On Basic Feasible Functionals and the Interpretation Method

Patrick Baillot, Ugo Dal Lago, Cynthia Kop, Deivid Vale

On Basic Feasible Functionals and the Interpretation Method*

Patrick Baillot[1], Ugo Dal Lago[2], Cynthia Kop[3], and Deivid Vale[4]

[1] Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France, patrick.baillot@univ-lille.fr
[2] University of Bologna & INRIA Sophia Antipolis, ugo.dallago@unibo.it
[3] Radboud University Nijmegen, c.kop@cs.ru.nl
[4] Radboud University Nijmegen, deividvale@cs.ru.nl

Abstract. The class of basic feasible functionals (BFF) is the analog of FP (polynomial time functions) for type-2 functionals, that is, functionals that can take (first-order) functions as arguments. BFF can be defined through Oracle Turing machines with running time bounded by second-order polynomials. On the other hand, higher-order term rewriting provides an elegant formalism for expressing higher-order computation. We address the problem of characterizing BFF by higher-order term rewriting. Various kinds of interpretations for *first-order* term rewriting have been introduced in the literature for proving termination and characterizing (first-order) complexity classes. In this paper, we consider a recently introduced notion of cost–size interpretations for higher-order term rewriting and see definitions as ways of computing functionals. We then prove that the class of functionals represented by higher-order terms admitting a certain kind of cost–size interpretation is exactly BFF.

Keywords: Basic Feasible Functions · Higher-Order Term Rewriting · Tuple Interpretations · Computational Complexity

1 Introduction

Computational complexity classes, and in particular those relating to polynomial time and space [19,10] capture the concept of a feasible problem, and as such have been scrutinized with great care by the scientific community in the last fifty years. The fact that even apparently simple problems, such as nontrivial separation between those classes, remain open today has highlighted the need for a comprehensive study aimed at investigating the deep nature of computational complexity. The so-called implicit computational complexity [7,29,32,12,4] fits into this picture, and is concerned with characterizations of complexity classes

* This work is supported by the NWO TOP project "Implicit Complexity through Higher-Order Rewriting", NWO 612.001.803/7571, the NWO VIDI project "Constrained Higher-Order Rewriting and Program Equivalence", NWO VI.Vidi.193.075, and the ERC CoG "Differential Program Semantics", GA 818616.

based on tools from mathematical logic and the theory of programming languages.

One of the areas involved in this investigation is certainly that of term rewriting [33], which has proved useful as a tool for the characterization of complexity classes. In particular, the class FP (i.e., of polytime first-order functions) has been characterized through variations of techniques originally introduced for *termination*, e.g., the interpretation method [30,28], path orders [14], or dependency pairs [15]. Some examples of such characterizations can be found in [6,8,9,1,3].

After the introduction of FP, it became clear that the study of computational complexity also applies to *higher-order functionals*, which are functions that take not only data but also other functions as inputs. The pioneering work of Constable [11], Mehlhorn [31], and Kapron and Cook [21] laid the foundations of the so-called higher-order complexity, which remains a prolific research area to this day. Some motivations for this line of work can be found e.g. in computable analysis [23], NP search problems [5], and programming language theory [13].

There have been several proposals for a class of type-two functionals that correctly generalizes FP. However, the most widely accepted one is the class BFF of *basic feasible functionals*. This class can be characterized based on function algebras, similar to Cobham-style, but it can also be described using Oracle Turing machines. The class BFF was then the object of study by the research community, which over the years has introduced a variety of characterizations, e.g., in terms of programming languages with restricted recursion schemes [20,13], typed imperative languages [16,17], and restricted forms of iteration in OTMs [22]. An investigation of higher-order complexity classes employing the higher-order interpretation method (in the context of a pure higher-order functional language) was also proposed in [18]. However, this paper does not provide a characterization of the standard BFF class. Instead, it characterizes a newly proposed class $\mathtt{SFF}_2$ (Safe Feasible Functionals) which is defined as the restriction of BFF to argument functions in FP (see Sect. 4.2 and the conclusion in [18]).

The studies cited above present structurally complex programming languages and logical systems, precisely due to the presence of higher-order functions. It is not currently known whether it is possible to give a characterization of BFF in terms of mainstream concepts of rewriting theory, although the latter has long been known to provide tools for the modeling and analysis of functional programs with higher-order functions [24].

This paper goes precisely in that direction by showing that the interpretation method in the form studied by Kop and Vale [26,25] provides the right tools to characterize BFF. More precisely, we consider a class of higher-order rewriting systems admitting cost–size tuple interpretations (with some mild upper-bound conditions on their cost and size components) and show that this class contains exactly the functionals in BFF. Such a characterization could not have been obtained employing classical integer interpretations as e.g. in [8] because BFF crucially relies on some conditions both on size and on time. This is the main contribution of our paper, formally stated in Theorem 2.

We believe that a benefit of this characterization is that it opens the way to effectively handling programs or executable specifications implementing BFF functions, in full generality. For instance, we expect that such a characterization could be integrated into rewriting-based tools for complexity analysis of term rewriting systems such as e.g. [2].

Our result is proved in two parts. We first prove that if any term rewriting system in this class computes a higher-order functional, then this functional has to be in BFF (*soundness*). Conversely, we prove that all functionals in BFF are computed by this class of rewriting systems (*completeness*). We argue that the key ingredient towards achieving this characterization is the ability to split the dual notions of cost and size given by the usage of tuple interpretations.

2 Preliminaries

2.1 Higher-Order Rewriting

We roughly follow the definition of *simply-typed term rewriting system* [27] (STRS): terms are applicative, and we limit our interest to *second-order* STRSs where all rules have base type. Reductions follow an innermost evaluation strategy.

Let $\mathbb{B}$ be a nonempty set whose elements are called *base types* and range over ι, κ, ν. The set $\mathbb{T}(\mathbb{B})$ of *simple types* over $\mathbb{B}$ is defined by the grammar $\mathbb{T}(\mathbb{B}) \coloneqq \mathbb{B} \mid \mathbb{T}(\mathbb{B}) \Rightarrow \mathbb{T}(\mathbb{B})$. Types from $\mathbb{T}(\mathbb{B})$ are ranged over by σ, τ, ρ. The $\Rightarrow$ type constructor is right-associative, so we write $\sigma \Rightarrow \tau \Rightarrow \rho$ for $(\sigma \Rightarrow (\tau \Rightarrow \rho))$. Hence, every type σ can be written as $\sigma_1 \Rightarrow \cdots \Rightarrow \sigma_n \Rightarrow \iota$. We may write such types as $\vec{\sigma} \Rightarrow \iota$. The *order* of a type is: $\mathtt{ord}(\iota) = 0$ for $\iota \in \mathbb{B}$ and $\mathtt{ord}(\sigma \Rightarrow \tau) = \max(1 + \mathtt{ord}(\sigma), \mathtt{ord}(\tau))$. A *signature* $\mathbb{F}$ is a triple $(\mathbb{B}, \Sigma, \mathtt{typeOf})$ where $\mathbb{B}$ is a set of base types, Σ is a nonempty set of symbols, and $\mathtt{typeOf} : \Sigma \longrightarrow \mathbb{T}(\mathbb{B})$. For each type σ, we assume given a set $\mathbb{X}_\sigma$ of countably many variables and assume that $\mathbb{X}_\sigma \cap \mathbb{X}_\tau = \emptyset$ if $\sigma \neq \tau$. We let $\mathbb{X}$ denote $\cup_\sigma \mathbb{X}_\sigma$ and assume that $\Sigma \cap \mathbb{X} = \emptyset$.

The set $\mathsf{T}(\mathbb{F}, \mathbb{X})$ — of *terms* built from $\mathbb{F}$ and $\mathbb{X}$ — collects those expressions s for which a judgment $s : \sigma$ can be deduced using the following rules:

$$\text{(ax)}\ \frac{x \in \mathbb{X}_\sigma}{x : \sigma} \qquad \text{(f-ax)}\ \frac{\mathsf{f} \in \Sigma \qquad \mathtt{typeOf}(\mathsf{f}) = \sigma}{\mathsf{f} : \sigma} \qquad \text{(app)}\ \frac{s : \sigma \Rightarrow \tau \qquad t : \sigma}{(s\,t) : \tau}$$

As usual, application of terms is left-associative, so we write $s\,t\,u$ for $((s\,t)\,u)$. Let $\mathtt{vars}(s)$ be the set of variables occurring in s. A term s is *ground* if $\mathtt{vars}(s) = \emptyset$. The *head symbol* of a term $\mathsf{f}\,s_1 \cdots s_n$ is f. We say t is a *subterm* of s (written $s \trianglerighteq t$) if either (a) $s = t$, or (b) $s = s'\,s''$ and $s' \trianglerighteq t$ or $s'' \trianglerighteq t$. It is a *proper subterm* of s if $s \neq t$. For a term s, $\mathtt{pos}(s)$ is the set of *positions* in s: $\mathtt{pos}(x) = \mathtt{pos}(\mathsf{f}) = \{\sharp\}$ and $\mathtt{pos}(s\,t) = \{\sharp\} \cup \{1 \cdot u \mid u \in \mathtt{pos}(s)\} \cup \{2 \cdot u \mid u \in \mathtt{pos}(t)\}$. For $p \in \mathtt{pos}(s)$, the subterm $s|_p$ at position p is given by: $s|_\sharp = s$ and $(s_1\,s_2)|_{i \cdot p} = s_i|_p$.

In this paper, we require that for all $\mathsf{f} \in \Sigma$, $\mathtt{ord}(\mathtt{typeOf}(\mathsf{f})) \leq 2$, so w.l.o.g., $\mathsf{f} : (\vec{\iota_1} \Rightarrow \kappa_1) \Rightarrow \cdots \Rightarrow (\vec{\iota_k} \Rightarrow \kappa_k) \Rightarrow \nu_1 \Rightarrow \cdots \Rightarrow \nu_l \Rightarrow \iota$. Hence, in a fully applied term $\mathsf{f}\,s_1 \ldots s_k\,t_1 \ldots t_l$ we say the s_i are the arguments of type-1 and the t_j are

the arguments of type-0 for f. A *substitution* γ is a type-preserving map from variables to terms such that $\{x \in \mathbb{X} \mid \gamma(x) \neq x\}$ is finite. We extend γ to terms as usual: $x\gamma = \gamma(x)$, $\mathsf{f}\gamma = \mathsf{f}$, and $(s\,t)\gamma = (s\gamma)\,(t\gamma)$. A *context* C is a term with a single occurrence of a variable $\Box$; the term $C[s]$ is obtained by replacing $\Box$ by s.

A *rewrite rule* $\ell \to r$ is a pair of terms of the same type such that $\ell = \mathsf{f}\,\ell_1 \cdots \ell_m$ and $\mathtt{vars}(\ell) \supseteq \mathtt{vars}(r)$. It is *left-linear* if no variable occurs more than once in ℓ. A *simply-typed term rewriting system* $(\mathbb{F}, \mathcal{R})$ is a set of rewrite rules $\mathcal{R}$ over $\mathsf{T}(\mathbb{F}, \mathbb{X})$. In this paper, we require that all rules have *base* type. An STRS is *innermost orthogonal* if all rules are left-linear, and for any two distinct rules $\ell_1 \to r_1, \ell_2 \to r_2$, there are no substitutions γ, δ such that $\ell_1\gamma = \ell_2\delta$. A *reducible expression* (redex) is a term of the form $\ell\gamma$ for a rule $\ell \to r$ and substitution γ. The *innermost rewrite relation* induced by $\mathcal{R}$ is defined as follows:

- $\ell\gamma \to_{\mathcal{R}} r\gamma$, if $\ell \to r \in \mathcal{R}$ and $\ell\gamma$ has no proper subterm that is a redex;
- $s\,t \to_{\mathcal{R}} u\,t$, if $s \to_{\mathcal{R}} u$ and $s\,t \to_{\mathcal{R}} s\,u$, if $t \to_{\mathcal{R}} u$.

We write $\to_{\mathcal{R}}^{+}$ for the transitive closure of $\to_{\mathcal{R}}$. An STRS $\mathcal{R}$ is *innermost terminating* if no infinite rewrite sequence $s \to_{\mathcal{R}} t \to_{\mathcal{R}} \dots$ exists. It is *innermost confluent* if $s \to_{\mathcal{R}}^{+} t$ and $s \to_{\mathcal{R}}^{+} u$ implies that some v exists with $t \to_{\mathcal{R}}^{+} v$ and $u \to_{\mathcal{R}}^{+} v$. It is well-known that innermost orthogonality implies innermost confluence. In this paper, we will typically drop the "innermost" adjective and simply refer to terminating/orthogonal/confluent STRSs.

Example 1. Let $\mathbb{B} = \{\mathsf{nat}\}$ and $\mathsf{0} : \mathsf{nat}, \mathsf{s} : \mathsf{nat} \Rightarrow \mathsf{nat}, \mathsf{add}, \mathsf{mult} : \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}$, and $\mathsf{funcProd} : (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}$. We then let $\mathcal{R}$ be given by:

$$\begin{array}{rr}
\mathsf{add}\,\mathsf{0}\,y \to y & \mathsf{add}\,(\mathsf{s}\,x)\,y \to \mathsf{s}\,(\mathsf{add}\,x\,y) \\
\mathsf{mult}\,\mathsf{0}\,y \to \mathsf{0} & \mathsf{mult}\,(\mathsf{s}\,x)\,y \to \mathsf{add}\,y\,(\mathsf{mult}\,x\,y) \\
\mathsf{funcProd}\,F\,\mathsf{0}\,y \to y & \mathsf{funcProd}\,F\,(\mathsf{s}\,x)\,y \to \mathsf{funcProd}\,F\,x\,(\mathsf{mult}\,y\,(F\,x))
\end{array}$$

Hereafter, we write $\ulcorner \mathsf{n} \urcorner$ for the term $\mathsf{s}\,(\mathsf{s}\,(\dots\,\mathsf{0}\dots))$ with n ss.

2.2 Cost–Size Interpretations

For sets A and B, we write $A \longrightarrow B$ for the set of functions from A to B. A *quasi-ordered set* $(A, \sqsupseteq)$ consists of a nonempty set A and a reflexive and transitive relation $\sqsupseteq$ on A. For quasi-ordered sets $(A_1, \sqsupseteq_1)$ and $(A_2, \sqsupseteq_2)$, we write $A_1 \Longrightarrow A_2$ for the set of functions $f \in A_1 \longrightarrow A_2$ such that $f(x) \sqsupseteq_2 f(y)$ whenever $x \sqsupseteq_1 y$, i.e., $A_1 \Longrightarrow A_2$ is the space of functions that preserve quasi-ordering.

For every $\iota \in \mathbb{B}$, let a quasi-ordered set $(\mathcal{S}_\iota, \sqsupseteq_\iota)$ be given. We extend this to $\mathbb{T}(\mathbb{B})$ by defining $\mathcal{S}_{\sigma\Rightarrow\tau} = (\mathcal{S}_\sigma \Longrightarrow \mathcal{S}_\tau, \sqsupseteq_{\sigma\Rightarrow\tau})$ where $f \sqsupseteq_{\sigma\Rightarrow\tau} g$ iff $f(x) \sqsupseteq_\tau f(x)$ for any $x \in \mathcal{S}_\sigma$. Given a function $\mathcal{J}^{\mathsf{s}}$ mapping $\mathsf{f} \in \Sigma$ to some $\mathcal{J}^{\mathsf{s}}_{\mathsf{f}} \in \mathcal{S}_{\mathtt{type0f(f)}}$ and a valuation α mapping $x \in \mathbb{X}_\sigma$ to $\mathcal{S}_\sigma$, we can map each term $s : \sigma$ to an element of $\mathcal{S}_\sigma$ naturally as follows: (a) $[\![x]\!]^{\mathsf{s}}_\alpha = \alpha(x)$; (b) $[\![\mathsf{f}]\!]^{\mathsf{s}}_\alpha = \mathcal{J}^{\mathsf{s}}_{\mathsf{f}}$; (c) $[\![s\,t]\!]^{\mathsf{s}}_\alpha = [\![s]\!]^{\mathsf{s}}_\alpha([\![t]\!]^{\mathsf{s}}_\alpha)$.

