Fundamenta Informaticae XXI (2001) 1001–1027
DOI 10.3233/FI-2016-0000
IOS Press

Computing with Infinite Terms and Infinite Reductions

Jeroen Ketema*
Department of Computing, Imperial College London
London, United Kingdom

Jakob Grue Simonsen
Department of Computer Science, University of Copenhagen (DIKU)
Copenhagen, Denmark

Abstract. We define computable infinitary rewriting by introducing computability to the study of strongly convergent infinite reductions over infinite first-order terms.

Given computable infinitary reductions, we show that descendants and origins—essential to proving fundamental properties such as compression and confluence—are computable across such reductions.

Keywords: Infinitary term rewriting, computability, descendants, origins, needed reductions

1. Introduction

In term rewriting—where terms and rule sets are finite—issues concerning computability pertain only to advanced properties such as deciding whether a term has a normal form. Most proofs of standard properties in term rewriting are constructive: if a proof states that some object exists (e.g., a normal form, or a common reduct of two reductions starting from the same term), then the object can usually be constructed directly from a number of inputs. In other words, the proof can be converted into a program that outputs the object when given appropriate inputs. As an example of this, consider the

*Jeroen Ketema contributed to this work while at Imperial College London. He is currently at ESI (TNO), the Netherlands.

plethora of proofs collected in [1] showing confluence of orthogonal rewrite systems; each of the proofs gives rise to a program that, on input of a peak $t\ {}^*\!\leftarrow s \rightarrow^* t'$, outputs a valley $t \rightarrow^* s'\ {}^*\!\leftarrow t'$.

In *infinitary* rewriting, in contrast, even basic constructs such as terms and reductions may fail to be computable. Proofs of standard properties are also rarely constructive, e.g., no constructive proofs are known showing confluence of orthogonal, non-collapsing infinitary rewriting systems, although several non-constructive proofs exist.

We believe the above discrepancy between ordinary and infinitary rewriting to be at odds with the original reasons for introducing infinitary rewriting, namely, as an *extension* of ordinary rewriting, and we aim to reduce the discrepancy by introducing computable infinitary rewriting.

Failure of the Usual Proof Methods To obtain a robust definition of computable infinitary rewriting, we need to understand why the usual proof methods employed in infinitary rewriting fail to be constructive. To this end, consider the *de facto* standard form of infinitary rewriting utilising *strong convergence*—a syntactic constraint on the depth of rewrite steps in reductions. A cursory glance at the literature on strong convergence reveals that almost all proofs depend on the ability to know from which point in a reduction onward all steps occur below a certain depth. Unfortunately, as we will show in Section 5, there is demonstrably no constructive way to determine such points.

We circumvent the failure of the usual proof methods by careful analysis of reductions and by requiring information about the convergence of reductions to be explicit through the notion of a *computable modulus of convergence*, a concept well-known from computable analysis [2].

Contributions We provide an account of computability of infinite terms, infinitary term rewriting systems, and strongly convergent transfinite reductions. Our methods allow for reductions of any computable ordinal length (i.e., of any length strictly less than the Church-Kleene ordinal ω_1^{CK}). We give a range of examples of computable infinitary rewriting and computable reductions.

Based on the introduced notion of computable infinitary rewriting, we show that sets of descendants and origins, tracking redexes forward and backward across reductions, respectively, are computable. Descendants and origins are fundamental to proofs of basic properties such as compression and confluence and, hence, showing computability of these notions forms a critical step towards obtaining further constructive results.

We believe all our constructions to be completely non-controversial: they use classical computability theory (including computable ordinals) straightforwardly, and our notion of a computable infinite reduction is, hence, robust. The primary, and non-trivial, technical difficulties consist of proving that the defined notions have desirable properties, i.e., in providing suitably constructive proofs.

2. Related Work

Infinitary rewriting was introduced as an extension of ordinary rewriting that (i) allows for structures that, in theory, are infinite (e.g., lazy data structures such as streams [3] and sentences in infinitary logic [4]), and (ii) allows for infinite reductions over the infinite structures (enabling, e.g., iteration through every element of an infinite list).

Infinitary analogues of many standard results from finitary first-order term rewriting have been shown to hold for strongly convergent infinitary rewriting. Major landmarks include confluence of (almost non-collapsing) orthogonal systems [5], modularity [6, 7], and normalisation [8, 9]. Similar results have been established for higher-order systems [10, 11, 12, 13, 14] and term graph rewriting [15, 16]. Several attempts have also been made at providing abstract foundations [17, 18, 19].

From a computability perspective, our work relates to a line of research that considers rewriting of rational terms (infinite terms with a finite number of syntactically distinct subterms), as rational terms can easily be represented finitely (see Example 4.7), e.g., using μ-terms. Employing a μ-term representation, [20] describes a two-phase approach to rewriting of rational terms: during each rewrite step (i) the representation of the rational term is changed, and (ii) a rewrite rule is applied to a subterm not below a μ. In [21, 22] parallel rewriting of rational terms is discussed, where the set of positions of redexes being rewritten is also rational. None of the aforementioned works explicitly discusses computability. In the setting of rational terms, computability issues seem to have been raised only in [23], which discusses a combination of the two-phase approach and the parallel rewriting approach.

Also related to our work is the line of research that tries to provide formalisations of infinitary rewriting in proof assistants based on constructive logics. An early formalisation is [24] that uses Coq to formalise Kahrs' transfinite reductions satisfying adherence [18] (a much weaker notion than strong convergence, only requiring the terms along a reduction to be eventually always close to the limit of the reduction). The formalisation in [24] stops short of proving any fundamental properties.

In [25], the authors employ a coinductive approach to define both infinite terms and infinite reductions, providing Coq formalisations of some of their constructions, notably a version of the Compression Lemma (relating any reduction of countable ordinal length to an equivalent reduction of length at most ω). The formalisation allows for terms to be defined using coinduction, and infinite reductions to be defined using mixed induction and coinduction. While this approach is general and elegant, and a viable setting for studying computable terms and reductions, establishing precise connections to computability theory remains to be done and may require adding further axioms (e.g., a version of Markov's Principle and the (recursive) countability of the set of partial functions on the natural numbers). We conjecture that doing so is indeed possible, and furthermore that a computable procedure can be extracted from the proof of the Compression Lemma of [25] that would allow computably strongly convergent reductions (in the sense of the present paper) of any countably ordinal length to be converted to computably strongly convergent reductions of length at most ω. However, we believe this to be out of the scope of our present work.

3. Preliminaries

We assume acquaintance with ordinary term rewriting (see, e.g., [26, 27, 28]), and with computability theory, in particular with Turing machines and the partial functions they compute (see, e.g., [29, 30, 31, 32]). We do not presuppose familiarity with *infinitary* rewriting, but a working knowledge of the subject will make the paper easier to read. For background on infinitary rewriting see [33].

Throughout the paper $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ denotes the set of non-negative integers, $\mathbb{N}^k$ (for $k \in \mathbb{N}$) denotes the k-ary Cartesian product over non-negative integers, and $\mathbb{N}^* = \bigcup_{n \in \mathbb{N}} \mathbb{N}^k$ denotes the set of a finite strings over non-negative integers, with ϵ the empty string. We write $p \cdot q$ for the concatenation

of p, $q \in \mathbb{N}^*$, and we denote the length of $p \in \mathbb{N}^*$ by $|p|$. We say that $p \in \mathbb{N}^*$ is a *prefix* of $q \in \mathbb{N}^*$, denoted $p \leq q$, if there exists a $p' \in \mathbb{N}^*$ with $p \cdot p' = q$, and write $p < q$ when $p' \neq \epsilon$. Furthermore, we say that p, $q \in \mathbb{N}^*$ are *parallel*, denoted $p \parallel q$, if neither $p \leq q$ nor $q \leq p$.

Unless stated otherwise, all Turing machines are multi-tape Turing machines with binary input, output, and worktape alphabets. For every $n \in \mathbb{N}$, we denote by $\langle n \rangle$ its standard binary encoding. Given a Turing machine M, we write φ_M for the partial function with domain and range $\mathbb{N}$ *computed* by M, i.e., if M halts on input $\langle n \rangle$ with output $\langle m \rangle$, then $\varphi_M(n) = m$, and $\varphi_M(n)$ is undefined otherwise. Recall that a partial function $\varphi : \mathbb{N} \longrightarrow \mathbb{N}$ is *computable* if φ is computed by some Turing machine M.

We assume a fixed computable bijection from the set of Turing machines to $\mathbb{N}$, and denote the element of $\mathbb{N}$ assigned to a Turing machine M under this bijection by $\langle M \rangle$. We also assume a fixed computable bijective *pairing function* from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$ and a fixed computable bijective function from $\mathbb{N}^*$ to $\mathbb{N}$ (see, e.g., [31, Section 5.6]). Recall that a set $A \subseteq \mathbb{N}$ is *recursively enumerable (r.e.)* if there exists a total computable function $\varphi_M : \mathbb{N} \longrightarrow \mathbb{N}$ with $\varphi_M(\mathbb{N}) = A$. Likewise, the set A is *recursive* if there exists a total computable function $\varphi_M : \mathbb{N} \longrightarrow \{0, 1\}$ such that $\varphi_M(n) = 1$ iff $n \in A$.[1]

Observe that we can use our assumed fixed computable bijective pairing function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$ to naturally extend r.e. sets and recursive sets to subsets of any Cartesian product $\mathbb{N}^k$.

4. Computable Infinite Terms and Substitutions

We begin by defining computable infinite terms and computable substitutions. This task requires all standard concepts from infinitary rewriting to be investigated for computability. We start from the basics with signatures and variables.

Definition 4.1. A *signature* is any set

$$\Sigma = \{(f_0, n_0), (f_1, n_1), \ldots\}$$

with f_i a *function symbol* and $n_i \in \mathbb{N}$ the *arity* of f_i for all $i \in \{0, 1, \ldots\}$, such that each function symbol f_i occurs at most once as the first component of a pair in Σ.

We generally suppress the arities of function symbols in signatures Σ, and simply say that a symbol f occurs in Σ, writing $f \in \Sigma$. If Σ is a signature and the set $\{f_1, f_2, \ldots\}$ is countable, then we may assume that all f_i are coded by their index i and, hence, that $\Sigma \subseteq \mathbb{N} \times \mathbb{N}$. This leads to the following:

Definition 4.2. A signature Σ is *recursively enumerable (r.e.)*, if $\Sigma \subseteq \mathbb{N} \times \mathbb{N}$ and Σ is an r.e. set.

Note that we could have alternatively defined 'computable' signatures as partial computable functions $\varphi_M : \mathbb{N} \longrightarrow \mathbb{N} \times \mathbb{N}$ with recursive domain. We have chosen not to, as this runs counter to the way signatures are normally defined in term rewriting.

[1] In the literature, r.e. is sometimes called *semi-decidable* or *computably enumerable*, and recursive is sometimes called *decidable*, *effective*, or *computable*.

