $\underline{A}$ by induction on t:

- if $t = x \in \mathcal{X}$, then $[t]$ is $\underline{x}$;
- if $t \in \mathcal{C} \cup \mathcal{F}$, $[t]$ is already defined;
- otherwise, if $t = (s\ t_1 \ldots t_n)$ then $[t] \equiv (\ldots([s][t_1])\ldots[t_n])$.

Observe that in practice, computing $[t]$ will in general require to do some β-reduction steps.

Lemma 13. *Let $t \in \mathcal{T}$ of type A and $FV(t) = \{y_1 : A_1, \ldots, y_n : A_n\}$, then: $[t]$ is a HOP, of type $\underline{A}$, and $FV([t]) = \{\underline{y_1} : \underline{A_1}, \ldots, \underline{y_n} : \underline{A_n}\}$.*

Proof. This follows from the definition of $[t]$, by induction on t. □

We extend the notion of assignments to contexts by setting: $[\,\bullet^A\,] = \bullet^{\underline{A}}$. So if $\mathcal{C}$ is a context then $[\mathcal{C}]$ is a HOP context, and we have:

Lemma 14. *If $\mathcal{C}$ is a context and t a term, then $[\mathcal{C}\{t\}] = [\mathcal{C}]\{[t]\}$.*

Now, if σ is a substitution, $[\sigma]$ is the HOP substitution defined as follows: for any variable x, $[\sigma](\underline{x}) = [\sigma(x)]$. We have:

Lemma 15. *If t is a term and σ a substitution, then $[t\sigma] = [t][\sigma]$.*

Let us now consider the semantic interpretation. If $t \in \mathcal{T}$ of type A and $FV(t) = \{y_1 : A_1, \ldots, y_n : A_n\}$, we will simply denote by $[\![t]\!]_\prec$ the element $[\![[t]]\!]_\prec$ of $[\![\underline{A_1} \times \ldots \times \underline{A_n} \to \underline{A}]\!]_\prec$. We get the following lemma, establishing the closure by context of this interpretation:

Lemma 16. *Let M and P be HOPs such that $[\![M]\!]_\prec \prec [\![P]\!]_\prec$ and $\mathcal{C}$ be a HOP context such that $\mathcal{C}\{M\}$ and $\mathcal{C}\{P\}$ are well-defined. Then we have that $[\![\mathcal{C}\{M\}]\!]_\prec \prec [\![\mathcal{C}\{P\}]\!]_\prec$. The same statement holds for terms: if t and s are two terms such that $[\![t]\!]_\prec \prec [\![s]\!]_\prec$ and $\mathcal{C}$ is a context such that $\mathcal{C}\{t\}$ and $\mathcal{C}\{s\}$ are well-defined, then we have that $[\![\mathcal{C}\{t\}]\!]_\prec \prec [\![\mathcal{C}\{s\}]\!]_\prec$.*

Now, we say that an assignment $[\,\cdot\,]$ is a *higher polynomial interpretation* or simply a *polynomial interpretation* for a STTRS R iff for every $l \to r \in R$, we have that $[\![r]\!]_\prec \prec [\![l]\!]_\prec$. Note that in the particular case where the program only contains first-order functions, this notion of polynomial interpretation coincides with the classical one for first-order TRSs. In the following, we assume that $[\,\cdot\,]$ is a polynomial interpretation for R. A key property is the following, which tells us that the interpretation of terms strictly decreases along any reduction step:

Lemma 17. *If $t \to s$, then $[\![s]\!]_\prec \prec [\![t]\!]_\prec$.*

Proof. if $t \to s$, then by definition there exists a rule $l \to r$ of R, a context $\mathcal{C}$ and a substitution σ such that $t = \mathcal{C}\{l\sigma\}$ and $s = \mathcal{C}\{r\sigma\}$. As $[\,\cdot\,]$ is a polynomial interpretation, we have that $[\![r]\!]_\prec \prec [\![l]\!]_\prec$. We then get:

$$\begin{array}{rcll} [\![r\sigma]\!]_\prec & = & [\![[r\sigma]]\!]_\prec, & \text{by definition,} \\ & = & [\![[r][\sigma]]\!]_\prec, & \text{by Lemma 15,} \\ & \prec & [\![[l][\sigma]]\!]_\prec, & \text{by Lemma 12,} \\ & = & [\![[l\sigma]]\!]_\prec, & \text{by Lemma 15 again,} \\ & = & [\![l\sigma]]\!]_\prec. & \end{array}$$

Then by Lemma 16 (second statement) we get $[\![\mathcal{C}\{r\sigma\}]\!]_\prec \prec [\![\mathcal{C}\{l\sigma\}]\!]_\prec$, which concludes the proof. □

As a consequence, the interpretation of terms (of base type) is itself a bound on the length of reduction sequences:

Proposition 18. *Let t be a closed term of base type D. Then $[t]$ has type $\mathbf{N}$ and any reduction sequence of t has length bounded by $[\![t]\!]_\prec$.*

Proof. It is sufficient to observe that by Lemma 17 any reduction step on t makes $[\![t]\!]_\prec$ decrease for $\prec$, and that as t is closed and of type $\mathbf{N}$ the order $\prec$ here is $\prec_{\mathcal{N}}$, which is the ordinary (strict) order on integers. □

4.4. A Complexity Criterion

Proving a STTRS to have a polynomial interpretation is not enough to guarantee its time complexity to be polynomially bounded. To ensure that, we need to impose some constraints on the way constructors are interpreted. We say that the assignment $[\cdot]$ is *additive* if for any constructor $\mathbf{c}$ of type $D_1 \times \cdots \times D_k \to D$, there exists an integer $n_{\mathbf{c}} \geq 1$ such that $[\mathbf{c}] \equiv \lambda x_1 \ldots \lambda x_k.(x_1 + x_2 + \cdots + x_k + n_{\mathbf{c}})$. Additivity ensures that the interpretation of first-order values is proportional to their size:

Lemma 19. *Let $[\,\cdot\,]$ be an additive assignment. Then there exists $\gamma \geq 1$ such that for any value v of type D, we have $[\![v]\!]_{\prec} \leq \gamma \cdot |v|$.*

A function $f : (\{0,1\}^*)^m \to \{0,1\}^*$ is said to be *representable* by a STTRS R if there is a function symbol $\mathtt{f}$ of type $(W_2)^n \to W_2$ in R which computes f in the obvious way, where we recall that W_2 stands for the type of binary words. We are now ready to prove the main result about polynomial interpretations, namely that they enforce reduction lengths to be bounded in an appropriate way:

Theorem 20 (Polynomial Bound). *Let R be a STTRS with an additive polynomial interpretation $[\,\cdot\,]$. Consider a function symbol $\mathtt{g}$ of type $(W_2)^n \to W_2$. Then, there exists a polynomial $p : \mathbb{N}^n \to \mathbb{N}$ such that, for any $w_1, \ldots, w_n \in \{0,1\}^\star$, any reduction of $(\mathtt{g}\ \underline{w_1} \ldots \underline{w_n})$ has length bounded by $p(|w_1|, \ldots, |w_n|)$. This holds more generally for $\mathtt{g}$ of type $D_1 \times \cdots \times D_n \to D$.*

Proof. By Prop. 18 we know that any reduction sequence has length bounded by: $[\![(\mathtt{g}\ \underline{w_1} \ldots \underline{w_n})]\!]_{\prec} = [\![\mathtt{g}]\!]_{\prec}\,([\![\underline{w_1}]\!]_{\prec}, \ldots, [\![\underline{w_n}]\!]_{\prec})$. By Lemma 9 there exists a polynomial function $q : \mathbb{N}^n \to \mathbb{N}$ such that $[\![\mathtt{g}]\!]_{\prec}$ is bounded by q. Moreover by Lemma 19 there exists $\gamma \geq 1$ such that: $[\![\underline{w_i}]\!]_{\prec} \leq \gamma|\underline{w_i}|$. So by defining $p : \mathbb{N}^n \to \mathbb{N}$ as the polynomial function such that $p(y_1, \ldots, y_n) = q(\gamma y_1, \ldots, \gamma y_n)$, we have that the length of the reduction sequence is bounded by $p(|w_1|, \ldots, |w_n|)$. □

Corollary 21. *The functions on binary words representable by STTRSs admitting an additive polynomial interpretation are exactly the polytime functions.*

Proof. We have two inclusions to prove: from left to right (*complexity soundness*), and from right to left (*completeness*):

- **Soundness**. Assume F is represented by a program $\mathtt{g}$ with type $W_2 \times \cdots \times W_2 \to W_2$ admitting an additive polynomial interpretation. Then by Theorem 20 we know that for any $w_1, \ldots, w_n \in \{0,1\}^\star$, any reduction of $(\mathtt{g}\ \underline{w_1} \ldots \underline{w_n})$ has a polynomial number of steps. By a result in [18], derivational complexity is an invariant cost model for TRSs, via graph rewriting. This result can be easily generalized to STTRSs.
- **Completeness**. It has been shown in [10] (Theorem 4, Section 4.2) that if F is polytime computable, then there exists a first-order rewriting system with an additive polynomial interpretation which computes F. Actually the rewriting systems considered in the cited paper are not assumed to be

typed, but can anyway be seen as simply-typed term rewriting systems in our sense: it suffices to consider just one base type *TERMS* and give to function symbols and constructors their natural type, e.g., a constructor $\mathbf{c}$ of arity n has type $\mathit{TERMS}^n \to \mathit{TERMS}$. When restricting to first-order typed rewriting systems, our notions of polynomial interpretation and of additive polynomial interpretation coincide with the notions they consider. Therefore any polytime function on binary words can be represented by a simply-typed term rewriting system with an additive polynomial interpretation.

This concludes the proof. □

The results we have just described are quite robust: one is allowed to extend C_P with new combinators, provided their set-theoretic semantics are strictly increasing functions for which Lemma 9 continues to hold. In particular if they are of type $\mathbf{N}^k \to \mathbf{N}$ they should be polynomially bounded. However, the class of polynomial time STTRSs which can be proved such by way of higher-order polynomial interpretations is quite restricted, as we are going to argue.

