Amortised Resource Analysis and Typed Polynomial Interpretations*

Martin Hofmann[1] and Georg Moser[2]

[1] Institute of Computer Science, LMU Munich, Germany email: hofmann@ifi.lmu.de
[2] Institute of Computer Science, University of Innsbruck, Austria, email: georg.moser@uibk.ac.at

Abstract. We introduce a novel resource analysis for typed term rewrite systems based on a potential-based type system. This type system gives rise to polynomial bounds on the innermost runtime complexity. We relate the thus obtained amortised resource analysis to polynomial interpretations and obtain the perhaps surprising result that whenever a rewrite system $\mathcal{R}$ can be well-typed, then there exists a polynomial interpretation that orients $\mathcal{R}$. For this we adequately adapt the standard notion of polynomial interpretations to the typed setting.

Keywords: term rewriting, types, amortised resource analysis, complexity of rewriting, polynomial interpretations

1 Introduction

In recent years there have been several approaches to the automated analysis of the complexity of programs. Without hope for completeness, we mention work by Albert et al. [1] that underlies COSTA, an automated tool for the resource analysis of Java programs. Related work, targeting C programs, has been reported by Alias et al. [2]. In Zuleger et al. [3] further approaches for the runtime complexity analysis of C programs is reported, incorporated into LOOPUS. Noschinski et al. [4] study runtime complexity analysis of rewrite systems, which has been incorporated in AProVE. Finally, the RaML prototype [5] provides an automated potential-based resource analysis for various resource bounds of functional programs and TcT [6] is one of the most powerful tools for complexity analysis of rewrite systems.

Despite the abundance in the literature on complexity analysis of programs, almost no comparison results are known that relate the sophisticated methods developed. Indeed a precise comparison proves often difficult. Consider Example 1; $\mathcal{R}_{\mathsf{que}}$ encodes an efficient implementation of a queue in functional programming. A queue is represented as a pair of two lists $\mathsf{que}(f, r)$, encoding the initial part f and the reversal of the remainder r. The invariant of the algorithm is that the first list never becomes empty, which is achieved by reversing r if necessary. Should the invariant ever be violated, an exception (err_head or err_tail) is raised.

* This research is partly supported by FWF (Austrian Science Fund) project P25781.

Example 1. Consider the following term rewrite system (TRS for short) $\mathcal{R}_{\mathsf{que}}$, encoding a variant of an example by Okasaki [7, Section 5.2].

$$
\begin{array}{rlrl}
1\colon & \mathsf{chk}(\mathsf{que}(\mathsf{nil}, r)) \to \mathsf{que}(\mathsf{rev}(r), \mathsf{nil}) & 7\colon & \mathsf{enq}(0) \to \mathsf{que}(\mathsf{nil}, \mathsf{nil}) \\
2\colon & \mathsf{chk}(\mathsf{que}(x \sharp xs, r)) \to \mathsf{que}(x \sharp xs, r) & 8\colon & \mathsf{rev}'(\mathsf{nil}, ys) \to ys \\
3\colon & \mathsf{tl}(\mathsf{que}(x \sharp f, r)) \to \mathsf{chk}(\mathsf{que}(f, r)) & 9\colon & \mathsf{rev}(xs) \to \mathsf{rev}'(xs, \mathsf{nil}) \\
4\colon & \mathsf{snoc}(\mathsf{que}(f, r), x) \to \mathsf{chk}(\mathsf{que}(f, x \sharp r)) & 10\colon & \mathsf{hd}(\mathsf{que}(x \sharp f, r)) \to x \\
5\colon & \mathsf{rev}'(x \sharp xs, ys) \to \mathsf{rev}'(xs, x \sharp ys) & 11\colon & \mathsf{hd}(\mathsf{que}(\mathsf{nil}, r)) \to \mathsf{err\_head} \\
6\colon & \mathsf{enq}(\mathsf{s}(n)) \to \mathsf{snoc}(\mathsf{enq}(n), n) & 12\colon & \mathsf{tl}(\mathsf{que}(\mathsf{nil}, r)) \to \mathsf{err\_tail}\ .
\end{array}
$$

We exemplify the physicist's method of amortised analysis [8]. We assign to every queue $\mathsf{que}(f, r)$ the length of r as *potential*. Then the amortised cost for each operation is constant, as the costly reversal operation is only executed if the potential can pay for the operation, cf. [7]. Thus, based on an amortised analysis, we deduce the optimal linear runtime complexity for $\mathcal{R}$. Let us attempt to apply the interpretation method instead. Termination proofs by interpretations are well-established and can be traced back to work by Turing [9]. It is straightforward to restrict *polynomial interpretations* [10] so that compatibility with a TRS $\mathcal{R}$ induces polynomial runtime complexity of $\mathcal{R}$, cf. [11]. Such polynomial interpretations are called *restricted*. However, we can see that no restricted polynomial interpretation can exist that is compatible with $\mathcal{R}_{\mathsf{que}}$. The constraints induced by $\mathcal{R}_{\mathsf{que}}$ imply that the function snoc has to be interpreted by a linear polynomial. Thus an exponential interpretation is required for enqueuing (enq). Looking more closely at the different proofs, we observe the following. While in the amortised analysis the potential of a queue $\mathsf{que}(f, r)$ depends only on the remainder r, the interpretation of que has to be monotone in both arguments by definition. This difference induces that snoc is assigned a *strongly linear* potential in the amortised analysis, while only a *linear* interpretation is possible for snoc.

Still it is possible to relate amortised analysis to polynomial interpretations if we base our investigation on many-sorted (or typed) TRSs and make suitable use of the concept of *annotated types* originally introduced in [12]. We note that Example 1 is also subject to other techniques like quasi-interpretations [13] and can also be handled fully automatically in AProVE or $\mathsf{T_C T}$. However, our interest in the example stems from the fact that it shows a separation between amortised analysis and restricted polynomial interpretations.

We establish a novel innermost runtime complexity analysis for typed constructor rewrite systems $\mathcal{R}$. This complexity analysis is based on a potential-based amortised analysis incorporated into a type system. From the annotated type of a term its derivation height with respect to innermost rewriting can be read off, inducing polynomial bounds on the runtime complexity with respect to $\mathcal{R}$ (see Theorem 12). The correctness proof of the obtained bound rests on an operational big-step semantics decorated with counters for the derivation height of the evaluated terms. We complement this big-step semantics with a similar decorated small-step semantics and prove equivalence between these semantics.

Furthermore we establish a second soundness result based on the small-step semantics (see Theorem 20). Exploiting the small-step semantics we prove our main result that from the well-typing of $\mathcal{R}$ we can read off a typed polynomial interpretation that orients $\mathcal{R}$ (see Theorem 23).

While the type system exhibited is inspired by Hoffmann et al. [14] we generalise their use of annotated types to arbitrary (data) types. Furthermore the introduced small-step semantics (and our main result) directly establish that any well-typed TRS is terminating, cf. [15]. As a corollary to our main result, we obtain that the physicist's method of amortised analysis conceptually amounts to the interpretation method, if we allow for the following changes: (i) every term bears a potential, not only values, (ii) polynomial interpretations are defined over annotated types, and (iii) compatibility is replaced by orientability.

Our study is purely theoretic, and we have not (yet) an implementation of the provided techniques. However, automation is straightforward and seems to yield fairly precise bounds on the runtime complexity. Furthermore, we have restricted our study to typed (constructor) TRSs. In the conclusion we sketch application of the established results to innermost runtime complexity analysis of untyped TRSs.

This paper is structured as follows. In the next section we cover basics. In Section 3 we provide our first soundness result. In Section 4 we establish our second soundness result. Our main result will be stated and proved in Section 5. Finally, we conclude in Section 6. Due to space limitations some proofs are only sketched, or have been completely omitted. The reader is kindly referred to the extended version of this paper [16].

2 Typed Term Rewrite Systems

Let $\mathcal{C}$ denote a finite, non-empty set of *constructor symbols* and $\mathcal{D}$ a finite set of *defined function symbols*. Let S be a finite set of (data) types. A family $(X_A)_{A\in S}$ of sets is called *S-typed* and denoted as X. Let $\mathcal{V}$ denote an S-typed set of *variables*, such that the $\mathcal{V}_A$ are pairwise disjoint. In the following, variables will be denoted by x, y, $\ldots$, possibly extended by subscripts.