Variables We assume a countable, infinite set of variables $V = \{x_0, x_1, \ldots\}$. By fixing the order of the elements in V and by identifying each variable with its index in this order, we can assume V to be $\mathbb{N}$, but for readability, we write variables as x, y, z, and so forth. By our assumptions on V, we have for any r.e. signature Σ that $\Sigma \cup V$ is r.e., and that it is decidable whether an element $x \in \Sigma \cup V$ originates from either Σ or V (by re-coding the representations of Σ and V if need be).

Computable Infinite Terms There are at least four equivalent ways [34, 25] to define infinite terms: by metric completion, by ideal completion, by employing partial functions, and by coinduction (these all define the same set of infinite terms, but offer different flexibilities [34]). Of these definitions, we find the definition by partial functions most easily amenable to computability, due to the emphasis on partial functions in classical recursion theory. The definition is as follows:

Definition 4.3. The set of (finite and infinite) *terms* over a signature Σ and a set of variables V is the set of all partial functions $t : \mathbb{N}^* \longrightarrow \Sigma \cup V$ such that $t(\epsilon)$ is defined and such that for each $p \in \mathbb{N}^*$:

- if $t(p)$ is defined, then $t(q)$ is defined for all $q < p$;
- if $t(p) \in \Sigma$, then $t(p \cdot i)$ is defined for all $0 \leq i < n$ with n the arity of $t(p)$, and $t(p \cdot i)$ is undefined otherwise;
- if $t(p) \in V$, then $t(q)$ is undefined for all $q > p$.

The domain over which a function (or term) t is defined is denoted by $\mathcal{P}os(t)$ and the elements from the domain are called *positions*. A term t is *finite* iff $\mathcal{P}os(t)$ is finite.

Observe that a term is finite in the sense of the above definition iff it is finite in the usual sense. Moreover, it is clear that every finite term over an *r.e.* signature can be encoded as an element of $\mathbb{N}$ and that the set of finite terms is an r.e. set.

An intuitive way to think about computable infinite terms over an r.e. signature is as follows: a term t is computable if there exists a Turing machine which on input of a position p outputs the function symbol at that position if p is a position of the term, and outputs an error message or a similar value representing 'undefined' otherwise. Formally:

Definition 4.4. The set of *computable terms* over an r.e. signature Σ and a set of variables V is the set of all terms t over Σ and V such that t is computable as a partial function.

An immediate consequence of Definition 4.4 is that every finite term is a computable term. We can also represent every computable term t by a natural number via the encoding $\langle M \rangle$ of the Turing machine M associated with t.

We now give some examples of computable infinite terms:

Example 4.5. Let the function symbols f and h have arities 1 and 2, respectively. The infinite terms represented by the recursion equations $s = f(s)$ and $t = h(t, t)$ (see also Figure 1) are computable. To see this, observe that the sets of positions $\{0\}^* \subseteq \mathbb{N}^*$ and $\{0, 1\}^* \subseteq \mathbb{N}^*$ are recursive and define:

$$s(p) = \begin{cases} f & \text{if } p \in \{0\}^* \\ \bot & \text{otherwise} \end{cases} \qquad t(p) = \begin{cases} h & \text{if } p \in \{0,1\}^* \\ \bot & \text{otherwise} \end{cases}$$

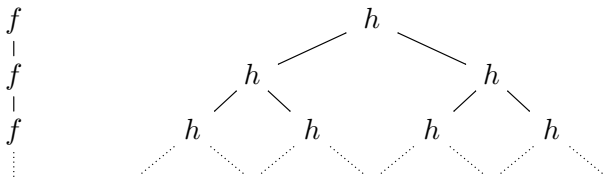

Figure 1. The computable terms from Example 4.5

Example 4.6. Let : ('cons') be a function symbol of arity 2 and let 0 and 1 be function symbols both of arity 0. Writing : in infix notation, we may consider all infinite terms of the form $s_1 : (s_2 : (s_3 : \cdots))$ with $s_1,\ s_2,\ s_3,\ \ldots \in \{0, 1\}$. The set of positions of each of these terms is $\{q, q \cdot 0 \mid q \in \{1\}^*\} \subseteq \mathbb{N}^*$, which is easily seen to be a recursive set.

The term $s = 0 : (0 : (0 : \cdots))$, consisting of all 0s, and the term $t = 1 : (1 : (1 : \cdots))$, consisting of all 1s, are clearly computable by the following functions:

$$s(p) = \begin{cases} 0 & \text{if } p \in \{q \cdot 0 \mid q \in \{1\}^*\} \\ : & \text{if } p \in \{q \mid q \in \{1\}^*\} \\ \bot & \text{otherwise} \end{cases} \qquad t(p) = \begin{cases} 1 & \text{if } p \in \{q \cdot 0 \mid q \in \{1\}^*\} \\ : & \text{if } p \in \{q \mid q \in \{1\}^*\} \\ \bot & \text{otherwise} \end{cases}$$

Example 4.7. Denote by $t|_p$ the subterm of t at position p (see also Definition 4.12) and recall that an infinite term t is *rational* if the set $\{t' \mid \exists p \in \mathcal{P}os(t).t' = t|_p\}$ is finite, i.e., if the set of syntactically distinct subterms of t is finite. Examples of rational terms are the two terms of Example 4.5, as are the terms $(0 : (0 : (0 : \cdots)))$ and $(1 : (1 : (1 : \cdots)))$ of Example 4.6.

It is easy to see that *every* rational term t is computable: recall from, e.g., [35], that rational terms are representable by finite systems of regular equations $S_t = \{(x_0, t_0), (x_1, t_1), \ldots, (x_n, t_n)\}$, with each x_i unique and each t_i a finite term, such that x_0 is the 'start' variable, i.e., a copy of t_0 occurs at the root of the rational term. Given S_t, there is an obvious top-down algorithm that computes the function symbol at each position p: the algorithm only needs to store the elements of S_t in a table and trace the path from the root of t_0 as defined by p. Once a variable is encountered, the algorithm looks up the corresponding term in the table (if it exists) and continues to trace along that term (and so on, recursively).

The existence of *un*computable infinite terms is immediate: given a finite signature Σ with two symbols of arity ≥ 1, there are uncountably many distinct infinite terms, while only countably many distinct partial computable functions exist. The following example exhibits an uncomputable term.

Example 4.8. Let f and g both have arity 1, and consider any uncomputable total function $\varphi : \mathbb{N} \longrightarrow \{0, 1\}$ (e.g., $\varphi(\langle M \rangle) = 1$, respectively $\varphi(\langle M \rangle) = 0$, if, on empty input, the Turing machine M halts, respectively does not halt). Let the domain of t be $\{0\}^* \subseteq \mathbb{N}^*$, and let $t(p) = f$, if $\varphi(|p|) = 0$, and $t(p) = g$, if $\varphi(|p|) = 1$. Although $\mathcal{P}os(t)$ is recursive, it is not possible to compute which symbol occurs at which position (otherwise φ would be computable).

Remark 4.9. A computable infinite term has a natural representation as an infinite tree. However, this tree may fail to have any computable infinite *path*. Consider a signature Σ with h of arity 2 as its sole function symbol and observe that any finite or infinite binary tree may be realised as a (finite or infinite) term over Σ (nodes are occurrences of h, leaves are variables). An infinite *path* of an infinite term t over Σ is any infinite sequence $i_1 \cdot i_2 \cdots$ with $i_j \in \{0, 1\}$ such that the finite sequence $i_1 \cdot i_2 \cdots i_{n-1} \cdot i_n$ is a position of t for each n. An infinite path is *computable* if there exists a computable function $\varphi_M : \mathbb{N} \to \{0, 1\}$ such that $\varphi_M(j) = i_j$ for each j. By the classic construction called the 'Kleene tree' [36, 37], it is now immediate that there exists a computable infinite term t over Σ such that t has an infinite number of positions but no infinite computable paths.

Observe that Remark 4.9 implies that we cannot have a 'computable' König's Lemma. Indeed, existence of the Kleene tree shows that König's lemma *fails* in certain variants of constructive mathematics (but does hold in other variants; see [38] for an overview).

Computable Substitutions We define computable substitutions. We limit ourselves to substitutions with finite domains. The restriction to finite domains suffices below because of the usual restriction in infinitary rewriting that all rules have *finite* left-hand sides.

Definition 4.10. A *substitution* (with finite domain) is a finite set of pairs

$$\sigma = \{(x_0, t_0), \ldots, (x_m, t_m)\}$$

with x_i a variable and t_i a term for every $i \in \{0, \ldots, m\}$, such that each variable x_i occurs at most once as the first component of a pair in σ. A substitution is *computable* if t_i is a computable term for every $i \in \{0, \ldots, m\}$.

Just as computable terms are terms in the usual sense, computable substitutions are substitutions in the usual sense. We can represent each element of a computable substitution by a natural number by (i) representing t_i as a natural number, as explained immediately below Definition 4.4, and by (ii) applying a computable pairing operation to pair t_i with x_i. Consequently, we can also represent a computable substitution by a natural number: simply apply a computable mapping from $\mathbb{N}^*$ to $\mathbb{N}$.

Applying a substitution σ to an infinite term t yields a well-defined infinite term $\sigma(t)$ [5, 33]. Naturally, if both t and σ are computable, we would like $\sigma(t)$ to be computable. In fact, we would like $\sigma(t)$ to be *uniformly* computable: there should exist a Turing machine that, on input (of representations of) t and σ, yields (a representation of) $\sigma(t)$. The existence of such a Turing machine is witnessed by the following lemma. In the lemma, and also in any further statements regarding Turing machines with certain behaviour, we assume both in- and output to be encoded as elements of $\mathbb{N}$; this is possible for terms and substitutions as explained above, and for combinations thereof by use of a suitable computable pairing operation.

Lemma 4.11. There exists a Turing machine M with the following behaviour:

Input – a computable term t and a computable substitution σ;

Output – a Turing machine N that computes $\sigma(t)$.

Proof:
Observe that it is decidable whether $x \in \mathcal{D}om(\sigma)$, as σ has finite domain. Let the Turing machine N for each $p \in \mathbb{N}^*$ trace the path from the root of t as defined by p. If a variable $x \in \mathcal{D}om(\sigma)$ is found at a position q such that $q \cdot q' = p$, then output $\sigma(x)(q')$, otherwise output $t(p)$. □

Hence, if t is a computable infinite term and σ is a computable substitution, then $\sigma(t)$ is a computable infinite term.

Subterms and Contexts We now define subterms and contexts.

Definition 4.12. Let t be a term and $p \in \mathcal{P}os(t)$. The *subterm* $t|_p$ at position p is defined as $t|_p(q) = t(p \cdot q)$ for each $q \in \mathbb{N}^*$.

A *one-hole context* $C[\Box]$ is a term over Σ and $V \cup \{\Box\}$ such that precisely one $\Box$ (the 'hole') occurs in $C[\Box]$. A one-hole context $C[\Box]$ is *computable* if the term $C[\Box]$ is computable. Given a substitution $\sigma = \{(\Box, t)\}$, $C[t]$ denotes $\sigma(C[\Box])$.