4.5. Examples

Example 3. *Consider the STTRS defined by the following rules:*

$$((\mathtt{map}\ f)\ \mathbf{nil}^D) \to \mathbf{nil}^E; \tag{1}$$

$$((\mathtt{map}\ f)\ (\mathbf{cons}^D\ x\ xs)) \to (\mathbf{cons}^E\ (f\ x)\ ((\mathtt{map}\ f)\ xs)); \tag{2}$$

with the following types:

$$f : D \to E; \qquad \mathtt{map} : (D \to E) \to L(D) \to L(E);$$
$$\mathbf{nil}^D : L(D); \qquad \mathbf{cons}^D : D \times L(D) \to L(D);$$
$$\mathbf{nil}^E : L(E); \qquad \mathbf{cons}^E : E \times L(E) \to L(E).$$

Here $D, E, L(D), L(E)$ are base types. For simplicity we use just one **cons** *and one* **nil** *notation for both types D and E. The interpretation below was given in [35] for proving termination, but here we show that it also gives a polynomial time bound. To simplify the reading of HOPs we use infix notations for $+$, and omit some brackets (because anyway we have associativity and commutativity for the denotations). Now, we choose the following assignment of HOPs:*

$$[\mathbf{nil}] = 2 : \mathbf{N};$$
$$[\mathbf{cons}] = \lambda n.\lambda m.(n + m + 1) : \mathbf{N} \to \mathbf{N} \to \mathbf{N};$$
$$[\mathtt{map}] = \lambda\phi.\lambda n.n \times (\phi\ n) : (\mathbf{N} \to \mathbf{N}) \to \mathbf{N} \to \mathbf{N}.$$

We then get the following interpretations of terms:

$$[(\mathtt{map}\ f\ \mathbf{nil})] = 2 \times (\underline{f}\ 2);$$
$$[(\mathtt{map}\ f\ \mathbf{cons}(x, xs))] = (\underline{x} + \underline{xs} + 1) \times (\underline{f}\ (\underline{x} + \underline{xs} + 1));$$
$$[(\mathbf{cons}\ (f\ x)\ (\mathtt{map}\ f\ xs))] = 1 + (\underline{f}\ \underline{x}) + \underline{xs} \times (\underline{f}\ \underline{xs}).$$

Let us check that the condition $[\![r]\!]_{\prec} \prec [\![l]\!]_{\prec}$ *holds for the rules above. By strict increasing, as* $1 \prec \underline{y}$ *we have that* $\underline{f}\ \underline{x} \prec \underline{f}(\underline{x} + \underline{y})$. *Therefore we have in particular:*

$$\underline{f}\ \underline{xs} \prec \underline{f}\ (\underline{x} + \underline{xs} + 1);$$
$$\underline{f}\ \underline{x} \prec \underline{f}\ (\underline{x} + \underline{xs} + 1).$$

We deduce from that:

$$1 + (\underline{f}\ \underline{x}) + \underline{xs} \times (\underline{f}\ \underline{xs}) \prec (\underline{x} + \underline{xs} + 1) \times (\underline{f}\ (\underline{x} + \underline{xs} + 1));$$

so $[\![r]\!]_{\prec} \prec [\![l]\!]_{\prec}$ *holds for (2). Similarly, we can check it for (1).*

Observe that choosing $[\mathbf{nil}] = 2$ was slightly unnatural, but it is necessary here to have a valid interpretation. We thus have an additive polynomial interpretation for `map`, therefore Corollary 21 applies and we can conclude that for any `f` also satisfying the criterion, (`map f`) computes a polynomial time function. Now, one might want to apply the same method to an iterator `iter`, of type $(D \to D) \times D \to NAT \to D$, which when fed with arguments f, d, n iterates f exactly n times starting from d. However there is no additive polynomial interpretation for this program. Actually, this holds for very good reasons: `iter` can produce an exponential-size function when fed with a fast-growing polynomial time function, *e.g.* `double` : $NAT \to NAT$.

One way to overcome this issue could be to show that `iter` *does* admit a valid polynomial interpretation, *provided* its domain is restricted to some particular functions, admitting a *not-so-fast-growing* polynomial interpretation, of the form $\lambda n.(n+c)$, for some constant c. This could be enforced by considering a refined type systems for HOPs. But the trouble is that there are very few programs which admit a polynomial interpretation of this form! Intuitively, the problem is that polynomial interpretations need to bound simultaneously the execution time *and* the size of the intermediate values. In the sequel we will see how to overcome this issue.

5. Beyond Interpretations: Quasi-Interpretations

The previous section has illustrated our approach. However we have seen that the intensional expressivity of higher-order polynomial interpretations is too limited. In the first-order setting this problem has been overcome by decomposing into *two distinct conditions* the role played by polynomial interpretations [32, 11]: (i) a termination condition, (ii) a condition enforcing a bound on the size of values occurring during the computation. In [11], this has been implemented by using: for (i) some specific path orderings, and for (ii) a notion of *quasi-interpretation*. We now examine how this methodology can be extended to the higher-order setting.

The first step will take the form of a termination criterion defined by a linear type system for STTRSs together with a path-like order, to be described in

$$\frac{\mathtt{f}^A \in \mathcal{NF}}{\Gamma \mid \Delta \vdash \mathtt{f} : A} \qquad \frac{\mathbf{c}^A \in \mathcal{C}}{\Gamma \mid \Delta \vdash \mathbf{c} : A} \qquad \frac{}{\Gamma \mid x : A, \Delta \vdash x : A} \qquad \frac{}{x : D, \Gamma \mid \Delta \vdash x : D}$$

$$\frac{\mathtt{f}^{A_1,\ldots,A_n \to B} \in \mathcal{RF}, \text{with arity } n \qquad \Gamma \mid \emptyset \vdash s_i : A_i}{\Gamma \mid \Delta \vdash (\!(\mathtt{f}\ s_1 \ldots s_n)\!) : B} \qquad \frac{\Gamma \mid \Delta \vdash t : A_1 \times \ldots \times A_n \to B \qquad \Gamma \mid \Delta_i \vdash s_i : A_i}{\Gamma \mid \Delta, \Delta_1, \ldots, \Delta_n \vdash (t\ s_1 \ldots s_n) : B}$$

Figure 2: A Linear Type System for STTRS terms.

Section 5.1 below. The second step consists in shifting from the semantic world of strictly increasing functions to one of increasing functions. This corresponds to a picture like the following, and is the subject of Section 5.3 and Section 5.5.

$$\text{STTRSs} \xrightarrow{[\cdot]} \text{HOMPs} \xrightarrow{[\![\cdot]\!]} \mathbb{FPOS}$$

5.1. The Termination Criterion

The termination criterion has two ingredients: a typing ingredient and a syntactic ingredient, expressed using an order $\sqsubset$ on the function symbols. Formally, introducing the typing ingredient requires splitting the class $\mathcal{F}$ into two disjoint classes $\mathcal{RF}$ and $\mathcal{NF}$. The intended meaning is that functions in $\mathcal{NF}$ cannot be defined in a recursive way, while functions in $\mathcal{RF}$ can. We further assume given a strict order $\sqsubset$ on $\mathcal{F}$ which is well-founded. If t is a term, $t \sqsubset \mathtt{f}$ means that for any $\mathtt{g}$ occurring in t we have $\mathtt{g} \sqsubset \mathtt{f}$. The rules of a linear type system for STTRS terms are in Figure 2. In a judgement $\Gamma \mid \Delta \vdash t : A$, the sub-context Δ is meant to contain linear variables while Γ is meant to contain non-linear variables. Note that in the bottom-right rule of Figure 2, the contexts Δ and Δ_i for $1 \leq i \leq n$ are assumed to have pairwise disjoint sets of variables.

A STTRS *satisfies the termination criterion* if every rule $(\!(\mathtt{f}\ p_1 \ldots p_k)\!) \to s$ satisfies:

1. either $\mathtt{f} \in \mathcal{RF}$, there are a term r and a sequence of patterns $q_1, \ldots, q_k$ such that $s = r\{x/(\!(\mathtt{f}\ q_1 \ldots q_k)\!)\}$ (in other words, $(\!(\mathtt{f}\ q_1 \ldots q_k)\!)$ is a recursive call of $\mathtt{f}$ in s), we have $\Gamma \mid x : B, \Delta \vdash r : B$, $r \sqsubset \mathtt{f}$, every q_i is subterm of p_i and there exists j s.t. $q_j \neq p_j$;
2. or we have $\Gamma \mid \Delta \vdash s : B$ and $s \sqsubset \mathtt{f}$.

Observe that because of the typability constraint in 1., this termination criterion implies that there is at most one recursive call in the right-hand-side s of a rule. Is the termination criterion too restrictive? Let us comment first on the syntactic ingredient:

- First consider the embedding of simply-typed λ-calculus given by the restriction of the embedding of PCF of Section 3. The function symbols used are $\mathtt{abs}_{M,x}$ for all typed term M and variable x. We define the order $\sqsubset$ by:

$$\mathtt{abs}_{L,y} \sqsubset \mathtt{abs}_{M,x} \text{ if } \lambda y.L \text{ is a subterm of } M.$$

Recall that $t \sqsubset \mathtt{f}$ if $\mathtt{g} \sqsubset \mathtt{f}$ for any function $\mathtt{g}$ in t. The STTRS encoding of M then satisfies, for any rule $(\!(\mathtt{f}\ \overline{p_1} \dots \overline{p_k})\!) \to s$, the condition

$$s \sqsubset \mathtt{f} \tag{3}$$

As the order $\sqsubset$ defined is well-founded, this implies the termination of this STTRS program.

- Now consider System T. We have seen that System T with weak reduction can also be embedded into a STTRS. Forget about functions `pred`, `succ`, *ifz* for simplification, and consider the new operator *rec*. We extend $\sqsubset$ by setting: $\mathtt{rec}_{M,L} \sqsubset \mathtt{rec}_{P,Q}$ if $(rec\ M\ L)$ is a subterm of either P or Q, and similarly for symbols in the form $\mathtt{abs}_{R,x}$. Then in the STTRS encoding of a system T term M, each rule $(\!(\mathtt{f}\ p_1 \dots p_k)\!) \to s$ satisfies (3) or the following Condition 4: there are a term r and patterns $q_1, \dots, q_k$ such that for any i, q_i is subterm of p_i, there exists j s.t. $q_j \neq p_j$ and

$$s = r\{x/(\!(\mathtt{f}\ q_1 \dots q_k)\!)\},\ \text{and}\ r \sqsubset \mathtt{f} \tag{4}$$

  In other words, $\langle \mathsf{T} \rangle$ satisfies the order-theoretic part of the termination, while of course it does not satisfy the typing-theoretic part of it.

So the syntactic ingredient is fairly expressive, since it will allow to validate system T programs.

Let us now examine the full termination criterion on an example.

Example 4. *Consider the program* `foldr` *given by:*

$$((\mathtt{foldr}\ f\ b)\ \mathbf{nil}) \to b; \tag{5}$$

$$((\mathtt{foldr}\ f\ b)\ (\mathbf{cons}\ x\ xs)) \to (f\ x\ ((\mathtt{foldr}\ f\ b)\ xs)); \tag{6}$$

where functions, variables and constructors have the following types:

$$\begin{aligned} \mathtt{foldr} &: (D \times E \to E) \times E \to L(D) \to E; \\ f &: D \times E \to E; \\ \mathbf{nil} &: L(D); \\ \mathbf{cons} &: D \times L(D) \to L(D). \end{aligned}$$

Examine rule (5). The function `foldr` *does not appear on the r.h.s. and we have* $b \sqsubset (\mathtt{foldr}\ f\ b)$*. So, by Case 2, this rule satisfies the condition. As to rule (6), it has a recursive call. Let us prove that it satisfies the condition by Case 1. Following the notations of the definition we have here* $s = r\{y/((\mathtt{foldr}\ f\ b)\ xs)\}$ *with* $r = (f\ x\ y)$*. We have* $\Gamma \mid y : E, \Delta \vdash r : E$*, as shown by the (simple) type derivation of Figure 3 (where we use the abbreviation* $F = D \times E \to E$*), with* $\Gamma = \{x : D\}$ *and* $\Delta = \{f : D \times E \to E\}$*. We also have* $r \sqsubset \mathtt{foldr}$*. Finally* f*,* b *are respectively subterms of* f*,* b*, and* xs *is a strict subterm of* $(\mathbf{cons}\ x\ xs)$*. So Case 1 is indeed satisfied.*

$$\dfrac{\dfrac{}{x:D \mid f:F \vdash f:F} \quad \dfrac{}{x:D \mid \emptyset \vdash x:D} \quad \dfrac{}{x:D \mid y:E \vdash y:E}}{x:D \mid y:E, f:F \vdash (f\ x\ y):E}$$

Figure 3: Type Derivation (Example 4)

More generally we believe that the termination criterion is general enough to embed a non-trivial fragment of Hofmann's system SLR [23], which is a restriction of system T based on safe recursion and using linear types, and which characterizes the class of polytime functions.