Following [17], a *type declaration* is of the form $[A_1 \times \cdots \times A_n] \to C$, where A_i and C are types. Type declarations serve as input-output specifications for function symbols. We write A instead of $[] \to A$. A *signature* $\mathcal{F}$ (with respect to the set of types S) is a mapping from $\mathcal{C} \cup \mathcal{D}$ to type declarations. We often write $f\colon [A_1 \times \cdots \times A_n] \to C$, if $\mathcal{F}(f) = [A_1 \times \cdots \times A_n] \to C$ and refer to a type declaration as a type, if no confusion can arise.

We define the S-typed set of terms $\mathcal{T}(\mathcal{D} \cup \mathcal{C}, \mathcal{V})$ (or $\mathcal{T}$ for short): (i) for each $A \in S$: $\mathcal{V}_A \subseteq \mathcal{T}_A$, (ii) for $f \in \mathcal{C} \cup \mathcal{D}$ such that $\mathcal{F}(f) = [A_1, \ldots, A_n] \to C$ and $t_i \in \mathcal{T}_{A_i}$, we have $f(t_1, \ldots, t_n) \in \mathcal{T}_C$. Type assertions are denoted $t\colon A$. Terms of type A will sometimes be referred to as instances of A: a term of list type, is simply called a list. If $t \in \mathcal{T}(\mathcal{C}, \varnothing)$ then t is called a *ground constructor term* or a *value*. The set of values is denoted $\mathcal{T}(\mathcal{C})$. The (S-typed) set of variables of a term t is denoted $\mathcal{V}\mathsf{ar}(t)$. The root of t is denoted $\mathsf{rt}(t)$ and the size of t, that is

the number of symbols in t, is denoted $|t|$. In the following terms are denoted by s, t, u, v, ..., possibly extended by subscripts. Furthermore, we use v (possibly extended by subscripts) to denote values.

A *substitution* σ is a mapping from variables to terms that respects types. Substitutions are denoted as sets of assignments: $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. We write $\mathsf{dom}(\sigma)$ ($\mathsf{rg}(\sigma)$) to denote the domain (range) of σ. Let σ be a substitution and V be a set of variables; $\sigma \restriction V$ denotes the restriction of the domain of σ to V. The substitution τ is called an *extension* of substitution σ if $\tau \restriction \mathsf{dom}(\sigma) = \sigma$. Let σ, τ be substitutions such that $\mathsf{dom}(\sigma) \cap \mathsf{dom}(\tau) = \varnothing$. Then we denote the (disjoint) union of σ and τ as $\sigma \uplus \tau$. We call a substitution σ *normalised* if all terms in the range of σ are values. In the following all considered substitutions will be normalised.

A *typing context* is a mapping from variables $\mathcal{V}$ to types. Type contexts are denoted by upper-case Greek letters. Let Γ be a context and let t be a term. The typing relation $\Gamma \vdash t\colon A$ expresses that based on context Γ, t has type A (with respect to the signature $\mathcal{F}$). The typing rules that define the typing relation are given in Figure 2, where we forget the annotations. In the sequel we sometimes make use of an abbreviated notation for sequences of terms $\boldsymbol{t} := t_1, \ldots, t_n$.

A *typed rewrite rule* is a pair $l \to r$ of terms, such that (i) the types of l and r coincide, (ii) $\mathsf{rt}(l) \in \mathcal{D}$, and (iii) $\mathcal{V}\mathsf{ar}(l) \supseteq \mathcal{V}\mathsf{ar}(r)$. An S-typed *term rewrite system* (*TRS* for short) over the signature $\mathcal{F}$ is a finite set of typed rewrite rules. We define the *innermost rewrite relation* $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ for typed TRSs $\mathcal{R}$. For well-typed terms s and t, $s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t$ holds, if there exists a context C, a normalised substitution σ and a rewrite rule $l \to r \in \mathcal{R}$ such that $s = C[l\sigma]$ and $t = C[r\sigma]$. In the sequel we are only concerned with *innermost* rewriting. A TRS is *orthogonal* if it is left-linear and non-overlapping [10,18]. A TRS is *completely defined* if all ground normal-forms are values. These notions naturally extend to typed TRS. In particular note that an orthogonal typed TRS is confluent. Let s and t be terms, such that t is in normal-form. Then an *(innermost) derivation* $D\colon s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ with respect to a TRS $\mathcal{R}$ is a finite sequence of rewrite steps. The *derivation height* of a term s with respect to a well-founded, finitely branching relation $\to$ is defined as: $\mathsf{dh}(s, \to) = \max\{n \mid \exists t\ s \to^n t\}$. A term $t = f(t_1, \ldots, t_k)$ is called *basic* if f is defined, and all $t_i \in \mathcal{T}(\mathcal{C}, \mathcal{V})$.

Definition 2. *We define the* runtime complexity *(with respect to $\mathcal{R}$):* $\mathsf{rc}_{\mathcal{R}}(n) := \max\{\mathsf{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}}) \mid t \text{ is basic and } |t| \leqslant n\}$.

We study *typed constructor* TRSs $\mathcal{R}$, that is, for each rule $f(l_1, \ldots, l_n) \to r$ we have that the arguments l_i are constructor terms. Furthermore, we restrict to *completely defined* and *orthogonal* systems. These restrictions are natural in the context of functional programming. If no confusion can arise from this, we simply call $\mathcal{R}$ a TRS. $\mathcal{F}$ denotes the signature underlying $\mathcal{R}$. In the sequel, $\mathcal{R}$ and $\mathcal{F}$ are kept fixed.

Example 3 (continued from Example 1). Consider the TRS $\mathcal{R}_{\mathsf{que}}$ and let $S = \{\mathsf{Nat}, \mathsf{List}, \mathsf{Q}\}$, where Nat, List, and Q represent the type of natural numbers, lists over natural numbers, and queues respectively. Then $\mathcal{R}_{\mathsf{que}}$ is an S-typed

$$\frac{x\sigma = v}{\sigma \mid\!\!\overset{0}{\vdash} x \Rightarrow v} \qquad \frac{c \in \mathcal{C} \quad x_1\sigma = v_1 \quad \cdots \quad x_n\sigma = v_n}{\sigma \mid\!\!\overset{0}{\vdash} c(x_1, \ldots, x_n) \Rightarrow c(v_1, \ldots, v_n)}$$

$$\frac{f(l_1, \ldots, l_n) \to r \in \mathcal{R} \quad \exists \tau\ \forall i \colon x_i\sigma = l_i\tau \quad \sigma \uplus \tau \mid\!\!\overset{m}{\vdash} r \Rightarrow v}{\sigma \mid\!\!\overset{m+1}{\vdash} f(x_1, \ldots, x_n) \Rightarrow v}$$

all x_i are fresh

$$\frac{\sigma \uplus \rho \mid\!\!\overset{m_0}{\vdash} f(x_1, \ldots, x_n) \Rightarrow v \quad \sigma \mid\!\!\overset{m_1}{\vdash} t_1 \Rightarrow v_1 \quad \cdots \quad \sigma \mid\!\!\overset{m_n}{\vdash} t_n \Rightarrow v_n \quad m = \sum_{i=0}^{n} m_i}{\sigma \mid\!\!\overset{m}{\vdash} f(t_1, \ldots, t_n) \Rightarrow v}$$

Here $\rho := \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$. Recall that σ, τ, and ρ are normalised.

Fig. 1. Operational Big-Step Semantics

TRSs over signature $\mathcal{F}$. We exemplify the signature of some constructors: $\mathsf{0}\colon \mathsf{Nat}$, $\mathsf{s}\colon [\mathsf{Nat}] \to \mathsf{Nat}$, $\mathsf{nil}\colon \mathsf{List}$, $\sharp\colon [\mathsf{Nat} \times \mathsf{List}] \to \mathsf{List}$, $\mathsf{que}\colon [\mathsf{List} \times \mathsf{List}] \to \mathsf{Q}$. Finally, consider $\mathsf{snoc}\colon [\mathsf{Q} \times \mathsf{Nat}] \to \mathsf{Q}$.

As $\mathcal{R}$ is completely defined, any derivation ends in a value. In connection with innermost rewriting this yields a *call-by-value* strategy. Furthermore, as $\mathcal{R}$ is non-overlapping any innermost derivation is determined modulo the order in which parallel redexes are contracted. This allows us to recast innermost rewriting into an operational big-step semantics instrumented with resource counters, cf. Figure 1. Its definition is instrumental in the proof of our first soundness theorem. The semantics resembles similar definitions given in the literature on amortised resource analysis (see for example [14, 19, 20]).