Do not conflate $t|_p$ and $t(p)$: the former denotes a function such that for each $q \in \mathbb{N}^*$ we have $t|_p(q) = t(p \cdot q)$, whereas the latter denotes the symbol at position $p \in \mathbb{N}^*$ in t (if any). We have:

Proposition 4.13. There exists a Turing machine M with the following behaviour:

Input – a computable term t and a position $p \in \mathcal{P}os(t)$;

Output – a Turing machine N that computes $t|_p$.

Proof:
Let N for each $q \in \mathbb{N}^*$ compute $p \cdot q$ (which is possible as p and q are finite) and output $t(p \cdot q)$. □

Hence, if t is a computable term and $p \in \mathcal{P}os(t)$, then $t|_p$ is a computable term. The behaviour of M in Proposition 4.13 on 'illegal' inputs, i.e., where p is not a position of t, is unspecified; whether the Turing machine returns a result or not, and what that result is, does not affect the technical development or proofs in any way.

5. Computable Rewrite Rules and Systems

We next define computable rewrite rules. We follow the standard definitions [5, 33], but as before insert notions from computability theory where needed.

Definition 5.1. An *infinitary rewrite rule* is a pair of terms (l, r), invariably written $l \to r$, such that l is finite with $l \notin V$, and such that each variable that occurs in r also occurs in l. A rewrite rule is *computable* if r is a computable term.

Observe that left-hand sides of rules are always computable because they are finite, and can thus be represented by natural numbers. Consequently, as we can also represent each computable term by a natural number (see immediately below Definition 4.4), the existence of a computable pairing operation implies that computable rewrite rules can be represented by natural numbers.

Remark 5.2. As with all other definitions in this paper involving computability, the onus is on the person defining a computable rewrite rule to check that all requirements from the definition are met (by providing proofs). For example, a proof must be provided showing that the left-hand side is finite, as our very liberal definition of computable terms entails that it is undecidable in general whether finiteness holds. Observe that in all papers on infinitary rewriting, except one[2], rules are always given by explicitly writing down finite representations of terms, which makes the rules obviously computable.

One-step rewriting is uniformly computable given the applied rule and position at which the rule is applied:

Lemma 5.3. There exists a Turing machine M with the following behaviour:

Input – a computable term t and a pair $(p, l \to r)$, with $p \in \mathcal{P}os(t)$ and $l \to r$ a computable rewrite rule, such that $t = C[\sigma(l)]$ for a context $C[\Box]$ and a substitution σ with $C[\Box]|_p = \Box$;

Output – a Turing machine N that computes $C[\sigma(r)]$.

Proof:
Let N compute σ by considering every $q \in \mathcal{P}os(l)$ and adding a pair $(l(q), t|_{p\cdot q})$ to σ for every $l(q) \in V$. Observe that $t|_{p\cdot q}$ is computable by Proposition 4.13. Note that, for every q and q' with $l(q) = l(q') \in V$, we have by definition of the input that $t|_{p\cdot q} = t|_{p\cdot q'}$. Hence, as l is a finite term, σ can be computed in finite time. Now, for each $q \in \mathbb{N}^*$, if $q \parallel p$ or $q < p$, let N output $t(q)$, otherwise, if $q = p \cdot p'$ for some $p' \in \mathbb{N}^*$, let N output $\sigma(r)(p')$, where $\sigma(r)$ is computable by Lemma 4.11. It is computable whether $q \parallel p$, $q < p$, or $q = p \cdot p'$, as p and q are finite. □

Following again the standard definitions [5, 33] and inserting notions from computability theory where needed, we define infinitary rewriting systems.

Definition 5.4. An *infinitary term rewriting system (iTRS)* R is a set of infinitary rewrite rules. An iTRS is *recursively enumerable (r.e.)* if it has an r.e. set of computable infinitary rewrite rules.

We employ r.e. iTRSs throughout the following. The restriction to r.e. iTRSs helps to ensure that every time we prove a computability result related to iTRSs, we will be able to prove that the result is *uniformly* computable. Uniform computability requires a suitable, finite representation of each input and output, and the most straightforward way to ensure this in the case of iTRSs is by assuming that rule sets are r.e. (i.e., that rule sets are representable by Turing machines enumerating the rules).

Observe that rule sets being r.e. is *insufficient* to ensure decidability of the deceptively simple problem of checking whether the topmost part of a term (up to some fixed depth k) contains a redex. If the rule sets were recursive and left-linear (see below), then the problem would be decidable (for every fixed k), but no result below depends on this.

[2]The exception is [39] where function symbols of infinite arity are considered. However, in [39], right-hand sides are constructed by induction, not co-induction, and, hence, each path through the tree-representation of a right-hand side of a rule is finite. In contrast, right-hand sides of rules in infinitary term rewriting systems allow for infinite paths.

Left-Linear Rules and Systems We employ the following common restriction on rules and iTRSs.

Definition 5.5. A rewrite rule $l \to r$ is *left-linear* if each variable occurs at most once in l. An iTRS is *left-linear* if all its rules are.

Looking back at Lemma 5.3, observe that it is undecidable in general whether a term t is equal to $C[\sigma(l)]$ and, hence, whether a specific non-left-linear rule can be applied at a specific position in t. The lemma circumvents the issue by requiring a 'witness' of $t = C[\sigma(l)]$. Thus, the Turing machine M from the lemma does not need to check for equality.

A witness of $t = C[\sigma(l)]$ is *not* required in the case of left-linear rules, because each variable occurs at most once. Hence, it is easy to see that for *left-linear* r.e. iTRSs the set of redexes of a computable term is also an r.e. set. Notwithstanding, even for left-linear systems it is generally undecidable whether a term is in normal form.

6. Computable Infinite Reductions

We next define computable infinite reductions. We start by recalling the standard definition of strongly convergent reductions [5, 33]:

Definition 6.1. Given an iTRS R, a *strongly convergent reduction* of ordinal length α over R, denoted $t_0 \twoheadrightarrow_R t_\alpha$, is a sequence of *terms* $(t_\beta)_{\beta<\alpha+1}$ together with a sequence of *steps* $(p_\beta, l_\beta \to r_\beta)_{\beta<\alpha}$ where $l_\beta \to r_\beta \in R$, $t_\beta = C_\beta[\sigma_\beta(l_\beta)] \to C_\beta[\sigma_\beta(r_\beta)] = t_{\beta+1}$, and $C_\beta[\Box]|_{p_\beta} = \Box$, such that for every limit ordinal $\beta \leq \alpha$ and $n \in \mathbb{N}$ there exists a $\gamma_n < \beta$ with $t_\delta(q) = t_\beta(q)$ and $|p_\delta| > n$ for all $q \in \mathbb{N}^*$, $|q| < n$, and $\gamma_n \leq \delta < \beta$.

If a reduction $t_0 \twoheadrightarrow_R t_\alpha$ has finite length, then we also denote the reduction by $t_0 \to^* t_\alpha$.

Computable Ordinals To obtain a computability-theoretic version of Definition 6.1, we employ computable ordinals (for an in-depth account of computable ordinals, see, e.g., [31]).

Definition 6.2. Let α be an ordinal and $A \subseteq \mathbb{N}$. A binary relation R over A is of *order type* α if there exists a bijection between A and α such both the bijection and its inverse are order preserving.

An ordinal α is *computable* (or *recursive*) if there exists an r.e. set $A \subseteq \mathbb{N}$ and a decidable binary relation over A that well-orders A with order type α.

Examples of computable ordinals are any finite ordinal, $\epsilon_0 = \lim\{\omega, \omega^\omega, \omega^{\omega^\omega}, \ldots\}$, any ordinal written as an arithmetic expression over successors and ω (e.g., $\omega^\omega + 1$), the Feferman-Schütte ordinal Γ_0, and the small and large Veblen ordinals $\phi_{\Omega^\omega}(0)$ and $\phi_{\Omega^\Omega}(0)$ [40]. All computable ordinals are countable; the least uncomputable ordinal, the *Church-Kleene* ordinal ω_1^{CK}, is also countable [31].

Although the computability-theoretic version of Definition 6.1 only depends on computable ordinals as defined above, obtaining further results requires representations of computable ordinals α that also allow us to (i) compute for each $\beta < \alpha$ if β is either 0, a successor ordinal, or a limit ordinal,

(ii) compute the predecessor of every successor ordinal, and (iii) decide for every β, $\gamma < \alpha$ whether $\beta < \gamma$. To this end we employ univalent, recursively related systems of notation [41, 31].[3]

Definition 6.3. Let α be an ordinal. A *system of notation* S for α is a function $\nu_S : D \longrightarrow \alpha$ with $D \subseteq \mathbb{N}$ such that the following Turing machines exist:

- M_k, computing the ordinal kind, such that (a) if $\nu_S(n) = 0$, then $\varphi_{M_k}(n) = 0$, (b) if $\nu_S(n)$ is a successor ordinal, then $\varphi_{M_k}(n) = 1$, and (c) if $\nu_S(n)$ is a limit ordinal, then $\varphi_{M_k}(n) = 2$;
- M_p, computing predecessors, such that if $\nu_S(n)$ is a successor ordinal, then $\varphi_{M_p}(n)$ is defined and $\nu_S(n) = \nu_S(\varphi_{M_p}(n)) + 1$;
- M_q, computing limits, such that if $\nu_S(n)$ is a limit ordinal, then $\varphi_{M_q}(n)$ is defined, say as $\langle M' \rangle$, with $\varphi_{M'}$ total, and

  $$\nu_S(\varphi_{M'}(0)),\ \nu_S(\varphi_{M'}(1)),\ \ldots,\ \nu_S(\varphi_{M'}(i)),\ \ldots$$

  an increasing sequence whose limit is $\nu_S(n)$.

A system of notation S is *univalent* if ν_S is injective and it is *recursively related* if the set of pairs $R_S = \{(m, n) \mid m,\, n \in D \wedge \nu_S(m) \leq \nu_S(n)\}$ is decidable.

Remark 6.4. Given a univalent, recursively related system of notation S for an ordinal α, we can compute the unique element of D representing 0, and the unique successor of every ordinal (if the successor is $< \alpha$).

For the element representing 0, compute $\varphi_{M_k}(n)$ in parallel for all $n \in \mathbb{N}$ recording the elements z such that $\varphi_{M_k}(z) = 0$. For each recorded z, use a Turing machine deciding R_S to decide whether $(z, z) \in R_S$ (at least one such element exists for all ordinals $\alpha > 0$, otherwise 0 would not have a representation). As soon as one element $(z, z) \in R_S$ is found, we know that $z \in D$. We claim that $\nu_S(z) = 0$; to see this, observe that if $\nu_S(z)$ is a successor or a limit ordinal, then $\varphi_{M_k}(z) \in \{1, 2\}$, a contradiction. By univalence of S it now follows that z is the unique element of D representing 0.