Now let us prove that this criterion indeed implies termination. We will proceed by a reducibility proof, this way making the argument independent on the type-theoretical condition, but only on the order-theoretical condition. Given a term t, its *definitional depth* is the maximum, over any function symbol `f` appearing in t, of the length of the longest descending $\sqsubset$-chain starting from `f`. The definitional depth of t is denoted as $\partial(t)$.

Theorem 22 (Termination). *If a STTRS satisfies the termination criterion, then any of its closed terms is strongly normalizing.*

Proof. This is a standard reducibility argument. First of all, one needs to define the notion of reducibility for *closed* terms, by induction on their types:

- A term t of base type is reducible iff it is strongly normalizing.
- A term t of type $A_1 \times \cdots \times A_n \to A$ is reducible iff it is strongly normalizing *and* for every reducible terms $s_1, \ldots, s_n$ of types $A_1, \ldots, A_n$, respectively, the term $(t\ s_1 \ldots s_n)$ is itself reducible.

Before going on, let us just state four lemmas:

1. A term t is strongly normalizing iff it is weakly normalizing. This follows from the way reduction is defined: it satisfies a diamond-like property, from which one can easily derive that weak-normalization implies strong normalization.
2. Suppose that t of type $A_1 \times \cdots \times A_n \to A$ and $s_1, \ldots, s_n$ of types $A_1, \ldots, A_n$ (respectively) are all reducible. Then, the term $(t\ s_1\ \ldots\ s_n)$ is itself reducible. This is an easy consequence of the definition of reducibility for t.
3. If $t \to^* s$, then t is reducible iff s is reducible. Let us proceed by induction on the structure of the type of t:
   - If t and s have base type, then we need to prove that t is strongly normalizing iff s is strongly normalizing. Of course, if t is strongly normalizing, also s is strongly normalizing. The converse also holds because of Point 1 above.
   - Let t and s have type $A_1 \times \cdots \times A_n \to A$, and suppose that $r_1, \ldots, r_n$ are reducible terms of type $A_1, \ldots, A_n$, respectively. By induction hypothesis, $(t\ r_1\ \ldots\ r_n)$ is reducible iff $(s\ r_1\ \ldots\ r_n)$ is reducible. Moreover, for reasons similar to the ones in the previous case, t is strongly normalizing iff s is strongly normalizing.

4. Every reducible term is strongly normalizing. This is trivial from the definition.

It is now possible to prove that any closed term t of a STTRS satisfying the termination criterion is reducible, by induction on $\partial(t)$. Indeed, one can prove that any function symbol $\mathtt{f}$ of such a STTRS is reducible, by induction on $\partial(\mathtt{f})$, and then argue by Point 2 above (since all constructor symbols are easily seen to be reducible). If $\mathtt{f}$ is a function symbol, then $\mathtt{f}$ is of course strongly normalizing, and if one applies it to reducible terms, one obtains a term which reduces to an application $(\mathtt{f}\ v_1, \ldots, v_n)$, itself either a value (and in this case one can apply the argument iteratively) or a redex which rewrites in one step (if $\mathtt{f} \in \mathcal{NF}$) or in many steps (if $\mathtt{f} \in \mathcal{RF}$) to a term which contains constructors, function symbols lower than $\mathtt{f}$, or values from $v_1, \ldots, v_n$, all of them reducible by Point 3. By Point 2, the obtained term is reducible, and thus by Point 3, $(\mathtt{f}\ v_1, \ldots, v_n)$ is itself reducible. This concludes the proof. □

5.2. On Base-Type Values and Time Complexity

In this section, we show that all that matters for the time complexity of STTRSs satisfying the termination criterion is the size of base-type values that can possibly appear along the reduction of terms. In other words, we are going to prove that if the latter is bounded, then the complexity of the starting term is known, modulo a fixed polynomial. Showing this lemma, which will be crucial in the following, requires introducing some auxiliary definitions and results.

First of all: when, formally, a natural number can be considered as a bound on the size of base-type values appearing along reduction of t? Given a term t and a natural number $n \in \mathbb{N}$, n is said to be *a bound of base-type values for* t if whenever $t \to^* s$, all base-type values v occurring in s are such that $|v| \leq n$. The following immediately follows from the aforementioned definition:

Lemma 23. *If $n \in \mathbb{N}$ is a bound of base-type values for t and $t \to^* s$, then n is also a bound of base-type values for s.*

Suppose a function symbol $\mathtt{f}$ takes n arguments of base types. Then $\mathtt{f}$ is said to have *base values bounded by a function* $q : \mathbb{N}^n \to \mathbb{N}$ if $(\mathtt{f}\ t_1 \ldots t_n)$ has $q(|t_1|, \ldots, |t_n|)$ as a bound of its base-type values whenever $t_1, \ldots, t_n$ are base-type values of the appropriate type.

It is convenient to put all terms occurring in the r.h.s. of rules defining a given function symbol in a set. Formally, given a function symbol $\mathtt{f}$, $\mathcal{R}(\mathtt{f})$ denotes the set of terms appearing in the right-hand side of rules for $\mathtt{f}$, not taking into account recursive calls. More formally, r belongs to $\mathcal{R}(\mathtt{f})$ iff there is a rule $((\mathtt{f}\ p_1 \ldots p_k)) \to s$ such that $s = r\{x/((\mathtt{f}\ q_1 \ldots q_k))\}$ (where x might not occur in r).

The central concept in this section is probably the weight of a term, which is defined as a quantity which is an upper bound both to the number of reduction steps and to the size of intermediate terms to its normal form. For technical reasons, it is convenient to define the weight as a polynomial rather than as a number. Formally, define *space-time weight* of a term t as a polynomial $\mathcal{TS}_t(X)$

$$\mathcal{TS}_v(X) = 1, \quad \text{if } v \text{ is a first order value,}$$

$$\mathcal{TS}_{(\!(\mathtt{f}\ t_1 \ldots t_n)\!)}(X) = 1 + \left(\sum_{\substack{t_j \in \mathcal{FO} \\ j \leq \mathit{arity}(\mathtt{f})}} \mathcal{TS}_{t_j}(X) \right) + \left(\sum_{\substack{t_j \in \mathcal{HO} \\ j \leq \mathit{arity}(\mathtt{f})}} \mathit{arity}(\mathtt{f}) \cdot X \cdot \mathcal{TS}_{t_j}(X) \right)$$
$$+ \left(\sum_{j \geq \mathit{arity}(\mathtt{f})+1} \mathcal{TS}_{t_j}(X) \right) + \left(\sum_{s \in \mathcal{R}(\mathtt{f})} \mathit{arity}(\mathtt{f}) \cdot X \cdot \mathcal{TS}_s(X) \right), \quad \text{if } \mathtt{f} \in \mathcal{RF};$$

$$\mathcal{TS}_{(\!(\mathtt{f}\ t_1 \ldots t_n)\!)}(X) = 1 + \left(\sum_{1 \leq j \leq n} \mathcal{TS}_{t_j}(X) \right) + \left(\sum_{s \in \mathcal{R}(\mathtt{f})} \mathcal{TS}_s(X) \right), \quad \text{if } \mathtt{f} \in \mathcal{NF};$$

$$\mathcal{TS}_{(\!(\mathtt{c}\ t_1 \ldots t_n)\!)}(X) = 1 + \left(\sum_{1 \leq j \leq n} \mathcal{TS}_{t_j}(X) \right);$$

$$\mathcal{TS}_{(\!(x\ t_1 \ldots t_n)\!)}(X) = 1 + \left(\sum_{1 \leq j \leq n} \mathcal{TS}_{t_j}(X) \right).$$

Figure 4: The Definition of $\mathcal{TS}_{(\cdot)}(X)$

on the indeterminate X, by induction on $(\partial(t), |t|)$, following the lexicographic order, as in Figure 4. We denote here by $\mathcal{FO}$ (resp. $\mathcal{HO}$) the set of base type (resp. functional type) terms.

The rewrite relation $\Rightarrow$ is defined as $\rightarrow$, except that whenever a recursive function symbol is unfolded, it is unfolded *completely* in just one rewriting step.

The last auxiliary definition we need is a modified notion of size, which attributes size 1 to all base-type values. Formally, the *collapsed size* $||t||$ of a term t is defined by induction on the structure of t:

$$||v|| = 1, \text{ if } v \text{ is a base-type value;}$$
$$||(t_0\ t_1\ \ldots t_n)|| = \sum_{i=0}^{n} ||t_i||;$$
$$||\alpha|| = 1, \text{ if } \alpha \in \mathcal{X} \cup \mathcal{C} \cup \mathcal{F}.$$

Example 5. *For the sake of clarifying the just-introduced concepts, let us give a simple example, namely the STTRS whose only rules are:*

$$(\mathtt{modadd}\ x\ \mathbf{0}\ f) \rightarrow (f\ x);$$
$$(\mathtt{modadd}\ x\ (\mathbf{s}\ y)\ f) \rightarrow (\mathbf{s}\ (\mathtt{modadd}\ x\ y\ f)).$$

Consider the term $t = (\mathtt{modadd}\ \mathbf{3}\ \mathbf{2}\ \mathbf{s})$*, where* $\mathbf{n}$ *stands, as usual, for the value of type NAT containing exactly* n *instances of* $\mathbf{s}$*. The collapsed size* $||t||$ *of* t *is*

just 4*, while* $t \Rightarrow \mathbf{6}$*. But* 6 *is also a bound on base-type values for* t*. Moreover:*

$$\begin{aligned}\mathcal{TS}_t(X) &= 1 + \mathcal{TS}_{\mathbf{3}}(X) + \mathcal{TS}_{\mathbf{2}}(X) + \mathit{arity}(\mathtt{modadd}) \cdot X \cdot \mathcal{TS}_{\mathbf{s}}(X) \\ &\quad + \mathit{arity}(\mathtt{modadd}) \cdot X \cdot \mathcal{TS}_{(\mathbf{s}\ x)}(X) + \mathit{arity}(\mathtt{modadd}) \cdot X \cdot \mathcal{TS}_{(f\ x)}(X) \\ &= 1 + 1 + 1 + 3 \cdot X \cdot 1 + 6 \cdot X + 6 \cdot X = 3 + 15 \cdot X.\end{aligned}$$

We are now ready to explain why the main result of this section holds. First of all, $\mathcal{TS}_t(X)$ is an upper bound on the collapsed size of t:

Lemma 24. *For every* $n \geq 1$ *and for every* t*,* $\mathcal{TS}_t(n) \geq ||t||$*.*

Proof. A simple induction on t. □

In the following we assume that t is well-typed in the linear type system (Figure 2) and we would like to prove that $\mathcal{TS}_t(X)$ decreases along any $\Rightarrow$ step if X is big enough (i.e., if it is a bound on base-type values for t). Preliminary to that is the following:

Lemma 25 (Substitution Lemma). $\mathcal{TS}_{t\{x/v\}}(X) \leq \mathcal{TS}_t(X) + \mathcal{TS}_v(X)$*.*

Proof. This can be proved by induction on t, and intuitively holds because either v has base type, x can occur in t possibly many times but $\mathcal{TS}_v(X) = 1$, or v is an higher-order value, and in that case x occurs at most once in t, at a place where its space-time weight is counted only additively while computing $\mathcal{TS}_t(X)$. □