Proposition 4. *Let f be a defined function symbol of arity n and σ a substitution. Then $\sigma \mid\!\!\overset{m}{\vdash} f(x_1, \ldots, x_n) \Rightarrow v$ holds iff $\mathsf{dh}(f(x_1\sigma, \ldots, x_n\sigma), \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = m$.*

Proof. In the proof of the direction form left to right, we show the stronger statement that $\sigma \mid\!\!\overset{m}{\vdash} t \Rightarrow v$ implies $\mathsf{dh}(t\sigma, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = m$ by induction on the derivation of $\sigma \mid\!\!\overset{m}{\vdash} t \Rightarrow v$. For the opposite direction, we show that if $\mathsf{dh}(t\sigma, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = m$, then $\sigma \mid\!\!\overset{m}{\vdash} t \Rightarrow v$ by induction on the length of the derivation $D\colon t\sigma \xrightarrow{\mathrm{i}}{}^{*}_{\mathcal{R}} v$. □

3 Annotated Types

Let S be a set of types. We call a type $A \in S$ *annotated*, if A is decorated with resource annotations. These annotations will allow us to read off the potential of a well-typed term t from the annotations.

Definition 5. *An* annotated type $A^{\boldsymbol{p}}$*, is a pair consisting of a type $A \in S$ and a vector $\boldsymbol{p} = (p_1, \ldots, p_k)$ over non-negative rational numbers, typically natural numbers. The vector $\boldsymbol{p}$ is called* resource annotation*.*

Resource annotations are denoted by $\boldsymbol{p}$, $\boldsymbol{q}$, $\boldsymbol{u}$, $\boldsymbol{v}$, ..., possibly extended by subscripts and we write $\mathcal{A}$ for the set of such annotations. For resource annotations (p) of length 1 we write p. We will see that a resource annotation does not change its meaning if zeroes are appended at the end, so, conceptually, we can identify () with (0) (and also with 0). If $\boldsymbol{p} = (p_1, \ldots, p_k)$ we write $k = |\boldsymbol{p}|$ and $\max \boldsymbol{p} = \max_i p_i$. We define the notations $\boldsymbol{p} \leqslant \boldsymbol{q}$ and $\boldsymbol{p} + \boldsymbol{q}$ and $\lambda \boldsymbol{p}$ for $\lambda \geqslant 0$ component-wise, filling up with 0s if needed. So, for example $(1,2) \leqslant (3,4,5)$ and $(1,2) + (3,4,5) = (4,6,5)$.

Furthermore, we recall the additive shift [14] given by $\triangleleft(p_1, \ldots, p_k) = (p_1 + p_2, p_2 + p_3, \ldots, p_{k-1} + p_k, p_k)$. We also define the interleaving $\boldsymbol{p} \| \boldsymbol{q}$ by $(p_1, q_1, p_2, q_2, \ldots, p_k, q_k)$ where, as before the shorter of the two vectors is padded with 0s. Finally, we use the notation $\Diamond \boldsymbol{p} = p_1$ for the first entry of an annotation vector. If no confusion can arise, we refer to annotated types simply as types. In contrast to Hoffmann et al. [14, 21], we generalise the concept of annotated types to arbitrary (data) types. In [14] only list types, in [21] list and tree types have been annotated.

Definition 6. *Let $\mathcal{F}$ be a signature. Suppose $\mathcal{F}(f) = [A_1 \times \cdots \times A_n] \to C$, such that the A_i $(i = 1, \ldots, n)$ and C are types. Consider the annotated types $A_i^{\boldsymbol{u_i}}$ and $A^{\boldsymbol{v}}$. Then an* annotated type declaration *for f is a type declaration over annotated types, decorated with $p \in \mathbb{N}$: $[A_1^{\boldsymbol{u_1}} \times \cdots \times A_n^{\boldsymbol{u_n}}] \xrightarrow{p} C^{\boldsymbol{v}}$. The set of annotated type declarations is denoted $\mathcal{F}_{\mathsf{pol}}$.*

We lift signatures to *annotated signatures* $\mathcal{F} \colon \mathcal{C} \cup \mathcal{D} \to (\mathcal{P}(\mathcal{F}_{\mathsf{pol}}) \setminus \varnothing)$ by mapping a function symbol to a non-empty set of annotated type declarations. Hence for any function symbol f we allow multiple types. If f has result type C, then for each annotation $C^{\boldsymbol{q}}$ there should exist exactly one declaration of the form $[A_1^{\boldsymbol{p}_1} \times \cdots \times A_n^{\boldsymbol{p}_n}] \xrightarrow{p} C^{\boldsymbol{q}}$ in $\mathcal{F}(c)$. Moreover, constructor annotations are to satisfy the *superposition principle*: If a constructor c admits the annotations $[A_1^{\boldsymbol{p_1}} \times \cdots \times A_n^{\boldsymbol{p_n}}] \xrightarrow{p} C^{\boldsymbol{q}}$ and $[A_1^{\boldsymbol{p'_1}} \times \cdots \times A_n^{\boldsymbol{p'_n}}] \xrightarrow{p'} C^{\boldsymbol{q'}}$ then it also has the annotations $[A_1^{\lambda \boldsymbol{p_1}} \times \cdots \times A_n^{\lambda \boldsymbol{p_n}}] \xrightarrow{\lambda p} C^{\lambda \boldsymbol{q}}$ $(\lambda \geqslant 0)$ and $[A_1^{\boldsymbol{p_1} + \boldsymbol{p'_1}} \times \cdots \times A_n^{\boldsymbol{p_n} + \boldsymbol{p'_n}}] \xrightarrow{p+p'} C^{\boldsymbol{q}+\boldsymbol{q'}}$.

Note that, in view of superposition and uniqueness, the annotations of a given constructor are uniquely determined once we fix the annotated types for result annotations of the form $(0, \ldots, 0, 1)$ (remember the implicit filling up with 0s). An annotated signature $\mathcal{F}$ is simply called signature, where we sometimes write $f \colon [A_1 \times \cdots \times A_n] \xrightarrow{p} C$ instead of $[A_1 \times \cdots \times A_n] \xrightarrow{p} C \in \mathcal{F}(f)$. Note that the A_i $(i = 1, \ldots, n)$ and C denote *annotated* types.

Example 7 (continued from Example 3). In order to extend $\mathcal{F}$ to an annotated signature we can set $\mathcal{F}(\mathsf{0}) := \{[] \xrightarrow{0} \mathsf{Nat}^{\boldsymbol{p}} \mid \boldsymbol{p} \in \mathcal{A}\}$ and $\mathcal{F}(\mathsf{s}) := \{[\mathsf{Nat}^{\triangleleft(\boldsymbol{p})}] \xrightarrow{\Diamond \boldsymbol{p}} \mathsf{Nat}^{\boldsymbol{p}} \mid \boldsymbol{p} \in \mathcal{A}\}$. Furthermore, we set $\mathcal{F}(\mathsf{nil}) := \{[] \xrightarrow{0} \mathsf{List}^{\boldsymbol{p}} \mid \boldsymbol{p} \in \mathcal{A}\}$ and $\mathcal{F}(\sharp) := \{[\mathsf{Nat}^0 \times \mathsf{List}^{\triangleleft(\boldsymbol{p})}] \xrightarrow{\Diamond \boldsymbol{p}} \mathsf{List}^{\boldsymbol{p}} \mid \boldsymbol{p} \in \mathcal{A}\}$ and $\mathcal{F}(\mathsf{que}) := \{[\mathsf{List}^{\boldsymbol{p}} \times \mathsf{List}^{\boldsymbol{q}}] \xrightarrow{0} \mathsf{Q}^{\boldsymbol{p} \| \boldsymbol{q}} \mid \boldsymbol{p}, \boldsymbol{q} \in \mathcal{A}\}$. In particular, we have the typings $\sharp \colon [\mathsf{Nat}^0 \times \mathsf{List}^7] \xrightarrow{7} \mathsf{List}^7$ and $\sharp \colon [\mathsf{Nat}^0 \times \mathsf{List}^{(10,7)}] \xrightarrow{3} \mathsf{List}^{(3,7)}$ and $\mathsf{que} \colon [\mathsf{List}^1 \times \mathsf{List}^3] \xrightarrow{0} \mathsf{Q}^{(1,3)}$.