For the successor, recursive relatedness implies that for each $m \in D$ we can enumerate all $n \in \mathbb{N}$ such that $(m, n) \in R_S$, and using M_k we can establish whether such an n is a successor ordinal. By univalence, the predecessor of n is unique, and we can compute it using M_p. If $\varphi_{M_p}(n) = m$, then we have found the successor of m, which is again unique by univalence.

The following is standard [41, 31]:

Theorem 6.5. For every computable ordinal there exists a univalent, recursively related system of notation.

Hence, without loss of generality we may use univalent, recursively related systems of notation.

An example of a system of notation is Kleene's system $\mathcal{O}$ [41, 31]. The system $\mathcal{O}$ is univalent but not recursively related. For 'small' computable transfinite ordinals, univalent, recursively related systems can be easily specified, e.g., by using the Cantor normal form notation for ordinals up to ϵ_0 and encoding the normal forms by finite trees or lists of pairs in the usual way [31, 42].

[3] The name *system of notation* is taken from [31]. In [41] these systems are called *r-systems*.

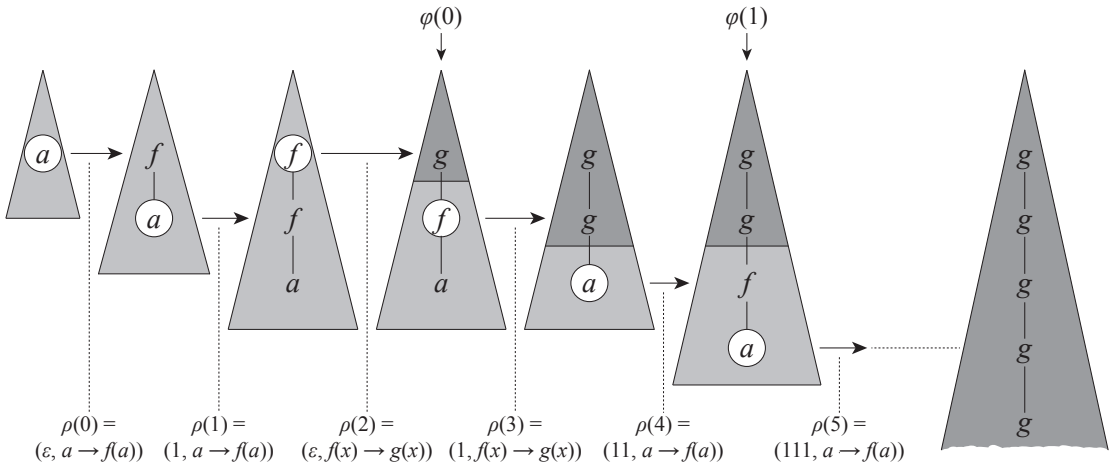

Figure 2. A computably strongly convergent reduction. Note that we have two computable functions: a function ρ computing each rewrite step, and a function φ computing a modulus of convergence for the reduction. The function $\rho(n)$ yields the nth rewrite step, whereas $\varphi(n)$ yields an ordinal such that all steps from the $\varphi(n)$th step onward will occur below depth n (above, $\varphi(0)$ and $\varphi(1)$ point to the left-hand sides of such steps). Observe that $\varphi(n)$ does not have to yield the minimum step from which point onward all steps occur below depth n (above, $\varphi(1)$ could have been placed one term further to the left).

Remark 6.6. Given a system of notation for an ordinal α, there exists a Turing machine M that on input $\beta < \alpha$ outputs the largest limit ordinal smaller than or equal to β, if it exists, and outputs 0 otherwise. Let M employ M_k to decide the ordinal kind of β. If β is a limit ordinal or 0, then M is done. Otherwise, β is a successor ordinal, in which case M may use M_p to compute its predecessor and continue recursively. As there exists no infinite descending chain of ordinals, M eventually halts.

From here onward, we will always work in the context of an explicitly specified computable ordinal α (typically, the length of a transfinite reduction). In each such context, we assume for α (or $\alpha + 1$, if α is a successor ordinal) a *fixed* univalent, recursively related system of notation, and *all* representations of ordinals $\beta < \alpha$ that we use in this context will come from the fixed system.

Computable Reductions With the machinery of computable ordinals in hand, defining computably strongly convergent reductions is now quite straightforward. The only difficulty that arises relates to requirement that, for each reduction and depth n, we want to be able to compute from which point in the reduction onward all contracted redexes occur below depth n. In other words, we want the *rate* of convergence of reductions to be computable. Unfortunately, as we show in Example 6.12, computing the rate of convergence is not possible in general.

To mitigate the above issue, we borrow the concept of *computable modulus of convergence* from computable analysis [2]. Consider the following strongly convergent reduction of length ω:

$$a \to f(a) \to f(f(a)) \to g(f(a)) \to g(g(a)) \to g(g(f(a))) \to g(g(f(f(a)))) \to \cdots g^{\omega}\,,$$

employing the rules $a \to f(a)$ and $f(x) \to g(x)$ (see also Figure 2). The steps of the reduction can clearly be specified by a computable function:

$$\rho(\beta) = \begin{cases} (0^k, a \to f(a)) & \text{if } \beta = 4k \\ (0^{k+1}, a \to f(a)) & \text{if } \beta = 4k+1 \\ (0^k, f(x) \to g(x)) & \text{if } \beta = 4k+2 \\ (0^{k+1}, f(x) \to g(x)) & \text{if } \beta = 4k+3 \end{cases}$$

It is also possible to construct a computable function which, for every depth n, yields the reduction step such that from that step onward we have that all steps occur below n:

$$\varphi(n) = \begin{cases} 4k+3 & \text{if } n = 2k \\ 4k+5 & \text{if } n = 2k+1 \end{cases}$$

The function φ is a computable modulus of convergence for the reduction. Although in this case a modulus is easily computed from ρ, this is not possible in general (see Example 6.12).

Oftentimes we will need to know the rate of convergence when approaching any arbitrary limit ordinal along a reduction. Thus, a modulus of convergence requires not only a *depth* argument but also a *limit ordinal* argument, and the above example presents a degenerate case, where the limit ordinal argument is always ω.

Call a computable map from a computable ordinal α to a set $S \subseteq \mathbb{N}$ a *computable sequence of ordinal length α over S*. We define the following.

Definition 6.7. A *proto-computably strongly convergent reduction* of ordinal length α over an r.e. iTRS R is a strongly convergent reduction of computable ordinal length α over R such that $(t_\beta)_{\beta<\gamma}$ is a computable sequence over computable terms, with $\gamma = \alpha$, if α is a limit ordinal, and $\gamma = \alpha + 1$ otherwise, and such that $(p_\beta, l_\beta \to r_\beta)_{\beta<\alpha}$ is a computable sequence over steps $(p, l \to r)$ with $l \to r$ a computable rewrite rule.

A *computable modulus of convergence* for a strongly convergent reduction of length α is a computable function $\varphi : \alpha \times \mathbb{N} \longrightarrow \gamma$, with $\gamma = \alpha$, if α is a limit ordinal, and $\gamma = \alpha + 1$ otherwise, such that for all $n \in \mathbb{N}$:

- if $\gamma_n \doteq \varphi(0, n)$, then $|p_\delta| > n$ for all $\gamma_n \leq \delta < \gamma$, and
- if $\gamma_n \doteq \varphi(\beta, n)$ with $\beta < \alpha$ a limit ordinal, then $\gamma_n < \beta$ and $|p_\delta| > n$ for all $\gamma_n \leq \delta < \beta$.

A pair $(s \twoheadrightarrow t, \varphi)$ consisting of a proto-computably strongly convergent reduction $s \twoheadrightarrow t$ and a computable modulus of convergence φ for $s \twoheadrightarrow t$ is called a *computably strongly convergent reduction*.

We usually suppress explicit mention of φ, referring to $s \twoheadrightarrow t$ as computably strongly convergent. We will also write $s \to^* t$ for a computably strongly convergent reduction when we can compute its length and that length is finite. In the case of finite reductions, we assume without loss of generality that the reduction is encoded by employing our fixed bijection from $\mathbb{N}^*$ to $\mathbb{N}$, where each natural number in the sequence represents a reduction step. Note that given a finite reduction using the fixed encoding, we can uniformly compute its length.

Each computable term t_β of a proto-computably strongly convergent reduction can be represented by a natural number using the encoding specified immediately below Definition 4.4. Similarly, each step $(p_\beta, l_\beta \to r_\beta)$ can be represented by a natural number: employ a computable pairing operation and make use of the fact that p_β is finite and that $l_\beta \to r_\beta$ has a natural number representation (as explained immediately below Definition 5.1). Hence, we may rightfully talk about computable sequences when defining proto-computably strongly convergent reductions.

Observe that we do not require the final term t_α of a proto-computably strongly convergent reduction to be computable if α is a limit ordinal (indeed, in this case t_α is not even explicitly given by the Turing machine computing the sequence of terms). There are two reasons for this. First, given a computable modulus of convergence φ, we can compute t_α, as we explain in Remark 6.11. Second, if t_α were given explicitly, one could compute it without considering the initial parts of the reduction or its converging behaviour, which we believe to be contrary to the common and accepted intuition regarding infinitary rewriting.

With regard to the modulus of convergence, observe that we only require it to produce correct results if the first argument is either 0 or a limit ordinal $< \alpha$. We do not care about the behaviour of the modulus otherwise. The use of $\varphi(0, n)$, instead of $\varphi(\alpha, n)$, is a technical convenience: it ensures that we do not need to know the length of the reduction, e.g., when computing the term t_α in Remark 6.11. As explained in Remark 6.4, $0 \in \alpha$ is computable because we assume a univalent, recursively related system of notation for α.

Example 6.8. Consider the computable terms defined by the recursion equations $s = f(s)$ and $t = g(t)$ (see also Example 4.5), and consider the rule $f(x) \to g(x)$, which is computable as both sides are finite. We have the following strongly convergent reduction of length ω, contracting the outermost redex in each step:

$$s \to g(s) \to g(g(s)) \to \cdots \to g^n(s) \to g^{n+1}(s) \to \cdots t\,,$$

The reduction is obviously proto-computable and has a computable modulus of convergence φ with $\varphi(0, n) = n + 1$.