We are finally able to prove the main result of this Section:

Lemma 26. *If* n *is a bound of base-type values for* t*, and* $t \Rightarrow s$*, then* $\mathcal{TS}_t(n) > \mathcal{TS}_s(n)$*.*

Proof. Suppose that $t \Rightarrow s$ and let $(\!(\mathtt{f}\ r_1 \dots r_m)\!)$ be the redex fired in t to produce s. Then we can say that there are a strictly increasing map $p : \mathbb{N} \to \mathbb{N}$ and a term q such that

$$\begin{aligned}\mathcal{TS}_t(n) &= p(\mathcal{TS}_{(\!(\mathtt{f}\ r_1 \dots r_m)\!)}(n)); \\ \mathcal{TS}_s(n) &= p(\mathcal{TS}_q(n)).\end{aligned}$$

Indeed, p is almost always in the form $p(X) = X + k$ for some k, the only exception being when $(\!(\mathtt{f}\ r_1 \dots r_m)\!)$ appears as an higher-order argument of a function $\mathtt{g} \in \mathcal{RF}$, in which case $p(X) = n \times l \times X + k$ for some k and l. Let us now distinguish two cases:

- If $\mathtt{f} \in \mathcal{RF}$, then q is obtained by at most $m \cdot n$ rewriting steps from $(\!(\mathtt{f}\ r_1 \dots r_m)\!)$ and does not contain any instance of $\mathtt{f}$ anymore. Actually, q consists of at most $n \cdot m$ copies of terms in $\mathcal{R}(\mathtt{f})$ (one applied to the next one), where at most one higher-order variable for each copy is substituted by

any $r_j \in \mathcal{HO}$, and base-type variables are substituted by base-type values. By an easy combinatorial argument, one realizes that, indeed,

$$\mathcal{TS}_q(n) \leq \left(\sum_{\substack{r_j \in \mathcal{HO} \\ j \leq arity(\mathtt{f})}} m \cdot n \cdot \mathcal{TS}_{r_j}(n) \right) + \left(\sum_{s \in \mathcal{R}(\mathtt{f})} m \cdot n \cdot \mathcal{TS}_s(n) \right),$$

which, by definition, is strictly smaller than $\mathcal{TS}_{(\!(\mathtt{f}\ r_1 \ldots r_m)\!)}(n)$.

- if $\mathtt{f} \in \mathcal{NF}$, then q is such that $(\!(\mathtt{f}\ r_1 \ldots r_m)\!) \Rightarrow q$. As a consequence

$$\mathcal{TS}_{(\!(\mathtt{f}\ r_1 \ldots r_m)\!)}(n) = 1 + \left(\sum_{1 \leq j \leq m} \mathcal{TS}_{r_j}(n) \right) + \left(\sum_{w \in \mathcal{R}(\mathtt{f})} \mathcal{TS}_w(n) \right)$$

while q, containing possibly at most one instance of each higher-order value in $r_1, \ldots, r_m$, is such that:

$$\mathcal{TS}_q(n) \leq \left(\sum_{1 \leq j \leq m} \mathcal{TS}_{r_j}(n) \right) + \left(\sum_{w \in \mathcal{R}(\mathtt{f})} \mathcal{TS}_w(n) \right).$$

This concludes the proof. □

It is now easy to reach our goal:

Proposition 27. *Moreover, suppose that* $\mathtt{f}$ *has base values bounded by a function* $q : \mathbb{N}^n \to \mathbb{N}$*. Then, there is a polynomial* $p : \mathbb{N} \to \mathbb{N}$ *such that if* $t_1, \ldots, t_n$ *are base-type values and* $(\mathtt{f}\ t_1 \ldots t_n) \to^m s$*, then* $m, |s| \leq p(q(|t_1|, \ldots, |t_n|))$.

Proof. Let us first prove the following statement: if $(\mathtt{f}\ t_1 \ldots t_n) \Rightarrow^m s$, then there is p polynomial such that $m, ||s|| \leq p(q(|t_1|, \ldots, |t_n|))$ Actually, the polynomial we are looking for is precisely

$$p(X) = \left(\sum_{s \in \mathcal{R}(\mathtt{f})} n \cdot X \cdot \mathcal{TS}_s(X) \right) + n.$$

Indeed, observe that, by definition, as $t_1 \ldots t_n$ are base-type values, we have $\mathcal{TS}_{(\mathtt{f}\ t_1 \ldots t_n)}(X) = q(X)$ and that, by lemmas 26 and 24, this is both a quantity that decreases at any reduction step and which bounds from above the collapsed size of any reduct of $(\mathtt{f}\ t_1 \ldots t_n)$. Now, observe that:

- If m is a bound of first order values for t, then $|t| \leq m \cdot ||t||$;
- If m is a bound of first order values for t, then the number of "real" reduction steps corresponding to each $\Rightarrow$-reduction step from t is bounded by $m \cdot k$, where k is the maximum arity of function symbols in the underlying STTRS;
- As we already mentioned, call-by-value is a constrained reduction relation, and as a consequence the possible number of reduction steps from a term does not depend on the specific reduction order. Similarly for the size of reducts.

This concludes the proof. □

To convince yourself that linearity is needed to get a result like Proposition 27, examine the following STTRS, whose terms cannot be typed in our linear type system.

Example 6. *Consider the program* `expid` *given by:*

$$\begin{aligned}((\texttt{comp}\ f\ g)\ z) &\to (f\ (g\ z));\\ (\texttt{autocomp}\ f) &\to (\texttt{comp}\ f\ f);\\ (\texttt{id}\ x) &\to x;\\ (\texttt{expid}\ \mathbf{0}) &\to \texttt{id};\\ (\texttt{expid}\ (\texttt{s}\ x)) &\to (\texttt{autocomp}\ (\texttt{expid}\ x)).\end{aligned}$$

Both `id` *and* $(\texttt{expid}\ t)$ *(for every value* t *of type* NAT*) can be given type* $NAT \to NAT$*. Actually, they all compute the same function, namely the identity. But try to see what happens if* `expid` *is applied to natural numbers of growing sizes: there is an exponential blowup going on which does not find any counterpart in base-type values.*

Now, to be able to use Proposition 27 we need a way to bound the values of a program, and this is precisely what quasi-interpretations are introduced for.

5.3. Higher-Order Max-Polynomials (HOMPs)

We want to refine the type system for higher-order polynomials, in order to be able to use types to restrict the domain of functionals. The grammar of types is now the following one:

$$S ::= \mathbf{N} \ \Big|\ S \multimap S; \qquad A ::= S \ \Big|\ A \to A.$$

Types of the first (resp. second) grammar are called linear types (resp. types) and denoted as $R, S \dots$ (resp. $A, B, C \dots$). The linear function type $\multimap$ is a subtype of $\to$, i.e., one defines a relation $\ll$ between types by stipulating that $S \multimap R \ll S \to R$ and by closing the rule above in the usual way, namely by imposing that $A \to B \ll C \to E$ whenever $C \ll A$ and $B \ll E$.

We now consider the following new set of constructors:

$$D_P = \{+ : \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}, max : \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}, \times : \mathbf{N} \to \mathbf{N} \to \mathbf{N}\} \cup \{n : \mathbf{N} \mid\ n \in \mathbb{N}^\star\},$$

and we define the following grammar of Church-typed terms

$$\begin{aligned}M, P := x^A \ \Big|\ \mathbf{c}^A \ \Big|\ (M^{A\to B} P^A)^B \ \Big|\ (\lambda x^A.M^B)^{A\to B} \ \Big|\\ (M^{S\multimap R} P^S)^R \ \Big|\ (\lambda x^S.M^R)^{S\multimap R}\end{aligned}$$

where $\mathbf{c}^A \in D_P$. We also require that:

- in $(\lambda x^A.M^B)^{A\to B}$, the variable x^A occurs *at least once* in M^B;
- in $(\lambda x^S.M^R)^{S\multimap R}$, the variable x^S occurs *exactly once* in M^R and in linear position (i.e., it cannot occur on the right-hand side of an application $P^{A\to B}L^A$).

One can check that this class of Church-typed terms is preserved by β-reduction. A *higher-order max-polynomial* (HOMP) is a term as defined above and which is in β-normal form.

Just to prevent confusion let us insist on the fact that the linear typing discipline used here for lambda-terms (HOMPs) is completely separate from the one used in Section 5.1 for STTRS as part of the termination criterion. In particular observe that in the type system of Section 5.1 (Figure 2) variables of base types can be used non-linearly, while it is not the case here.

5.4. Semantic Interpretation

We define the following objects and constructions on objects:

- $\mathcal{N}$ is the domain of *strictly positive* integers, equipped with the natural partial order, denoted here $\leq_{\mathcal{N}}$;
- 1 is the trivial order with one point;
- if σ, τ are objects, then $\sigma \times \tau$ is obtained by the product ordering,
- $\sigma \Rightarrow \tau$ is the set of increasing total functions from σ to τ, equipped with the extensional order: $f \leq_{\sigma\Rightarrow\tau} g$ if for any a in σ we have $f(a) \leq_\tau g(a)$.

This way, one obtains a subcategory $\mathbb{FPOS}$ of the category $\mathbb{POS}$ with partial orders as objects and *increasing* total functions as morphisms. As before with $\prec$ we define $\leq$ so as to compare the semantics of terms which do not have the same free variables.

Sizes. In order to interpret the $\multimap$ construction of the HOMP types in this category we now introduce a notion of *size*. A size is a (finite) multiset of elements of $\mathbb{N}$. The empty multiset will be denoted as $\emptyset$. Given a multiset $\mathcal{S}$, we denote by $\max\mathcal{S}$ its maximal element and by $\sum\mathcal{S}$ the sum of its elements. By convention $\max\emptyset = \sum\emptyset = 0$.

Now, given an object σ of the category $\mathbb{FPOS}$, we say that an element $e \in \sigma$ *admits a size* in the following cases:

- If σ is $\mathcal{N}$, then e is an integer n, and $\mathcal{S}$ is a size of e iff we have: $\max\mathcal{S} \leq n \leq \sum\mathcal{S}$.
- If $\sigma = \sigma_1 \times \cdots \times \sigma_n$, then $\mathcal{S}$ is a size of $e = (e_1, \ldots, e_n)$ iff there exists for any $i \in \{1, \ldots, n\}$ a multiset $\mathcal{S}_i$ which is a size of e_i, and such that $\mathcal{S} = \cup_{i=1}^n \mathcal{S}_i$.
- If $\sigma = \tau \Rightarrow \rho$, then $\mathcal{S}$ is a size of e iff for any f of τ which has a size $\mathcal{T}$, $\mathcal{S} \cup \mathcal{T}$ is a size of $e(f)$. Now, $\tau \twoheadrightarrow \rho$ is defined as the subset of all those functions in σ which admit a size.

We denote by $[\![A]\!]_\leq$ the semantics of A as an object of $\mathbb{FPOS}$, where $\mathbf{N}$ is mapped to $\mathcal{N}$, $\to$ is mapped to $\Rightarrow$ and $\multimap$ to $\twoheadrightarrow$. As for HOPs, any HOMP M can be naturally interpreted as an increasing function between the appropriate partial orders, which we denote by $[\![M]\!]_\leq$. We will speak of the *size* of an HOMP M, by which we mean a size of its interpretation $[\![M]\!]_\leq$. Note that not all terms

admit a size. For instance $\times : \mathbf{N} \to \mathbf{N} \to \mathbf{N}$ does not admit a size. If M reduces to P, then they have the same sizes, if any.