We omit annotations for the defined symbols and refer to Example 13 for a complete signature with a different annotation for the constructor symbol que.

The next definition introduces the notion of the potential of a value.

Definition 8. *Let $v = c(v_1, \dots, v_n) \in \mathcal{T}(\mathcal{C})$ and let C be an annotated type. The* potential *of v under C, written $\Phi(v\colon C)$, is defined recursively by $\Phi(v\colon C) := p + \Phi(v_1\colon A_1) + \cdots + \Phi(v_n\colon A_n)$ when $[A_1 \times \cdots \times A_n] \xrightarrow{p} C \in \mathcal{F}(c)$.*

Note that by assumption the declaration in $\mathcal{F}(c)$ is unique.

Example 9 (continued from Example 7). It is easy to see that for any term t of type Nat, we have $\Phi(t\colon \mathsf{Nat}^0) = 0$ and $\Phi(t\colon \mathsf{Nat}^\lambda) = \lambda t$.

If l is a list then $\Phi(l\colon \mathsf{List}^{(p,q)}) = p \cdot |l| + q \cdot \binom{|l|}{2}$. where $|l|$ denotes the length of l, that is the number of $\sharp$ in l. More generally, we have $\Phi(l\colon \mathsf{List}^{\boldsymbol{p}}) = \sum_i p_i \binom{|l|}{i}$. Finally, if $\mathsf{que}(l, k)$ has type Q then $\Phi(\mathsf{que}(l, k)\colon \mathsf{Q}^{\boldsymbol{p}\|\|\boldsymbol{q}}) = \Phi(l\colon \mathsf{List}^{\boldsymbol{p}}) + \Phi(k\colon \mathsf{List}^{\boldsymbol{q}})$.

The *sharing relation* $\Upsilon(A^{\boldsymbol{p}} \mid A^{\boldsymbol{p_1}}, A^{\boldsymbol{p_2}})$ holds if $\boldsymbol{p_1} + \boldsymbol{p_2} = \boldsymbol{p}$. The *subtype relation* is defined as follows: $A^{\boldsymbol{p}} <\colon B^{\boldsymbol{q}}$, if $A = B$ and $\boldsymbol{p} \geqslant \boldsymbol{q}$.

Lemma 10. *If $\Upsilon(A^{\boldsymbol{p}} \mid A^{\boldsymbol{p_1}}, A^{\boldsymbol{p_2}})$ then $\Phi(v\colon A^{\boldsymbol{p}}) = \Phi(v\colon A^{\boldsymbol{p_1}}) + \Phi(v\colon A^{\boldsymbol{p_2}})$ holds for any value of type A. If $A^{\boldsymbol{p}} <\colon B^{\boldsymbol{q}}$ then $\Phi(v\colon A^{\boldsymbol{p}}) \geqslant \Phi(v\colon B^{\boldsymbol{q}})$ again for any $v\colon A$.*

Proof. The proof of the first claim is by induction on the structure of v. We note that by superposition together with uniqueness the additivity property propagates to the argument types. For example, if we have the annotations $\mathsf{s} : [\mathsf{Nat}^2] \xrightarrow{4} \mathsf{Nat}^3$ and $\mathsf{s} : [\mathsf{Nat}^4] \xrightarrow{6} \mathsf{Nat}^5$ and $\mathsf{s} : [\mathsf{Nat}^x] \xrightarrow{10} \mathsf{Nat}^y$ then we can conclude $x = 6$, $y = 8$, for this annotation must be present by superposition and there can only be one by uniqueness.

The second claim follows from the first one and nonnegativity of potentials. □

The set of typing rules for TRSs $\mathcal{R}$ are given in Figure 2. Observe that the type system employs the assumption that $\mathcal{R}$ is left-linear. In a nutshell, the method works as follows: Let Γ be a typing context and let us consider the typing judgement $\Gamma \stackrel{p}{\vdash} t\colon A$ derivable from the type rules. Then p is an upper-bound to the amortised cost required for reducing t to a value. The derivation height of $t\sigma$ (with respect to innermost rewriting) is bound by the difference in the potential before and after the evaluation plus p. Thus if the sum of the potential of the arguments of $t\sigma$ is in $\mathsf{O}(n^k)$, where n is the size of the arguments, then the runtime complexity of $\mathcal{R}$ lies in $\mathsf{O}(n^k)$.

Recall that any rewrite rule $l \to r \in \mathcal{R}$ can be written as $f(l_1, \dots, l_n) \to r$ with $l_i \in \mathcal{T}(\mathcal{C}, \mathcal{V})$. We introduce *well-typed* TRSs.

Definition 11. *Let $f(l_1, \dots, l_n) \to r$ be a rewrite rule in $\mathcal{R}$ and let $\mathcal{V}\mathsf{ar}(f(\boldsymbol{l})) = \{y_1, \dots, y_\ell\}$. Then $f \in \mathcal{D}$ is* well-typed *wrt. $\mathcal{F}$, if we obtain*

$$y_1\colon B_1, \dots, y_\ell\colon B_\ell \stackrel{p-1+\sum_{i=1}^n k_i}{\vdash} r\colon C \;, \tag{1}$$

$$\frac{f \in \mathcal{C} \cup \mathcal{D} \quad [A_1 \times \cdots \times A_n] \xrightarrow{p} C \in \mathcal{F}(f)}{x_1\colon A_1, \ldots, x_n\colon A_n \mid\!\!\frac{p}{\vdash} f(x_1, \ldots, x_n)\colon C} \qquad \frac{\Gamma \mid\!\!\frac{p}{\vdash} t\colon C \quad p' \geqslant p}{\Gamma \mid\!\!\frac{p'}{\vdash} t\colon C}$$

$$\frac{\begin{array}{l}\text{all } x_i \text{ are fresh} \qquad\qquad p = \sum_{i=0}^{n} p_i \\ x_1\colon A_1, \ldots, x_n\colon A_n \mid\!\!\frac{p_0}{\vdash} f(x_1, \ldots, x_n)\colon C \quad \Gamma_1 \mid\!\!\frac{p_1}{\vdash} t_1\colon A_1 \;\cdots\; \Gamma_n \mid\!\!\frac{p_n}{\vdash} t_n\colon A_n\end{array}}{\Gamma_1, \ldots, \Gamma_n \mid\!\!\frac{p}{\vdash} f(t_1, \ldots, t_n)\colon C}$$

$$\frac{\Gamma \mid\!\!\frac{p}{\vdash} t\colon C}{\Gamma, x\colon A \mid\!\!\frac{p}{\vdash} t\colon C} \qquad \frac{\Gamma, x\colon A_1, y\colon A_2 \mid\!\!\frac{p}{\vdash} t[x, y]\colon C \quad \curlyvee(A \,|\, A_1, A_2) \quad x,\, y \text{ are fresh}}{\Gamma, z\colon A \mid\!\!\frac{p}{\vdash} t[z, z]\colon C}$$

$$\frac{\Gamma, x\colon B \mid\!\!\frac{p}{\vdash} t\colon C \quad A <\colon B}{\Gamma, x\colon A \mid\!\!\frac{p}{\vdash} t\colon C} \qquad \frac{}{x\colon A \mid\!\!\frac{0}{\vdash} x\colon A} \qquad \frac{\Gamma \mid\!\!\frac{p}{\vdash} t\colon D \quad D <\colon C}{\Gamma \mid\!\!\frac{p}{\vdash} t\colon C}$$

Fig. 2. Type System for Rewrite Systems

for all $[A_1 \times \cdots \times A_n] \xrightarrow{p} C \in \mathcal{F}(f)$*, for all types* B_j *(*$j \in \{1, \ldots, \ell\}$*), and all costs* k_i*, such that* $y_1\colon B_1, \ldots, y_\ell\colon B_\ell \mid\!\!\frac{k_i}{\vdash} l_i\colon A_i$ *is derivable. A TRS* $\mathcal{R}$ *over* $\mathcal{F}$ *is* well-typed *if any defined* f *is well-typed.*

Let Γ be a typing context and let σ be a substitution. We call σ *well-typed (with respect to* Γ*)* if for all $x \in \mathsf{dom}(\Gamma)$, $x\sigma$ is of type $\Gamma(x)$. We extend the definition of potential to substitutions σ and typing contexts Γ. Suppose σ is well-typed with respect to Γ. Then $\Phi(\sigma\colon\Gamma) := \sum_{x \in \mathsf{dom}(\Gamma)} \Phi(x\sigma\colon\Gamma(x))$. We establish our first soundness result.