For every type σ with $\mathtt{ord}(\sigma) \leq 2$, we define $\mathcal{C}_\sigma$ as follows: (a) $\mathcal{C}_\kappa = \mathbb{N}$ for $\kappa \in \mathbb{B}$; (b) $\mathcal{C}_{\iota\Rightarrow\tau} = \mathcal{S}_\iota \Longrightarrow \mathcal{C}_\tau$ for $\iota \in \mathbb{B}$; and (c) $\mathcal{C}_{\sigma\Rightarrow\tau} = \mathcal{C}_\sigma \Longrightarrow \mathcal{S}_\sigma \Longrightarrow \mathcal{C}_\tau$ if $\mathtt{ord}(\sigma) = 1$. We want to interpret terms $s : \sigma$ where both σ and all variables

occurring in s are of type order either 0 or 1, as is the case for the left- and right-hand side of rules. Thus, we let $\mathcal{J}^{\mathsf{c}}$ be a function mapping $\mathsf{f} \in \Sigma$ to some $\mathcal{J}^{\mathsf{c}}_{\mathsf{f}} \in \mathcal{C}_{\mathtt{typeOf}(\mathsf{f})}$ and assume given, for each type σ, *valuations* $\alpha : \mathbb{X}_\sigma \longrightarrow \mathcal{S}_\sigma$ and $\zeta : \mathbb{X}_\sigma \longrightarrow \mathcal{C}_\sigma$. We then define:

$$\begin{aligned}[\![x\, s_1 \cdots s_n]\!]^{\mathsf{c}}_{\alpha,\zeta} &= \zeta(x)([\![s_1]\!]^{\mathsf{s}}_\alpha, \ldots, [\![s_n]\!]^{\mathsf{s}}_\alpha)\\ [\![\mathsf{f}\, s_1 \cdots s_k\, t_1 \cdots t_n]\!]^{\mathsf{c}}_{\alpha,\zeta} &= \mathcal{J}^{\mathsf{c}}_{\mathsf{f}}([\![s_1]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_1]\!]^{\mathsf{s}}_\alpha, \ldots, [\![s_k]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_k]\!]^{\mathsf{s}}_\alpha, [\![t_1]\!]^{\mathsf{s}}_\alpha, \ldots, [\![t_n]\!]^{\mathsf{s}}_\alpha)\end{aligned}$$

We let $\mathsf{cost}(s)_{\alpha,\zeta} = \sum\{[\![t]\!]^{\mathsf{c}}_{\alpha,\zeta} \mid s \trianglerighteq t \text{ and } t \text{ is a non-variable term of base type}\}$. This is all well-defined under our assumptions that all variables have a type of order 0 or 1, and $\mathsf{f} : (\vec{\iota_1} \Rightarrow \kappa_1) \Rightarrow \cdots \Rightarrow (\vec{\iota_k} \Rightarrow \kappa_k) \Rightarrow \nu_1 \Rightarrow \cdots \Rightarrow \nu_l \Rightarrow \iota$. We also define $\mathsf{cost}'(s)_{\alpha,\zeta} = \sum\{[\![t]\!]^{\mathsf{c}}_{\alpha,\zeta} \mid s \trianglerighteq t \text{ and } t \notin \mathbb{X} \text{ is of base type not in normal form}\}$.

A *cost–size interpretation* $\mathcal{F}$ for a second order signature $\mathbb{F} = (\mathbb{B}, \Sigma, \mathtt{typeOf})$ is a choice of a quasi-ordered set $\mathcal{S}_\iota$, for each $\iota \in \mathbb{B}$, along with cost- and size-interpretations $\mathcal{J}^{\mathsf{c}}$ and $\mathcal{J}^{\mathsf{s}}$ defined as above. Let $(\mathbb{F}, \mathcal{R})$ be an STRS over $\mathbb{F}$. We say $(\mathbb{F}, \mathcal{R})$ is *compatible* with a cost–size interpretation if for any valuations α and ζ, we have (a) $[\![\ell]\!]^{\mathsf{c}}_{\alpha,\zeta} > \mathsf{cost}(r)_{\alpha,\zeta}$ and (b) $[\![\ell]\!]^{\mathsf{s}}_\alpha \sqsupseteq [\![r]\!]^{\mathsf{s}}_\alpha$, for all rules $\ell \to r$ in $\mathcal{R}$. In this case we say such cost–size interpretation *orients* all rules in $\mathcal{R}$.

Theorem 1 (Innermost Compatibility). *Suppose $\mathcal{R}$ is an STRS compatible with a cost–size interpretation $\mathcal{F}$, then for any valuations α and ζ we have $\mathsf{cost}'(s)_{\alpha,\zeta} > \mathsf{cost}'(t)_{\alpha,\zeta}$ and $[\![s]\!]^{\mathsf{s}}_\alpha \sqsupseteq [\![t]\!]^{\mathsf{s}}_\alpha$ whenever $s \to_{\mathcal{R}} t$.*

From compatibility, we have that if $s_0 \to_{\mathcal{R}} \cdots \to_{\mathcal{R}} s_n$, then $n \leq \mathsf{cost}'(s_0)$. Hence, $\mathsf{cost}'(s)$ bounds the *derivation height* of s. This follows from [25, Corollary 34], although we significantly simplified the presentation: the limitation to second-order fully applied rules and the lack of abstraction terms allow us to avoid many of the complexities in [25]. We also adapted it to *innermost* rather than *call-by-value* evaluation. A correctness proof of this version is supplied in the Appendix A. Since α and ζ are universally quantified, we typically omit them, and just write x instead of $\alpha(x)$ and F^c instead of $\zeta(F)$.

Example 2. We let $\mathcal{S}_{\mathsf{nat}} = (\mathbb{N}, \geq)$ and assign $\mathcal{J}^{\mathsf{s}}_{\mathsf{0}} = 0$ and $\mathcal{J}^{\mathsf{s}}_{\mathsf{s}} = \boldsymbol{\lambda} x.x + 1$, as well as $\mathcal{J}^{\mathsf{c}}_{\mathsf{0}} = 0$ and $\mathcal{J}^{\mathsf{c}}_{\mathsf{s}} = \boldsymbol{\lambda} x.0$. This gives us $[\![\ulcorner \mathsf{n} \urcorner]\!]^{\mathsf{s}} = n$ for all $n \in \mathbb{N}$, and $[\![\ulcorner \mathsf{n} \urcorner]\!]^{\mathsf{c}} = \mathsf{cost}(n) = 0$. Now, we let $\mathcal{J}^{\mathsf{s}}_{\mathsf{add}} = \boldsymbol{\lambda} xy.x + y$ and $\mathcal{J}^{\mathsf{s}}_{\mathsf{mult}} = \boldsymbol{\lambda} xy.x * y$; then indeed $[\![\ell]\!]^{\mathsf{s}} \geq [\![r]\!]^{\mathsf{s}}$ for the first four rules of Example 1 (e.g., $[\![\mathsf{mult}\,(\mathsf{s}\,x)\,y]\!]^{\mathsf{s}} = (x+1) * y \geq y + (x * y) = [\![\mathsf{add}\,y\,(\mathsf{mult}\,x\,y)]\!]^{\mathsf{s}}$). Moreover, let us choose $\mathcal{J}^{\mathsf{c}}_{\mathsf{add}} = \boldsymbol{\lambda} xy.x + 1$ and $\mathcal{J}^{\mathsf{c}}_{\mathsf{mult}} = \boldsymbol{\lambda} xy.x * y + x + 1$. Then also $[\![\ell]\!]^{\mathsf{c}} > \mathsf{cost}(r)$ for all rules; for example, $[\![\mathsf{mult}\,(\mathsf{s}\,x)\,y]\!]^{\mathsf{c}} = (x+1) * y + 2 * x + 3 > (y+1) + (x * y + 2 * x + 1) = [\![\mathsf{add}\,y\,(\mathsf{mult}\,x\,y)]\!]^{\mathsf{c}} + [\![\mathsf{mult}\,x\,y]\!]^{\mathsf{c}} = \mathsf{cost}(\mathsf{add}\,y\,(\mathsf{mult}\,x\,y))$. Regarding $\mathsf{funcProd}$, we can orient both rules by choosing $\mathcal{J}^{\mathsf{s}}_{\mathsf{funcProd}} = \boldsymbol{\lambda} Fxy.y * \max(F(x), 1)^x$ and $\mathcal{J}^{\mathsf{c}}_{\mathsf{funcProd}} = \boldsymbol{\lambda} FGxy.2 * x * y * \max(F(x), 1)^{x+1} + x * G(x) + 2 * x + 1$. This works due to the monotonicity assumption, which provides, e.g., $G(x+1) \geq G(x)$. (This function is not polynomial, but that is allowed in the general case.)

2.3 Basic Feasible Functionals

We assume familiarity with Turing machines. In this paper, we consider *deterministic multi-tape Turing machines*. Those are, conceptually, machines consist-

ing of a finite set of states, one or more (but a fixed number of) right-infinite *tapes* divided into cells. Each tape is equipped with a tape head that scans the symbols on the tape's cells and may write on it. The head can move to the left or right. Let $W = \{0,1\}^*$. A k-ary *Oracle Turing Machine* (OTM) is a deterministic multi-tape Turing machine with at least $2k+1$ tapes: one main tape for (input/output), k designated *query* tapes, and k designated *answer* tapes. It also has k distinct *query states* q_i and k *answer states* a_i.

A computation with a k-ary OTM M requires k fixed *oracle functions* $f_1, \ldots, f_k : W \longrightarrow W$. We write $M_{\vec{f}}$ to denote a run of M with these functions. A run of $M_{\vec{f}}$ on w starts with w written in the main tape. It ends when the machine halts, and yields the word that is written in the main tape as output. As usual, we only consider machines that halt on all inputs. The computation proceeds as usual for non-query states. To query the value of f_i on w, the machine writes w on the corresponding query tape and enters the query state q_i. Then, *in one step*, the machine transitions to the answer state a_i as follows: (a) the query value w written in the query tape for f_i is read; (b) the contents of the answer tape for f_i are changed to $f_i(w)$; (c) the query value w is erased from the query tape; and (d) the head of the answer tape is moved to its first symbol. The *running time* of $M_{\vec{f}}$ on w is the number of steps used in the computation.

A *type-1 function* is a mapping in $W \longrightarrow W$. A *type-2 functional* of rank (k,l) is a mapping in $(W \longrightarrow W)^k \longrightarrow W^l \longrightarrow W$.

Definition 1. *We say an OTM M* ***computes*** *a type-2 functional Ψ of rank (k,l) iff for all type-1 functions $f_1, \ldots, f_k$ and $x_1, \ldots, x_l \in W$, whenever $M_{f_1,\ldots,f_k}$ is started with $x_1, \ldots, x_l$ written on its main tape (separated by blanks), it halts with $\Psi(f_1, \ldots, f_k, x_1, \ldots, x_l)$ written on its main tape.*

Definition 2. *Let $\{F_1, \ldots, F_k\}$ be a set of* type-1 variables *and $\{x_1, \ldots, x_l\}$ a set of* type-0 variables. *The set $\mathtt{Pol}^2_{\mathbb{N}}[F_1, \ldots, F_k; x_1, \ldots, x_l]$ of* ***second-order polynomials*** *over $\mathbb{N}$ with indeterminates $F_1, \ldots, F_k, x_1, \ldots, x_l$ is generated by:*

$$P, Q \coloneqq n \mid x \mid P + Q \mid P * Q \mid F(Q)$$

where $n \in \mathbb{N}$, $x \in \{x_1, \ldots, x_l\}$, and $F \in \{F_1, \ldots, F_k\}$.

Notice that a polynomial expression can be viewed as a type-2 functional in the natural way, e.g., $P(F,x) = 3 * F(x) + x$ is a second-order polynomial functional. Given $w \in W$, we write $|w|$ for its length and define the length $|f|$ of $f : W \longrightarrow W$ as $|f| = \boldsymbol{\lambda} n. \max_{|y| \leq n} |f(y)|$. This allows us to define $\mathtt{BFF}$ as the class of functionals computable by OTMs with running time bounded by a second-order polynomial.

Definition 3. *A type-2 functional Ψ is in* $\mathtt{BFF}$ *iff there exist an OTM M and a second-order polynomial P such that M computes Ψ and for all $\vec{f}$ and $\vec{x}$: the running time of $M_{f_1,\ldots,f_k}$ on $x_1, \ldots, x_l$ is at most $P(|f_1|, \ldots, |f_k|, |x_1|, \ldots, |x_l|)$.*

3 Statement of the Main Result

The main result of this paper roughly states that BFF consists exactly of those type-2 functionals computed by an STRS compatible with a polynomially bounded cost–size tuple interpretation. To formally state this result, we must first define what it means for an STRS to compute a type-2 functional and define precisely the class of cost–size interpretations we are interested in.

Indeed, let us start by encoding words in W as terms. We let $\mathsf{bit}, \mathsf{word} \in \mathbb{B}$ and introduce symbols $\mathsf{o}, \mathsf{i} : \mathsf{bit}$ and $[] : \mathsf{word}$, $:: : \mathsf{bit} \Rightarrow \mathsf{word} \Rightarrow \mathsf{word}$. Then for instance 001 is encoded as the term $::\mathsf{o}\,(::\mathsf{o}\,(::\mathsf{i}\,[]))$. We use the cleaner list-like notation $[\mathsf{o};\mathsf{o};\mathsf{i}]$ in practice. Let $\underline{\mathsf{w}}$ denote the term encoding of a word w. Next, we encode type-1 functions as a possibly infinite set of one-step rewrite rules.

Definition 4. *Consider a type-1 function $f : W \longrightarrow W$ and let $\mathsf{S}_f : \mathsf{word} \Rightarrow \mathsf{word}$ be a fresh function symbol. A set of rules $\mathcal{R}_f$ **defines** f **by way of** S_f if for each $w \in W$ there is exactly one rule of the form $\mathsf{S}_f\,\underline{\mathsf{w}} \to \underline{\mathsf{f(w)}}$ in $\mathcal{R}_f$.*

Henceforth, we assume given that our STRS $(\mathbb{F}, \mathcal{R})$ at hand is such that $\mathbb{F}$ contains $\mathsf{o}, \mathsf{i}, [], ::$ typed as above and a distinguished symbol $\mathsf{F} : (\mathsf{word} \Rightarrow \mathsf{word})^k \Rightarrow \mathsf{word}^l \Rightarrow \mathsf{word}$. Given type-1 functions $f_1, \ldots, f_k$, we write $\mathbb{F}_{\vec{f}}$ for $\mathbb{F}$ extended with function symbols $\mathsf{S}_{f_i} : \mathsf{word} \Rightarrow \mathsf{word}$, with $1 \leq i \leq k$, and let $\mathcal{R}_{+\vec{f}} = \mathcal{R} \cup \bigcup_{i=1}^{k} \mathcal{R}_f$. Now we can define the notion of type-2 computability for such STRSs.

Definition 5. *Let $(\mathbb{F}, \mathcal{R})$ be an STRS. We say that F **computes** the type-2 functional Ψ in $(\mathbb{F}, \mathcal{R})$ iff for all type-1 functions $f_1, \ldots, f_k$ and all $w_1, \ldots, w_l \in W$, $\mathsf{F}\,\mathsf{S}_{f_1} \cdots \mathsf{S}_{f_k}\,\underline{\mathsf{w}_1} \cdots \underline{\mathsf{w}_l} \to^{+}_{\mathcal{R}_{+\vec{f}}} \underline{\mathsf{u}}$, where $u = \Psi(f_1, \ldots, f_k, w_1, \ldots, w_l)$.*

Next, we define what we mean by polynomially bounded interpretation.

Definition 6. *We say an STRS $(\mathbb{F}, \mathcal{R})$ **admits** a polynomially bounded interpretation iff $(\mathbb{F}, \mathcal{R})$ is compatible with a cost–size interpretation such that:*

- $\mathcal{S}_{\mathsf{word}} = (\mathbb{N}, \geq)$;
- $\mathcal{J}^{\mathsf{c}}_{\mathsf{o}} = \mathcal{J}^{\mathsf{c}}_{\mathsf{i}} = \mathcal{J}^{\mathsf{c}}_{[]} = 0$, $\mathcal{J}^{\mathsf{c}}_{::} = \boldsymbol{\lambda} xy.0$, *and* $\mathcal{J}^{\mathsf{s}}_{::} = \boldsymbol{\lambda} xy.x + y + c$ *for some* $c \geq 1$;
- $\mathcal{J}^{\mathsf{c}}_{\mathsf{F}}$ *is bounded by a polynomial in* $\mathtt{Pol}^2_{\mathbb{N}}[F^{\mathsf{c}}_1, F^{\mathsf{s}}_1, \ldots, F^{\mathsf{c}}_k, F^{\mathsf{s}}_k; x_1, \ldots, x_l]$.

Finally, we can formally state our main result.

Theorem 2. *A type-2 functional Ψ is in* BFF *if and only if there exists a finite orthogonal STRS $(\mathbb{F}, \mathcal{R})$ such that the distinguished symbol F computes Ψ in $(\mathbb{F}, \mathcal{R})$ and $\mathcal{R}$ admits a polynomially bounded cost–size interpretation.*

We prove this result in two parts. First, we prove soundness in Section 4 which states that every type-2 functional computed by an STRS as above is in BFF. Then in Section 5 we prove completeness which states that every functional in BFF can be computed by such an STRS. In order to simplify proofs, we only consider type-2 functions of rank (1,1). We claim that the results can be easily generalized, but the proofs become more tedious when handling multiple arguments.