Example 6.9. Expanding on the previous example, consider the terms defined by the recursion equations $s = f(s)$, $t = g(t)$, $u = h(u)$, and consider the rules $f(x) \to g(x)$ and $g(x) \to h(x)$. We have the following strongly convergent reduction of length $\omega \cdot 2$ that starts by repeatedly contracting outermost f-redexes until no such redex is left, and by then repeatedly contracting outermost g-redexes:

$$\begin{aligned} s \to g(s) \to g(g(s)) \to \cdots \to g^n(s) \to g^{n+1}(s) \to \cdots \\ t \to h(t) \to h(h(t)) \to \cdots \to h^n(t) \to h^{n+1}(t) \to \cdots u\,, \end{aligned}$$

To see that the reduction is computable, first observe that we can define a univalent, recursively related system of notation S for $\omega \cdot 2$ with $\nu_S : \mathbb{N} \longrightarrow \omega \cdot 2$ defined as:

$$\nu_S(n) = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even} \\ \omega + \frac{n-1}{2} & \text{if } n \text{ is odd} \end{cases}$$

The steps of the reduction can now be described by a computable function with domain $\mathbb{N}$ (the domain of ν_S):

$$\rho(n) = \begin{cases} (0^k, f(x) \to g(x)) & \text{if } n \text{ is even and } k = \frac{n}{2} \\ (0^k, g(x) \to h(x)) & \text{if } n \text{ is odd and } k = \frac{n-1}{2} \end{cases}$$

and similarly for the terms of the reduction. There also exists a computable modulus of convergence φ, where both the first argument and the range are over the domain of ν_S, with $\varphi(0, i) = 2i + 3 = \nu_S^{-1}(\omega + i + 1)$ and $\varphi(1, i) = \varphi(\nu_S^{-1}(\omega), i) = 2i + 2 = \nu_S^{-1}(i + 1)$.

Remark 6.10. Not every strongly convergent reduction is proto-computable. Let $\psi : \mathbb{N} \longrightarrow \mathbb{N}$ be a strictly increasing uncomputable function and let $s = f(s)$ and $f(x) \to g(x)$. Consider the reduction that in each step $n \in \mathbb{N}$ contracts the redex at position p_n with $|p_n| = \psi(n)$. The reduction is of length ω, and strongly convergent because ψ is strictly increasing. The reduction is, however, not computable, as ψ is not.

We end this section with a number of observations that should convince the reader that the choice of the representation $(s \twoheadrightarrow t, \varphi)$ is the 'right' one for computably strong convergence, and that the inclusion of a computable modulus of convergence is necessary.

To start, observe that computable moduli of convergence support the computational intuition behind strongly convergent rewriting that we believe is prevalent in infinitary rewriting: if a program computes some 'infinite data structure', then for each natural number j the reduction will 'stabilise' after a finite number n_j of computation steps such that the 'initial part' of the data structure of size j can then be read without fear that the part will change later in the computation. Indeed, a modulus φ computes an upper bound on n_j when given input j: "to safely read data in the result up to depth j, perform $n = \varphi(0, j)$ steps."

Remark 6.11. Given a computable modulus φ for a reduction $t_0 \twoheadrightarrow t_\alpha$, we can compute the term t_α in the following manner: for each $p \in \mathbb{N}^*$ compute from which term t_β onward all reduction steps occur below $|p|$ (i.e., compute $\varphi(0, |p|)$) and take $t_\beta(p)$.

Yet another reason for defining computably strongly convergent reductions with an explicit demand for a computable modulus of convergence is that certain 'wild' computable reductions may be strongly convergent, but fail to have a computable modulus, as shown in the following example.

Example 6.12. Let $S \subseteq \mathbb{N}$ be an infinite set that is r.e., but not co-r.e. (i.e., $\mathbb{N} \setminus S$ is not r.e.). Consider an injective total computable function ψ enumerating S (i.e., for each $i \in \mathbb{N}$, $\psi(i)$ computes the ith element in the enumeration). The function ψ is eventually increasing (but not necessarily monotone), because ψ is injective and because the number of elements in S smaller than n is finite for each $n \in \mathbb{N}$.

Given the term $s = f(s)$ and the rule $f(x) \to g(x)$, define a reduction of length ω starting from s which for each $i \in \mathbb{N}$ contracts in the ith step the unique redex at position p with $|p| = \psi(i)$. As ψ is computable and eventually increasing, the reduction is proto-computably strongly convergent. However, no computable modulus of convergence φ exists. Otherwise, S would be co-r.e.: for $n \in \mathbb{N}$, compute $\varphi(0, n)$ and, next, compute the first $\varphi(0, n)$ steps of the reduction. If no redex was contracted at position q such that $|q| = n$, then $n \notin S$.

As the above example shows, a computable modulus of convergence does not always exist for a proto-computably strongly convergent reduction. Even stronger, it is generally not possible to compute a modulus for a proto-computably strongly convergent reduction even if one is known to exist.

Example 6.13. For each inputless Turing machine M, we may define a computably strongly convergent reduction based on the iTRS from Example 6.9: start from $s = f(s)$, and, if M has not halted in n steps, contract the unique redex at depth n in the nth step of the reduction (note that the first step of the reduction occurs at depth 0, i.e., at the root). If M halts in precisely n steps, contract the redex at the root in the nth step of the reduction, and for every $k > n$, contract the redex at depth $k - n$.

Observe that all defined reductions are computably strongly convergent reductions of length ω. If M does not halt, then the function $\varphi(0, j) = j + 1$ defines a computable modulus of convergence for the reduction, and if M does halt after n steps, then the function $\varphi(0, j) = n + j + 1$ defines a computable modulus of convergence. However, there does not exist a Turing machine which, on input of one of these reductions, outputs a modulus of convergence, otherwise the halting problem would be decidable. On input M, compute φ and $\varphi(0, 0) = m$, and compute the depth at which the first m steps occur. Now, M halts iff among the m steps there are exactly two steps at the root.

We hope that our efforts above have convinced the reader that omitting the modulus of convergence from Definition 6.7 is incompatible with the aim of being able to compute the rate of convergence of strongly convergent reductions.

7. Descendants and Origins

We devote the remainder of the paper to laying the groundwork that will enable future efforts to prove computable versions of the standard compression and confluence results from infinitary rewriting [5, 33]. The groundwork consists of defining descendants and origins [43] in the current section, and showing in Section 8 that they are uniformly computable.

Note that none of the notions defined in the current section depend on computability of the assumed iTRSs and reductions. Computability will only become relevant again once we arrive at Section 8.

7.1. Descendants

Descendants formalise how a position from the *initial* term of a reduction 'contributes' to the positions of the *final* term. We follow the usual definition from strongly convergent infinitary rewriting [5, 33]:

Definition 7.1. Let $s \to t$ be a rewrite step contracting a redex at position $p \in \mathcal{P}os(s)$ and employing a rule $l \to r$. If $q \in \mathcal{P}os(s)$, then the set of *descendants of q across $s \to t$*, denoted $q\backslash(s \to t)$, is the subset of positions of t defined as:

- $\{q\}$ if $q \parallel p$ or $q < p$,
- $\emptyset$ if $q = p \cdot p'$ with $l(p') \in \Sigma$, and
- $\{p \cdot q' \cdot p'' \mid r(q') = l(p')\}$ if $q = p \cdot p' \cdot p''$ with $l(p') \in V$

If $Q \subseteq \mathcal{P}os(s)$ is *any* set of positions, then $Q\backslash(s \to t) = \bigcup_{q \in Q} q\backslash(s \to t)$.

Let $t_0 \to^* t_n$ be a finite reduction and let $t_0 \twoheadrightarrow t_\alpha$ be strongly convergent. If $Q \subseteq \mathcal{P}os(t_0)$ is *any* set of positions, then

$$Q\backslash(t_0 \to^* t_n) = \begin{cases} Q & \text{if } t_0 \to^* t_n \text{ is empty} \\ (Q\backslash(t_0 \to^* t_{n-1}))\backslash(t_{n-1} \to t_n) & \text{otherwise} \end{cases}$$

Furthermore,

$$Q\backslash(t_0 \twoheadrightarrow t_\alpha) = (Q\backslash(t_0 \twoheadrightarrow t_\beta))\backslash(t_\beta \to^* t_\alpha)\,,$$

where β is the largest limit ordinal smaller than or equal to α (or 0 if no such ordinal exists), and $q \in Q\backslash(t_0 \twoheadrightarrow t_\beta)$ iff $q \in Q\backslash(t_0 \twoheadrightarrow t_\gamma)$ and each step in $t_\gamma \twoheadrightarrow t_\beta$ occurs at a position p with $|p| > |q|$.

Across a single rewrite step there are three possible cases: either (i) $q \parallel p$ or $q < p$, in which case the rewrite step does not affect the descendants of q, (ii) $q = p \cdot p'$ with p' the position of a function symbol in l, in which case q does not have any descendants, or (iii) $q = p \cdot p' \cdot p''$ with p' the position of a variable, say x, in l, in which case q has a descendant for each occurrence of x in r. Across finite reductions we simply iterate the definition of descendants across rewrite steps. Additionally, across infinite reductions we make use of strong convergence.

Observe that the set of descendants of a position may be infinite across a single rewrite step, as there are no constraints on the sizes of right-hand sides of rewrite rules. Across a reduction, the set of descendants may also be infinite because there may be an infinite number of steps where across each single step the third clause from the definition applies with the variable $l(p') \in V$ occurring more often on the right-hand side of the rule than on the left-hand side.

Example 7.2. Consider the rules $f(x) \to h(x, x)$ and $g(x) \to x$. We have

$$\{\epsilon, 0, 00\}\backslash(f(g(a)) \to h(g(a), g(a)) \to h(a, g(a))) \\ = \{0, 00, 1, 10\}\backslash(h(g(a), g(a)) \to h(a, g(a))) = \{0, 1, 10\}\,.$$

7.2. Origins

Origins formalise how a position in the *final* term of a reduction is 'contributed' to by the positions of the *initial* term. Hence, origins can be considered to be a sort of 'inverse' of descendants. We define origins as as in [44] (see [43] for a definition in the context of finitary term rewriting and further discussion):

Definition 7.3. Let $s \to t$ be a rewrite step contracting a redex at a position $p \in \mathcal{P}os(s)$ and employing a rule $l \to r$. If $q \in \mathcal{P}os(t)$, then the set of *origins of* q *across* $s \to t$, denoted $(s \to t)^{\backslash}q$, is the subset of positions of s defined as:

- $\{q\}$ if $q \parallel p$ or $q < p$,
- $\{p \cdot q' \mid l(q') \in \Sigma\}$ if $q = p \cdot p'$ with $r(p') \in \Sigma$, and

- $\{p \cdot q' \cdot p'' \mid l(q') = r(p')\} \cup \{p \cdot q' \mid l(q') \in \Sigma \wedge r(\epsilon) \in V\}$ if $q = p \cdot p' \cdot p''$ with $r(p') \in V$.

If $Q \subseteq \mathcal{P}os(t)$ is a *finite* set of positions, then $(s \to t)\restriction Q = \bigcup_{q \in Q} (s \to t)\restriction q$.

Let $t_0 \to^* t_n$ be a finite reduction and let $t_0 \twoheadrightarrow t_\alpha$ be strongly convergent. If $Q \subseteq \mathcal{P}os(t_n)$ is a *finite* set of positions, then

$$(t_0 \to^* t_n)\restriction Q = \begin{cases} Q & \text{if } t_0 \to^* t_n \text{ is empty} \\ (t_0 \to^* t_{n-1})\restriction((t_{n-1} \to t_n)\restriction Q) & \text{otherwise} \end{cases}$$

Furthermore, if $Q \subseteq \mathcal{P}os(t_\alpha)$ is a *finite* set of positions, then

$$(t_0 \twoheadrightarrow t_\alpha)\restriction Q = (t_0 \twoheadrightarrow t_\gamma)\restriction((t_\gamma \to^* t_\beta)\restriction Q)\,,$$

where γ is the largest limit ordinal smaller than or equal to β (or 0 if no such ordinal exists), and β is such that each step in $t_\beta \twoheadrightarrow t_\alpha$ occurs at a position p with $|p| > |q|$ for all $q \in Q$.