Theorem 12. *Let* $\mathcal{R}$ *and* σ *be well-typed. Suppose* $\Gamma \mid\!\!\frac{p}{\vdash} t\colon A$ *and* $\sigma \mid\!\!\frac{m}{\vdash} t \Rightarrow v$*. Then* $\Phi(\sigma\colon\Gamma) - \Phi(v\colon A) + p \geqslant m$*.*

Proof. Let Π be the proof deriving $\sigma \mid\!\!\frac{m}{\vdash} t \Rightarrow v$ and let Ξ be the proof of $\Gamma \mid\!\!\frac{p}{\vdash} t\colon A$. The proof of the theorem proceeds by main-induction on the length of Π and by side-induction on the length of Ξ.

We exemplify the pattern of the proof on one case. We employ the notation from Figure 1. Suppose the last rule in Π has the form

$$\frac{\sigma \uplus \rho \mid\!\!\frac{m_0}{\vdash} f(x_1, \ldots, x_n) \Rightarrow v \quad \sigma \mid\!\!\frac{m_1}{\vdash} t_1 \Rightarrow v_1 \quad \cdots \quad \sigma \mid\!\!\frac{m_n}{\vdash} t_n \Rightarrow v_n}{\sigma \mid\!\!\frac{m}{\vdash} f(t_1, \ldots, t_n) \Rightarrow v},$$

where $m = \sum_{i=0}^{n} m_i$. W.l.o.g. we can assume that t is linear. Otherwise we would consider the case, where Ξ ends with the type rule for sharing. Thus, we assume the last rule in the type inference Ξ is of the following form.

$$\frac{\overbrace{x_1\colon A_1, \ldots, x_n\colon A_n}^{=:\Delta} \mid\!\!\frac{p_0}{\vdash} f(\boldsymbol{x})\colon C \quad \Gamma_1 \mid\!\!\frac{p_1}{\vdash} t_1\colon A_1 \quad \cdots \quad \Gamma_n \mid\!\!\frac{p_n}{\vdash} t_n\colon A_n}{\Gamma_1, \ldots, \Gamma_n \mid\!\!\frac{p}{\vdash} f(t_1, \ldots, t_n)\colon C},$$

such that $p = \sum_{i=0}^{n} p_i$. By induction hypothesis: $\Phi(\sigma\colon \Gamma_i) - \Phi(v_i\colon A_i) + p_i \geqslant m_i$ for all $i = 1, \ldots, n$. Hence (i) $\sum_{i=1}^{n} \Phi(\sigma\colon \Gamma_i) - \sum_{i=1}^{n} \Phi(v_i\colon A_i) + \sum_{i=1}^{n} p_i \geqslant \sum_{i=1}^{n} m_i$ Again by induction hypothesis we obtain: (ii) $\Phi(\sigma \uplus \rho\colon \Delta) - \Phi(v\colon C) + p_0 \geqslant m_0$ Now $\Phi(\sigma\colon \Gamma) = \sum_{i=1}^{n} \Phi(\sigma\colon \Gamma_i)$ and $\Phi(\sigma \uplus \rho\colon \Delta) = \Phi(\rho\colon \Delta) = \sum_{i=1}^{n} \Phi(v_i\colon A_i)$. By (i) and (ii), we obtain

$$\begin{aligned}\Phi(\sigma\colon \Gamma) + \sum_{i=0}^{n} p_i &= \sum_{i=1}^{n} \Phi(\sigma\colon \Gamma_i) + \sum_{i=1}^{n} p_i + p_0 \\ &\geqslant \sum_{i=1}^{n} \Phi(v_i\colon A_i) + \sum_{i=1}^{n} m_i + p_0 \geqslant \Phi(v\colon C) + \sum_{i=0}^{n} m_i \ ,\end{aligned}$$

and thus $\Phi(\sigma\colon \Gamma) - \Phi(v\colon C) + p \geqslant m$. □

Example 13 (continued from Example 1). We extend our example signature with annotated typings for the defined functions as follows.

$$\begin{array}{lll} \mathsf{chk}\colon [\mathsf{Q}^{(0,1)}] \xrightarrow{3} \mathsf{Q}^{(0,1)} & \mathsf{tl}\colon [\mathsf{Q}^{(0,1)}] \xrightarrow{4} \mathsf{Q}^{(0,1)} & \mathsf{hd}\colon [\mathsf{Q}^{(0,1)}] \xrightarrow{1} \mathsf{Nat}^0 \\ \mathsf{rev}'\colon [\mathsf{List}^1 \times \mathsf{List}^0] \xrightarrow{1} \mathsf{List}^0 & \mathsf{rev}\colon [\mathsf{List}^1] \xrightarrow{2} \mathsf{List}^0 & \\ \mathsf{snoc}\colon [\mathsf{Q}^{(0,1)} \times \mathsf{Nat}^0] \xrightarrow{5} \mathsf{Q}^{(0,1)} & \mathsf{enq}\colon [\mathsf{Nat}^6] \xrightarrow{1} \mathsf{Q}^{(0,1)} \ , & \end{array}$$

where the annotations of the constructors are as in Example 7, with the exception of $\mathsf{que}\colon [\mathsf{List}^0 \times \mathsf{List}^1] \xrightarrow{0} \mathsf{Q}^{(0,1)}$. It is not difficult to verify that $\mathcal{R}_{\mathsf{que}}$ is well-typed wrt. $\mathcal{F}$. We show in detail that enq is well-typed. Consider rule 6. First, we observe that 6 resource units become available for the recursive call, as $n\colon \mathsf{Nat}^6 \mid^{6}\!\vdash \mathsf{s}(n)\colon \mathsf{Nat}^6$ is derivable. Second, we have the following partial type derivation; missing parts are easy to fill in.

$$\dfrac{\dfrac{q\colon \mathsf{Q}^{(0,1)}, m\colon \mathsf{Nat}^0 \mid^{5}\!\vdash \mathsf{snoc}(q,m)\colon \mathsf{Q}^{(0,1)} \qquad n_1\colon \mathsf{Nat}^6 \mid^{1}\!\vdash \mathsf{enq}(n_1)\colon \mathsf{Q}^{(0,1)}}{n_1\colon \mathsf{Nat}^6, n_2\colon \mathsf{Nat}^0 \mid^{6}\!\vdash \mathsf{snoc}(\mathsf{enq}(n_1), n_2)\colon \mathsf{Q}^{(0,1)}} \qquad n_2\colon \mathsf{Nat}^0 \mid^{0}\!\vdash n_2\colon \mathsf{Nat}^0}{n\colon \mathsf{Nat}^6 \mid^{6}\!\vdash \mathsf{snoc}(\mathsf{enq}(n), n)\colon \mathsf{Q}^{(0,1)}}$$

Considering rule 7, we see that $n\colon \mathsf{Nat}^6 \mid^{0}\!\vdash \mathsf{que}(\mathsf{nil}, \mathsf{nil})\colon \mathsf{Q}^{(0,1)}$ is derivable. Thus enq is well-typed and we conclude optimal linear runtime complexity of $\mathcal{R}_{\mathsf{que}}$.

Polynomial bounds Note that if the type annotations are chosen such that for each type A we have $\Phi(v\colon A) \in \mathsf{O}(n^k)$ for $n = |v|$ then $\mathsf{rc}_{\mathcal{R}}(n) \in \mathsf{O}(n^k)$ as well. The following proposition gives a sufficient condition as to when this is the case and in particular subsumes the type system in [14].