Example 3. Let us consider the type-2 functional defined by $\Psi := \boldsymbol{\lambda} f x.\ \sum_{i<|x|} f(i)$. Notice that Ψ adds all $f(i)$ over each word $i \in W$ whose value (as a natural number) is smaller than the length of x. This functional was proved to lie in BFF in [20], where the authors utilized an encoding of Ψ as a BTLP_2 program. We can encode Ψ as an STRS as follows. Let us consider ancillary symbols $\mathsf{lengthOf} : \mathsf{word} \Rightarrow \mathsf{nat}$ and $\mathsf{toBin} : \mathsf{nat} \Rightarrow \mathsf{word}$. The former computes the length of a given word and the latter converts a number from unary to binary representation. We also consider rules for addition on binary words, i.e., $+_{\mathsf{B}} : \mathsf{word} \Rightarrow \mathsf{word} \Rightarrow \mathsf{word}$, which we use in infix notation below.

$$\begin{aligned} \mathsf{compute}\, F\, x\, \mathsf{0}\, acc &\to acc \\ \mathsf{compute}\, F\, x\, (\mathsf{s}\, i)\, acc &\to \mathsf{compute}\, F\, x\, i\, (acc +_{\mathsf{B}} F(\mathsf{toBin}\, i)) \\ \mathsf{start}\, F\, x &\to \mathsf{compute}\, F\, x\, (\mathsf{lengthOf}\, x)\, [] \end{aligned}$$

Now, if we want to compute $\Psi(f, x)$ we simply reduce the term $\mathsf{start}\, \mathsf{S}_f\, \underline{\mathsf{x}}$ to normal form. To show that this system is in BFF via our rewriting formalism, we need to exhibit a cost–size tuple interpretation for it that satisfies Definition 6.

4 Soundness

In order to prove soundness, let us consider a fixed finite orthogonal STRS $\mathcal{R}$ admitting a polynomially bounded cost–size interpretation such that it computes a type-2 functional Ψ. We proceed to show that Ψ is in BFF roughly as follows:

1. Since $\mathcal{R}$ computes Ψ and admits a polynomially bounded interpretation, we show that so does the extended system $\mathcal{R}_{+f}$ (Definition 5). The restriction on $\mathcal{J}^{\mathsf{s}}_{::}$ (Definition 6) implies that $[\![\mathsf{F}\, \mathsf{S}_f\, \underline{\mathsf{w}}]\!]^{\mathsf{c}}$ is bounded by a second-order polynomial over $|f|, |w|$. We show this in Lemma 1. By compatibility (Theorem 1), we can do at most polynomially many steps when reducing $\mathsf{F}\, \mathsf{S}_f\, \underline{\mathsf{w}}$.
2. The cost polynomial restricts the size of any input that the function variable F is applied to (e.g., a cost bound of $3 + F^{\mathsf{c}}(m)$ implies that F is never called on a term with size interpretation $> m$). This is the subject of Lemma 3.
3. Using the observations above, we then show that by graph rewriting we can simulate $\mathcal{R}_{+f}$ and compute each $\mathcal{R}_{+f}$-reduction step in polynomial time on an OTM. This guarantees that Ψ is in BFF, Theorem 3.

4.1 Interpreting The Extended STRS, Polynomially

Our first goal is to provide a polynomially bounded cost–size interpretation to the extended system $\mathcal{R}_{+f}$. We start with the observation that the size interpretation of words in W is proportional to their length. Indeed, since $\mathcal{J}^{\mathsf{s}}_{::} = \boldsymbol{\lambda} xy.x + y + c$ (Definition 6) let $\mu := \max(\mathcal{J}^{\mathsf{s}}_{\mathsf{o}}, \mathcal{J}^{\mathsf{s}}_{\mathsf{i}}) + c$ and $\nu := \mathcal{J}^{\mathsf{s}}_{[]}$. Consequently, for all $w \in W$:

$$|w| \leq [\![\underline{\mathsf{w}}]\!]^{\mathsf{s}} \leq \mu * |w| + \nu \tag{1}$$

Recall that by Definition 4 the extended system $\mathcal{R}_{+f}$ has possibly infinitely many rules of the form $\mathsf{S}_f\underline{\mathsf{w}} \to \underline{\mathsf{f(w)}}$. Such rules S_f represent calls for an oracle to compute f in a single step. Thus, we set their cost to 1. The size should be given by the length of the oracle output, taking the overhead of interpretation into account. Hence, we obtain:

$$\mathcal{J}^{\mathsf{c}}_{\mathsf{S}_f} = \boldsymbol{\lambda} x.1 \qquad \mathcal{J}^{\mathsf{s}}_{\mathsf{S}_f} = \boldsymbol{\lambda} x.\mu * |f|(x) + \nu$$

This is weakly monotonic because $|f|$ is. It orients the rules in $\mathcal{R}_f$ because $[\![\mathsf{S}_f\,\underline{\mathsf{w}}]\!]^{\mathsf{c}} = 1 > 0 = \mathsf{cost}(\underline{\mathsf{f(w)}})$, and $[\![\mathsf{S}_f\,\underline{\mathsf{w}}]\!]^{\mathsf{s}} = \mu * |f|([\![\underline{\mathsf{w}}]\!]^{\mathsf{s}}) + \nu \geq \mu * |f|(|w|) + \nu \geq \mu * |f(w)| + \nu$ by definition of $|f|$, which is superior or equal to $[\![\underline{\mathsf{f(w)}}]\!]^{\mathsf{s}}$.

As $\mathcal{J}^{\mathsf{c}}_{\mathsf{F}}$ is bounded by a second-order polynomial $\boldsymbol{\lambda} F^c F^s x.P$, we can let $D(F, n) := P(\boldsymbol{\lambda} x.1, \boldsymbol{\lambda} x.\mu * F(x) + \nu, \mu * n + \nu)$. Then D is a second-order polynomial, and $D(|f|, |w|) \geq \mathcal{J}^{\mathsf{c}}_{\mathsf{F}}(\mathcal{J}^{\mathsf{c}}_{\mathsf{S}_f}, \mathcal{J}^{\mathsf{s}}_{\mathsf{S}_f}, [\![\underline{\mathsf{w}}]\!]^{\mathsf{s}}) = \mathsf{cost}(\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}})$. By Theorem 1 we see:

Lemma 1. *There exists a second-order polynomial D so that $D(|f|, |w|)$ bounds the derivation height of $\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}}$ for any $f \in W \longrightarrow W$ and $w \in W$.*

Notice that this lemma does not imply that Ψ is in BFF. It only guarantees that there is a polynomial bound to the number of rewriting steps for such systems. However, it does not immediately follow that this number is a reasonable bound for the actual computational cost of simulating a reduction on an OTM. Consider for example a rule $\mathsf{f}\,(\mathsf{s}\,n)\,t \to \mathsf{f}\,n\,(\mathsf{c}\,t\,t)$. Every step doubles the size of the term. A naive implementation – which copies the duplicated term in each step – would take exponential time. Moreover, a single step using the oracle can create a very large output, which is not considered part of the cost of the reduction, even though an OTM would be unable to use it without first fully reading it.

Therefore, in order to prove soundness, we show how to realize a reasonable implementation of rewriting w.r.t. OTMs. In essence, we will show that (1) oracle calls are not problematic in the presence of polynomially bounded interpretations, and (2) we can handle duplication with an appropriate representation of rewriting.

4.2 Bounding The Oracle Input

We first deal with the reasonability of oracle calls. We will show that there exists a second-order polynomial B such that if an oracle call $\mathsf{S}_f\,\underline{\mathsf{x}}$ occurs anywhere along the reduction $\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}} \to^+_{\mathcal{R}} \underline{\mathsf{v}}$, then $|x| \leq B(|f|, |w|)$. From this, we know that the growth of the overall term size during an oracle call is at most $|f|(B(|f|, |w|))$.

Let P again be the polynomial bounding $\mathcal{J}^{\mathsf{c}}_{\mathsf{F}}$. Since P is a second-order polynomial, each occurrence of a sub-expression $F^c(E)$ in P is a second-order polynomial, and so is E. Let us enumerate these arguments as $E_1, \ldots, E_n$. We can then form the new polynomial Q defined as

$$Q := \sum_i E_i \quad \text{where occurrences of } F^c(E'_j) \text{ inside } E_i \text{ are replaced by } 1$$

We let $B(G, y) := Q(\boldsymbol{\lambda} z.\mu * G(z) + \nu, \mu * y + \nu)$.

Example 4. If $P = \boldsymbol{\lambda} F^{\mathsf{c}} F^{\mathsf{s}} x.x * F^{\mathsf{c}}(3 + F^{\mathsf{s}}(9 * x)) + F^{\mathsf{c}}(12) * F^{\mathsf{c}}(3 + x * F^{\mathsf{c}}(2)) + 5$, then $Q = 3 + F^{\mathsf{s}}(9 * x) + 12 + 3 + x * 1 + 2 = 20 + F^{\mathsf{s}}(9 * x) + x$. We have $B(G, x) = 20 + \mu * G(9 * (\mu * x + \nu)) + \nu + (\mu * x + \nu) = 20 + 2 * \nu + G(9 * \mu * x + 9 * \nu) + \mu * x$.

Now B gives an upper bound to the argument values for F^{c} that are considered: if a function differs from $\mathcal{J}^{\mathsf{c}}_{\mathsf{S}_f}$ only on argument values greater than $B(|f|, |w|)$, then we can use it in P and obtain the same result. Formally:

Lemma 2. *Fix f, w. Let $G \in \mathbb{N} \longrightarrow \mathbb{N}$ with $G(z) = 1$ if $z \leq B(|f|, |w|)$. Then $P(G, \mathcal{J}^{\mathsf{s}}_{\mathsf{S}_f}, [\![\underline{\mathsf{w}}]\!]^{\mathsf{s}}) = P(\mathcal{J}^{\mathsf{c}}_{\mathsf{S}_f}, \mathcal{J}^{\mathsf{s}}_{\mathsf{S}_f}, [\![\underline{\mathsf{w}}]\!]^{\mathsf{s}})$.*

This is proved by induction on the form of P, using that G is never applied on arguments larger than $B(|f|, |w|)$. Lemma 2 is used in the following key result:

Lemma 3 (Oracle Subterm Lemma). *Let $f : W \longrightarrow W$ be a type-1 function and $w \in W$. If $\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}} \to^*_{\mathcal{R}_{+f}} C[\mathsf{S}_f\,\underline{\mathsf{x}}]$ for some context C, then $|x| \leq B(|f|, |w|)$.*

Proof. In view of a contradiction, suppose there exist f, w, and x such that $\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}} \to^*_{\mathcal{R}_{+f}} C[\mathsf{S}_f\,\underline{\mathsf{x}}]$ for some context C, and $|x| > B(|f|, |w|)$. Let us now construct an alternative oracle: let $\mathsf{0} : \mathsf{nat}, \mathsf{s} : \mathsf{nat} \Rightarrow \mathsf{nat}, \mathsf{S}'_f : \mathsf{word} \Rightarrow \mathsf{word}$ and $\mathsf{helper} : \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}$, and for $N := D(|f|, |w|)$, let $\mathcal{R}'_{f,w}$ be given by:

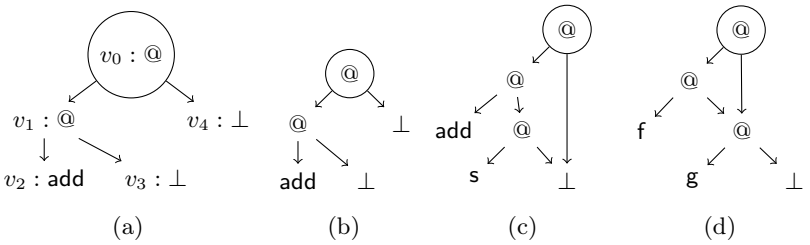
$$\begin{array}{ll} \mathsf{S}'_f\,\underline{\mathsf{x}} \to \underline{\mathsf{f(x)}} & \text{if } |x| \leq B(|f|, |w|) \\ \mathsf{S}'_f\,\underline{\mathsf{x}} \to \mathsf{helper}\,\ulcorner \mathsf{N} \urcorner\,\underline{\mathsf{f(x)}} & \text{otherwise} \end{array} \qquad \begin{array}{l} \mathsf{helper}\,\mathsf{0}\,y \to y \\ \mathsf{helper}\,(\mathsf{s}\,x)\,y \to \mathsf{helper}\,x\,y \end{array}$$

Where $\ulcorner \mathsf{N} \urcorner$ is the unary number encoding of N as introduced in Section 2.1. Notice that by definition, the rules for S'_f will produce $\underline{\mathsf{f(x)}}$ in one step if $|x| \leq B(|f|, |w|)$, but they will take $N + 2$ steps otherwise. Also observe that S_f and S'_f behave the same; that is, $\mathsf{S}_f\,\underline{\mathsf{x}}$ and $\mathsf{S}'_f\,\underline{\mathsf{x}}$ have the same normal form on any input $\underline{\mathsf{x}}$. We extend the interpretation function of the original signature with:

$$\mathcal{J}^{\mathsf{c}}_{\mathsf{S}'_f} = \boldsymbol{\lambda} x. \begin{cases} 1 & \text{if } x \leq B(|f|, |n|) \\ N + 2 & \text{if } x > B(|f|, |n|) \end{cases} \qquad \mathcal{J}^{\mathsf{s}}_{\mathsf{S}'_f} = \mathcal{J}^{\mathsf{s}}_{\mathsf{S}_f}(y)$$

$$\mathcal{J}^{\mathsf{c}}_{\mathsf{helper}} = \boldsymbol{\lambda} xy.x + 1 \qquad \mathcal{J}^{\mathsf{s}}_{\mathsf{helper}} = \boldsymbol{\lambda} xy.y \qquad \mathcal{J}^{\mathsf{s}}_{\mathsf{0}} = 0 \qquad \mathcal{J}^{\mathsf{s}}_{\mathsf{s}} = \boldsymbol{\lambda} x.x + 1$$

We easily see that this orients all rules in $\mathcal{R}_{f,w}$. Then, by Lemma 2, $\mathsf{cost}(\mathsf{F}\,\mathsf{S}'_f\,\underline{\mathsf{w}}) \leq P(\mathcal{J}^{\mathsf{c}}_{\mathsf{S}'_f}, \mathcal{J}^{\mathsf{s}}_{\mathsf{S}'_f}, [\![\underline{\mathsf{w}}]\!]^{\mathsf{s}}) = P(\mathcal{J}^{\mathsf{c}}_{\mathsf{S}_f}, \mathcal{J}^{\mathsf{s}}_{\mathsf{S}_f}, [\![\underline{\mathsf{w}}]\!]^{\mathsf{s}}) \leq D(|f|, |w|) = N$. Yet, as we have $\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}} \to^*_{\mathcal{R}_{+f}} C[\mathsf{S}_f\,\underline{\mathsf{x}}]$, we also have $\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}} \to_{\mathcal{R} \cup \mathcal{R}'_{f,w}} C'[\mathsf{S}'_f\,\underline{\mathsf{x}}]$, where C' is obtained from C by replacing all occurrences of S_f by S'_f. Since $|x| > B(|f|, |w|)$ by assumption, the reduction $\mathsf{F}\,\mathsf{S}'_f\,\underline{\mathsf{w}} \to^*_{\mathcal{R} \cup \mathcal{R}'_{f,w}} C[\mathsf{S}'_f\,\underline{\mathsf{w}}] \to^*_{\mathcal{R} \cup \mathcal{R}_{f,w'}} C[\underline{\mathsf{f(x)}}]$ takes strictly more than N steps, contradicting Theorem 1. ◻



(a) (b) (c) (d)

Fig. 1: A term graph, its simplified version, and two graphs with sharing

4.3 Graph Rewriting

Lemma 1 guarantees that if $\mathcal{R}$ is compatible with a suitable interpretation, then at most polynomially many $\mathcal{R}_{+f}$-steps can be performed starting in $\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}}$. However, as observed in Section 4.1, this does not yet imply that a type-2 functional computed by an STRS with such an interpretation is in BFF. To simulate a reduction on an OTM, we must find a representation whose size does not increase too much in any given step. The answer is *graph rewriting*.

Definition 7. *A* ***term graph*** *for a signature Σ is a tuple $(V, \mathtt{label}, \mathtt{succ}, \Lambda)$ with V a finite nonempty set of vertices; $\Lambda \in V$ a designated vertex called the* root*; $\mathtt{label} : V \longrightarrow \Sigma \cup \{@\}$ a partial function with @ fresh; and $\mathtt{succ} : V \longrightarrow V^*$ a total function such that $\mathtt{succ}(v) = v_1 v_2$ when $\mathtt{label}(v) = @$ and $\mathtt{succ}(v) = \varepsilon$ otherwise. We view this as a directed graph, with an edge from v to v' if $v' \in \mathtt{succ}(v)$, and require that this graph is* acyclic *(i.e., there is no path from any v to itself). Given term graph G, we will often directly refer to $V_G, \mathtt{label}_G$, etc.*

Term graphs can be denoted visually in an intuitive way. For example, using Σ from Example 1, the graph with $V = \{v_0, \ldots, v_4\}$, $\mathtt{label} = \{v_0, v_1 \mapsto @,\ v_2 \mapsto \mathsf{add}\}$, $\mathtt{succ} = \{v_0 \mapsto v_1 v_4,\ v_1 \mapsto v_2 v_3,\ v_3, v_4, v_5 \mapsto \varepsilon\}$ amd $\Lambda = v_0$ is pictured in Figure 1a. We use $\bot$ to indicate unlabeled vertices and a circle for Λ. We will typically omit vertex names, as done in Figure 1b. Note that the definition allows multiple vertices to have the same vertex as successor; these successor vertices with in-degree > 1 are *shared*. Two examples are denoted in Figures 1c and 1d.