Example 7. *We illustrate the sizes of several terms:*

- *The term n of type $\mathbf{N}$ admits the following sizes: $[n]$, $\underbrace{[1, \ldots, 1]}_{k \text{ times}}$ with $k \geq n$, and more generally $[n_1, \ldots, n_k]$ such that $\forall i \in \{1, k\}$, $n_i \leq n$ and $\sum_{i=1}^{k} n_i \geq n$.*
- *The terms $\lambda x.(x + 3)$, $\lambda x. \max(x, 3)$ of type $\mathbf{N} \multimap \mathbf{N}$ both have size $[3]$, or any $\mathcal{S}$ which is a size of 3.*
- *The terms $\max$ and $+$ of type $\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$ admit as sizes $\emptyset$ and $[0]$.*
- *The term $M = \lambda f.(f\ 2\ 3)$ of type $(\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}) \multimap \mathbf{N}$ has size $[2, 3]$. On the other hand note that e.g. $[4]$ or $[1, 1, 1]$ are not sizes of M because by definition $(M +) \equiv 5$ does not admit as sizes $[4]$ or $[1, 1, 1]$.*

Actually, if we consider first-order terms with types of the form $\mathbf{N} \multimap \ldots \multimap \mathbf{N}$, it is sufficient to consider singletons as sizes. If we only wanted to deal with these terms we could thus use integers for sizes, instead of multisets. Non-singleton multisets only become necessary when we move to higher-order types, as in the last example above:

Proposition 28. *Let M be a closed HOMP of type A. Then $[\![M]\!]_{\leq} \in [\![A]\!]_{\leq}$. Moreover, if A is a linear type S, then $[\![M]\!]_{\leq}$ admits a size.*

Proof. We will focus here on proving that: if M is a closed HOMP of linear type S, then M admits a size. For that we will follow a reducibility-like method. Consider the following predicate $\mathcal{P}(M)$: *if M is a term of type S with free variables $x_1 : S_1, \ldots, x_n : S_n$ which are linear in M, then there exists a multiset $\mathcal{T}$ such that: for any closed $M_i : S_i$ with size $\mathcal{S}_i$, for $1 \leq i \leq n$, $M\{x_1/M_1, \ldots, x_n/M_n\}$ admits size $\cup_{i=1}^{n} \mathcal{S}_i \cup \mathcal{T}$.* We will prove $\mathcal{P}(M)$ for all M in normal form, by induction on M.

- If $M = x$, then x is a free variable, say x_1. The multiset $[0]$ is a size for M.
- If $M = n : \mathbf{N}$, then $[n]$ is a size for M.
- If $M = +$ or $max : \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$, then $[0]$ is a size for M.
- If $M = \lambda x.P$, then M has type $R_1 \multimap R_2$. By definition of HOMPs x is a free variable of P, and we know it has type R_1 which is linear. Denote by $x_1 : S_1, \ldots, x_n : S_n$ the free variables of M, which are linear. As $\mathcal{P}(P)$ holds for P which has free variables $x : R_1, x_1 : S_1, \ldots, x_n : S_n$, we easily deduce that $\mathcal{P}(M)$ holds.
- If $M = (P^{R_2 \multimap S}\ L^{R_2})$: by i.h. we know that $\mathcal{P}(P)$ and $\mathcal{P}(L)$ hold. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two multisets obtained respectively by $\mathcal{P}(P)$ and $\mathcal{P}(L)$. Let $x_1 : S_1, \ldots, x_n : S_n$ be the free variables of M and I (resp. J) be the set of i such that x_i occurs in P (resp. L). Thus I, J form a partition of $\{1, \ldots, n\}$. Let $M_i : S_i$ be closed terms with size $\mathcal{S}_i$, for $1 \leq i \leq n$. We have:
  - $P\{x_1/M_1, \ldots, x_n/M_n\}$ has type $R_2 \multimap S$ and by $\mathcal{P}(P)$ it has size $\cup_{i \in I} \mathcal{S}_i \cup \mathcal{T}_1$;
  - $L\{x_1/M_1, \ldots, x_n/M_n\}$ has type R_2 and by $\mathcal{P}(L)$ it has size $\cup_{i \in J} \mathcal{S}_i \cup \mathcal{T}_2$.

Therefore by definition of the size for $R_2 \multimap S$ we get that $M\{x_1/M_1, \ldots, x_n/M_n\}$ has size $(\cup_{i\in I}\mathcal{S}_i \cup \mathcal{T}_1) \cup (\cup_{i\in J}\mathcal{S}_i \cup \mathcal{T}_2)$, so $\cup_{1\leq i\leq n}\mathcal{S}_i \cup (\mathcal{T}_1 \cup \mathcal{T}_2)$. So $(\mathcal{T}_1 \cup \mathcal{T}_2)$ shows that $\mathcal{P}(M)$ holds. Note that we have used here the fact that the free variables occur linearly in M, in order to deduce that I and J are disjoint.

- If $M = (P^{A\to S}\ L^A)$: as M is assumed to be in normal form it can be written as $M = (\ldots(P\ P_1)\ldots P_k)$ where P is either a variable or a constant. Moreover given the grammar of types, as $A \to S$ is not linear, P must have a non linear type $A_1 \to A_2$. Let us now distinguish the two cases:
  - If P is a variable x, then it is a free variable of M. So M has a free variable with a non linear type, hence the property holds.
  - If P is a constant, as it has a type $A_1 \to A_2$ the only possibility is that $P = \times$, and as M has a linear type S we must have $k = 2$ and $S = \mathbf{N}$. Now, if P_1 or P_2 has a free variable then it is not linear in M, therefore the property holds. Assume that P_1 and P_2 are closed, then M is closed of type $\mathbf{N}$. Let $m = [\![M]\!]_{\leq}$, then $[m]$ is a size for M.

This concludes the proof. □

Proposition 29. *If M is a HOMP of type $\mathbf{N}^k \multimap \mathbf{N}$ with free variables $x_1 : \mathbf{N}, \ldots, x_n : \mathbf{N}$ which are linear in M, then it admits a size of the form $[m]$ where $m \in \mathbb{N}$.*

Proof. The proof is similar to that of Proposition 28. We prove by induction on M the following property, called $\mathcal{Q}(M)$: *if M is a term of type $\mathbf{N} \multimap \mathbf{N} \multimap \ldots \multimap \mathbf{N}$ or $\mathbf{N}$ with free variables $x_1 : \mathbf{N}, \ldots, x_n : \mathbf{N}$ which are linear in M, then there exists a singleton multiset $[m]$ such that: for any closed $M_i : S_i$ with size $[k_i]$, for $1 \leq i \leq n$, $M\{x_1/M_1, \ldots, x_n/M_n\}$ admits size $[k_1, \ldots, k_n, m]$.* See the Appendix for the full proof. □

Before going on let us examine a concrete example on how to compare the interpretation of some HOMPs in $\mathbb{FPOS}$.

Example 8. *Consider the two following HOMPs, which will be useful later for the* `foldr` *program (in Example 9):*

$$\begin{aligned} M &= \lambda f^{\mathbf{N}\multimap\mathbf{N}\multimap\mathbf{N}}.\lambda b^{\mathbf{N}}.\lambda x^{\mathbf{N}}.\lambda y^{\mathbf{N}}.b + (x + y + 1) \times (f\ 1\ 1) \\ &\quad : (\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}) \to \mathbf{N} \to \mathbf{N} \to \mathbf{N} \to \mathbf{N} \\ P &= \lambda f^{\mathbf{N}\multimap\mathbf{N}\multimap\mathbf{N}}.\lambda b^{\mathbf{N}}.\lambda x^{\mathbf{N}}.(f\ x\ (b + y \times (f\ 1\ 1))) \\ &\quad : (\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}) \to \mathbf{N} \to \mathbf{N} \to \mathbf{N} \to \mathbf{N} \end{aligned}$$

We want here to show that $[\![P]\!]_{\leq} \leq [\![M]\!]_{\leq}$. For that consider an element $\phi \in \mathcal{N} \twoheadrightarrow \mathcal{N} \twoheadrightarrow \mathcal{N}$. The crucial point is that we know that ϕ has a size, and thus there exists a $c \geq 0$ such that for every $x, y \in \mathcal{N}$,

$$c \leq \phi\ x\ y \leq x + y + c. \tag{7}$$

Then we have:

$$\begin{array}{rcl}(f\ x\ (b + y \times (f\ 1\ 1)) & \leq & x + b + y \times (f\ 1\ 1) + c \\ & \leq & x \times (f\ 1\ 1) + b + y \times (f\ 1\ 1) + c \\ & \leq & b + (x + y + 1) \times (f\ 1\ 1),\end{array}$$

where for the two last steps we used $(f\ 1\ 1) \geq 1$ *and* $(f\ 1\ 1) \geq c$ *(because of (7)). So we have* $[\![P]\!]_{\leq} \leq [\![M]\!]_{\leq}$.

Now we will establish two lemmas will be useful to obtain later the Subterm Property (Prop. 35):

Lemma 30. *If* M *is a HOMP of type* $\mathbf{N}^m \to \mathbf{N}$ *(with* m *arguments) and such that* $FV(M) = \{y_1 : \mathbf{N}, \ldots, y_k : \mathbf{N}\}$*, then the function* $[\![M]\!]$ *is bounded by a polynomial and satisfies the following inequality for every* i *between* 1 *and* $k + m$*:*

$$([\![M]\!](x_1, \ldots, x_k))(x_{k+1}, \ldots, x_{k+m}) \geq x_i.$$

Proof. We prove the statement by induction on M. The key case is that where M is an application. In that case it can be written as

$$M = ((\ldots((M_0\ )\ M_1\ )\ldots)\ M_n)$$

where M_0 is not an application and $n \geq 1$. Moreover M_0 cannot be an abstraction since M is in β-normal form, and it cannot be a variable y since M can only have free variables of type $\mathbf{N}$. So $M_0 = \mathbf{c}$ for $\mathbf{c} = +, max$ or $\times$, and therefore $n \leq 2$. We obtain that the M_is for $1 \leq i \leq 2$ are of type $\mathbf{N}$ hence also satisfy the hypothesis, and thus by i.h. they satisfy the claim. Therefore the claim is also valid for M. See the Appendix for the other cases of the induction. □

Lemma 31. *For every type* A *there is a closed HOMP of type* A*.*

Proof. By induction on A:

- If A is simply $\mathbf{N}$, one can take 1 for the required HOMP;
- If A is $A_1 \mapsto_1 \ldots A_n \mapsto_n \mathbf{N}$ (where $\mapsto_i$ is either $\to$ or $\multimap$), then the required HOMP is

$$\lambda x_1^{A_1} \ldots . \lambda x_n^{A_n} . (x_1 M_1^1 \ldots M_1^{m_1}) + \ldots + (x_n M_n^1 \ldots M_n^{m_1}),$$

  where the HOMPs M_i^j exist by induction hypothesis.

This concludes the proof. □

5.5. Higher-Order Quasi-Interpretations

Now, a HOMP assignment $[\,\cdot\,]$ is defined by: for any $f^A \in \mathcal{X}$ (resp. $f^A \in \mathcal{C} \cup \mathcal{F}$), $[f]$ is a variable $\underline{f}$ (resp. a closed HOMP M) with a type B, where B is obtained from (the curryfication of) A by:

- replacing each occurrence of a base type D by $\mathbf{N}$,
- replacing each occurrence of $\to$ in A by either $\to$ or $\multimap$.