Theorem 14. *Suppose that for each constructor c with $[A_1^{\boldsymbol{u_1}} \times \cdots \times A_n^{\boldsymbol{u_n}}] \xrightarrow{p} C^{\boldsymbol{w}} \in \mathcal{F}(c)$, there exists $\boldsymbol{r_i} \in \mathcal{A}$ such that $\boldsymbol{u_i} \leqslant \boldsymbol{w} + \boldsymbol{r_i}$ where $\max \boldsymbol{r}_i \leqslant \max \boldsymbol{w} =: r$ and $p \leqslant r$ with $|\boldsymbol{r_i}| < |\boldsymbol{w}| =: k$. Then $\Phi(v\colon C^{\boldsymbol{w}}) \leqslant r|v|^k$.*

$$\frac{x\sigma = v}{\vdash^0 \langle x, \sigma\rangle \to \langle v, \sigma\rangle} \qquad \frac{c \in \mathcal{C} \quad x_1\sigma = v_1 \quad \cdots \quad x_n\sigma = v_n}{\vdash^0 \langle c(x_1, \ldots, x_n), \sigma\rangle \to \langle c(v_1, \ldots, v_n), \sigma\rangle}$$

$$\frac{\forall i\colon v_i \text{ is a value} \quad \rho = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\} \quad f \text{ is defined and all } x_i \text{ are fresh}}{\vdash^0 \langle f(v_1, \ldots, v_n), \sigma\rangle \to \langle f(x_1, \ldots, x_n), \sigma \uplus \rho\rangle}$$

$$\frac{f(l_1, \ldots, l_n) \to r \in \mathcal{R} \quad \forall i\colon x_i\sigma = l_i\tau}{\vdash^1 \langle f(x_1, \ldots, x_n), \sigma\rangle \to \langle r, \sigma \uplus \tau\rangle} \qquad \frac{\vdash^1 \langle t_i, \sigma\rangle \to \langle u, \sigma'\rangle}{\vdash^1 \langle f(\ldots, t_i, \ldots), \sigma\rangle \to \langle f(\ldots, u, \ldots), \sigma'\rangle}$$

Note that the substitutions σ, σ', τ, and ρ are normalised.

Fig. 3. Operational Small-Step Semantics

Proof. The proof is by induction on the size of v. Note that, if $k = 0$ then $\Phi(v\colon C^{\boldsymbol{w}}) = 0$. Otherwise, we have $\Phi(c(v_1, \ldots, v_n)\colon C^{\boldsymbol{w}}) \leqslant r + \Phi(v_1\colon A_1^{\boldsymbol{w}+\boldsymbol{r_1}}) + \cdots + \Phi(v_n\colon A_n^{\boldsymbol{w}+\boldsymbol{r_n}}) \leqslant r(1 + |v_1|^k + |v_1|^{k-1} + \cdots + |v_n|^k + |v_n|^{k-1})$ by application of the induction hypothesis in conjunction with Lemma 10. The latter quantity can be bounded by $r(1 + |v_1| + \cdots + |v_n|)^k = r|v|^k$ due to the multinomial theorem. □

We note that our running example satisfies the premise to the proposition. In concrete cases more precise bounds than those given by Theorem 14 can be computed as has been done in Example 9. The next example clarifies that potentials are not restricted to polynomials.

Example 15. Consider that we annotate the constructors for natural numbers as $\mathsf{0}\colon [] \xrightarrow{0} \mathsf{Nat}^{\boldsymbol{p}}$ and $\mathsf{s}\colon [\mathsf{Nat}^{2\boldsymbol{p}}] \xrightarrow{\Diamond \boldsymbol{p}} \mathsf{Nat}^{\boldsymbol{p}}$. We then have, for example, $\Phi(t\colon \mathsf{Nat}^1) = 2^{t+1} - 1$.

As mentioned in the introduction, foundational issues are our main concern. However, the potential-based method detailed above are susceptible to automation. One conceives the resource annotations as variables and encodes the constraints of the typing rules in Figure 2 over these resource variables.

4 Small-Step Semantics

The big-step semantics, the type system, and Theorem 12 provide a potential-based resource analysis for typed TRSs that yields polynomial bounds. However, Theorem 12 is not directly applicable, if we want to link this analysis to the interpretation method. We recast the method and present a small-step semantics, used in our second soundness result (Theorem 20 below), cf. Figure 3. As the big-step semantics, the small-step semantics is decorated with counters for the derivation height of the evaluated terms. Its definition is instrumental in the proof of our second soundness theorem.

The transitive closure of the judgement $\vdash^m \langle s, \sigma\rangle \to \langle t, \tau\rangle$ is defined as follows:

– $\left|\!\!\frac{m}{}\right. \langle s,\sigma\rangle \twoheadrightarrow \langle t,\tau\rangle$ if $\left|\!\!\frac{m}{}\right. \langle s,\sigma\rangle \to \langle t,\tau\rangle$ $(m \in \{0,1\})$
– $\left|\!\!\frac{m_1+m_2}{}\right. \langle s,\sigma\rangle \twoheadrightarrow \langle u,\rho\rangle$ if $\left|\!\!\frac{m_1}{}\right. \langle s,\sigma\rangle \to \langle t,\tau\rangle$ and $\left|\!\!\frac{m_2}{}\right. \langle t,\tau\rangle \twoheadrightarrow \langle u,\rho\rangle$.

The next lemma proves the equivalence of big-step and small-step semantics.

Lemma 16. *Let σ be a substitution, let t be a term, $\mathcal{V}\mathsf{ar}(t) \subseteq \mathsf{dom}(\sigma)$, and let v be a value. Then $\sigma \left|\!\!\frac{m}{}\right. t \Rightarrow v$ if and only if $\left|\!\!\frac{m}{}\right. \langle t,\sigma\rangle \twoheadrightarrow \langle v,\sigma'\rangle$, where σ' is an extension of σ.*

Proof. Let Π be the proof deriving $\sigma \left|\!\!\frac{m}{}\right. t \Rightarrow v$ and let D denote the sequence of reductions that make up $\left|\!\!\frac{m}{}\right. \langle t,\sigma\rangle \twoheadrightarrow \langle v,\sigma'\rangle$.

One proves the direction left-to-right by induction on the length of Π. We observe that if $\sigma \left|\!\!\frac{m}{}\right. t \Rightarrow v$ and if σ' is an extension of σ, then $\sigma' \left|\!\!\frac{m}{}\right. t \Rightarrow v$. Furthermore the sizes of the derivations of the corresponding judgements are the same. This follows by straightforward inductive argument.

In proof of the lemma, we consider one, significant case. We employ the notation from Figure 3. Suppose the last rule in Π has the form

$$\frac{\sigma \uplus \rho \left|\!\!\frac{m_0}{}\right. f(x_1,\ldots,x_n) \Rightarrow v \quad \sigma \left|\!\!\frac{m_1}{}\right. t_1 \Rightarrow v_1 \quad \cdots \quad \sigma \left|\!\!\frac{m_n}{}\right. t_n \Rightarrow v_n}{\sigma \left|\!\!\frac{m}{}\right. f(t_1,\ldots,t_n) \Rightarrow v} \;,$$

where $t = f(t_1,\ldots,t_n)$ and $m = \sum_{i=0}^{n} m_i$. By induction hypothesis we have for all $i = 1,\ldots,n$: $\left|\!\!\frac{m_i}{}\right. \langle t_1,\sigma_{i-1}\rangle \twoheadrightarrow \langle v_1,\sigma_i\rangle$, where we set $\sigma_0 = \sigma$ and note that all σ_i are extensions of σ. From $\left|\!\!\frac{0}{}\right. \langle f(v_1,\ldots,v_n),\sigma_n\rangle \to \langle f(x_1,\ldots,x_n),\sigma_n \uplus \rho\rangle$ we obtain:

$$\left|\!\!\frac{\sum_{i=1}^{n} m_i}{}\right. \langle f(t_1,\ldots,t_n),\sigma\rangle \twoheadrightarrow \langle f(x_1,\ldots,x_n),\sigma_n \uplus \rho\rangle \;.$$

Furthermore, by the above and the induction hypothesis there exists a substitution σ' such that $\left|\!\!\frac{m_0}{}\right. \langle f(x_1,\ldots,x_n),\sigma_n \uplus \rho\rangle \twoheadrightarrow \langle v,\sigma'\rangle$ where σ' extends $\sigma_n \uplus \rho$ (and thus also σ as $\mathsf{dom}(\sigma_n) \cap \mathsf{dom}(\rho) = \varnothing$). From above, we obtain $\left|\!\!\frac{m}{}\right. \langle t,\sigma\rangle \twoheadrightarrow \langle v,\sigma'\rangle$.

The direction from right to left follows by induction on the sum of the size of the proofs of the single-step execution in D. The proof is based on the observation that if $\left|\!\!\frac{m}{}\right. \langle s,\sigma\rangle \to \langle t,\sigma'\rangle$, $m \in \mathbb{N}$, then σ' extends σ and $s\sigma = s\sigma'$. □

We extend the notion of potential (cf. Definition 8) to ground terms. Recall that by assumption the declaration in $\mathcal{F}(f)$ is unique.