Each term has a natural representation as a tree. Formally, for a term s we let $[s]_{\mathbb{G}} = (\mathtt{pos}(s), \mathtt{label}, \mathtt{succ}, \sharp)$ where $\mathtt{label}(p) = @$ if $s|_p = s_1 s_2$ and $\mathtt{label}(p) = \mathsf{f}$ if $s|_p = \mathsf{f}$; $\mathtt{label}(p)$ is not defined if $s|_p$ is a variable; and $\mathtt{succ}(p) = (1 \cdot p)(2 \cdot p)$ if $s|_p = s_1\, s_2$ and $\mathtt{succ}(p) = \varepsilon$ otherwise. Essentially, $[s]_{\mathbb{G}}$ maintains the positioning structure of s and forgets variable names. For example, Figure 1b denotes both $[\mathsf{add}\, x\, y]_{\mathbb{G}}$ and $[\mathsf{add}\, x\, x]_{\mathbb{G}}$.

Our next step is to *reduce* term graphs using rules. We limit interest to *left-linear* rules, which includes all rules in $\mathcal{R}_{+f}$ (as $\mathcal{R}$ is orthogonal, and the rules in $\mathcal{R}_f$ are ground). To define reduction, we will need some helper definitions.

Definition 8. *Let $G = (V, \mathtt{label}, \mathtt{succ}, \Lambda), v \in V$. The* ***subgraph*** $\mathsf{reach}(G, v)$ *of G rooted at v is the term graph $(V', \mathtt{label}', \mathtt{succ}', v)$ where V' contains those $v' \in V$ such that a path from v to v' exists, and $\mathtt{label}', \mathtt{succ}'$ are respectively the limitations of* $\mathtt{label}$ *and* $\mathtt{succ}$ *to V'.*

Definition 9. *A* ***homomorphism*** *between two term graphs G and H is a function $\phi : V_G \longrightarrow V_H$ with $\phi(\Lambda_G) = \Lambda_H$, and for $v \in V_G$ such that $\mathtt{label}_G(v)$ is defined, $\mathtt{label}_H(\phi(v)) = \mathtt{label}_G(v)$ and $\mathtt{succ}_H(\phi(v)) = \phi(v_1) \dots \phi(v_k)$ when $\mathtt{succ}_G(v) = v_1 \dots v_k$. (If $\mathtt{label}_G(v)$ is undefined, $\mathtt{succ}_H(\phi(v))$ may be anything.)*

Definition 10. *A* ***redex*** *in G is a triple (ρ, v, ϕ) consisting of some rule $\rho = \ell \to r \in \mathcal{R}_{+f}$, a vertex v in V_G, and a homomorphism $\phi : [\ell]_{\mathbb{G}} \longrightarrow \mathsf{reach}(G, v)$.*

Definition 11. *Let G be a term graph and v_1, v_2 vertices in G. The* ***redirection*** *of v_1 to v_2 is the term graph $G[v_1 \gg v_2] \equiv (V_G, \mathtt{label}_G, \mathtt{succ}_{G'}, \Lambda'_G)$ with*

$$\mathtt{succ}_{G'}(v)_i = \begin{cases} v_2, & \text{if } \mathtt{succ}_G(v)_i = v_1 \\ \mathtt{succ}_G(v)_i, & \text{otherwise} \end{cases} \qquad \Lambda'_G = \begin{cases} v_2 & \text{if } \Lambda_G = v_1 \\ \Lambda_G & \text{otherwise} \end{cases}$$

That is, we replace every reference to v_1 by a reference to v_2. With these definitions in hand, we can define *contraction* of term graphs:

Definition 12. *Let G be a term graph, and (ρ, v, ϕ) a redex in G with $\rho \in \mathcal{R}_{+f}$, such that no other vertex v' in $\mathsf{reach}(G, v)$ admits a redex (so v is an* innermost redex position*). Denote a_x for the position of variable x in ℓ, and recall that a_x is a vertex in $[\ell]_{\mathbb{G}}$. By left-linearity, a_x is unique for $x \in \mathtt{vars}(\ell)$. The* ***contraction*** *of (ρ, v, ϕ) in G is the term graph J produced after the following steps: H (building), I (redirection), and J (garbage collection).*

(building) *Let $H = (V_H, \mathtt{label}_H, \mathtt{succ}_H, \Lambda_G)$ where:*
- *$V_H = V_G \uplus \{\overline{p} \in \mathtt{pos}(r) \mid r|_p$ is not a variable$\}$ ($\uplus$ means disjoint union);*
- *for $v \in V_G$: $\mathtt{label}_H(v) = \mathtt{label}_G(v)$ and $\mathtt{succ}_H(v) = \mathtt{succ}_G(v)$*
- *for $p \in V_H$ with $r|_p$ not a variable:*
  - *$\mathtt{label}_H(\overline{p}) = \mathsf{f}$ if $r|_p = \mathsf{f}$ and $\mathtt{label}_H(\overline{p}) = @$ otherwise*
  - *$\mathtt{succ}_H(\overline{p}) = \varepsilon$ if $r|_p = \mathsf{f}$; otherwise, $\mathtt{succ}_H(\overline{p}) = \psi(1 \cdot p)\psi(2 \cdot p)$*

  Here, $\psi(q) = \overline{q}$ if $r|_q$ is not a variable; if $r|_q = x$ then $\psi(q) = \phi(a_x)$.

(redirection) *If r is a variable x (so $H = G$), then let $I = G[v \gg \phi(a_x)]$. Otherwise, let $I = H[v \gg \overline{\sharp}]$, so with all references to v redirected to the root vertex for r.*

(garbage collection) *Let $J := \mathsf{reach}(I, \Lambda_I)$ (so remove unreachable vertices).*

We then write $G \rightsquigarrow J$ in one step, and $G \rightsquigarrow^n J$ for the n-step reduction.

We illustrate this with two examples. First, we aim to rewrite the graph of Figure 2a with a rule $\mathsf{add}\,\mathsf{0}\,y \to y$ at vertex v. Since the right-hand side is a variable, the building phase does nothing. The result of the redirection phase is given in Figure 2b, and the result of the garbage collection in Figure 2c.

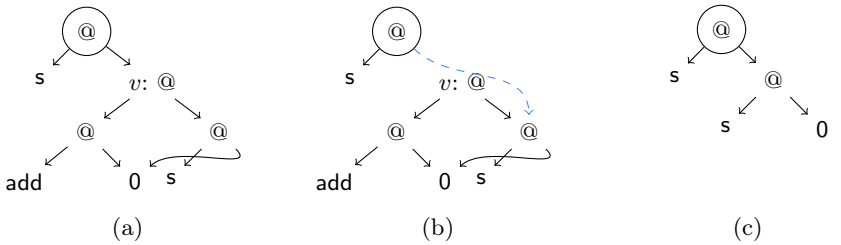


Fig. 2: Reducing a graph with the rule $\mathsf{add}\,\mathsf{0}\,y \to y$

Second, we consider a reduction by $\mathsf{mult}\,(\mathsf{s}\,x)\,y \to \mathsf{add}\,y\,(\mathsf{mult}\,x\,y)$. Figure 3a shows the result of the building phase, with the vertices and edges added during this phase in red. Redirection sets the root to the squared node (the root of the right-hand side), and the result after garbage collection is in Figure 3b.

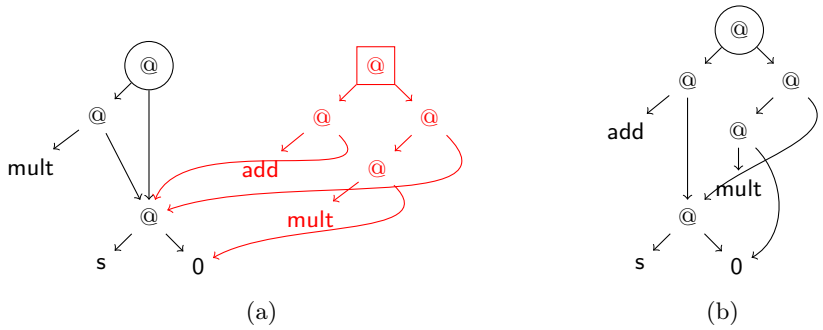


Fig. 3: Reducing a term graph with substantial sharing

Note that, even when a term graph G is not a tree, we can find a corresponding term: we assign a variable $var(v)$ to each unlabeled vertex v in G, and let:

$$\theta(v) = \begin{cases} \theta(v_1)\,\theta(v_2) & \text{if } \mathtt{label}(v) = @ \text{ and } \mathtt{succ}(v) = v_1 v_2 \\ \mathsf{f} & \text{if } \mathtt{label}(v) = \mathsf{f} \\ var(v) & \text{if } \mathtt{label}(v) \text{ is undefined} \end{cases}$$

Then we may define $[G]_{\mathbb{G}}^{-1} = \theta(\Lambda_G)$. For a linear term, clearly $[[s]_{\mathbb{G}}]_{\mathbb{G}}^{-1} = s$ (modulo variable renaming). We make the following observation:

Lemma 4. *Assume given a term graph G such that there is a path from Λ_G to every vertex in V_G, and let $[G]_{\mathbb{G}}^{-1} = s$. If $G \rightsquigarrow H$ then $[G]_{\mathbb{G}}^{-1} \to_{\mathcal{R}+f}^{+} [H]_{\mathbb{G}}^{-1}$. Moreover, if $s \to_{\mathcal{R}+f} t$ for some t, then there exists H such that $G \rightsquigarrow H$.*

Consequently, if $\to_{\mathcal{R}+f}$ is terminating, then so is $\rightsquigarrow$; and if $[s]_{\mathbb{G}} \rightsquigarrow^n G$ for some ground term s then $s \to_{\mathcal{R}+f}^{*} [G]_{\mathbb{G}}^{-1}$ in at least n steps. Notice that if G does not

admit any redex, then $[G]_{\mathbb{G}}^{-1}$ is in normal form. Moreover, since $\mathcal{R}_{+f} = \mathcal{R} \cup \mathcal{R}_f$ is orthogonal (as $\mathcal{R}$ is orthogonal and the $\mathcal{R}_f$ rules are non-overlapping) and therefore confluent, this is the *unique* normal form of s. We conclude:

Corollary 1. *If* $[\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}}]_{\mathbb{G}} \rightsquigarrow^n G$, *then* $n \leq D(|f|, |w|)$; *and if* G *is in normal form, then* $[G]_{\mathbb{G}}^{-1} = \underline{\Psi(\mathsf{f}, \mathsf{w})}$.

4.4 Bringing Everything Together

We are now ready to complete the soundness proof following the recipe at the start of the section. Towards the third bullet point, we make the following observation.

Lemma 5. *There is a constant* a *such that, whenever* $G \rightsquigarrow H$ *by a rule in* $\mathcal{R}$, *then* $|H| \leq |G| + a$, *where* $|G|$ *denotes the total number of nodes in the graph* G.

Proof. In a step using a rule $\ell \to r$, the number of nodes in the graph can be increased at most by $|[r]_{\mathbb{G}}|$. As there are only finitely many rules in $\mathcal{R}$, we can let a be the number of nodes in the largest graph for a right-hand side r. □

To see that graph rewriting with S_f can be implemented in an efficient way, we observe that the size of any intermediary graph in the reduction $[\mathsf{G}\,\underline{\mathsf{w}}]_{\mathbb{G}} \to_{\mathcal{R}}^{+} [q]_{\mathbb{G}}$ is polynomially bounded by a second-order polynomial over $|f|, |w|$:

Lemma 6. *There is a second-order polynomial* Q *such that if* $[\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}}]_{\mathbb{G}} \rightsquigarrow^* H$, *then* $|H| \leq Q(|f|, |w|)$.

Proof. Let $Q(F, x) := x + D(F, x) * (a + F(B(F, x)))$, where D is the polynomial from Lemma 1, a is the constant from Lemma 5, and B is the polynomial from Section 4.2. This suffices, because there are at most $D(|f|, |w|)$ steps (Lemma 1, Corollary corollary 1), each of which increases the graph size by at most $\max(a, |f|(B(|f|, |w|)))$. □

All in all, we are finally ready to prove the *soundness* side of the main theorem:

Theorem 3. *Let* $\mathcal{R}$ *be a finite orthogonal STRS admitting a polynomially bounded interpretation. If* F *computes a type-2 functional* Ψ, *then* $\Psi \in \mathtt{BFF}$.

Proof. Given $(\mathbb{F}, \mathcal{R})$, we can construct an OTM M so that for a given $f \in W \longrightarrow W$, the machine M_f executed on $w \in W$ computes the normal form of $\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}}$ under $\to_{\mathcal{R}_{+f}}$ using graph rewriting. We omit the exact construction, but observe:

- that we can represent each graph in polynomial space in the size of the graph;
- that we can do a rewriting step that does not call the oracle (so using a rule in $\mathcal{R}$) following the contraction algorithm we defined in Definition 12, which is clearly feasible to do in polynomial time in the size of the graph;

- and that each oracle call (implemented in rewriting by a $\mathcal{R}_f$-step $\mathsf{S}_f\,\underline{\mathsf{x}} \to \underline{\mathsf{y}}$) is resolved by copying $\underline{\mathsf{x}}$ to the query tape, transitioning to the query state, and from the answer state copying $\underline{\mathsf{y}}$ from the answer tape to the main tape.

By Lemma 3 this is doable in polynomial time in $|f|, |w|$ and the graph size. By Lemma 6, graph sizes are bounded by a polynomial over $|f|, |w|$, so using the above reasoning, the same holds for the cost of each reduction step. In summary: the total cost of M_f running on w is bounded by a second-order polynomial in terms of $|f|$ and $|w|$. As M_f simulates $\mathcal{R}_{+f}$ via graph rewriting and $\mathcal{R}_{+f}$ computes Ψ, M also computes Ψ. By Definition 3, Ψ is in BFF. □

5 Completeness

Recall from Section 3 that to prove completeness we have to show the following: if a given type-2 functional Ψ is in BFF, then there exists an orthogonal STRS that computes Ψ and admits a polynomially bounded interpretation. We prove this by providing an encoding of OTMs as STRSs that admit a polynomially bounded interpretation.

The encoding is divided into three steps. In Section 5.1, we will define the function symbols that will allow us to encode any possible machine configuration as terms. In Section 5.2, we will encode transitions as reduction rules that rewrite configuration terms. Lastly, we will design an STRS to simulate a complete execution of an OTM in polynomially many steps. Achieving this polynomial bound is non-trivial and is done in Sections 5.3–5.4.

Henceforth, we assume given a fixed OTM M, and a second-order polynomial P_M, such that M operates in time P_M. For simplicity, we assume the machine has only three tapes (one input/output tape, one query tape, one answer tape); that each non-oracle transition only operates on one tape (i.e., reading/writing and moving the tape head); and that we only have tape symbols $\{0, 1, \mathtt{B}\}$.

5.1 Representing Configurations

Following 3, we have $\mathsf{o}, \mathsf{i} : \mathsf{bit}$, $:: : \mathsf{bit} \Rightarrow \mathsf{word} \Rightarrow \mathsf{word}$ and $[] : \mathsf{word}$. To represent a (partial) tape, we also introduce $\mathsf{b} : \mathsf{bit}$ for the blank symbol. Now for instance a tape with content $011\mathtt{B}01\mathtt{BB}\cdots$ (followed by infinitely many blanks) may be represented as the list $[\mathsf{o}; \mathsf{i}; \mathsf{i}; \mathsf{b}; \mathsf{o}; \mathsf{i}]$ of type word. We may also add an arbitrary number of blanks at the end of the representation; e.g., $[\mathsf{o}; \mathsf{i}; \mathsf{i}; \mathsf{b}; \mathsf{o}; \mathsf{i}; \mathsf{b}; \mathsf{b}]$.