Across a single rewrite step there are three possible cases: either (i) $q \parallel p$ or $q < p$, in which case the rewrite step does not affect the origins of q, (ii) $q = p \cdot p'$ with p' the position of a function symbol in r, in which case all function symbol positions of l act as origins of q, or (iii) $q = p \cdot p' \cdot p''$ with p' the position of a variable, say x, in r, in which case q has an origin for each occurrence of x in l, and, in addition, when $r = x$, all function symbol positions of l act as origins of q. Across finite reductions we iterate the definition of origins across rewrite steps. Additionally, across infinite reductions we make use of strong convergence.

Observe that, contrary to descendants, we only define origins for finite sets of positions. The reason for this asymmetry stems from the difference in use cases of descendants and origins: descendants are used to track redexes across reductions (with a potentially *infinite* number of copies of a tracked redex being created across a reduction), while origins are used to construct sub-reductions of strongly convergent reductions with the final terms corresponding up to a certain *finite* depth (so-called *needed* reductions, defined below).

The set of origins is well-defined in the limit ordinal case, because for all p with $|p| \leq |q|$ and $q \in Q$ we have that $t_\beta(p) = t_\alpha(p)$, and because all descending chains of ordinals are finite. By the previous observation and the fact that we only allow rewrite rules with finite left-hand sides, it also follows that a finite set of positions always has a finite set of origins across a reduction. Furthermore, if Q is a prefix-closed set of positions (i.e., for each $q \in Q$ we have that $p \in Q$ for all $p < q$), then the set of origins is also prefix-closed.

Example 7.4. Consider the rules $f(x) \to h(x, x)$ and $g(x) \to x$ from Example 7.2. We have

$$(f(g(a)) \to h(g(a), g(a)) \to h(a, g(a)))\restriction\{0, 1, 10\} \\ = (f(g(a)) \to h(g(a), g(a)))\restriction\{0, 00, 1, 10\} = \{0, 00\}$$

and

$$(f(g(a)) \to h(g(a), g(a)) \to h(a, g(a)))\restriction\{\epsilon, 1\} \\ = (f(g(a)) \to h(g(a), g(a)))\restriction\{\epsilon, 1\} = \{\epsilon, 0\}\,.$$

Needed Reductions Having defined origins, we next define the related concept of needed reductions for *left-linear* iTRSs, again following [44]. Intuitively, a needed reduction is a sub-reduction of a strongly convergent reduction $s \twoheadrightarrow t$ that retains a certain amount of structure with respect to a set of positions of the final term t. In order to define needed reductions, we first need to define needed steps.

Definition 7.5. Let $t_0 \twoheadrightarrow t_\alpha$ be a strongly convergent reduction and let $Q \subseteq \mathcal{P}os(t_\alpha)$ be finite and prefix-closed. A step $(p_\beta, l_\beta \to r_\beta)$ of $t_0 \twoheadrightarrow t_\alpha$ is *needed for* Q iff $p_\beta \in (t_\beta \twoheadrightarrow t_\alpha)\backslash Q$.

Hence, needed steps are steps that occur at origin positions of Q along $t_0 \twoheadrightarrow t_\alpha$. The set of needed steps for Q will always be finite for a finite set Q by strong convergence of the reduction and the fact that $(t_\beta \twoheadrightarrow t_\alpha)\backslash Q$ is always finite (as we observed when we defined origins).

Example 7.6. Consider the rules $f(x) \to h(x, x)$ and $g(x) \to x$ and the reduction

$$f(g(a)) \to h(g(a), g(a)) \to h(a, g(a))$$

from Example 7.4. Given the finite set $Q = \{\epsilon, 1\} \subseteq \mathcal{P}os(h(a, g(a)))$, the definition of origins yields that only the first step of the reduction is needed for Q, as $0 \notin \{\epsilon, 1\}$.

Needed steps give rise to finite reductions in the case of *left-linear* systems:

Lemma 7.7. Let R be a left-linear iTRS, $s \twoheadrightarrow_R t$ a strongly convergent reduction, and $Q \subseteq \mathcal{P}os(t)$ finite and prefix-closed. The finite subsequence of $s \twoheadrightarrow t$ formed by the needed steps for Q defines a finite reduction $s \to_R^* t'$ such that $t'(q) = t(q)$ for all $q \in Q$.

Proof:
Let n be the length of the finite subsequence of $s \twoheadrightarrow t$ formed by the needed steps for Q. We proceed by induction on n. If $n = 0$, define $s \to_R^* t'$ to be the empty reduction. We have $t' = s$. That $s(q) = t(q)$ for all $q \in Q$ follows once we observe that each step in $s \twoheadrightarrow t$ occurs at a position p with for all $q \in Q$ either $p > q$ or $p \parallel q$ by the definition of neededness.

If $n = n' + 1$, write $t_0 \twoheadrightarrow t_\alpha$ for $s \twoheadrightarrow t$ and suppose the last step of the finite subsequence of needed steps for Q is the βth step of $t_0 \twoheadrightarrow t_\alpha$, where the step contracts a redex at position p employing rule $l \to r$. By the induction hypothesis, there exists a finite reduction $t_0 \to^* t'_{n'}$ such that $t'_{n'}(q) = t_\beta(q)$ for all $q \in (t_\beta \twoheadrightarrow t_\alpha)\backslash Q$. Hence, by left-linearity of R, a redex employing $l \to r$ occurs at position p in $t'_{n'}$. Define $t_0 \to^* t'_n$ by appending to $t_0 \to^* t'_{n'}$ the step contracting the $l \to r$-redex at position p in $t'_{n'}$. Observe that $t'_n(q) = t_{\beta+1}(q)$ for all $q \in (t_{\beta+1} \twoheadrightarrow t_\alpha)\backslash Q$. Moreover, $t'_n(q) = t_\alpha(q)$ for all $q \in Q$, as each step in $t_{\beta+1} \twoheadrightarrow t_\alpha$ occurs at a position p with for all $q \in Q$ either $p > q$ or $p \parallel q$ by the definition of neededness. The complete result now follows once we observe that the needed steps of $t_0 \twoheadrightarrow t_\beta$ for $(t_\beta \twoheadrightarrow t_\alpha)\backslash Q$ are precisely the first n' needed steps of $t_0 \twoheadrightarrow t_\alpha$ for Q. □

With the above lemma in hand, we can define needed reductions.

Definition 7.8. Let R be a left-linear iTRS, $s \twoheadrightarrow_R t$ a strongly convergent reduction, and $Q \subseteq \mathcal{P}os(t)$ finite and prefix-closed. The *needed reduction* for $s \twoheadrightarrow t$ and Q is the finite reduction whose existence is guaranteed by Lemma 7.7

Take heed that both the definition and all results related to needed reductions *only* apply to *left-linear* iTRSs.

Example 7.9. Continuing Example 7.6, we obtain for the reduction

$$f(g(a)) \to h(g(a), g(a)) \to h(a, g(a))$$

and the finite, prefix-closed set $\{\epsilon, 1\} \subseteq \mathcal{P}os(h(a, g(a)))$ the needed reduction

$$f(g(a)) \to h(g(a), g(a))\,,$$

with the function symbols of $h(g(a), g(a))$ and $h(a, g(a))$ equal at positions ϵ and 1, as implied by Lemma 7.7.

7.3. Relating Descendants and Origins

As mentioned above, descendants and origins differ in that the former are defined over all sets of positions, while the latter are only defined over finite sets. Besides this, descendants and origins are also not fully symmetric with regard to positions occurring in redex patterns and their substitute patterns (compare Example 7.2 and the first half of Example 7.4): a position occurring in the redex pattern of a contracted redex does not have any descendants, while a position occurring in the substitute pattern of a contracted redex has all positions in the redex pattern as its origins. As before, the difference stems from the distinct intended uses of descendants and origins: descendants are intended to track non-contracted redexes across reductions, while origins are intended to construct needed reductions (which requires identifying contracted redexes and, hence, one or more positions of those redexes).

Although the definitions of descendants and origins are not fully symmetric, we can still relate them in the case of positions that are not part of redex patterns and substitute patterns. To start, if a set of positions P of the initial term of a reduction does not overlap with the origins of a set of positions Q of the final term, then the same holds for the descendants of P and for Q itself.

Proposition 7.10. Let R be an iTRS and $s \twoheadrightarrow_R t$ a strongly convergent reduction. If $P \subseteq \mathcal{P}os(s)$, $Q \subseteq \mathcal{P}os(t)$ finite, and $P \cap ((s \twoheadrightarrow t)\nwarrow Q) = \emptyset$, then $(P\searrow(s \twoheadrightarrow t)) \cap Q = \emptyset$.

Proof:
Let α be the length of $s \twoheadrightarrow t$. We proceed by transfinite induction on α. If $\alpha = 0$, then we have $P\searrow(s \twoheadrightarrow t) = P$ and $(s \twoheadrightarrow t)\nwarrow Q = Q$, and the result is immediate.

If $\alpha = \alpha' + 1$, write $s \twoheadrightarrow t$ as $s \twoheadrightarrow t' \to t$. By the induction hypothesis, $(P\searrow(s \twoheadrightarrow t')) \cap ((t' \to t)\nwarrow Q)$ is empty. To see that $(P\searrow(s \twoheadrightarrow t)) \cap Q$ is also empty, suppose to the contrary that there exists a position $p \in P\searrow(s \twoheadrightarrow t)$ such that $(p\searrow(t' \to t)) \cap Q$ is non-empty. By definition of descendants and origins, it is now immediate that for any $q \in (p\searrow(t' \to t)) \cap Q$ we have $p \in (P\searrow(s \twoheadrightarrow t)) \cap ((t' \to t)\nwarrow q)$. Hence, $(P\searrow(s \twoheadrightarrow t')) \cap ((t' \to t)\nwarrow Q)$ is non-empty, contradicting the induction hypothesis.

If α is a limit ordinal, then by the induction hypothesis and strong convergence there exists a $\beta < \alpha$ such that $(P\searrow(s \twoheadrightarrow t_\beta)) \cap ((t_\beta \twoheadrightarrow t)\nwarrow Q)$ is empty and such that each step in $t_\beta \twoheadrightarrow t$ occurs a position greater than $|q|$ for all $q \in Q$. Hence, by the definition of descendants and origins, we also have that $(P\searrow(s \twoheadrightarrow t)) \cap Q$ is empty. □

Besides the above, if some descendant of a position p occurs in a set of positions Q across a reduction, then p itself occurs in the set of origins of Q.