Definition 17. *Let $t = f(t_1,\ldots,t_n) \in \mathcal{T}(\mathcal{D} \cup \mathcal{C})$ and let $[A_1 \times \cdots \times A_n] \xrightarrow{p} C \in \mathcal{F}(f)$. Then the* potential *of t is defined as follows: $\Phi(t\colon C) := p + \Phi(t_1\colon A_1) + \cdots + \Phi(t_n\colon A_n)$.*

Example 18 (continued from Example 13). Recall the type of chk. Let $q = \mathsf{que}(f,r)$ be a queue. We obtain $\Phi(\mathsf{chk}(q)\colon \mathsf{Q}^{(0,1)}) = 3 + \Phi(q\colon \mathsf{Q}^{(0,1)}) = 3 + \Phi(f\colon \mathsf{List}^{0}) + \Phi(r\colon \mathsf{List}^{1}) = 3 + |r|$.

Lemma 19. *Let $\mathcal{R}$ and σ be well-typed. Suppose $\Gamma \left|\!\!\frac{p}{}\right. t\colon A$. Then we have $\Phi(\sigma\colon \Gamma) + p \geqslant \Phi(t\sigma\colon A)$.*

Proof. Let Ξ denote the proof of $\Gamma \vdash^{p} t\colon A$. Then the lemma follows by induction on Ξ. □

We obtain our second soundness result.

Theorem 20. *Let $\mathcal{R}$ and σ be well-typed. If $\Gamma \vdash^{p} t\colon A$ and $\vdash^{m} \langle t, \sigma\rangle \twoheadrightarrow \langle u, \sigma'\rangle$, then $\Phi(\sigma\colon\Gamma) - \Phi(u\sigma'\colon A) + p \geqslant m$. Thus if for all ground basic terms t and types A: $\Phi(t\colon A) \in \mathsf{O}(n^k)$, where $n = |t|$, then $\mathsf{rc}_{\mathcal{R}}(n) \in \mathsf{O}(n^k)$.*

Proof. Let D denote the derivation of $\vdash^{m} \langle t, \sigma\rangle \twoheadrightarrow \langle u, \sigma'\rangle$ and let Ξ denote the proof of $\Gamma \vdash^{p} t\colon A$. The proof proceeds by main induction on the size of D and by side induction on the length of Ξ.

We exemplify the pattern of the proof on one case. Let Π denote the proof of the judgement $\vdash^{m_1} \langle t, \sigma\rangle \to \langle w, \gamma\rangle$ where $\vdash^{m_2} \langle w, \gamma\rangle \twoheadrightarrow \langle u, \sigma'\rangle$ and $m = m_1 + m_2$. Suppose Π has the form

$$\frac{f(l_1, \ldots, l_n) \to r \in \mathcal{R} \quad \forall i\colon x_i\sigma = l_i\tau}{\vdash^{1} \langle f(x_1, \ldots, x_n), \sigma\rangle \to \langle r, \sigma \uplus \tau\rangle} \ .$$

Then $t = f(x_1, \ldots, x_n)$ and $f(x_1, \ldots, x_n)\sigma = f(l_1, \ldots, l_n)\tau$. Let $\mathcal{V}\mathsf{ar}(f(\boldsymbol{l})) = \{y_1, \ldots, y_\ell\}$ and let $\mathcal{V}\mathsf{ar}(l_i) = \{y_{i1}, \ldots, y_{il_i}\}$ for $i \in \{1, \ldots, n\}$. As $\mathcal{R}$ is left-linear we have $\mathcal{V}\mathsf{ar}(f(l_1, \ldots, l_n)) = \biguplus_{i=1}^{n} \mathcal{V}\mathsf{ar}(l_i)$. We set $\Gamma = x_1\colon A_1, \ldots, x_n\colon A_n$. By the assumption $\Gamma \vdash^{p} t\colon A$ and well-typedness of $\mathcal{R}$ we obtain

$$\overbrace{y_1\colon B_1, \ldots, y_\ell\colon B_\ell}^{=:\Delta} \vdash^{p-1+\sum_{i=1}^{n} k_i} r\colon A \ , \tag{2}$$

similar to (1). We have $\Phi(\sigma\colon\Gamma) + p = \sum_{i=1}^{n} (k_i + \Phi(y_{i1}\tau\colon B_{i1}) + \cdots + \Phi(y_{il_i}\tau\colon B_{il_i})) + p = \Phi(\tau\colon\Delta) + \sum_{i=1}^{n} k_i + (p-1) + 1$. The first equality follows by an inspection on the cases for the constructors. Furthermore note that $r\tau = r(\sigma \uplus \tau)$, as $\mathsf{dom}(\sigma) \cap \mathsf{dom}(\tau) = \varnothing$. The theorem follows by application of the main induction hypothesis on r in conjunction with the typing judgement (2). □

5 Typed Polynomial Interpretations

We adapt the concept of polynomial interpretation to typed TRSs. For that we suppose a mapping $[\![\cdot]\!]$ that assigns to every *annotated* type C a subset of the natural numbers, whose elements are ordered with $>$ in the standard way. The set $[\![C]\!]$ is called the *interpretation* of C.

Definition 21. *An* interpretation γ of function symbols *is a mapping from function symbols and types to functions over $\mathbb{N}$. Consider a function symbol f and an annotated type C such that $[A_1 \times \cdots \times A_n] \xrightarrow{p} C \in \mathcal{F}(f)$. Then the interpretation $\gamma(f, C)\colon [\![A_1]\!] \times \cdots \times [\![A_n]\!] \to [\![C]\!]$ of f is defined as: $\gamma(f, C)(x_1, \ldots, x_n) := x_1 + \cdots + x_n + p$.*

Note that by assumption the declaration in $\mathcal{F}(f)$ is unique and thus $\gamma(f, C)$ is unique. Interpretations of function symbols naturally extend to interpretations on ground terms. $[\![f(t_1, \ldots, t_n)\colon C]\!]^\gamma := \gamma(f, C)([\![t_1\colon A_1]\!]^\gamma, \ldots, [\![t_n\colon A_n]\!]^\gamma)$. Let $\mathcal{R}$ be well-typed and let the interpretation γ of function symbols in $\mathcal{F}$ be induced by the well-typing of $\mathcal{R}$ as in Definition 21. Then by construction $[\![t\colon A]\!]^\gamma = \Phi(t\colon A)$.

Example 22 (continued from Example 13). We obtain the following definitions of the interpretation of function symbols γ. We start with the constructor symbols.

$$\gamma(\mathsf{0}, \mathsf{Nat}^p) = 0 \qquad \gamma(\mathsf{s}, \mathsf{Nat}^p)(x) = x + p \qquad \gamma(\mathsf{err\_head}, \mathsf{Nat}^p) = 0$$
$$\gamma(\mathsf{nil}, \mathsf{List}^q) = 0 \qquad \gamma(\sharp, \mathsf{List}^q)(x, y) = x + y + q \quad \gamma(\mathsf{err\_tail}, \mathsf{Q}^{(0,1)}) = 0$$
$$\gamma(\mathsf{que}, \mathsf{Q}^{(0,1)})(x, y) = x + y \ ,$$

where $p, q \in \mathbb{N}$. Similarly the definition of γ for defined symbols follows from the signature detailed in Example 13. Then for any rule $l \to r \in \mathcal{R}_{\mathsf{que}}$ and any substitution σ, we obtain $[\![l\sigma\colon C]\!]^\gamma > [\![r\sigma\colon C]\!]^\gamma$. We show this for rule 1.

$$\begin{aligned}[\![\mathsf{chk}(\mathsf{que}(\mathsf{nil}, r\sigma))\colon \mathsf{Q}^{(0,1)}]\!]^\gamma = [\![r\sigma\colon \mathsf{List}^1]\!]^\gamma + 3 &> 0 \\ &= [\![\mathsf{rev}(r\sigma)\colon \mathsf{List}^0]\!]^\gamma + [\![\mathsf{nil}\colon \mathsf{List}^1]\!]^\gamma \\ &= [\![\mathsf{que}(\mathsf{rev}(r\sigma), \mathsf{nil})\colon \mathsf{Q}^{(0,1)}]\!]^\gamma \ .\end{aligned}$$

Orientability of $\mathcal{R}_{\mathsf{que}}$ with the above given interpretation implies the optimal linear innermost runtime complexity.