We can think of a *tape configuration* — the combination of a tape and the position of the tape head — as a finite word $w_1 \dots w_{p-1} \# w_p w_{p+1} \dots w_k$ (followed by infinitely many blanks). Here, the tape's head is reading the symbol w_p. We can split this tape into two components: the *left* word $w_1 \dots w_{p-1}$, and the *right* word $w_p \dots w_k$. To represent a tape configuration, we introduce three symbols:

$$\mathsf{L} : \mathsf{word} \Rightarrow \mathsf{left} \qquad \mathsf{R} : \mathsf{word} \Rightarrow \mathsf{right} \qquad \mathsf{split} : \mathsf{left} \Rightarrow \mathsf{right} \Rightarrow \mathsf{tape}$$

Here, L, R hold the content of the left and right split of the tape, respectively. While we technically do not need these two constructors (we could have split :

$\mathsf{word} \Rightarrow \mathsf{word} \Rightarrow \mathsf{tape}$), they serve to make configurations more human-readable. For convenience in rewriting transitions, later on, we will encode the left side of the split in reverse order. Specifically, we encode $w_1 \dots w_{p-1} \# w_p w_{p+1} \dots w_k$ as

$$\mathsf{split}\,(\mathsf{L}\,[w_{p-1}; \dots; w_2; w_1])\,(\mathsf{R}\,[w_p; \dots; w_{k-1}; w_k])$$

The symbol currently being read is the first element of the list below R; in case of $\mathsf{R}\,[]$, this symbol is $\mathtt{B}$. For a concrete example, a tape configuration $1\mathtt{B}0\#10$ is represented by: $\mathsf{split}\,(\mathsf{L}\,[\mathsf{o}; \mathsf{b}; \mathsf{i}])\,(\mathsf{R}\,[\mathsf{i}; \mathsf{o}])$. Since we have assumed an OTM with three tapes, a configuration of the machine at any moment is a tuple (q, t_1, t_2, t_3), with q a state and t_1, t_2, t_3 tape configurations. To represent machine configurations, we introduce, for each state q, a symbol $\mathsf{q} : \mathsf{tape} \Rightarrow \mathsf{tape} \Rightarrow \mathsf{tape} \Rightarrow \mathsf{config}$. Thus, a configuration (q, t_1, t_2, t_3) is represented by a term $\mathsf{q}\,T_1\,T_2\,T_3$.

Example 5. The initial configuration for a machine M_f on input w is a tuple $(q_0, \#w, \#\mathtt{B}, \#\mathtt{B})$. This is represented by the term

$$\mathsf{initial}(w) := \mathsf{q}_0\,(\mathsf{split}\,(\mathsf{L}\,[])\,(\mathsf{R}\,\underline{\mathsf{w}}))\,(\mathsf{split}\,(\mathsf{L}\,[])\,(\mathsf{R}\,[]))\,(\mathsf{split}\,(\mathsf{L}\,[])\,(\mathsf{R}\,[]))$$

To interpret the symbols from this section, we let $(\mathcal{S}_\iota, \sqsupseteq_\iota) := (\mathbb{N}, \geq)$ for all ι, let $\mathcal{J}^{\mathsf{c}}_{\mathsf{f}} = \boldsymbol{\lambda} x_1 \dots x_m.0$ whenever f takes m arguments, and for the sizes:

$$\begin{array}{lllll} \mathcal{J}^{\mathsf{s}}_{\mathsf{o}} = 0 & \mathcal{J}^{\mathsf{s}}_{\mathsf{b}} = 0 & \mathcal{J}^{\mathsf{s}}_{\mathsf{L}} = \boldsymbol{\lambda} x.x & \mathcal{J}^{\mathsf{s}}_{::} = \boldsymbol{\lambda} xy.x + y + 1 & \mathcal{J}^{\mathsf{s}}_{\mathsf{q}} = \boldsymbol{\lambda} xyz.x + y \\ \mathcal{J}^{\mathsf{s}}_{\mathsf{i}} = 0 & \mathcal{J}^{\mathsf{s}}_{[]} = 0 & \mathcal{J}^{\mathsf{s}}_{\mathsf{R}} = \boldsymbol{\lambda} x.x & \mathcal{J}^{\mathsf{s}}_{\mathsf{split}} = \boldsymbol{\lambda} x.xy.x + y & \text{(for all states } q\text{)} \end{array}$$

Hence, $[\![\underline{\mathsf{w}}]\!]^{\mathsf{s}} = |w|$, which satisfies the requirements of Theorem 2; the size of a tape configuration $w_1 \dots w_{p-1} \# w_p \dots w_k$ is k, and the size of a configuration is the size of its first and second tapes combined. We do *not* include the third tape, as it does not directly affect either the result yielded by the final configuration (this is read from the first tape), nor the size of a word the oracle f is applied on.

5.2 Executing The Machine

A single step in an OTM can either be an oracle call (a transition from the `query` state to the `answer` state), or a traditional step: we assume that an OTM M has a fixed set $\mathcal{T}$ of *transitions* $q \xrightarrow[t]{r/i,\ d} l$ where q is the *input state*, l the *output state*, $t \in \{1, 2, 3\}$ the tape considered (recall that we have assumed that a non-oracle transition only operates on one tape), $r, i \in \{0, 1, \mathtt{B}\}$ respectively the symbol being read and the symbol being written, and $d \in \{L, R\}$ the direction for the read head of tape t to move. We will model the computation of M as rules that simulate the small step semantics for the machine.

To encode a single transition, let $\mathsf{step} : (\mathsf{word} \Rightarrow \mathsf{word}) \Rightarrow \mathsf{config} \Rightarrow \mathsf{config}$. For any transition of the form $q \xrightarrow[1]{r/i,\ L} l$ (so a transition operating on tape 1, and moving left), we introduce a rule (where we write $\underline{0} = \mathsf{o}$, $\underline{1} = \mathsf{i}$, $\underline{\mathtt{B}} = \mathsf{b}$):

$$\mathsf{step}\,F\,(\mathsf{q}\,(\mathsf{split}\,(\mathsf{L}\,(x{::}y))\,(\mathsf{R}\,(\underline{r}{::}z)))\,u\,v) \to \mathtt{l}\,(\mathsf{split}\,(\mathsf{L}\,y)\,(\mathsf{R}\,(x{::}\underline{i}{::}z)))\,u\,v$$

Moreover, for transitions $q \xrightarrow[1]{\mathtt{B}/w,\ L} l$ (so where $\mathtt{B}$ is read), we add a rule:

$$\mathsf{step}\, F\, (\mathsf{q}\, (\mathsf{split}\, (\mathsf{L}\, (x{::}y))\, (\mathsf{R}\, []))\, u\, v) \to \mathtt{l}\, (\mathsf{split}\, (\mathsf{L}\, y)\, (\mathsf{R}\, (x{::}\underline{\mathsf{i}}{::}[])))\, u\, v$$

These rules respectively handle the steps where a tape configuration is changed from $u_1 \dots u_{p-1} u_p \# r u_{p+2} \dots u_k$ to $u_1 \dots u_{p-1} \# u_p i u_{p+2} \dots u_k$, and where a tape configuration is changed from $u_1 \dots u_k \#$ to $u_1 \dots \# u_k i$.

Transitions where $d = R$, or on the other two tapes, are encoded similarly.

Next, we encode oracle calls. Recall that, to query the machine for the value of f at u, we write u on the second tape, move its head to the leftmost position, and enter the query state. Then, the content of this tape is erased and the image of f over u is written in the third tape. Visually, this step is represented as:

$$(\mathtt{query}, \langle \mathrm{tape}_1 \rangle, v_1 \dots v_p \# \underline{u\mathtt{B}} \dots, \langle \mathrm{tape}_3 \rangle) \rightsquigarrow (\mathtt{answer}, \langle \mathrm{tape}_1 \rangle, \#\mathtt{B}, \#\underline{\mathsf{f}(\mathsf{u})})$$

This is implemented by the following rules:

$$\mathsf{step}\, F\, (\mathtt{query}\, t_1\, (\mathsf{split}\, x\, (\mathsf{R}\, y))\, t_3) \to \mathtt{answer}\, t_1\, (\mathsf{split}\, (\mathsf{L}\, [])\, (\mathsf{R}\, []))\, (\mathsf{split}\, (\mathsf{L}\, [])\, (\mathsf{R}\, (F\, (\mathsf{clean}\, y))))$$

$$\begin{array}{rr} \mathsf{clean}\, (\mathsf{o}{::}x) \to \mathsf{o}{::}(\mathsf{clean}\, x) & \mathsf{clean}\, (\mathsf{b}{::}x) \to [] \\ \mathsf{clean}\, (\mathsf{i}{::}x) \to \mathsf{i}{::}(\mathsf{clean}\, x) & \mathsf{clean}\, [] \to [] \end{array}$$

Here, $\mathsf{clean} : \mathsf{word} \Rightarrow \mathsf{word}$ turns a word that may have blanks in it into a bitstring, by reading until the next blank; for instance replacing $[\mathsf{o}; \mathsf{i}; \mathsf{b}; \mathsf{i}]$ by $[\mathsf{o}; \mathsf{i}]$.

The various step rules, as well as the clean rules, are non-overlapping because we consider *deterministic* OTMs. They are also left-linear, and are oriented using:

$$\begin{array}{rr} \mathcal{J}^{\mathsf{s}}_{\mathsf{clean}} = \boldsymbol{\lambda} x.x & \mathcal{J}^{\mathsf{c}}_{\mathsf{clean}} = \boldsymbol{\lambda} x.x + 1 \\ \mathcal{J}^{\mathsf{s}}_{\mathsf{step}} = \boldsymbol{\lambda} F x.x + 1 & \mathcal{J}^{\mathsf{c}}_{\mathsf{step}} = \boldsymbol{\lambda} F^c F^s x.F^c(x) + x + 2 \end{array}$$

(Note that $\mathcal{J}^{\mathsf{s}}_{\mathsf{step}}$ is so simple because the size of a configuration does not include the size of the answer tape.) From the rules, the following result is obvious:

Lemma 7. *Let M_f be an OTM and C, C' be machine configurations of M_f such that $C \rightsquigarrow C'$. Then $\mathsf{step}\, \mathsf{S}_f\, [C] \to^+_{\mathcal{R}} [C']$, where $[C]$ is the term encoding of C.*

5.3 A Bound on the Number of Steps

To generalize from performing a single step of the machine to tracing a full computation on the machine level, the natural idea would be to define rules such as:

$$\begin{array}{r} \mathsf{execute}\, F\, (\mathsf{q}\, x\, y\, z) \to \mathsf{execute}\, F\, (\mathsf{step}(\mathsf{q}\, x\, y\, z)) \text{ for } \mathtt{q} \neq \mathtt{end} \\ \mathsf{execute}\, F\, (\mathtt{end}\, (\mathsf{split}\, (\mathsf{L}\, x)\, (\mathsf{R}\, w))\, y\, z) \to \mathsf{clean}\, w \end{array}$$

Then, reducing $\mathsf{execute}\, \mathsf{S}_f\, \mathsf{initial}(w)$ to normal form simulates a full OTM execution of M_f on input w. Unfortunately, this rule does not admit an interpretation,

as it may be non-terminating. A solution could be to give $\mathsf{execute}$ an additional argument $\ulcorner \mathsf{N} \urcorner$ suggesting an execution in at most N steps; this argument would ensure termination, and could be used to find an interpretation.

The challenge, however, is to compute a bound on the number of steps in the OTM: the obvious thought is to compute $P_M(|f|, |w|)$, but this cannot in general be done in polynomial time because the STRS does not have access to $|f|$: since $|f|(i) = \max\{x \in \mathbb{N} \mid |x| \leq i\}$, there are exponentially many choices for x.

To solve this, and following [21, Proposition 2.3], we observe that it suffices to know a bound for $f(x)$ for only those x on which the oracle is actually questioned. That is, for $A \subseteq W$, let $|f|_A = \boldsymbol{\lambda} n.\, \max\{|f(x)| \mid x \in A \wedge |x| \leq n\}$. Then:

Lemma 8. *Suppose an OTM M_f runs in time bounded by $P_M(|f|, |w|)$ on input w. If M_f transitions in N steps from its initial state to some configuration C, calling the oracle only on words in $A \subseteq W$, then $N \leq P_M(|f|_A, |w|)$.*

Proof (Sketch). We construct f' with $f'(x) = 0$ if $x \notin A$ and $f'(x) = f(x)$ if $x \in A$. Then $|f'| = |f|_A$, and $M_{f'}$ runs the same on input w as M_f does. □

Now, for A encoded as a term A (using symbols $\emptyset : \mathsf{set}$, $\mathsf{setcons} : \mathsf{word} \Rightarrow \mathsf{set} \Rightarrow \mathsf{set}$), we can compute $|f|_A$ using the rules below, where we use unary integers as in Example 1 ($\mathsf{0} : \mathsf{nat}, \mathsf{s} : \mathsf{nat} \Rightarrow \mathsf{nat}$), and defined symbols $\mathsf{len} : \mathsf{word} \Rightarrow \mathsf{nat}$, $\mathsf{max} : \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}$, $\mathsf{limit} : \mathsf{word} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{word}$, $\mathsf{retif} : \mathsf{word} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{word} \Rightarrow \mathsf{word}$, $\mathsf{tryapply} : (\mathsf{word} \Rightarrow \mathsf{word}) \Rightarrow \mathsf{word} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}$, $\mathsf{tryall} : (\mathsf{word} \Rightarrow \mathsf{word}) \Rightarrow \mathsf{set} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}$. By design, $\mathsf{retif}\, \underline{\mathsf{x}} \ulcorner \mathsf{n} \urcorner \underline{\mathsf{y}}$ reduces to $\underline{\mathsf{y}}$ if $|x| \leq n$ and to $[]$ otherwise; $\mathsf{tryapply}\, \mathsf{S}_f\, \underline{\mathsf{x}} \ulcorner \mathsf{n} \urcorner$ reduces to the unary encoding of $|F|_{\{x\}}(n)$ and $\mathsf{tryall}\, \mathsf{a}\, \underline{\mathsf{x}} \ulcorner \mathsf{n} \urcorner$ yields $|F|_A(n)$.

$$
\begin{array}{rrr}
\mathsf{len}\,[] \to \mathsf{0} & \mathsf{len}\,(x{::}y) \to \mathsf{s}\,(\mathsf{len}\,y) & \\
\mathsf{max}\,\mathsf{0}\,m \to m & \mathsf{max}\,(\mathsf{s}\,n)\,\mathsf{0} \to \mathsf{s}\,n & \mathsf{max}\,(\mathsf{s}\,n)\,(\mathsf{s}\,m) \to \mathsf{s}\,(\mathsf{max}\,n\,m) \\
\mathsf{limit}\,[]\,n \to [] & \mathsf{limit}\,(x{::}y)\,\mathsf{0} \to [] & \mathsf{limit}\,(x{::}y)\,(\mathsf{s}\,n) \to x{::}(\mathsf{limit}\,y\,n) \\
\mathsf{retif}\,[]\,n\,z \to z & \mathsf{retif}\,(x{::}y)\,\mathsf{0}\,z \to [] & \mathsf{retif}\,(x{::}y)\,(\mathsf{s}\,n)\,z \to \mathsf{retif}\,y\,n\,z \\
 & & \\
 & \mathsf{tryapply}\,F\,a\,n \to \mathsf{len}\,(\mathsf{retif}\,a\,n\,(F\,(\mathsf{limit}\,a\,n))) & \\
\mathsf{tryall}\,F\,\emptyset\,n \to \mathsf{0} & \mathsf{tryall}\,F\,(\mathsf{setcons}\,a\,tl)\,n \to \mathsf{max}\,(\mathsf{tryapply}\,F\,a\,n)\,(\mathsf{tryall}\,F\,tl\,n) &
\end{array}
$$

An interpretation is provided in Appendix B. Importantly, the limit function ensures that, in $\mathsf{tryall}\,F\,n$ we never apply F to a word w with $|w| > n$. Therefore we can let $[\![\mathsf{A}]\!]^{\mathsf{s}} = |A|$, the number of words in A, and have $\mathcal{J}^{\mathsf{s}}_{\mathsf{tryall}} = \boldsymbol{\lambda} F a n.F(n)$ and $\mathcal{J}^{\mathsf{c}}_{\mathsf{tryall}} = \boldsymbol{\lambda} F^c F^s a n.1 + a + F^c(n) + 2 * F^s(n) + 2 * n + 6$.

Now, for a given second-order polynomial P, fixed f, n, and a term A encoding a set $A \subseteq W$, we can construct a term $\Theta^P_{\mathsf{S}_f;\ulcorner \mathsf{n} \urcorner;\mathsf{A}}$ that computes $P(|f|_A, n)$ using tryall and the functions $\mathsf{add}, \mathsf{mult}$ from Example 1. By induction on P, we have $[\![\Theta^P_{\mathsf{S}_f;\ulcorner \mathsf{n} \urcorner;\mathsf{A}}]\!]^{\mathsf{s}} = P(|f|, n)$, while its cost is bounded by a polynomial over $|f|, n, |A|$.

5.4 Finalising Execution

Now, we can define execution in a way that can be bounded by a polynomial interpretation. We let $\mathsf{execute} : (\mathsf{word} \Rightarrow \mathsf{word}) \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nnat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{set} \Rightarrow$

$\mathsf{config} \Rightarrow \mathsf{word}$ and will define rules to reduce expressions $\mathsf{execute}\,F\,n\,m\,z\,a\,c$ where

- F is the function to be used in oracle calls.
- $n-1$ is a bound on the number of steps that can be done before the next oracle call (or until the machine completes execution).
- m is essentially a natural number that represents the number of steps that have been done so far. We use a new sort nnat with function symbols $\mathsf{o} : \mathsf{nnat}$ and $\mathsf{n} : \mathsf{nnat} \Rightarrow \mathsf{nnat}$ because we will let $\mathcal{S}_{\mathsf{nnat}} = (\mathbb{N}, \leq)$, so ordered in the other direction. This will be essential to find an interpretation for $\mathsf{execute}$.
- z is a unary representation of $|w|$, where w is the input to the OTM.
- c is the current configuration.