For instance if $A = (D_1 \to D_2) \to D_3$ we can take for B any of the types: $(\mathbf{N} \multimap \mathbf{N}) \to \mathbf{N}$, $(\mathbf{N} \to \mathbf{N}) \to \mathbf{N}$, etc. In the sequel we will write $\underline{A}$ for any of these types B. Now, if $t = (t_0\ t_1 \dots t_n)$ then $[t]$ is defined if for any $0 \leq i \leq n$, $[t_i]$ is defined and if $[t] \equiv (\dots([t_0][t_1])\dots[t_n])$ is well-typed. We have:

Lemma 32. *Let $t \in \mathcal{T}$ of type A and $FV(t) = \{y_1 : A_1, \dots, y_n : A_n\}$, then: if $[t]$ is defined then it is a HOMP, with a type $\underline{A}$, and $FV([t]) = \{\underline{y_1} : \underline{A_1}, \dots, \underline{y_n} : \underline{A_n}\}$ for some types $\underline{A_i}$, $1 \leq i \leq n$.*

Additive HOMP assignments are defined just as additive HOP assignments.

Lemma 33. *Let $[\,\cdot\,]$ be an additive HOMP assignment. Then there exists $\gamma \geq 1$ such that for any value v of type D, where D is a data type, we have $[\![v]\!]_{\prec} \leq \gamma \cdot |v|$.*

Now, we say that an assignment $[\,\cdot\,]$ is a *quasi-interpretation for R* if for any rule $l \to r$ of R, $[l]$ and $[r]$ are defined and have the same type, and it holds that $[\![l]\!]_{\leq} \geq [\![r]\!]_{\leq}$. Observe that contrarily to the case of polynomial interpretations, these inequalities are not strict, and moreover they are stated with respect to the new domains, taking into account the distinction between the two connectives $\to$ and $\multimap$.

The interpretation of a term does not, like in the strict case, necessarily decrease along a reduction step. However, it cannot increase:

Lemma 34. *If $[\,\cdot\,]$ is a quasi-interpretation and if $t \to^* s$, then $[\![s]\!]_{\leq} \leq [\![t]\!]_{\leq}$.*

Proof. [Sketch] This can be done in a way analogous to what has been done for polynomial interpretations with Lemma 17, using intermediary lemmas for substitutions and contexts similar to lemmas 15 and 16, but it is actually easier because here we are not considering a strict order. □

The previous lemma, together with the possibility of forming HOMPs of arbitrary type (Lemma 31) implies the following, crucial, property:

Proposition 35 (Subterm Property). *Suppose that an STTRS R has an additive quasi-interpretation $[\,\cdot\,]$. Then, for every function symbol $\mathtt{f}$ of arity n with base arguments, there is a polynomial $p : \mathbb{N}^n \to \mathbb{N}$ such that if $(\mathtt{f}\ t_1 \dots t_n) \to^* s$ and if s contains an occurrence of a base term r, then $|r| \leq p(|t_1|, \dots, |t_n|)$.*

Proof. Denote $t = (\mathtt{f}\ t_1 \dots t_n)$. Its type A can be written as $A = A_1, \dots, A_k \to D$, where D is a base type. By Lemma 31, for any $j \in \{1, \dots, k\}$ there is a closed HOMP M_j of type $\underline{A_j}$. Consider now $t' = (\mathtt{f}\ x_1 \dots x_n)$ where for $i \in \{1, \dots, n\}$, x_i is a free variable of same type as t_i. Then $M = [(\mathtt{f}\ x_1 \dots x_n)]M_1\ \dots\ M_k$ is a HOMP of type $\mathbf{N}$ with free variables $\underline{x_i}$ of type $\mathbf{N}$, for $i \in \{1, \dots, n\}$. By Lemma 30 we deduce from that that there exists a polynomial q such that, for $i \in \{1, \dots, n\}$,

$$y_i \leq [\![M]\!]_{\leq}(y_1, \dots, y_n) \leq q(y_1, \dots, y_n).$$

Moreover we have:

$$\begin{aligned}\llbracket([t]\ M_1\ \ldots\ M_k)\rrbracket_{\leq} &= \llbracket([t']\ M_1\ \ldots\ M_k)\rrbracket_{\leq}(\llbracket t_1\rrbracket_{\leq},\ldots,\llbracket t_n\rrbracket_{\leq})\\ &= \llbracket M\rrbracket_{\leq}(\llbracket t_1\rrbracket_{\leq},\ldots,\llbracket t_n\rrbracket_{\leq})\\ &\leq q(\llbracket t_1\rrbracket_{\leq},\ldots,\llbracket t_n\rrbracket_{\leq})\\ &\leq q(\alpha|t_1|,\ldots,\alpha|t_n|)\end{aligned}$$

for some α, because $[\cdot]$ is an additive quasi-interpretation, thanks to Lemma 33. So finally there is a polynomial p such that:

$$\llbracket([t]\ M_1\ \ldots\ M_k)\rrbracket_{\leq} \leq p(|t_1|,\ldots,|t_n|).$$

Now, as $t \to^* s$, by Lemma 34 we have $\llbracket t\rrbracket_{\leq} \geq \llbracket s\rrbracket_{\leq}$. Therefore $\llbracket[t]\ M_1 \ldots\ M_k\rrbracket_{\leq} \geq \llbracket[s]\ M_1 \ldots\ M_k\rrbracket_{\leq}$. So we get: $\llbracket[s]\ M_1 \ldots\ M_k\rrbracket_{\leq} \leq p(|t_1|,\ldots,|t_n|)$. Besides, by assumption we know that s can be written as $s = s'\{y/r\}$. By Lemma 30 we have $\llbracket[s']\ M_1 \ldots\ M_k\rrbracket_{\leq}(y) \geq y$, because $([s']\ M_1 \ldots\ M_k)$ has type $\mathbf{N}$ and only one free variable y which is also of type $\mathbf{N}$. Therefore we get $\llbracket[s]\ M_1 \ldots\ M_k\rrbracket_{\leq} = \llbracket[s']\ M_1 \ldots\ M_k\rrbracket_{\leq}(\llbracket r\rrbracket_{\leq}) \geq \llbracket r\rrbracket_{\leq}$. So finally by combining the two inequalities we obtained we get $\llbracket r\rrbracket_{\leq} \leq \llbracket[s]\ M_1 \ldots\ M_k\rrbracket_{\leq} \leq p(|t_1|,\ldots,|t_n|)$. □

And here is the main result of this Section:

Theorem 36 (Polytime Soundness). *If an STTRS R has an additive quasi-interpretation, R satisfies the termination criterion and* `f` *has arity n with base type arguments, then there is a polynomial $p : \mathbb{N}^n \to \mathbb{N}$ such that whenever $(\texttt{f}\ t_1 \ldots t_n) \to^m s$, it holds that $m, |s| \leq p(|t_1|,\ldots,|t_n|)$. So if* `f` *has a type $D_1 \times \cdots \times D_n \to D$ then instances of* `f` *can be computed in polynomial time.*

Proof. This is obtained by combining Proposition 35 and Proposition 27. □

Notice how Theorem 36 is proved by first observing that terms of STTRSs having a quasi-interpretation are bounded by natural numbers which are not too big with respect to the input, thus relying on the termination criterion to translate these bounds to *complexity* bounds.

Higher-order quasi interpretations, like their strict siblings, can be extended by enlarging D_P so as to include more combinators, provided they are increasing and bounded by polynomials.

5.6. Examples

Example 9. *Consider again the program* `foldr` *already mentioned in Example 4:*

$$((\texttt{foldr}\ f\ b)\ \mathbf{nil}) \to b; \tag{8}$$

$$((\texttt{foldr}\ f\ b)\ (\mathbf{cons}\ x\ xs)) \to (f\ x\ ((\texttt{foldr}\ f\ b)\ xs)); \tag{9}$$

where functions, variables and constructors have the following types:

$$\begin{aligned}\texttt{foldr} &: (D_1 \times D_2 \to D_2) \times D_2 \to L(D_1) \to D_2;\\ f &: D_1 \times D_2 \to D_2;\\ \mathbf{nil} &: L(D_1);\\ \mathbf{cons} &: D_1 \times L(D_1) \to L(D_1).\end{aligned}$$

We examine the two conditions of our complexity criterion, namely the termination criterion and the existence of quasi-interpretations:

- *Termination criterion: it has been checked in Example 4.*
- *As for quasi-interpretations, we choose as assignment:*

$$[\mathbf{nil}] = 1 : \mathbf{N}; \qquad [\mathbf{cons}] = \lambda n.\lambda m.n + m + 1 : \mathbf{N} \to \mathbf{N} \to \mathbf{N};$$
$$[\texttt{foldr}] = \lambda\phi.\lambda p.\lambda n.p + n \times (\phi\ 1\ 1) : (\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}) \to \mathbf{N} \to \mathbf{N} \to \mathbf{N}.$$

  Observe the $\multimap$ in the type of the first argument of $[\texttt{foldr}]$ *which is the way to restrict the domain of arguments. We then obtain the following interpretations of terms:*

$$[((\texttt{foldr}\ f\ b)\ \mathbf{nil})] = \underline{b} + 1 \times (\underline{f}\ 1\ 1);$$
$$[((\texttt{foldr}\ f\ b)\ (\mathbf{cons}\ x\ xs))] = \underline{b} + (\underline{x} + \underline{xs} + 1) \times (\underline{f}\ 1\ 1);$$
$$[(f\ x\ ((\texttt{foldr}\ f\ b)\ xs))] = \underline{f}\ \underline{x}\ (\underline{b} + \underline{xs} \times (\underline{f}\ 1\ 1)).$$

  The condition $[\![r]\!]_{\leq} \leq [\![l]\!]_{\leq}$ holds for (8), because we have that $p \leq p + \phi(1,1)$ holds for any p. As to rule (9), we have proven in Example 8 that $[\![r]\!]_{\leq} \leq [\![l]\!]_{\leq}$ holds. Therefore this assignment is an additive quasi-interpretation.