Proposition 7.11. Let R be an iTRS and $s \twoheadrightarrow_R t$ a strongly convergent reduction. If $p \in \mathcal{P}os(s)$, $Q \subseteq \mathcal{P}os(t)$ finite, and $(\{p\}\searrow(s \twoheadrightarrow t)) \cap Q \neq \emptyset$, then $p \in (s \twoheadrightarrow t)\nwarrow Q$.

Proof:
Let α be the length of $s \twoheadrightarrow t$. We proceed by transfinite induction on α. If $\alpha = 0$, then we have $\{p\}\searrow(s \twoheadrightarrow t) = \{p\}$ and $(s \twoheadrightarrow t)\nwarrow Q = Q$, and the result is immediate.

If $\alpha = \alpha' + 1$, write $s \twoheadrightarrow t$ as $s \twoheadrightarrow t' \to t$. By the definition of descendants and origins, it follows immediately from the non-emptiness of $(\{p\}\searrow(s \twoheadrightarrow t)) \cap Q$ that $(\{p\}\searrow(s \twoheadrightarrow t')) \cap ((t' \to t)\nwarrow Q)$ is non-empty. Hence, by the induction hypothesis, $p \in (s \twoheadrightarrow t')\nwarrow((t' \to t)\nwarrow Q) = (s \twoheadrightarrow t)\nwarrow Q$, as required.

If α is a limit ordinal, then, by the definition of descendants and origins and strong convergence, we have that $(\{p\}\searrow(s \twoheadrightarrow t_\beta)) \cap ((t_\beta \twoheadrightarrow t)\nwarrow Q)$ is non-empty, with $\beta < \alpha$ such that each step in $t_\beta \twoheadrightarrow t$ occurs a position greater than $|q|$ for all $q \in Q$. Hence, by the induction hypothesis we have that $p \in (s \twoheadrightarrow t_\beta)\nwarrow Q = (s \twoheadrightarrow t)\nwarrow Q$. □

8. Computability of Descendants and Origins

Having defined descendants, origins, and the related notion of needed reductions, we now show uniform computability of all these notions with respect to computably strongly convergent reductions. As computability of descendants depends on the computability of needed steps (which are defined by means of origins), we start by considering origins and needed reductions.

8.1. Computability of Origins and Needed Reductions

We first prove that origins are uniformly computable.

Lemma 8.1. There exists a Turing machine M with the following behaviour:

Input – an r.e. iTRS R, a computably strongly convergent reduction $t_0 \twoheadrightarrow_R t_\alpha$, and a finite set $Q \subseteq \mathcal{P}os(t_\alpha)$;

Output – the finite set $(t_0 \twoheadrightarrow t_\alpha)\nwarrow Q$.

Proof:
Compute $h = \max\{|p| \mid p \in Q\}$, which is possible as Q is finite, and employ the modulus of convergence of $t_0 \twoheadrightarrow t_\alpha$ to compute for h an ordinal β satisfying the conditions from Definition 7.3. As in Remark 6.6, compute the largest limit ordinal γ smaller than or equal to β (or 0 if no such ordinal exists). To compute $(t_0 \twoheadrightarrow t_\alpha)\nwarrow Q = (t_0 \twoheadrightarrow t_\gamma)\nwarrow((t_\gamma \to^* t_\beta)\nwarrow Q)$, first compute the finite set $Q' = (t_\gamma \to^* t_\beta)\nwarrow Q$, which is possible in finite time as the reduction is finite and as rules have finite left-hand sides. If $\gamma = 0$, we are done. Otherwise, if $\gamma > 0$, repeat with Q' in the place of Q and $t_0 \twoheadrightarrow t_\gamma$ in the place of $t_0 \twoheadrightarrow t_\alpha$, where we have for the modulus of convergence φ' of $t_0 \twoheadrightarrow t_\gamma$

that $\varphi'(0,n) = \varphi(\gamma,n)$ and $\varphi'(\kappa,n) = \varphi(\kappa,n)$ otherwise, with φ the modulus of convergence of $t_0 \twoheadrightarrow t_\alpha$. Only a finite number of repetitions can occur as there are no infinite descending chains of ordinals. □

Closely following the above proof, we can also show that needed steps are uniformly computable.

Lemma 8.2. There exists a Turing machine M with the following behaviour:

Input – an r.e. iTRS R, a computably strongly convergent reduction $t_0 \twoheadrightarrow_R t_\alpha$, and a finite, prefix-closed set $Q \subseteq \mathcal{P}os(t_\alpha)$;

Output – (i) the finite subsequence of needed steps for Q, and (ii) the length of the subsequence.

Proof:
Observe that $(t_\beta \twoheadrightarrow t_\alpha)\restriction Q = Q$ in case all steps in $t_\beta \twoheadrightarrow t_\alpha$ occur at positions p such that $|p| > |q|$ for all $q \in Q$. Hence, it suffices to consider $t_0 \twoheadrightarrow t_\beta$, as no needed steps occur in $t_\beta \twoheadrightarrow t_\alpha$, where β can be computed using the modulus of convergence employing the fact that Q is finite. As in Remark 6.6, compute the largest limit ordinal γ smaller than or equal to β (or 0 if no such ordinal exists). For each $\gamma \leq \delta < \beta$, compute $(t_\delta \twoheadrightarrow t_\alpha)\restriction Q$ by computing $(t_\delta \to^* t_\beta)\restriction Q$, which is possible in finite time as the reduction is finite and as rules have finite left-hand sides. As each $(t_\delta \twoheadrightarrow t_\alpha)\restriction Q$ is finite, we can compute whether $(p_\delta, l_\delta \to r_\delta)$ is needed. Next, if $\gamma = 0$, we are done. Otherwise, repeat with $Q' = (t_\gamma \twoheadrightarrow t_\alpha)\restriction Q$ in the place of Q and $t_0 \twoheadrightarrow t_\gamma$ in the place of $t_0 \twoheadrightarrow t_\alpha$, where we have for the modulus of convergence φ' of $t_0 \twoheadrightarrow t_\gamma$ that $\varphi'(0,n) = \varphi(\gamma,n)$ and $\varphi'(\kappa,n) = \varphi(\kappa,n)$ otherwise, with φ the modulus of convergence of $t_0 \twoheadrightarrow t_\alpha$. Only a finite number repetitions can occur as there are no infinite descending chains of ordinals. □

With the above lemma in hand, we can show that needed reductions are uniformly computable.

Proposition 8.3. There exists a Turing machine M with the following behaviour:

Input – a left-linear r.e. iTRS R, a computably strongly convergent reduction $s \twoheadrightarrow_R t$, and a finite, prefix-closed set $Q \subseteq \mathcal{P}os(t)$;

Output – the needed reduction $s \to_R^* t'$ for $s \twoheadrightarrow t$ and Q.

Proof:
The finite subsequence of needed steps is computable by Lemma 8.2, and, hence, the needed reduction is of finite length and can be computed by means of the Turing machine of Lemma 5.3. □

8.2. Computability of Descendants

We next show that descendants are uniformly computable. Observe that uniform computability requires us to somehow restrict the sets of positions we consider, as infinite sets are in general not finitely representable. To this end, we restrict ourselves to recursive sets of positions. This mirrors

the situation for origins in the sense that it is decidable for both finite and recursive sets of positions whether a position occurs in the set.

To facilitate our presentation below, we ensure that the considered sets of positions are defined by so-called *position functions*.

Definition 8.4. Let s be a term. A *position function π of s* is a function $\pi : \mathbb{N} \longrightarrow \wp(\mathcal{P}os(s))$, with $\wp(\mathcal{P}os(s))$ the power set of $\mathcal{P}os(s)$, such that $|q| = n$ for all $q \in \pi(n)$ and $n \in \mathbb{N}$.

Observe that $\pi(n)$ is finite for all n and that $|\pi(n)|$ is uniformly computable in n if π is computable.

For each recursive set of positions of a computable term t there exists a computable position function, and vice versa: in either case proceed by a breadth-first search over t starting from the root. By a similar argument, there exists for each computable term t and variable x a computable position function π_x such that $\bigcup_{n\in\mathbb{N}} \pi_x(n) = \{p \mid t(p) = x\}$.

We now prove that descendants across rewrite steps are uniformly computable.

Proposition 8.5. There exists a Turing machine M with the following behaviour:

Input – an r.e. iTRS R, a computable reduction step $s \to_R t$, and a computable position function π of s with $\bigcup_{n\in\mathbb{N}} \pi(n) = Q$ for some $Q \subseteq \mathcal{P}os(s)$;

Output – a computable position function π' of t with $\bigcup_{n\in\mathbb{N}} \pi'(n) = Q\backslash(s \to t)$.

Proof:
Let $l \to r$ be the rewrite rule employed in $s \to t$ and observe, by the definition of descendants, that for every descendant $q' \in \mathcal{P}os(t)$ of a position $q \in \mathcal{P}os(s)$ we have that $|q'| \geq |q| - h$ where $h = \max\{|p| \mid l(p) \in \Sigma\}$. Hence, to compute $\pi'(n)$ for a given $n \in \mathbb{N}$ it suffices to consider the finite set of positions $Q_n = \bigcup_{n'\leq n+h} \pi(n')$, which can be computed as π is computable. By definition of descendants and the fact that a computable position function π'_x exists for each variable x in r, there also exists a computable position function π'_q such that $\bigcup_{n\in\mathbb{N}} \pi'_q(n) = q\backslash(s \to t)$ for each $q \in \mathcal{P}os(s)$. Hence, $\pi'(n)$ can be computed by first computing the finite set Q_n and then taking $\bigcup_{q\in Q_n} \pi'_q(n)$. □

Proposition 8.5 immediately implies that the descendants of sets of positions (represented by computable position functions) are computable across finite reductions. Employing needed steps, we can also show that descendants are computable across computably strongly convergent reductions.

Lemma 8.6. There exists a Turing machine with the following behaviour:

Input – an r.e. iTRS R, a computably strongly convergent reduction $s \twoheadrightarrow_R t$, and a computable position function π of s with $\bigcup_{n\in\mathbb{N}} \pi(n) = P$ for some $P \subseteq \mathcal{P}os(s)$;

Output – a computable position function π' of t with $\bigcup_{n\in\mathbb{N}} \pi'(n) = Q\backslash(s \twoheadrightarrow t)$.

Proof:
Observe that the definition of descendants depends on the positions of the contracted redexes and the employed rewrite rules, but not on the actual terms that occur along reductions. Now, for each $n \in \mathbb{N}$, compute $\pi'(n)$ by computing a computable position function π'_n such that $\bigcup_{n\in\mathbb{N}} \pi'_n(n)$ is the set of descendants of Q across the steps of $s \twoheadrightarrow t$ needed for the finite, prefix-closed set of positions $\{q \in \mathcal{P}os(t) \mid |q| \leq n\}$, and define $\pi'(n) = \pi'_n(n)$.