Using helper symbols $\mathsf{F}' : (\mathsf{word} \Rightarrow \mathsf{word}) \Rightarrow \mathsf{nat} \Rightarrow \mathsf{config} \Rightarrow \mathsf{word}$, $\mathsf{execute}' : (\mathsf{word} \Rightarrow \mathsf{word}) \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nnat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{set} \Rightarrow \mathsf{config} \Rightarrow \mathsf{word}$, $\mathsf{extract} : \mathsf{tape} \Rightarrow \mathsf{word}$ and $\mathsf{minus} : \mathsf{nat} \Rightarrow \mathsf{nnat} \Rightarrow \mathsf{nat}$, we introduce the rules:

$$\mathsf{F}\,F\,w \to \mathsf{F}'\,F\,(\mathsf{len}\,w)\,(\mathsf{q}_0\,(\mathsf{split}(\mathsf{L}\,[])\,(\mathsf{R}\,w))\,(\mathsf{split}(\mathsf{L}\,[])\,(\mathsf{R}\,[]))\,(\mathsf{split}(\mathsf{L}\,[])\,(\mathsf{R}\,[])))$$
$$\mathsf{F}'\,F\,z\,c \to \mathsf{execute}\,F\,\Theta^{P_M+1}_{F;z;\emptyset}\,\mathsf{o}\,z\,\emptyset\,c$$

$$\mathsf{execute}\,F\,(\mathsf{s}\,n)\,m\,z\,a\,(\mathsf{q}\,t_1\,t_2\,t_3) \to \mathsf{execute}\,F\,n\,(\mathsf{n}\,m)\,z\,(\mathsf{step}\,F\,(\mathsf{q}\,t_1\,t_2\,t_3)) \text{ for } \mathtt{q} \notin \{\mathtt{query}, \mathtt{end}\}$$
$$\mathsf{execute}\,F\,(\mathsf{s}\,n)\,m\,z\,a\,(\mathtt{query}\,t_1\,t_2\,t_3) \to \mathsf{execute}'\,F\,n\,(\mathsf{n}\,m)\,z\,(\mathsf{setcons}\,(\mathsf{extract}\,t_2)\,a)\,(\mathtt{query}\,t_1\,t_2\,t_3)$$
$$\mathsf{execute}'\,F\,n\,m\,z\,a\,c \to \mathsf{execute}\,F\,(\mathsf{minus}\,\Theta^{P_M+1}_{F;z;a}\,m)\,m\,z\,a\,(\mathsf{step}\,F\,c)$$
$$\mathsf{execute}\,F\,n\,m\,z\,a\,(\mathtt{end}\,t_1\,t_2\,t_3) \to \mathsf{extract}\,t_1$$
$$\mathsf{extract}\,(\mathsf{split}\,(\mathsf{L}\,x)\,(\mathsf{R}\,y)) \to \mathsf{clean}\,y$$
$$\mathsf{minus}\,x\,\mathsf{o} \to x \qquad \mathsf{minus}\,\mathsf{0}\,(\mathsf{n}\,y) \to \mathsf{o} \qquad \mathsf{minus}\,(\mathsf{s}\,x)\,(\mathsf{n}\,y) \to \mathsf{minus}\,x\,y$$

That is, an execution on $\mathsf{F}\,\mathsf{S}_f\,\underline{\mathsf{w}}$ starts by computing the length of w and $P_M(|f|_\emptyset, |w|)$, and uses these as arguments to $\mathsf{execute}$. Each normal transition lowers the number n of steps we are allowed to do and increases the number n of steps we have done. Each oracle transition updates A, and either lowers n by one, or updates it to the new value $P_M(|f|_A, |w|) - m$, since we have already done m steps. Once we read the final state, the answer is read off the first tape.

For the interpretation, note that the unusual size set of nnat allows us to choose $\mathcal{J}^{\mathsf{s}}_{\mathsf{minus}} = \boldsymbol{\lambda} xy.\max(x-y, 0)$ without losing monotonicity. Hence, in every step $\mathsf{execute}\,F\,n\,m\,z\,a\,c$, the value $\max(P_M(\llbracket F \rrbracket^{\mathsf{s}}, \llbracket z \rrbracket^{\mathsf{s}}) + 1 - \llbracket m \rrbracket^{\mathsf{s}}, \llbracket n \rrbracket^{\mathsf{s}})$ decreases by at least one. Since $\llbracket \Theta^{P_M+1} F; z; a \rrbracket^{\mathsf{s}} = P_M(\llbracket F \rrbracket^{\mathsf{s}}, \llbracket z \rrbracket^{\mathsf{s}})$ regardless of a, we can use this component as part of the interpretation. The full interpretation functions for $\mathsf{execute}$ and F are long and complex, so we will not supply them here. They can be found in Appendix B. We will only conclude the other side of Theorem 2:

Theorem 4. *If $\Psi \in \mathtt{BFF}$, then there exists a finite orthogonal STRS $\mathcal{R}$ such that F computes Ψ in $\mathcal{R}$ and $\mathcal{R}$ admits a polynomially bounded interpretation.*

6 Conclusions and Future Work

In this paper, we have shown that BFF can be characterized through second-order term rewriting systems admitting polynomially bounded cost–size interpretations. This is arguably the first characterization of the basic feasible functionals purely in terms of rewriting theoretic concepts.

For the purpose of presentation, we have imposed some mild restrictions that we believe are not essential in practice. In future extensions, we can eliminate these restrictions, such as allowing lambda-abstraction, non-base type rules, and higher-order functions (assuming that F is still second-order). We can also allow arbitrary inductive data structures as input.

Another direction we definitely wish to explore is the characterization of polynomial time complexity for functionals of order strictly higher than two. It is well known that the underlying theory in this case becomes less robust than in type-2 complexity. As such, it is not clear which of the existing proposals for complexity classes of higher-order polytime complexity we can hope to capture within our framework.

References

1. Avanzini, M., Moser, G.: Polynomial path orders. Log. Methods Comput. Sci. **9**(4) (2013). https://doi.org/10.2168/LMCS-9(4:9)2013
2. Avanzini, M., Moser, G., Schaper, M.: Tct: Tyrolean complexity tool. In: Chechik, M., Raskin, J. (eds.) Proceedings of TACAS 2016 conference. Lecture Notes in Computer Science, vol. 9636, pp. 407–423. Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_24
3. Baillot, P., Dal Lago, U.: Higher-order interpretations and program complexity. In: Proceedings of CSL 2012. LIPIcs, vol. 16, pp. 62–76. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2012). https://doi.org/10.4230/LIPICS.CSL.2012.62, A journal version in Information and Computation (248), 2016
4. Baillot, P., De Benedetti, E., Ronchi Della Rocca, S.: Characterizing polynomial and exponential complexity classes in elementary lambda-calculus. Inf. Comput. **261**, 55–77 (2018). https://doi.org/10.1016/J.IC.2018.05.005
5. Beame, P., Cook, S.A., Edmonds, J., Impagliazzo, R., Pitassi, T.: The relative complexity of NP search problems. J. Comput. Syst. Sci. **57**(1), 3–19 (1998). https://doi.org/10.1006/JCSS.1998.1575
6. Beckmann, A., Weiermann, A.: A term rewriting characterization of the polytime functions and related complexity classes. Arch. Math. Log. **36**(1), 11–30 (1996). https://doi.org/10.1007/s001530050054
7. Bellantoni, S.J., Cook, S.A.: A new recursion-theoretic characterization of the polytime functions. Comput. Complex. **2**, 97–110 (1992). https://doi.org/10.1007/BF01201998
8. Bonfante, G., Cichon, A., Marion, J., Touzet, H.: Algorithms with polynomial interpretation termination proof. J. Funct. Program. **11**(1), 33–53 (2001). https://doi.org/10.1017/S0956796800003877
9. Bonfante, G., Marion, J., Moyen, J.: Quasi-interpretations a way to control resources. Theor. Comput. Sci. **412**(25), 2776–2796 (2011). https://doi.org/10.1016/j.tcs.2011.02.007

10. Cobham, A.: The intrinsic computational difficulty of functions. In: Bar-Hillel, Y. (ed.) Logic, Methodology and Philosophy of Science: Proceedings of the 1964 International Congress (Studies in Logic and the Foundations of Mathematics), pp. 24–30. North-Holland Publishing (1965)
11. Constable, R.L.: Type two computational complexity. In: Aho, A.V., Borodin, A., Constable, R.L., Floyd, R.W., Harrison, M.A., Karp, R.M., Strong, H.R. (eds.) Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA. pp. 108–121. ACM (1973). https://doi.org/10.1145/800125.804041
12. Dal Lago, U., Hofmann, M.: Realizability models and implicit complexity. Theor. Comput. Sci. **412**(20), 2029–2047 (2011). https://doi.org/10.1016/J.TCS.2010.12.025
13. Danner, N., Royer, J.S.: Adventures in time and space. In: Morrisett, J.G., Jones, S.L.P. (eds.) Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006, Charleston, South Carolina, USA, January 11-13, 2006. pp. 168–179. ACM (2006). https://doi.org/10.1145/1111037.1111053
14. Dershowitz, N.: Orderings for term-rewriting systems. Theor. Comput. Sci. **17**, 279–301 (1982). https://doi.org/10.1016/0304-3975(82)90026-3
15. Giesl, J., Thiemann, R., Schneider-Kamp, P.: The dependency pair framework: Combining techniques for automated termination proofs. In: Baader, F., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Montevideo, Uruguay, March 14-18, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3452, pp. 301–331. Springer (2004). https://doi.org/10.1007/978-3-540-32275-7_21
16. Hainry, E., Kapron, B.M., Marion, J., Péchoux, R.: A tier-based typed programming language characterizing feasible functionals. In: Hermanns, H., Zhang, L., Kobayashi, N., Miller, D. (eds.) LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020. pp. 535–549. ACM (2020). https://doi.org/10.1145/3373718.3394768
17. Hainry, E., Kapron, B.M., Marion, J., Péchoux, R.: Complete and tractable machine-independent characterizations of second-order polytime. In: Bouyer, P., Schröder, L. (eds.) Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13242, pp. 368–388. Springer (2022). https://doi.org/10.1007/978-3-030-99253-8_19
18. Hainry, E., Péchoux, R.: Theory of higher order interpretations and application to basic feasible functions. Log. Methods Comput. Sci. **16**(4) (2020), https://lmcs.episciences.org/6973
19. Hartmanis, J., Stearns, R.E.: Automata-based computational complexity. Inf. Sci. **1**(2), 173–184 (1969). https://doi.org/10.1016/0020-0255(69)90014-0
20. Irwin, R.J., Royer, J.S., Kapron, B.M.: On characterizations of the basic feasible functionals (part i). J. Funct. Program. **11**(1), 117–153 (2001). https://doi.org/10.1017/s0956796800003841
21. Kapron, B.M., Cook, S.A.: A new characterization of type-2 feasibility. SIAM J. Comput. **25**(1), 117–132 (1996). https://doi.org/10.1137/S0097539794263452
22. Kapron, B.M., Steinberg, F.: Type-two polynomial-time and restricted lookahead. In: Dawar, A., Grädel, E. (eds.) Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018. pp. 579–588. ACM (2018). https://doi.org/10.1145/3209108.3209124

23. Kawamura, A., Cook, S.A.: Complexity theory for operators in analysis. ACM Trans. Comput. Theory **4**(2), 5:1–5:24 (2012). https://doi.org/10.1145/2189778.2189780
24. Klop, J.W., van Oostrom, V., van Raamsdonk, F.: Combinatory reduction systems: Introduction and survey. Theor. Comput. Sci. **121**(1&2), 279–308 (1993). https://doi.org/10.1016/0304-3975(93)90091-7
25. Kop, C., Vale, D.: Cost-size semantics for call-by-value higher-order rewriting. In: Proc. FSCD. LIPIcs, vol. 260, pp. 15:1–15:19 (2023). https://doi.org/10.4230/LIPIcs.FSCD.2023.15
26. Kop, C., Vale, D.: Tuple interpretations for higher-order complexity. In: Kobayashi, N. (ed.) 6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference). LIPIcs, vol. 195, pp. 31:1–31:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.FSCD.2021.31
27. Kusakari, K.: On proving termination of term rewriting systems with higher-order variables. IPSJ Transactions on Programming **42**(SIG 7 (PRO 11)), 35–45 (2001), http://id.nii.ac.jp/1001/00016864/
28. Lankford, D.S.: On proving term rewriting systems are noetherian. Memo MTP-3 (1979), https://www.ens-lyon.fr/LIP/REWRITING/TERMINATION/Lankford_Poly_Term.pdf
29. Leivant, D.: A foundational delineation of computational feasiblity. In: Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991. pp. 2–11. IEEE Computer Society (1991). https://doi.org/10.1109/LICS.1991.151625
30. Manna, Z., Ness, S.: On the termination of Markov algorithms. In: Proceedings of the Third Hawaii International Conference on System Science. pp. 789–792 (1970)
31. Mehlhorn, K.: Polynomial and abstract subrecursive classes. J. Comput. Syst. Sci. **12**(2), 147–178 (1976). https://doi.org/10.1016/S0022-0000(76)80035-9
32. Oitavem, I.: Implicit characterizations of pspace. In: Kahle, R., Schroeder-Heister, P., Stärk, R.F. (eds.) Proof Theory in Computer Science, International Seminar, PTCS 2001, Dagstuhl Castle, Germany, October 7-12, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2183, pp. 170–190. Springer (2001). https://doi.org/10.1007/3-540-45504-3_11
33. Terese: Term rewriting systems, Cambridge tracts in theoretical computer science, vol. 55. Cambridge University Press (2003)

A Innermost Compatibility Theorem for STRSs

In this section, we prove Theorem 1. The program is the same as in [25], with some adaptations to the fact that we do not use lambdas and the cost component is explicit as a $\mathsf{cost}'(\cdot)$ function. Recall that in this paper, all rules are of base type, i.e., they are fully applied. Since reduction is innermost, we have that for a rule to be fired, the matching substitution (i.e., the substitution γ on the base case $\ell\gamma \to r\gamma$), does not map any variable to a term containing a redex. We restrict to this type of substitutions and notice that $\mathsf{cost}'(x\gamma) = 0$ for any variable x.

We first make the following observation about our definitions:

Lemma 9. *For all terms $s\,t$ with t of base type: $[\![s\,t]\!]^{\mathsf{s}}_{\alpha} = [\![s]\!]^{\mathsf{s}}([\![t]\!]^{\mathsf{s}})_{\alpha}$ and $[\![s\,t]\!]^{\mathsf{c}}_{\alpha,\zeta} = [\![s]\!]^{\mathsf{c}}([\![t]\!]^{\mathsf{s}})_{\alpha,\zeta}$ for all α, ζ.*

Proof. By an easy case analysis: this holds both if $s = x\,s_1 \cdots s_n$ and $s = \mathsf{f}\,s_1 \cdots s_k\,t_1 \cdots t_n$ (since t has base type, all higher-type arguments to f are given). □

Given a valuation α and substitution γ, we denote the γ-extention of α by α^γ; the valuation defined by $\alpha^\gamma(x) = [\![x\gamma]\!]^{\mathsf{s}}_\alpha$. With that in mind we start with some substitution lemmata.

Lemma 10. *Let γ be a substitution mapping all variables to irreducible terms and α be a valuation. Then, for any term s, $[\![s\gamma]\!]^{\mathsf{s}}_\alpha = [\![s]\!]^{\mathsf{s}}_{\alpha^\gamma}$.*

Proof. By induction on the structure of s.

- If s is a variable, we have $[\![x\gamma]\!]^{\mathsf{s}}_\alpha = \alpha^\gamma(x) = [\![x]\!]^{\mathsf{s}}_{\alpha^\gamma}$.
- If $s = t\,u$ is an application, we have

$$\begin{aligned}[\![(t\,u)\gamma]\!]^{\mathsf{s}} &= [\![t\gamma]\!]^{\mathsf{s}}_\alpha([\![u\gamma]\!]^{\mathsf{s}}_\alpha) \\ &\overset{IH}{=} [\![t]\!]^{\mathsf{s}}_{\alpha^\gamma}([\![u]\!]^{\mathsf{s}}_{\alpha^\gamma}) = [\![t\,u]\!]^{\mathsf{s}}_{\alpha^\gamma}\end{aligned}$$

□

Let us move on to cost versions of substitution lemmata. First, notice that we cannot directly define a γ-extention for cost valuations. Indeed, $[\![\cdot]\!]^{\mathsf{c}}_{\alpha,\zeta}$ also depends on a size valuation α. So given a size valuation α, we write ζ^γ_α to denote the valuation $\zeta^\gamma_\alpha = [\![\cdot]\!]^{\mathsf{c}}_{\alpha,\zeta} \circ \gamma$.