Summing up, we can apply Theorem 36 and conclude that if the termination criterion is satisfied by all functions, if $t^{D_1 \times D_2 \to D_2}$, b^{D_2} are terms and $[t]$ is a HOMP with type $\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$, then $(\texttt{foldr}\ t\ b)$ is a polynomial time program of type $L(D_1) \to D_2$. Let us give a concrete example of usage of `foldr`*. Consider the following rules defining* `append`*:*

$$\texttt{append}\ \mathbf{nil}\ ys \to ys \tag{10}$$
$$\texttt{append}\ (\mathbf{cons}\ x\ xs)\ ys \to \mathbf{cons}\ x\ (\texttt{append}\ xs\ ys) \tag{11}$$

with the type $\texttt{append} : L(D) \times L(D) \to L(D)$*. It is easy to check that these rules satisfy the termination criterion. As to the assignments we take:* $[\mathbf{nil}]$ *and* $[\mathbf{cons}]$ *as above, and set*

$$[\texttt{append}] = \lambda n.\lambda m.(n + m + 1) : \mathbf{N} \to \mathbf{N} \to \mathbf{N};$$

Now, the term $\lambda n.\lambda m.(n + m + 1)$ can also be given the type $\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$. We reconsider the typing of `foldr` *by renaming both D_1 and D_2 as $L(D)$, and define* `listappend` *of type $L(L(D)) \to L(D)$ by:*

$$\texttt{listappend}\ x \to ((\texttt{foldr}\ \texttt{append}\ \mathbf{nil})\ x); \tag{12}$$

Then `listappend` *admits as quasi-interpretation the normal form of the HOMP* $([\texttt{foldr}][\texttt{append}]\ [\mathbf{nil}])$*, which is well-defined. Thus the whole program satisfies the complexity criterion and* `listappend` *is polynomial time.*

Example 10. *The second example will be close to Example 3 in Section 4.5, that we used to illustrate higher-order interpretations. The point of this new example will be to show that HOQIs are slightly more natural, and to prepare for Example 11. We consider a program* `filter` *which given a predicate f on D and a list of elements in D, will return the sublist of elements satisfying predicate f. We define for that:*

$$((\texttt{filter}\ f)\ \mathbf{nil}) \to \mathbf{nil}; \tag{13}$$

$$((\texttt{filter}\ f)\ (\mathbf{cons}\ x\ xs)) \to \texttt{cond}\ (f\ x)\ x\ (\texttt{filter}\ f\ xs); \tag{14}$$

$$(\texttt{cond}\ \texttt{true}\ x\ y) \to \mathbf{cons}\ x\ y \tag{15}$$

$$(\texttt{cond}\ \texttt{false}\ x\ y) \to y \tag{16}$$

where functions, variables and constructors have the following types:

$$\begin{aligned} \texttt{filter} &: (D \to BOOL) \to L(D) \to L(D);\\ f &: D \to BOOL;\\ \texttt{true} &: BOOL;\\ \texttt{false} &: BOOL;\\ \texttt{cond} &: BOOL \times D \times L(D) \to L(D); \end{aligned}$$

- *As for the termination criterion, we set* $\texttt{cond} \sqsubset \texttt{filter}$*. The only rule with a recursive call is (14). Observe that the r.h.s. is of the form $r\{y/(\texttt{filter}\ f\ xs)\}$, with $r = \texttt{cond}\ (f\ x)\ x\ y$. The term r is typable in the linear type system, we have $r \sqsubset \texttt{filter}$ and in the recursive call xs is a strict subterm of $(\mathbf{cons}\ x\ xs)$. So this rule satisfies the condition. So the termination criterion is satisfied.*
- *We choose as assignment:*

$$\begin{aligned} [\mathbf{nil}] &= 1 : \mathbf{N};\\ [\mathbf{cons}] &= \lambda n.\lambda m.n + m + 1 : \mathbf{N} \to \mathbf{N} \to \mathbf{N};\\ [\texttt{true}] &= [\texttt{false}] = 1 : \mathbf{N};\\ [\texttt{filter}] &= \lambda\phi.\lambda n.n \times (\phi\ n) : (\mathbf{N} \to \mathbf{N}) \to \mathbf{N} \to \mathbf{N};\\ [\texttt{cond}] &= \lambda k.\lambda m.\lambda n.(k + m + n + 1) : \mathbf{N} \to \mathbf{N} \to \mathbf{N} \to \mathbf{N}. \end{aligned}$$

Note that contrary to what happened in Section 4.5, we do not choose here $[\mathbf{nil}] = 2$, but $[\mathbf{nil}] = 1$, which is slightly more natural. One can then check that all rules satisfy the condition $[\![r]\!]_{\leq} \leq [\![l]\!]_{\leq}$.

Example 11. *Now we want to consider a program* `2filter` *which given a predicate g on $D \times D$ and two lists l_1, l_2 in $L(D)$, will return a list consisting of all pairs (x_1, x_2) of elements of l_1, l_2 which satisfy the predicate g. Note that this program can have a quadratic size output. We define:*

$$((\texttt{2filter}\ g)\ \mathbf{nil}\ ys) \to \mathbf{nil}; \tag{17}$$

$$((\texttt{2filter}\ g)\ (\mathbf{cons}\ x\ xs)\ ys) \to \texttt{append}\ (\texttt{filter}\ (g\ x)\ ys)\ ((\texttt{2filter}\ g)\ xs\ ys); \tag{18}$$

where functions, variables and constructors have the following types:

$$\begin{aligned} \texttt{2filter} &: (D \to D \to BOOL) \to L(D) \times L(D) \to L(D \times D);\\ g &: D \to D \to BOOL;\\ \texttt{append} &: L(D) \times L(D) \to L(D); \end{aligned}$$

Now:

- *Take for the order:* $\texttt{filter} \sqsubset \texttt{2filter}$ *and* $\texttt{append} \sqsubset \texttt{2filter}$. *Concerning the rule (18) note that the term* $(\texttt{append}\ (\texttt{filter}\ (g\ x)\ ys)\ z)$ *is typable in the linear type system. Moreover the condition on recursive calls is satisfied. It is easy to check that the rule (11) also satisfies the conditions. So the termination criterion is satisfied.*
- *We choose the following assignments (and define* $[\mathbf{nil}]$, $[\mathbf{cons}]$, $[\texttt{filter}]$ *as above):*

  $$\begin{aligned} [\texttt{append}] &= \lambda n.\lambda m.(n+m+1) : \mathbf{N} \to \mathbf{N} \to \mathbf{N};\\ [\texttt{2filter}] &= \lambda\psi.\lambda n.\lambda m.n \times m \times (\psi\ n\ m) : (\mathbf{N} \to \mathbf{N} \to \mathbf{N}) \to \mathbf{N} \to \mathbf{N} \to \mathbf{N}. \end{aligned}$$

  Now let us examine the interpretation of the terms of the l.h.s. and r.h.s. of (18), where we use as notation $\psi = \underline{f}$, $n = \underline{x}$, $ns = \underline{xs}$, $m = \underline{ys}$:

  $$\begin{aligned} &[(\texttt{2filter}\ g)\ (\mathbf{cons}\ x\ xs)\ ys] =\\ &\qquad = (n + ns + 1) \times y \times (\psi\ (n + ns + 1)\ m);\\ &[\texttt{append}\ (\texttt{filter}\ (g\ x)\ ys)\ ((\texttt{2filter}\ g)\ xs\ ys)]\\ &\qquad = [\texttt{append}]([\texttt{filter}]\ (\psi\ n)\ m)([\texttt{2filter}]\ \psi\ ns\ m)\\ &\qquad = 1 + m \times (\psi\ n\ m) + ns \times m \times (\psi\ ns\ m) \end{aligned}$$

  We can then observe that we have:

  $$(n+ns+1) \times y \times (\psi\ (n+ns+1)\ m) \geq 1 + m \times (\psi\ n\ m) + ns \times m \times (\psi\ ns\ m),$$

  by using the increasing property of ψ *and the fact that all elements are in* $\mathcal{N}$, *hence superior or equal to 1. So rule (18) satisfies the condition* $[\![r]\!]_{\leq} \leq [\![l]\!]_{\leq}$. *The same can be verified for the other three rules.*

This example is interesting for comparison with related work. Indeed a program such as `2filter` *cannot be typed in the language* LFPL *of [25] because it has a quadratic size result.*

6. Embeddings

In this section, we compare two heterogeneous ICC systems to our higher-order quasi-interpretation approach. First, we show how Bellantoni and Cook's function algebra [8] can be embedded in STTRS programs satisfying our criterion. Then we consider non-size-increasing polynomial time functions and explain why we conjecture they could also be reproved polynomial time sound using the present criterion.

6.1. Bellantoni and Cook's Function Algebra

We assume some familiarity with the algebra BC of safe recursive definitions (see [8] for some details). The only datatype one needs here is W_2, with associated constructors $\mathbf{empty} : W_2$ and $\mathbf{c}_0, \mathbf{c}_1 : W_2 \to W_2$. To every n-ary BC function f one can associate a STTRS R_f on a function symbol $\mathtt{f}$ computing f; the definition goes as expected by induction on the proof that f is indeed a BC function. As an example, if f is defined by composition from $g_1, \ldots, g_{k+p}$ and h, then the rules of R_f will be the ones of $R_{g_1}, \ldots, R_{g_{k+p}}, R_h$, plus the following one:

$$(\mathtt{f}\ \vec{x}; \vec{y}) \to (\mathtt{h}\ (\mathtt{g}_1\ \vec{x};), \ldots, (\mathtt{g}_k\ \vec{x};); (\mathtt{g}_{k+1}\ \vec{x}; \vec{y}), \ldots, (\mathtt{g}_{k+p}\ \vec{x}; \vec{y})).$$

where $\vec{x}$ stands for $x_1 \ldots x_n$, and $\vec{y}$ for $y_1, \ldots, y_m$. The fact any such R_f satisfies the termination criterion is quite easy to verify: the syntactic ingredient is a consequence of the inductive construction behind R_f, while the typing ingredient is trivially verified (there is not any higher-order variable around). About the existence of additive quasi-interpretations for every such R_f: they can all be defined as a function in the form $\max\{x_1, \ldots, x_n\} + p(y_1 + \ldots + y_m)$ where $\overline{x}$ are the parameters corresponding to safe arguments, while $\overline{y}$ correspond to normal arguments. This, by the way, very much follows the original soundness proof in [8] (and also the embedding of BC into TRSs from [12]).

6.2. Non-Size-Increasing Polytime Functions

LFPL is a calculus for non-size-increasing functions introduced by Hofmann [25]. Again, in the following we assume familiarity with it. Terms of a significant fragment of LFPL can be turned into STTRSs following a scheme similar to the one we used for the simply-typed λ-calculus. The key ingredients of the encoding are as follows:

- The type $\diamond$ is mapped into the data-type $D_\diamond$ which only has one 0-ary constructor $\star : D_\diamond$. The term $\star$, however does *not* occur in the encoding of LFPL terms. There is also a type NAT with the usual constructors $\mathbf{s}$ and $\mathbf{0}$
- LFPL's successor, which has type $\diamond \multimap \mathbf{N} \multimap \mathbf{N}$, is *not* mapped into $\mathbf{s}$ but rather to a *function* symbol of type $D_\diamond \to NAT \to NAT$.
- Applications and λ-abstractions are translated in the natural way, as in T.
- Iteration is itself translated in a natural way: given two LFPL closed terms $M : \diamond \multimap A \multimap A$ and $L : \diamond \multimap A$ which are translated to STTRSs and terms t and s of corresponding types, the iteration of M and L becomes a function symbol $\mathtt{f}_{t,s} : NAT \to A$ with the following rules:

$$(\mathtt{f}_{t,s}\ \mathbf{0}) \to (s\ \star);$$
$$(\mathtt{f}_{t,s}\ (\mathbf{s}\ x)) \to ((t\ \star)\ (\mathtt{f}_{t,s}\ x)).$$

The termination criterion is easily seen to be satisfied. Assembling an assingment for the function symbols in such a way as to get a quasi-intepretation is harder. Such an assignment should *only* make use of the linear function space $\multimap$ and not $\to$. Moreover, the size of the term interpreting any LFPL term M

should be the empty multiset. Unfortunately, HOMPs lack a construct which iterates a function $A \multimap A$ (of empty size) and turns it into something of type $\mathbf{N} \multimap A$ (which is essential), and whose semantics would be perfectly consistent with the interpretation of HOMPs from Section 5.4. We conjecture that this way LFPL can be (re)proved polytime sound. Please observe that the first-order fragment of LFPL is not problematic in this respect, because, essentially, there is only one HOMP from $\mathbf{N}$ to $\mathbf{N}$ of empty size (see also [1]).