We lift the standard order $>$ on the interpretation domain $\mathbb{N}$ to an order on terms. Let s and t be terms of type A. Then $s > t$ if for all well-typed substitutions σ we have $[\![s\sigma\colon A]\!]^\gamma > [\![t\sigma\colon A]\!]^\gamma$.

Theorem 23. *Let $\mathcal{R}$ be well-typed TRS over signature $\mathcal{F}$ and let the interpretation of function symbols γ be induced by the type system. Then $l > r$ for any rule $l \to r \in \mathcal{R}$. Thus if for all ground basic terms t and types A: $[\![t\colon A]\!]^\gamma \in \mathsf{O}(n^k)$, where $n = |t|$, then $\mathsf{rc}_{\mathcal{R}}(n) \in \mathsf{O}(n^k)$.*

Proof. Let $l = f(l_1, \ldots, l_n)$ and let $x_1, \ldots, x_n$ be fresh variables. Suppose further $[A_1 \times \cdots \times A_n] \xrightarrow{p} C \in \mathcal{F}(f)$. As $\mathcal{R}$ is well-typed we have

$$\overbrace{x_1\colon A_1, \ldots, x_n\colon A_n}^{=:\Gamma} \vdash^p f(x_1, \ldots, x_n)\colon C \ ,$$

for $p \in \mathbb{N}$. Now suppose that τ denotes any well-typed substitution for the rule $l \to r$. In the standard way, we extend τ to a well-typed substitution σ such that $l\tau = f(x_1, \ldots, x_n)\sigma$. By definition of the small-step semantics, we obtain $\vdash^1 \langle f(x_1, \ldots, x_n), \sigma\rangle \to \langle r, \sigma \uplus \tau\rangle$. Then by Theorem 20, $\Phi(\sigma\colon \Gamma) + p > \Phi(r(\sigma \uplus \tau)\colon C)$ and by definitions, we have:

$$\Phi(l\tau\colon C) = \Phi(f(x_1\sigma, \ldots, x_n\sigma)\colon C) = \sum_{i=1}^{n} \Phi(x_i\sigma\colon A_i) + p = \Phi(\sigma\colon \Gamma) + p \ .$$

Furthermore, observe that $r(\sigma \uplus \tau) = r\tau$ as $\mathsf{dom}(\sigma) \cap \mathsf{dom}(\tau) = \varnothing$. In sum, we obtain $\Phi(l\tau\colon C) > \Phi(r\tau\colon C)$, from which we conclude $[\![l\tau\colon C]\!]^\gamma > [\![r\tau\colon C]\!]^\gamma$. As τ was chosen arbitrarily, we obtain $\mathcal{R} \subseteq >$. □

We say that an interpretation *orients* a typed TRS $\mathcal{R}$, if $\mathcal{R} \subseteq >$. As an immediate consequence of the theorem, we obtain the following corollary.

Corollary 24. *Let $\mathcal{R}$ be a well-typed and constructor TRS. Then there exists a typed polynomial interpretation over $\mathbb{N}$ that orients $\mathcal{R}$.*

At the end of Section 3 we have remarked on the automatabilty of the obtained amortised analysis. Observe that Theorem 23 gives rise to a related but conceptually different implementation. Instead of encoding the constraints of the typing rules in Figure 2 one directly encodes the orientability constraints for each rule, cf. [22].

6 Conclusion

This paper is concerned with the connection between amortised resource analysis, originally introduced for functional programs, and polynomial interpretations, which are frequently used in complexity and termination analysis of rewrite systems.

In order to study this connection we have established a novel resource analysis for typed term rewrite systems based on a potential-based type system. This type system gives rise to polynomial bounds for innermost runtime complexity. A key observation is that the classical notion of potential can be altered so that not only values but any term is assigned a potential. Ie. the potential function Φ is conceivable as an interpretation. Based on this observation we have shown that well-typedness of a TRSs $\mathcal{R}$ induces a typed polynomial interpretation that orients $\mathcal{R}$.

Apart from clarifying the connection between amortised resource analysis and polynomial interpretation our results induce two new methods for the innermost runtime complexity of typed TRSs. If we restrict the length of the resource annotations, standard techniques for type inference performed on our type system yield linear constraints that can be solved with a linear constraint solver. On the other hand considering the synthesis of typed polynomial interpretations on fixes abstract polynomials and feeds the obtained constraints into an SMT solver. A prototype is in preparation.

We emphasise that these methods are not restricted to typed TRSs, as our cost model gives rise to a *persistent* property. A property is called persistent if for any typed TRS $\mathcal{R}$ the property holds iff it holds for the corresponding untyped TRS $\mathcal{R}'$. While termination is in general not persistent [18], it is not difficult to see that runtime complexity is persistent. This is due to the restricted set of starting terms. Thus the proposed techniques directly give rise to novel methods of automated runtime complexity analysis. In future work we will clarify whether the established results extend to the multivariate amortised resource analysis presented in [20].

References

1. Albert, E., Arenas, P., Genaim, S., Puebla, G.: Closed-form upper bounds in static cost analysis. JAR **46**(2) (2011)
2. Alias, C., Darte, A., Feautrier, P., Gonnord, L.: Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In: Proc. 17th SAS. Volume 6337 of LNCS. (2010) 117–133
3. Zuleger, F., Gulwani, S., Sinn, M., Veith, H.: Bound analysis of imperative programs with the size-change abstraction. In: Proc. of 18th SAS. Volume 6887 of LNCS. (2011) 280–297
4. Noschinski, L., Emmes, F., Giesl, J.: Analyzing innermost runtime complexity of term rewriting by dependency pairs. JAR **51**(1) (2013) 27–56
5. Hoffmann, J., Aehlig, K., Hofmann, M.: Resource aware ML. In: Proc. 24th CAV. Volume 7358 of LNCS. (2012) 781–786
6. Avanzini, M., Moser, G.: Tyrolean complexity tool: Features and usage. In: Proc. 24th RTA. Volume 21 of LIPIcs. (2013) 71–80
7. Okasaki, C.: Purely functional data structures. Cambridge University Press (1999)
8. Tarjan, R.: Amortized computational complexity. SIAM J. Alg. Disc. Meth **6**(2) (1985) 306–318
9. Turing, A.: Checking a large routine. In: In Report of a Conference on High Speed Automatic Calculating Machines, Cambridge University (1949) 67–69
10. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
11. Bonfante, G., Cichon, A., Marion, J.Y., Touzet, H.: Algorithms with polynomial interpretation termination proof. JFP **11**(1) (2001) 33–53
12. Hofmann, M., Jost, S.: Static prediction of heap space usage for first-order functional programs. In: Proc. 30th POPL, ACM (2003) 185–197
13. Bonfante, G., Marion, J.Y., Moyen, J.Y.: Quasi-interpretations a way to control resources. TCS **412**(25) (2011) 2776–2796
14. Hoffmann, J., Hofmann, M.: Amortized resource analysis with polynomial potential. In: Proc. 19th ESOP. Volume 6012 of LNCS. (2010) 287–306
15. Hoffmann, J., Hofmann, M.: Amortized resource analysis with polymorphic recursion and partial big-step operational semantics. In: Proc. 8th APLAS. Volume 6461 of LNCS. (2010) 172–187
16. Hofmann, M., Moser, G.: Amortised resource analysis and typed polynomial interpretations (extended version). CoRR, cs.LO (2014) Available at `http://arxiv.org/abs/1402.1922`.
17. Jouannaud, J.P., Rubio, A.: The higher-order recursive path ordering. In: Proc. 14th LICS, IEEE Computer Society (1999) 402–411
18. TeReSe: Term Rewriting Systems. Volume 55 of Cambridge Tracks in Theoretical Computer Science. Cambridge University Press (2003)
19. Jost, S., Loidl, H.W., Hammond, K., Scaife, N., Hofmann, M.: "Carbon Credits" for resource-bounded computations using amortised analysis. In: Proc. 2nd FM. Volume 5850 of LNCS., Springer Verlag (2009) 354–369
20. Hoffmann, J., Aehlig, K., Hofmann, M.: Multivariate amortized resource analysis. TOPLAS **34**(3) (2012) 14
21. Hoffmann, J.: Types with Potential: Polynomial Resource Bounds via Automatic Amortized Analysis. PhD thesis, Ludwig-Maximilians-Universiät München (2011)
22. Contejean, E., Marché, C., Tomás, A.P., Urbain, X.: Mechanically proving termination using polynomial interpretations. JAR **34**(4) (2005) 325–363