Lemma 11. *Given cost–size valuations α, γ and a term s such that both s and all its variables have a type of order at most 1. Then $[\![s\gamma]\!]^{\mathsf{c}}_{\alpha,\zeta} = [\![s]\!]^{\mathsf{c}}_{\alpha^\gamma,\zeta^\gamma}$.*

Proof. We consider two cases:

- For the first case, we get $s = x\,s_1 \ldots s_n$, and
  - If $n = 0$, we have $[\![x\gamma]\!]^{\mathsf{c}}_{\alpha,\zeta} = \zeta^\gamma_\alpha(x) = [\![x]\!]^{\mathsf{c}}_{\alpha^\gamma,\zeta^\gamma}$ by definition.
  - If $n > 0$, we have

$$\begin{aligned}[\![(x\,s_1 \ldots s_n)\gamma]\!]^{\mathsf{c}}_{\alpha,\zeta} &= [\![(x\gamma)\,(s_1\gamma) \ldots (s_n\gamma)]\!]^{\mathsf{c}}_{\alpha,\zeta} \\ &\overset{\text{lemma 9}}{=} [\![x\gamma]\!]^{\mathsf{c}}_{\alpha,\zeta}([\![s_1\gamma]\!]^{\mathsf{s}}_\alpha, \ldots, [\![s_n\gamma]\!]^{\mathsf{s}}_\alpha) \\ &\overset{\text{lemma 10}}{=} [\![x]\!]^{\mathsf{c}}_{\alpha^\gamma,\zeta^\gamma}([\![s_1]\!]^{\mathsf{s}}_{\alpha^\gamma}, \ldots, [\![s_n]\!]^{\mathsf{s}}_{\alpha^\gamma}) \\ &= [\![(x\,s_1 \ldots s_n)]\!]^{\mathsf{c}}_{\alpha^\gamma,\zeta^\gamma}\end{aligned}$$

- For the second case we have $s = \mathsf{f}\,s_1 \ldots s_k\,t_1 \ldots t_n$. Recall that we fixed $\mathsf{f} : (\vec{\iota_1} \Rightarrow \kappa_1) \Rightarrow \cdots \Rightarrow (\vec{\iota_k} \Rightarrow \kappa_k) \Rightarrow \nu_1 \Rightarrow \cdots \Rightarrow \nu_l \Rightarrow \iota$ as the general type for f. Hence, since we consider s of type order at most 1, f must take at least k arguments, and $0 \le n \le l$.

$$\begin{aligned}&[\![(\mathsf{f}\,s_1 \ldots s_k\,t_1 \ldots t_n)\gamma]\!]^{\mathsf{c}}_{\alpha,\zeta} \\ &= [\![\mathsf{f}\,(s_1\gamma) \ldots (s_k\gamma)\,(t_1\gamma) \ldots (t_n\gamma)]\!]^{\mathsf{c}}_{\alpha,\zeta} \\ &= \mathcal{J}_{\mathsf{f}}([\![s_1\gamma]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_1\gamma]\!]^{\mathsf{s}}_\alpha, \ldots, [\![s_k\gamma]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_k\gamma]\!]^{\mathsf{s}}_\alpha, [\![t_1\gamma]\!]^{\mathsf{s}}_\alpha, \ldots, [\![t_n\gamma]\!]^{\mathsf{s}}_\alpha) \\ &\overset{IH}{=} \mathcal{J}_{\mathsf{f}}([\![s_1]\!]^{\mathsf{c}}_{\alpha^\gamma,\zeta^\gamma}, [\![s_1]\!]^{\mathsf{s}}_{\alpha^\gamma}, \ldots, [\![s_k]\!]^{\mathsf{c}}_{\alpha^\gamma,\zeta^\gamma}, [\![s_k]\!]^{\mathsf{s}}_{\alpha^\gamma}, [\![t_1]\!]^{\mathsf{s}}_{\alpha^\gamma}, \ldots, [\![t_n]\!]^{\mathsf{s}}_{\alpha^\gamma}) \\ &= [\![\mathsf{f}\,s_1 \ldots s_k\,t_1 \ldots t_n]\!]^{\mathsf{c}}_{\alpha^\gamma,\zeta^\gamma}\end{aligned}$$

□

Next, we connect the relationship between the two cost functions we defined.

Lemma 12. *For any term $s : \iota$ so that both s and all its variables have type order 0 or 1, and any normalized substitution γ, we have that* $\mathsf{cost}(s)_{\alpha^\gamma,\zeta^\gamma} \geq \mathsf{cost}'(s\gamma)_{\alpha,\zeta}$.

Proof. We again consider two cases:

- For the first case, let $s = x\, s_1 \dots s_n$. If $n = 0$ then $\mathsf{cost}(x)_{\alpha^\gamma,\zeta^\gamma} = 0$ by definition, and since we assumed that γ is normalized, also $\mathsf{cost}'(x\gamma)_{\alpha,\zeta} = 0$. If $n > 0$ and s has base type, then $\mathsf{cost}(s)_{\alpha^\gamma,\zeta^\gamma} = \zeta^\gamma(x)([\![s_1]\!]^{\mathsf{s}}_{\alpha^\gamma,\zeta^\gamma}, \dots, [\![s_n]\!]^{\mathsf{s}}_{\alpha^\gamma,\zeta^\gamma}) + \Sigma_{i=1}^n \mathsf{cost}(s_i)_{\alpha^\gamma,\zeta^\gamma} = [\![\gamma(x)]\!]^{\mathsf{c}}_{\alpha,\zeta}([\![s_1\gamma]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_n\gamma]\!]^{\mathsf{s}}_{\alpha}) + \Sigma_{i=1}^n \mathsf{cost}(s_i)_{\alpha^\gamma,\zeta^\gamma}$ by Lemmas 10 and 11, which by Lemma 9 and the induction hypothesis $\geq [\![(x\gamma)\,(s_1\gamma)\cdots(s_n\gamma)]\!]^{\mathsf{c}} + \Sigma_{i=1}^n \mathsf{cost}'(s_i\gamma)_{\alpha,\zeta}$. Since $x\gamma$ is in normal form, either this is exactly $\mathsf{cost}'(s\gamma)$, or $\mathsf{cost}'(s\gamma) = 0$ and we are done regardless. If $n > 0$ and s does not have base type, we complete quickly with the induction hypothesis.
- For the second case, let $s = \mathsf{f}\, s_1 \dots s_k\, t_1 \dots t_n$ We have two cases whether $s\gamma$ is in normal form or not. In the first case, $\mathsf{cost}'(s\gamma)_{\alpha,\zeta} = 0$ and certainly $\mathsf{cost}(s)_{\alpha^\gamma,\zeta^\gamma} \geq 0$. For the second case, s is not in normal form.
  If s has base type, then:

$$\begin{aligned}
\mathsf{cost}(s)_{\alpha^\gamma,\zeta^\gamma} &= \mathsf{cost}(\mathsf{f}\, s_1 \dots s_k\, t_1 \dots t_n)_{\alpha^\gamma,\zeta^\gamma} \\
&= [\![s]\!]^{\mathsf{c}}_{\alpha^\gamma,\zeta^\gamma} + \sum_{i=1}^{k} \mathsf{cost}(s_i)_{\alpha^\gamma,\zeta^\gamma} + \sum_{j=1}^{n} \mathsf{cost}(s_j)_{\alpha^\gamma,\zeta^\gamma} \\
&\overset{\text{lemma 11}}{=} [\![s\gamma]\!]^{\mathsf{c}}_{\alpha,\zeta} + \sum_{i=1}^{k} \mathsf{cost}(s_i)_{\alpha^\gamma,\zeta^\gamma} + \sum_{j=1}^{n} \mathsf{cost}(s_j)_{\alpha^\gamma,\zeta^\gamma} \\
&\overset{IH}{\geq} [\![s\gamma]\!]^{\mathsf{c}}_{\alpha,\zeta} + \sum_{i=1}^{k} \mathsf{cost}'(s_i\gamma)_{\alpha,\zeta} + \sum_{j=1}^{n} \mathsf{cost}'(s_j\gamma)_{\alpha,\zeta} \\
&= \mathsf{cost}'(s\gamma)_{\alpha,\zeta}
\end{aligned}$$

  If not, then:

$$\begin{aligned}
\mathsf{cost}(s)_{\alpha^\gamma,\zeta^\gamma} &= \sum_{i=1}^{k} \mathsf{cost}(s_i)_{\alpha^\gamma,\zeta^\gamma} + \sum_{j=1}^{n} \mathsf{cost}(s_j)_{\alpha^\gamma,\zeta^\gamma} \\
&\overset{IH}{\geq} \sum_{i=1}^{k} \mathsf{cost}'(s_i\gamma)_{\alpha,\zeta} + \sum_{j=1}^{n} \mathsf{cost}'(s_j\gamma)_{\alpha,\zeta} \\
&= \mathsf{cost}'(s\gamma)_{\alpha,\zeta}
\end{aligned}$$

□

The lemma below is restricted to type-1 terms and we assume that size compatibility holds. This lemma is only needed in one specific step of the inductive step prof of Theorem 1.

Lemma 13. *Let $(\mathbb{F}, \mathcal{R})$ be a STRS satisfying the compatibility conditions of Theorem 1 and s, t be type-1 terms of the same type. Assume that $[\![s]\!]^{\mathsf{s}}_{\alpha} \sqsupseteq [\![t]\!]^{\mathsf{s}}_{\alpha}$. Then $[\![s]\!]^{\mathsf{c}}_{\alpha,\zeta} \geq [\![t]\!]^{\mathsf{c}}_{\alpha,\zeta}$ whenever $s \to_{\mathcal{R}} t$.*

Proof. The proof is by induction on $s \to_{\mathcal{R}} t$.

- For the base case we get:

$$\begin{aligned}
[\![\ell\gamma]\!]^{\mathsf{c}}_{\alpha,\zeta} &= [\![\ell]\!]^{\mathsf{c}}_{\alpha^{\gamma},\zeta^{\gamma}} \\
&> \mathsf{cost}(r)_{\alpha^{\gamma},\zeta^{\gamma}}, \text{by compatibility} \\
&= [\![r]\!]^{\mathsf{c}}_{\alpha^{\gamma},\zeta^{\gamma}} + \sum [\![t]\!]^{\mathsf{c}}_{\alpha^{\gamma},\zeta^{\gamma}}, \text{ where } r \trianglerighteq t_i \\
&\geq [\![r]\!]^{\mathsf{c}}_{\alpha^{\gamma},\zeta^{\gamma}} = [\![r\gamma]\!]^{\mathsf{c}}_{\alpha,\zeta} \text{by lemma 11}
\end{aligned}$$

- For the second part, we recall that to get a type-1 term of arrow type, we need to partially apply a function symbol or a variable, and since rules are of base type, reduction does not occur at head position in s. Then we get two cases:
  - First, $s = x\, s_1 \dots s_n$, and assume w.l.g that $x : \iota_1 \Rightarrow \cdots \Rightarrow \iota_k \Rightarrow \kappa$ and $n < k$. So we get $x\, s_1 \dots s_i \dots s_n \to_{\mathcal{R}} x\, s_1 \dots s_i \dots s_n$ with $s_i \to_{\mathcal{R}} s_i'$.

$$\begin{aligned}
[\![x\, s_1 \dots s_i \dots s_n]\!]^{\mathsf{c}}_{\alpha,\zeta} &= \zeta(x)([\![s_1]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_i]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_n]\!]^{\mathsf{s}}_{\alpha}) \\
&\sqsupseteq \zeta(x)([\![s_1]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_i']\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_n]\!]^{\mathsf{s}}_{\alpha}) \\
&= [\![x\, s_1 \dots s_i' \dots s_n]\!]^{\mathsf{c}}_{\alpha,\zeta}
\end{aligned}$$

  - The case for $\mathsf{f}\, s_1 \dots s_i \dots s_n$ is similar to the variable one, with the observation that by assumption $[\![s_i]\!]^{\mathsf{s}}_{\alpha} \sqsupseteq [\![s_i']\!]^{\mathsf{s}}$ holds. Then we use the monotonicity of $\mathcal{J}^{\mathsf{s}}_{\mathsf{f}}$.

□

Finally, we can state and prove the innermost compatibility theorem.

Theorem 1 (Innermost Compatibility). *Suppose $\mathcal{R}$ is an STRS compatible with a cost–size interpretation $\mathcal{F}$, then for any valuations α and ζ we have $\mathsf{cost}'(s)_{\alpha,\zeta} > \mathsf{cost}'(t)_{\alpha,\zeta}$ and $[\![s]\!]^{\mathsf{s}}_{\alpha} \sqsupseteq [\![t]\!]^{\mathsf{s}}_{\alpha}$ whenever $s \to_{\mathcal{R}} t$.*

Proof. The proof follows by induction on the reduction $s \to_{\mathcal{R}} t$.

Size Case.

- In the base case, we have $s \to_{\mathcal{R}} t$ by $\ell\gamma \to r\gamma$. Then we combine the substitution lemma with the compatibility requirement for size, i.e., $[\![\ell]\!]^{\mathsf{s}}_{\alpha} \sqsupseteq [\![r]\!]^{\mathsf{s}}_{\alpha}$, as follows:

$$[\![\ell\gamma]\!]^{\mathsf{s}}_{\alpha} = [\![\ell]\!]^{\mathsf{s}}_{\alpha^{\gamma}} \sqsupseteq [\![r]\!]^{\mathsf{s}}_{\alpha^{\gamma}} = [\![r\gamma]\!]^{\mathsf{s}}_{\alpha}$$

- In the application case, we simply apply the induction hypothesis and the fact that in $[\![s\, t]\!]^{\mathsf{s}} = [\![s]\!]^{\mathsf{s}}([\![t]\!]^{\mathsf{s}})$, the function $[\![s]\!]^{\mathsf{s}}$ is weakly monotonic.

Cost Case.

- For the base case, we have that

$$\mathsf{cost}'(\ell\gamma)_{\alpha,\zeta} = [\![\ell\gamma]\!]^{\mathsf{c}}_{\zeta} = [\![\ell]\!]^{\mathsf{c}}_{\zeta^\gamma} > \mathsf{cost}(r)_{\alpha^\gamma,\zeta^\gamma} \geq \mathsf{cost}'(r\gamma)_{\alpha,\zeta}$$

- For the application case with a variable root symbol, we have that $x\,t_1 \dots t_i \dots t_n \to_{\mathcal{R}} x\,t_1 \dots t'_i \dots t_n$ with $t_i \to_{\mathcal{R}} t'_i$. By induction we get $\mathsf{cost}'(t_i) > \mathsf{cost}'(t'_i)$ and also use the size part $[\![t_i]\!]^{\mathsf{s}} \sqsupseteq [\![t'_i]\!]^{\mathsf{s}}$. Then:

$$\begin{aligned}
&\mathsf{cost}'(x\,t_1 \dots t_i \dots t_n)_{\alpha,\zeta} \\
&= [\![x\,t_1 \dots t_i \dots t_n]\!]^{\mathsf{c}}_{\alpha,\zeta} + \sum_{j=1}^{n} \mathsf{cost}'(t_j)_{\alpha,\zeta} \\
&= \zeta(x)([\![s_1]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_i]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_n]\!]^{\mathsf{s}}_{\alpha}) + \sum_{\substack{j=1\dots n \\ j\neq i}} \mathsf{cost}'(t_j)_{\alpha,\zeta} + \mathsf{cost}'(t_i)_{\alpha,\zeta} \\
&\geq \zeta(x)([\![s_1]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s'_i]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_n]\!]^{\mathsf{s}}_{\alpha}) + \sum_{\substack{j=1\dots n \\ j\neq i}} \mathsf{cost}'(t_j)_{\alpha,\zeta} + \mathsf{cost}'(t_i)_{\alpha,\zeta}, \\
&> \zeta(x)([\![s_1]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s'_i]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_n]\!]^{\mathsf{s}}_{\alpha}) + \sum_{\substack{j=1\dots n \\ j\neq i}} \mathsf{cost}'(t_j)_{\alpha,\zeta} + \mathsf{cost}'(t'_i)_{\alpha,\zeta} \\
&= \mathsf{cost}'(x\,t_1 \dots t'_i \dots t_n)_{\alpha,\zeta}
\end{aligned}$$

- For the application case with a function root symbol where the reduction is done in a base-type argument, we have that $\mathsf{f}\,s_1 \dots s_k\,t_1 \dots t_i \dots t_n \to_{\mathcal{R}} \mathsf{f}\,s_1 \dots s_k\,t_1 \dots t'_i \dots t_n$ with $t_i \to_{\mathcal{R}} t'_i$. Let us write $\vec{s}$ for $s_1 \dots s_k$ and $\mathsf{c}(s)$ for $\sum_{j=1}^{k} \mathsf{cost}'(s_i)_{\alpha,\zeta}$ below. We also abuse notation and write $[\![\vec{s}]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![\vec{s}]\!]^{\mathsf{s}}_{\alpha}$ for $[\![s_1]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_1]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_k]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_k]\!]^{\mathsf{s}}_{\alpha}$.