7. Discussion

The authors believe that the interest of the present work does not lie much in bringing yet another ready-to-use ICC system but rather in offering a new *framework* in which to design ICC systems and prove their complexity properties. Indeed, considered as an ICC system our setting presents two limitations:

1. given a program one needs to *find* an assignment and to *check* that it is a valid quasi-interpretation, which in general will be difficult to automatize;
2. the termination criterion currently does not allow to reuse higher-order arguments in full generality.

To overcome 2. we think it would be possible to design more liberal termination criteria, while attacking 1. could possibly consist in defining type systems such that if a program is well-typed, then it admits a quasi-interpretation, and for which one could devise type-inference algorithms. On the other hand, recently introduced techniques for inferring higher-order polynomial interpretations [21] could shed some light on this issue, which is however outside the scope of this paper.

8. Relations to Other ICC Systems

Let us first compare our approach to other frameworks for proving complexity soundness results. At first-order, we have already emphasized the fact that our setting is an extension of the quasi-interpretation approach [11] (see also [1] for the relation with non-size-increasing, at first-order). We could examine whether various flavours of termination criteria and interpretations (e.g. sup-interpretations) could suggest ideas in our higher-order setting. At higher-order, various approaches based on realizability have been used [17, 14]. While these approaches were developed for logics or System T-like languages, our setting is adapted to a language with recursion and pattern-matching. We think it might also be easier to use in practice.

Let us now discuss the relations with known ICC systems. Several variants of System T based on restriction of recursion and linearity conditions [24, 9, 15] have been proposed which characterize polynomial time. With respect to Hofmann's LFPL [25], the advantages we bring are a slightly more general handling of higher-order arguments, but also the possibility to capture size-increasing polytime algorithms. As an example, we are able to assign a quasi-interpretation

to (STTRSs computing) functions in Bellantoni and Cook's algebra BC [8] (see Section 6).

Some other works are based on type systems built out of variants of linear logic [7, 22, 5]. They are less expressive for first-order functions but offer more liberal disciplines for handling higher-order arguments. In future work we will examine if they could suggest a more flexible termination condition, maybe itself based on quasi-interpretations, following [16].

9. Conclusions

We have advocated the usefulness of simply-typed term rewriting systems to smoothly extend notions from first-order rewriting systems to the higher-order setting. Our main contribution is a new framework for studying (and distilling) ICC systems for higher-order languages. While up to now quite distinct techniques had been successful for providing expressive criteria for polynomial time complexity at first-order and at higher-order respectively, our approach brings together these techniques: interpretation methods on the one hand, and semantic domains and type systems on the other. We have illustrated the strength of this framework by designing an ICC system for polynomial time based on a termination criterion and on quasi-interpretations, which allows to give some sufficient conditions for programs built from higher-order combinators (like `foldr`) to work in bounded time. We think this setting should allow in future work to devise new, more expressive, systems for ensuring complexity bounds for higher-order languages.

References

[1] Roberto Amadio. Synthesis of max-plus quasi-interpretations. *Fundam. Inform.*, 65:29–60, 2005.

[2] Takahito Aoto and Toshiyuki Yamada. Termination of simply typed term rewriting by translation and labelling. In *RTA 2003*, volume 2706 of *LNCS*. Springer, 2003.

[3] Martin Avanzini, Ugo Dal Lago, and Georg Moser. Analysing the complexity of functional programs: higher-order meets first-order. In *ICFP 2015*, pages 152–164, 2015.

[4] Martin Avanzini and Georg Moser. A combination framework for complexity. In *RTA 2013*, volume 21 of *LIPIcs*, pages 55–70, 2013.

[5] Patrick Baillot, Marco Gaboardi, and Virgile Mogbil. A polytime functional language from light linear logic. In *ESOP 2010*, volume 6012 of *LNCS*, pages 104–124, 2010.

[6] Patrick Baillot and Ugo Dal Lago. Higher-order interpretations and program complexity. In *CSL 2012*, volume 16 of *LIPIcs*, pages 62–76, 2012.

[7] Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda calculus. *Inf. Comput.*, 207(1):41–62, 2009.

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

[9] Stephen J. Bellantoni, Karl-Heinz Niggl, and Helmut Schwichtenberg. Higher type recursion, ramification and polynomial time. *Ann. Pure Appl. Logic*, 104(1-3):17–30, 2000.

[10] Guillaume Bonfante, Adam Cichon, Jean-Yves Marion, and Hélène Touzet. Algorithms with polynomial interpretation termination proof. *J. Funct. Program.*, 11(1):33–53, 2001.

[11] Guillaume Bonfante, J.-Y. Marion, and Jean-Yves Moyen. Quasi-interpretations a way to control resources. *Theor. Comput. Sci.*, 412(25):2776–2796, 2011.

[12] Guillaume Bonfante, Jean-Yves Marion, and Jean-Yves Moyen. On lexicographic termination ordering with space bound certifications. In *Ershov Memorial Conference*, volume 2244 of *LNCS*, pages 482–493. Springer, 2001.

[13] Guillaume Bonfante, Jean-Yves Marion, and Romain Péchoux. Quasi-interpretation synthesis by decomposition. In *ICTAC 2007*, volume 4711 of *LNCS*, pages 410–424. Springer, 2007.

[14] Aloïs Brunel and Kazushige Terui. Church => Scott = Ptime: an application of resource sensitive realizability. In *DICE 2010*, volume 23 of *EPTCS*, pages 31–46, 2010.

[15] Ugo Dal Lago. The geometry of linear higher-order recursion. In *LICS 2005*, pages 366–375, 2005.

[16] Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. In *LICS 2011*, pages 133–142, 2011.

[17] Ugo Dal Lago and Martin Hofmann. Realizability models and implicit complexity. *Theor. Comput. Sci.*, 412(20):2029–2047, 2011.

[18] Ugo Dal Lago and Simone Martini. Derivational complexity is an invariant cost model. In *FOPARA 2009*, volume 6324 of *LNCS*, pages 88–101. Springer, 2009.

[19] Ugo Dal Lago and Simone Martini. On constructor rewrite systems and the lambda-calculus. In *ICALP 2009*, volume 5556 of *LNCS*, pages 163–174. Springer, 2009.

[20] Nachum Dershowitz. Orderings for term-rewriting systems. *Theor. Comput. Sci.*, 17(3):279–301, 1982.

[21] Carsten Fuhs and Cynthia Kop. Polynomial interpretations for higher-order rewriting. In *RTA 2012*, volume 15 of *LIPIcs*, pages 176–192. Schloss Dagstuhl, 2012.

[22] Marco Gaboardi and Simona Ronchi Della Rocca. A soft type assignment system for lambda -calculus. In *CSL 2007*, volume 4646 of *LNCS*, pages 253–267. Springer, 2007.

[23] Martin Hofmann. A mixed modal/linear lambda calculus with applications to Bellantoni-Cook safe recursion. In *CSL 1997*, volume 1414 of *LNCS*, pages 275–294, 1997.

[24] Martin Hofmann. Safe recursion with higher types and BCK-algebra. *Ann. Pure Appl. Logic*, 104(1-3):113–166, 2000.

[25] Martin Hofmann. Linear types and non-size-increasing polynomial time computation. *Inf. Comput.*, 183(1):57–85, 2003.

[26] Jean-Pierre Jouannaud and Mitsuhiro Okada. A computation model for executable higher-order algebraic specification languages. In *LICS 1991*, pages 350–361, 1991.

[27] Jean-Pierre Jouannaud and Albert Rubio. The higher-order recursive path ordering. In *LICS 1999*, pages 402–411, 1999.

[28] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theor. Comput. Sci.*, 121(1&2):279–308, 1993.

[29] D. Lankford. On proving term rewriting systems are noetherian. Technical Report MTP-3, Louisiana Tech. University, 1979.

[30] D. Leivant. Predicative recurrence and computational complexity I: word recurrence and poly-time. In *Feasible Mathematics II*, pages 320–343. Birkhauser, 1994.

[31] Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of poly-time. *Fundam. Inform.*, 19(1/2), 1993.

[32] Jean-Yves Marion and Jean-Yves Moyen. Efficient First Order Functional Program Interpreter with Time Bound Certifications. In *LPAR 2000*, volume 1955 of *LNAI*, pages 25–42. Springer, 2000.

[33] Georg Moser and Andreas Schnabl. The derivational complexity induced by the dependency pair method. *Logical Methods in Computer Science*, 7(3), 2011.

[34] Jaco van de Pol. *Termination of Higher-order Rewrite Systems*. PhD thesis, Utrecht University, 1996.

[35] Toshiyuki Yamada. Confluence and termination of simply typed term rewriting systems. In *RTA 2001*, volume 2051 of *LNCS*, pages 338–352. Springer, 2001.

Appendix

Proof. [of Prop. 29] We will prove by induction on M the following property $\mathcal{Q}(M)$: *if M is a term of type $\mathbf{N} \multimap \mathbf{N} \multimap \ldots \multimap \mathbf{N}$ or $\mathbf{N}$ with free variables $x_1 : \mathbf{N}, \ldots, x_n : \mathbf{N}$ which are linear in M, then there exists a singleton multiset $[m]$ such that: for any closed $M_i : S_i$ with size $[k_i]$, for $1 \leq i \leq n$, $M\{x_1/M_1, \ldots, x_n/M_n\}$ admits size $[k_1, \ldots, k_n, m]$.*

- If M is a variable or $M = +$ or max, $\mathcal{Q}(M)$ holds with the size $[0]$.
- If $M = \lambda x.P$, then by i.h. $\mathcal{Q}(P)$ holds with a size $[p]$. It is easy to check that $\mathcal{Q}(M)$ also holds with the size $[p]$.
- If $M = (P^{A \to S}\ L^A)$: this case is handled as in the proof of Prop. 28.
- If $M = (P^{R_2 \multimap S}\ L^{R_2})$: then as M is assumed to be in normal form it can be written as $M = (\ldots (J\ J_1) \ldots J_k)$ where J is either a variable or a constant. As M only has free variables of type $\mathbf{N}$ we deduce that J is a constant, and it can thus be either $+$ or max and we have $k = 1$ or 2. Take $P = max$ and $k = 2$ (the other cases are similar). We know by i.h. that $\mathcal{Q}(J_1)$ and $\mathcal{Q}(J_2)$ hold, so let $[n_1]$ and $[n_2]$ be two multisets for these properties. We take $n = max(n_1, n_2)$. One can then check that $[n]$ establishes property $\mathcal{Q}(M)$.

□

Proof. [of Lemma 30] We prove the statement by induction on M:

- If $M = x$ then the statement holds.
- If $M = n, +, max$ or $\times$, then the statement is obviously also true (note that for $\times$ we are using the fact that the base domain is $\mathbb{N}^*$ and not $\mathbb{N}$).
- If M is an application, it can be written as $M = (\ldots (M_0\ )\ M_1\ ) \ldots)\ M_n)$ where M_0 is not an application and $n \geq 1$. Moreover M_0 cannot be an abstraction since M is in β-normal form, and it cannot be a variable y since M can only have free variables of type $\mathbf{N}$. So $M_0 = \mathbf{c}$ for $\mathbf{c} \in \{+, max, \times\}$, and therefore $n \leq 2$. We obtain that the M_is for $1 \leq i \leq 2$ are of type $\mathbf{N}$ hence also satisfy the hypothesis, and thus by i.h. they satisfy the claim. Therefore the claim is also valid for M.
- Finally the only possibility left is $M = \lambda x.P$. By definition of HOMPs we know then that x is a free variable of P of type $\mathbf{N}$, and as P satisfies the hypothesis, by i.h. we know that P satisfies the claim. Therefore the claim is valid for M.

This concludes the proof. □