Considering the needed steps for $\{q \in \mathcal{P}os(t) \mid |q| \leq n\}$ suffices by Proposition 7.11 and the definition of needed steps, although the needed steps do not necessarily form a reduction, as R is not assumed to be left-linear. Each position function π'_n can be computed by repeated application of the Turing machine of Proposition 8.5. □

9. Conclusion and Future Work

We have defined computable infinitary rewriting by introducing concepts from recursion theory and computable analysis to the domain of strongly convergent infinitary rewriting. We have also shown that descendants and origins—used to track redexes forward and backward across reductions—are uniformly computable for computable infinitary reductions. Our work is the first to comprehensively treat computability of both infinite terms and infinite reductions; all our definitions and results use classical notions from computability theory with no extra assumptions.

We believe that our results lay the groundwork for establishing computable versions of the usual compression and confluence theorems from strongly convergent infinitary rewriting [5, 33]. Furthermore, we believe that the following two questions warrant further study:

- How can computational *complexity* for infinitary rewriting be defined? What are natural problems or transformations on infinite terms that can be computed in polynomial time? A starting point could be a lifting of similar notions from computable analysis [45, 2].
- Can suitable *subclasses* of computable infinite terms or computable convergent reductions be defined that have nice closure properties? The set of rational terms affords one such class, but other computationally limited notions might be possible, e.g., an 'infinite tree analogue' of automatic sequences [46].

Acknowledgements The authors thank Patrick Bahr and the reviewers of various drafts for their constructive comments. Figure 2 was inspired by Figure 3.3 of Patrick Bahr's PhD thesis [47].

References

[1] Klop JW, van Oostrom V, de Vrijer R. Orthogonality. In: Term Rewriting Systems [28], Chapter 4, pp. 88–148, 2003.

[2] Weihrauch K. Computable Analysis: An Introduction. Springer, New York, 2000. ISBN 978-3642569999.

[3] Tucker JV, Zucker JI. Theory of Computation over Stream Algebras, and its Applications. In: Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science (MFCS'92),

volume 629 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1992 pp. 62–80. doi:0.1007/3-540-55808-X_6.

[4] Barwise J. Infinitary Logic and Admissible Sets. *Journal of Symbolic Logic*, 1969. **34**(2):226–252. doi:10.2307/2271099.

[5] Kennaway R, Klop JW, Sleep R, de Vries FJ. Transfinite Reductions in Orthogonal Term Rewriting Systems. *Information and Computation*, 1995. **119**(1):18–38. doi:10.1006/inco.1995.1075.

[6] Simonsen JG. On modularity in infinitary term rewriting. *Information and Computation*, 2006. **204**(6):957–988. doi:10.1016/j.ic.2006.02.005.

[7] Kahrs S. Modularity of Convergence and Strong Convergence in Infinitary Rewriting. *Logical Methods in Computer Science*, 2010. **6**(3:18):1–27. doi:10.2168/LMCS-6(3:18)2010.

[8] Klop JW, de Vrijer RC. Infinitary Normalization. In: We Will Show Them! Essays in Honour of Dov Gabbay, volume 2. College Publications, London. ISBN 978-1904987123, 2005 pp. 169–192.

[9] Zantema H. Normalization of Infinite Terms. In: Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA 2008), volume 5117 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2008 pp. 441–455. doi:10.1007/978-3-540-70590-1_30.

[10] Kennaway R, Klop JW, Sleep MR, de Vries FJ. Infinitary Lambda Calculus. *Theoretical Computer Science*, 1997. **175**(1):93–125. doi:10.1016/S0304-3975(96)00171-5.

[11] Barendregt H, Klop JW. Applications of infinitary lambda calculus. *Information and Computation*, 2009. **207**(5):559–582. doi:10.1016/j.ic.2008.09.003.

[12] Ketema J, Simonsen JG. Infinitary Combinatory Reduction Systems. *Information and Computation*, 2011. **209**(6):893–926. doi:10.1016/j.ic.2011.01.007.

[13] Ketema J, Simonsen JG. Infinitary Combinatory Reduction Systems: Normalising Reduction Strategies. *Logical Methods in Computer Science*, 2010. **6**(1:7):1–35. doi:10.2168/LMCS-6(1:7)2010.

[14] Ketema J, Simonsen JG. Infinitary Combinatory Reduction Systems: Confluence. *Logical Methods in Computer Science*, 2009. **5**(4:3):1–29. doi:10.2168/LMCS-5(4:3)2009.

[15] Bahr P. Infinitary Term Graph Rewriting is Simple, Sound and Complete. In: Proceedings of the 23rd International Conference on Rewriting Techniques and Applications (RTA 2012), volume 15 of *Leibniz International Proceedings in Informatics*. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, Dagstuhl, 2012 pp. 69–84. doi:10.4230/LIPIcs.RTA.2012.69.

[16] Bahr P. Modes of Convergence for Term Graph Rewriting. *Logical Methods in Computer Science*, 2012. **8**(2:6):1–60. doi:10.2168/LMCS-8(2:6)2012.

[17] Kahrs S. Infinitary rewriting: meta-theory and convergence. *Acta Informatica*, 2007. **44**:91–121. doi:10.1007/s00236-007-0043-2.

[18] Kahrs S. Infinitary Rewriting: Foundations Revisited. In: Proceedings of the 21st International Conference on Rewriting Techniques and Applications (RTA 2010), volume 6 of *Leibniz International Proceedings in Informatics*. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, Dagstuhl, 2010 pp. 161–176. doi:10.4230/LIPIcs.RTA.2010.161.

[19] Bahr P. Abstract Models of Transfinite Reductions. In: Proceedings of the 21st International Conference on Rewriting Techniques and Applications (RTA 2010), volume 6 of *Leibniz International Proceedings in Informatics*. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, Dagstuhl, 2010 pp. 49–66. doi:10.4230/LIPIcs.RTA.2010.49.

[20] Inverardi P, Zilli MV. Rational Rewriting. In: Proceedings of the 19th International Symposium on Mathematical Foundations of Computer Science (MFCS'94), volume 841 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994 pp. 433–442. doi:10.1007/3-540-58338-6_90.

[21] Corradini A, Gadducci F. Rational Term Rewriting. In: Proceedings of the 1st International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'98), volume 1378 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998 pp. 156–171. doi:10.1007/BFb0053548.

[22] Corradini A, Drewes F. Term Graph Rewriting and Parallel Term Rewriting. In: Proceedings of the 6th International Workshop on Computing with Terms and Graphs (TERMGRAPH 2011), volume 48 of *Electronic Proceedings in Theoretical Computer Science*. Open Publishing Association, Australia, 2011 pp. 3–18. doi:10.4204/EPTCS.48.3.

[23] Aoto T, Ketema J. Rational Term Rewriting Revisited: Decidability and Confluence. In: Proceedings of the 6th International Conference on Graph Transformations (ICGT 2012), volume 7562 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2012 pp. 172–186. doi:10.1007/978-3-642-33654-6_12.

[24] Vermaat M. Infinitary Rewriting in Coq. Master's thesis, Vrije Universiteit, Amsterdam, 2010.

[25] Endrullis J, Hansen HH, Hendriks D, Polonsky A, Silva A. Coinductive Foundations of Infinitary Rewriting and Infinitary Equational Logic. *Logical Methods in Computer Science*, 2018. **14**(1:3):1–44. doi:10.23638/LMCS-14(1:3)2018.

[26] Baader F, Nipkow T. Term Rewriting and All That. Cambridge University Press, Cambridge, 1998. ISBN 978-0521779203.

[27] Klop JW. Term Rewriting Systems. In: Abramsky S, Gabbay D, Maibaum T (eds.), Handbook of Logic in Computer Science, volume 2, pp. 1–116. Oxford University Press, Oxford. ISBN 978-0198537618, 1992.

[28] Terese. Term Rewriting Systems, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 2003. ISBN 978-0521391153.

[29] Fernàndez M. Models of Computation: An Introduction to Computability Theory. Undergraduate Topics in Computer Science. Springer, New York, 2009. ISBN 978-1848824331.

[30] Jones ND. Computability and Complexity from a Programming Perspective. The MIT Press, Cambridge, MA, 1997. ISBN 978-0262100649.

[31] Rogers Jr H. Theory of Recursive Functions and Effective Computability. The MIT Press, Cambridge, MA, paperback edition, 1987. ISBN 978-0262680523.

[32] Sipser M. Introduction to the Theory of Computation. Thomson Course Technology, Boston, 2nd edition, 2006. ISBN 978-0534950972.

[33] Kennaway R, de Vries FJ. Infinitary rewriting. In: Term Rewriting Systems [28], Chapter 12, pp. 668–711, 2003.

[34] Ketema J. Böhm-Like Trees for Rewriting. Ph.D. thesis, Vrije Universiteit, Amsterdam, 2006.

[35] Courcelle B. Fundamental Properties of Infinite Trees. *Theoretical Computer Science*, 1983. **25**(2):95–169. doi:10.1016/0304-3975(83)90059-2.

[36] Kleene SC. Recursive Functions and Intuitionistic Mathematics. In: Proceedings of the International Conference of Mathematicians, August 30–September 6, 1950, volume 1. American Mathematical Society, Providence, RI, USA, 1952 pp. 679–685.

[37] Bauer A. König's Lemma and the Kleene Tree, 2006. Unpublished tutorial note.

[38] Bridges D, Richman F. Varieties of Constructive Mathematics, volume 97 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, Cambridge, 1987. ISBN 978-0521318020.

[39] Rodenburg PH. Termination and Confluence in Infinitary Term Rewriting. *Journal of Symbolic Logic*, 1998. **63**(4):1286–1296. doi:10.2307/2586651.

[40] Miller LW. Normal Functions and Constructive Ordinal Notations. *Journal of Symbolic Logic*, 1976. **41**(2):439–459. doi:10.2307/2272243.

[41] Kleene SC. On Notation for Ordinal Numbers. *Journal of Symbolic Logic*, 1938. **3**(4):150–155. doi: 10.2307/2267778.

[42] Phillips IC. Recursion Theory. In: Abramsky S, Gabbay DM, Maibaum TSE (eds.), Handbook of Logic in Computer Science, volume 1, pp. 79–188. Oxford University Press, Oxford. ISBN 978-0198537359, 1992.

[43] Bethke I, Klop JW, de Vrijer RC. Descendants and Origins in Term Rewriting. *Information and Computation*, 2000. **159**(1–2):59–124. doi:10.1006/inco.2000.2876.

[44] Ketema J. Reinterpreting Compression in Infinitary Rewriting. In: Proceedings of the 23th International Conference on Rewriting Techniques and Applications (RTA 2012), volume 15 of *Leibniz International Proceedings in Informatics*. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, Dagstuhl, 2012 pp. 209–224. doi:10.4230/LIPIcs.RTA.2012.209.

[45] Ko KI. Complexity Theory of Real Functions. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991. ISBN 978-1468468045.

[46] Allouche JP, Shallit J. Automatic Sequences: Theory, Application, Generalizations. Cambridge University Press, Cambridge, 2003. ISBN 978-0521823326.

[47] Bahr P. Modular Implementation of Programming Languages and a Partial-Order Approach to Infinitary Rewriting. Ph.D. thesis, University of Copenhagen, 2012.