$$\begin{aligned}
&\mathsf{cost}'(\mathsf{f}\,\vec{s}\,t_1 \dots t_i \dots t_n) \\
&= [\![\mathsf{f}\,\vec{s}\,t_1 \dots t_i \dots t_n]\!]^{\mathsf{c}}_{\alpha,\zeta} + \mathsf{c}(s) + \sum_{j=1}^{n} \mathsf{cost}'(t_j)_{\alpha,\zeta} \\
&= \mathcal{J}^{\mathsf{c}}_{\mathsf{f}}([\![\vec{s}]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![\vec{s}]\!]^{\mathsf{s}}_{\alpha}, [\![t_1]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![t_i]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![t_n]\!]^{\mathsf{s}}_{\alpha}) + \mathsf{c}(s) + \sum_{j=1}^{n} \mathsf{cost}'(t_j)_{\alpha,\zeta} \\
&\geq \mathcal{J}^{\mathsf{c}}_{\mathsf{f}}([\![\vec{s}]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![\vec{s}]\!]^{\mathsf{s}}_{\alpha}, [\![t_1]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![t'_i]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![t_n]\!]^{\mathsf{s}}_{\alpha}) + \mathsf{c}(s) + \sum_{j=1}^{n} \mathsf{cost}'(t_j)_{\alpha,\zeta} \\
&> \mathsf{cost}'(\mathsf{f}\,\vec{s}\,t_1 \dots t'_i \dots t_n)
\end{aligned}$$

  where in the last step we use $\mathsf{cost}'(t_i) > \mathsf{cost}'(t'_i)$, given by the IH.
- For the application case with a function root symbol where the reduction is done in a higher-type argument, we have that $\mathsf{f}\,s_1 \dots s_i \dots s_k\,t_1 \dots t_n \to_{\mathcal{R}} \mathsf{f}\,s_1 \dots s'_i \dots s_k\,t_1 \dots t_n$ with $s_i \to_{\mathcal{R}} s'_i$. Recall that by IH we get $\mathsf{cost}'(s_i) >$

$\mathsf{cost}'(s_i')$. Also, s_i is a type-1 term and here we are under the compatibility conditions and $[\![s_i]\!]^{\mathsf{s}} \sqsupseteq [\![s_i']\!]^{\mathsf{s}}$ is valid by the size part of the theorem. Hence the conditions of Lemma 13 are satisfied, so we get $[\![s_i]\!]^{\mathsf{c}} \geq [\![s_i']\!]^{\mathsf{c}}$, as well.
With this in hand we reason as follows:

$$
\begin{aligned}
&\mathsf{cost}'(\mathsf{f}\, s_1 \dots s_i \dots s_k\, \vec{t})_{\alpha,\zeta} \\
&= \mathcal{J}^{\mathsf{c}}_{\mathsf{f}}([\![s_1]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_1]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_i]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_i]\!]^{\mathsf{s}}_{\alpha} \dots, [\![s_k]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_k]\!]^{\mathsf{s}}_{\alpha}, [\![\vec{t}]\!]^{\mathsf{s}}_{\alpha}) \\
&\qquad + \sum_{j=1\dots k, j\neq i} \mathsf{cost}'(s_j) + \mathsf{cost}'(s_i) + \sum_{j=1}^{n} \mathsf{cost}'(t_j) \\
&\text{by monotonicity of } \mathcal{J}^{\mathsf{c}}_{\mathsf{f}} \text{ and } [\![s_i]\!]^{\mathsf{c}} \geq [\![s_i']\!]^{\mathsf{c}}, [\![s_i]\!]^{\mathsf{s}} \sqsupseteq [\![s_i']\!]^{\mathsf{s}}, \text{ we get} \\
&\geq \mathcal{J}^{\mathsf{c}}_{\mathsf{f}}([\![s_1]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_1]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_i']\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_i']\!]^{\mathsf{s}}_{\alpha} \dots, [\![s_k]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_k]\!]^{\mathsf{s}}_{\alpha}, [\![\vec{t}]\!]^{\mathsf{s}}_{\alpha}) \\
&\qquad + \sum_{j=1\dots k, j\neq i} \mathsf{cost}'(s_j) + \mathsf{cost}'(s_i) + \sum_{j=1}^{n} \mathsf{cost}'(t_j) \\
&> \mathcal{J}^{\mathsf{c}}_{\mathsf{f}}([\![s_1]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_1]\!]^{\mathsf{s}}_{\alpha}, \dots, [\![s_i']\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_i']\!]^{\mathsf{s}}_{\alpha} \dots, [\![s_k]\!]^{\mathsf{c}}_{\alpha,\zeta}, [\![s_k]\!]^{\mathsf{s}}_{\alpha}, [\![\vec{t}]\!]^{\mathsf{s}}_{\alpha}) \\
&\qquad + \sum_{j=1\dots k, j\neq i} \mathsf{cost}'(s_j) + \mathsf{cost}'(s_i') + \sum_{j=1}^{n} \mathsf{cost}'(t_j) \\
&= \mathsf{cost}'(\mathsf{f}\, s_1 \dots s_i' \dots s_k\, \vec{t})_{\alpha,\zeta}
\end{aligned}
$$

□

B Interpretations for Section 5

B.1 Interpretations for section 5.3

The omitted interpretation functions in section 5.3 are:

$$
\begin{aligned}
\mathcal{J}^{\mathsf{s}}_{\mathsf{len}} &= \boldsymbol{\lambda} x.x & \mathcal{J}^{\mathsf{c}}_{\mathsf{len}} &= \boldsymbol{\lambda} x.x+1 \\
\mathcal{J}^{\mathsf{s}}_{\mathsf{max}} &= \boldsymbol{\lambda} nm.\max(n,m) & \mathcal{J}^{\mathsf{c}}_{\mathsf{max}} &= \boldsymbol{\lambda} nm.n+1 \\
\mathcal{J}^{\mathsf{s}}_{\mathsf{limit}} &= \boldsymbol{\lambda} xn.n & \mathcal{J}^{\mathsf{c}}_{\mathsf{limit}} &= \boldsymbol{\lambda} xn.n+1 \\
\mathcal{J}^{\mathsf{s}}_{\mathsf{retif}} &= \boldsymbol{\lambda} xnz.z & \mathcal{J}^{\mathsf{c}}_{\mathsf{retif}} &= \boldsymbol{\lambda} xnz.n+1
\end{aligned}
$$

It is easy to see that the corresponding rules are all oriented.

For $\mathsf{tryapply}$, note that $\mathsf{tryapply}\, F\, a \ulcorner \mathsf{n} \urcorner$ reduces to $\ulcorner |\mathsf{F(a)}| \urcorner$ if $|a| \leq n$, and to $\ulcorner \mathsf{0} \urcorner$ otherwise. Thus, it indeed returns exactly $|F|_{\{a\}}(n)$.

$$
\mathcal{J}^{\mathsf{s}}_{\mathsf{tryapply}} = \boldsymbol{\lambda} Fan.F(n) \qquad \mathcal{J}^{\mathsf{c}}_{\mathsf{tryapply}} = \boldsymbol{\lambda} F^c F^s an.F^c(n) + F^s(n) + 2 * n + 4
$$

We easily see that $[\![\mathsf{tryapply}\, a\, n]\!]^{\mathsf{s}} = [\![\mathsf{len}\, (\mathsf{retif}\, a\, n\, (F\, (\mathsf{limit}\, a\, n)))]\!]^{\mathsf{s}}$. As for the cost, note that

$$
\begin{aligned}
&\mathsf{cost}(\mathsf{len}\, (\mathsf{retif}\, a\, n\, (F\, (\mathsf{limit}\, a\, n)))) \\
&= [\![\mathsf{len}\, (\mathsf{retif}\, a\, n\, (F\, (\mathsf{limit}\, a\, n)))]\!]^{\mathsf{c}} + [\![\mathsf{retif}\, a\, n\, (F\, (\mathsf{limit}\, a\, n))]\!]^{\mathsf{c}} + \\
&\quad [\![F\, (\mathsf{limit}\, a\, n]\!]^{\mathsf{c}} + [\![\mathsf{limit}\, a\, n]\!]^{\mathsf{c}} \\
&= (F^c(n)+1) + (n+1) + F^s(n) + (n+1) = F^c(n) + F^s(n) + 2n + 3
\end{aligned}
$$

Hence, also the tryapply rule is oriented.

To interpret sets and the apply rule, we use:

$$\begin{array}{rrrr} \mathcal{J}^{\mathsf{s}}_{\emptyset} = 0 & \mathcal{J}^{\mathsf{c}}_{\emptyset} = 0 & \mathcal{J}^{\mathsf{s}}_{\mathsf{setcons}} = \boldsymbol{\lambda} xy.y + 1 & \mathcal{J}^{\mathsf{s}}_{\mathsf{setcons}} = \boldsymbol{\lambda} xy.0 \\ \mathcal{J}^{\mathsf{s}}_{\mathsf{tryall}} = \boldsymbol{\lambda} Fan.F(n) & & & \\ \mathcal{J}^{\mathsf{c}}_{\mathsf{tryall}} = \boldsymbol{\lambda} F^c F^s an.1 + a * (F^c(n) + 2 * F^s(n) + 2 * n + 6) & & & \end{array}$$

To see that the rule is oriented, note:

$$\begin{aligned} [\![\mathsf{tryall}\, F\, (\mathsf{setcons}\, a\, tl)\, n]\!]^{\mathsf{s}} &= F^s(n) \\ &= \max(F^s(n), F^s(n)) \\ &= [\![\mathsf{max}\, (\mathsf{tryapply}\, F\, a\, n)\, (\mathsf{tryall}\, F\, tl\, n)]\!]^{\mathsf{s}} \end{aligned}$$

and

$$\begin{aligned} & [\![\mathsf{tryall}\, F\, (\mathsf{setcons}\, a\, tl)\, n]\!]^{\mathsf{c}} \\ &= 1 + (tl + 1) * (F^c(n) + 2 * F^s(n) + 2 * n + 6) \\ &= 1 + tl * (F^c(n) + 2 * F^s(n) + 2 * n + 6) \\ &\qquad + 1 * (F^c(n) + 2 * F^s(n) + 2 * n + 6) \\ &= [\![\mathsf{tryall}\, F\, tl\, n]\!]^{\mathsf{c}} + (F^c(n) + 2 * F^s(n) + 2 * n + 6) \\ &= [\![\mathsf{tryall}\, F\, tl\, n]\!]^{\mathsf{c}} + [\![\mathsf{tryapply}\, F\, a\, n]\!]^{\mathsf{c}} + F^s(n) + 2 \\ &= [\![\mathsf{tryall}\, F\, tl\, n]\!]^{\mathsf{c}} + [\![\mathsf{tryapply}\, F\, a\, n]\!]^{\mathsf{c}} + [\![\mathsf{max}\, (\mathsf{tryapply}\, F\, a\, n)\, (\mathsf{tryall}\, F\, tl\, n)]\!]^{\mathsf{c}} + 1 \\ &> \mathsf{cost}(\mathsf{max}\, (\mathsf{tryapply}\, F\, a\, n)\, (\mathsf{tryall}\, F\, tl\, n)) \end{aligned}$$

B.2 Interpretations for section 5.4

We first supply the interpretation functions for the nnat symbols and the two simple rules:

$$\begin{array}{rr} \mathcal{J}^{\mathsf{s}}_{\mathsf{o}} = 0 & \mathcal{J}^{\mathsf{c}}_{\mathsf{o}} = 0 \\ \mathcal{J}^{\mathsf{s}}_{\mathsf{n}} = \boldsymbol{\lambda} x.x + 1 & \mathcal{J}^{\mathsf{c}}_{\mathsf{n}} = \boldsymbol{\lambda} x.0 \\ \mathcal{J}^{\mathsf{s}}_{\mathsf{extract}} = \boldsymbol{\lambda} x.x & \mathcal{J}^{\mathsf{c}}_{\mathsf{extract}} = \boldsymbol{\lambda} x.x + 2 \\ \mathcal{J}^{\mathsf{s}}_{\mathsf{minus}} = \boldsymbol{\lambda} xy.\max(x - y, 0) & \\ \mathcal{J}^{\mathsf{c}}_{\mathsf{minus}} = \boldsymbol{\lambda} xy.x & \end{array}$$

These functions are all monotonic, and their rules are oriented (as can easily be checked).

By induction on the polynomial P, we can find polynomials A_P, B_P such that $\mathsf{cost}(\Theta^P_{F;z;a}) \leq [\![a]\!]^{\mathsf{s}} * A_P(F^c, F^s, [\![z]\!]^{\mathsf{s}}) + B_P(F^c, F^s, [\![z]\!]^{\mathsf{s}})$, assuming F, z and a are in normal form.

To define our remaining interpretation functions, first let:

- $\theta_{F,z,n,m} := \max(P_M(F^s, z) + 1 - m, n)$
- $\mathtt{POLY}_{F,z}[x] := x * A_{P_M+1}(F^c, F^s, [\![z]\!]^{\mathsf{s}}) + B_{P_M+1}(F^c, F^s, [\![z]\!]^{\mathsf{s}})$, so the polynomial bounding $\mathsf{cost}(\Theta^{P_M+1}_{F;z;a})$ if $[\![a]\!]^{\mathsf{s}} = x$.

Then, we can orient the size interpretations of the rewrite rules by the following interpretation:

$$\begin{aligned}
\mathcal{J}^{\mathsf{s}}_{\mathsf{F}} &= \boldsymbol{\lambda} Fn.n + P_M(F,n) + 1 \\
\mathcal{J}^{\mathsf{s}}_{\mathsf{F}'} &= \boldsymbol{\lambda} Fzc.c + P_M(F,z) + 1 \\
\mathcal{J}^{\mathsf{s}}_{\mathsf{execute}} &= \boldsymbol{\lambda} Fnmzac.c + \theta_{F,z,n,m} \\
\mathcal{J}^{\mathsf{s}}_{\mathsf{execute}'} &= \boldsymbol{\lambda} Fnmzac.c + 1 + \theta_{F,z,n,m}
\end{aligned}$$

And the cost interpretations by:

$$\begin{aligned}
\mathcal{J}^{\mathsf{c}}_{\mathsf{F}} &= \boldsymbol{\lambda} Fn.(P_M(F^s,n)+1) * (\\
&\qquad 8 + 3 * P_M(F^s,n) + 2 * n + F^c(P_M(F^s,n)+n+1) + \\
&\qquad \mathtt{POLY}_{F,z}[P_M(F^s,n)+1] \\
&\quad) + 6 + 2 * n + \mathtt{POLY}_{F,z}[0] \\
\mathcal{J}^{\mathsf{c}}_{\mathsf{F}'} &= \boldsymbol{\lambda} Fzc.(P_M(F^s,z)+1) * (\\
&\qquad 8 + 3 * P_M(F^s,z) + 2 * c + F^c(P_M(F^s,z)+1+c) + \\
&\qquad \mathtt{POLY}_{F,z}[P_M(F^s,z)+1] \\
&\quad) + 4 + c + \mathtt{POLY}_{F,z}[0] \\
\mathcal{J}^{\mathsf{c}}_{\mathsf{execute}} &= \boldsymbol{\lambda} Fnmzac.\theta_{F,z,n,m} * (\\
&\qquad 5 + 2 * (\theta_{F,z,n,m} + c) + F^c(\theta_{F,z,n,m} + c) + \\
&\qquad \mathtt{POLY}_{F,z}[\theta_{F,z,n,m} + a] + P_M(F^s,z) \\
&\quad) + 3 + \theta_{F,z,n,m} + c \\
\mathcal{J}^{\mathsf{c}}_{\mathsf{execute}'} &= \boldsymbol{\lambda} Fnmzac.(\theta_{F,z,n,m}+1) * (\\
&\qquad 5 + 2 * (\theta_{F,z,n,m} + c + 1) + F^c(\theta_{F,z,n,m} + c + 1) + \\
&\qquad \mathtt{POLY}_{F,z}[\theta_{F,z,n,m} + a] + P_M(F^s,z) \\
&\quad) + 1
\end{aligned}$$

To see that these interpretations are correct, we first observe:

$$\begin{aligned}
\theta_{F,z,\mathsf{s}\,n,m} &= \max(P_M(F^s,z) + 1 - m, n + 1) \\
&= \max(P_M(F^s,z) + 1 - (m+1), n) + 1 \\
&= \theta_{F,z,n,m+1} + 1
\end{aligned}$$

(Because $\max(a+1, b+1) = \max(a,b) + 1$.) We also have, for all a:

$$\begin{aligned}
\theta_{F,z,n,m} &= \max(P_M(F^s,z) + 1 - m, n) \\
&\geq \max(P_M(F^s,z) + 1 - m, 0) \\
&= \max(P_M(F^s,z) + 1 - m, \max(P_M(F^s,z) + 1 - m), 0) \\
&= \theta_{F,z,[\![\mathsf{minus}\,\Theta^{P_M+1}_{F;z;m}]\!]^{\mathsf{s}},m}
\end{aligned}$$

The inequalities now follow by writing out definitions.