Faculty of Mathematics, Computer Science and Natural Sciences Research Group Computer Science 2

RWTH Aachen University, Germany

Bachelor Thesis

A Term Encoding to Analyze Runtime Complexity via Innermost Runtime Complexity

submitted by

Simeon Valchev

Coming soon

Reviewers
Prof. Dr. Jürgen Giesl
apl. Prof. Dr. Thomas Noll

Supervisor
Stefan Dollase, M.Sc.

Statutory Declaration in Lieu of an Oath

Abstract

The automation of complexity analysis of term rewrite systems (TRSs) is quite a difficult problem. The tools currently in use to analyze full runtime complexity of TRS can be further improved, as is shown in the results of the Termination Competition. Comparatively, analyzing innermost runtime complexity is stronger, creating the opportunity to use its techniques to infer upper bounds on full runtime complexity. A TRS can be encoded into a new relative TRS, such that its innermost runtime complexity equals the full runtime complexity of the original, allowing us to use these stronger techniques for a different type of analysis. Our approach was formalized, partially proven and implemented. It was tested manually to show an improvement in the full runtime complexity analysis of some examples.

Contents

1. Introduction

Term rewrite systems(TRSs) offer a theoretical foundation to which computer programs can be abstracted to and analyzed. One of the topics of research concerning TRSs has to deal with the worst-case lengths of rewrite sequences, more commonly known as complexity. In its analysis a distinction is made based on the starting term of the rewrite sequence: *derivational complexity*, considers any term, and *runtime complexity*, considers only basic terms. The more intuitive notion is for a function to be called with some data objects as inputs, which is analogous to the evaluation of basic terms, and is what we focus on. A further distinction is made according to the evaluation strategy. There is *innermost* runtime complexity (irc), whose eager strategy never evaluates a term before its subterms are in normal form, i.e. can no longer be evaluated, and *full* runtime complexity (rc), which utilizes any strategy.

Most of the work in the field has been focused on irc, which is closer to the evaluation of imperative programs. While the topic of rc may not be as well studied, some notable papers on it are [1, 2, 3] with the most recent being [4], which introduced a novel technique on inferring upper bounds for rc by analyzing irc. However, there is still a wide gap in the automatic complexity analysis on rc as can be observed in the results of the annual *Termination Competition* [5]. Of the 959 TRSs analyzed for rc, an upper bound was discover by at least one of the participants in only 345 cases (36%).

The technique in [4] identifies some TRSs as non-dup generalized (ndg) and shows they have the property $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}}$. This combined with the much more powerful analysis of irc allows inferring upper bounds on rc from upper bounds on irc. All non-ndg TRSs are ignored by this method, leaving a gap, which this thesis aims to fill. This is done by extending the technique from [4] by encoding non-ndg TRSs in a way that $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\Phi}(\mathcal{R})$, where $\Phi(\mathcal{R})$ is the result of the encoding of $\mathcal{R}$. The two main problems in defining the encoding are calculating the set of non-ndg positions in a TRS and determining if the addition of non-terminating rules is necessary. The latter will make more sense when we dive deeper into the subject. Both of these problems are tackled in this thesis, but also leave room for improvement in terms of precision. A remaining problem not discussed here is the analysis of the encoded TRS is confined to the starting terms of the original TRS.

The structure of the thesis is as follows. Chapter 2 introduces some of the preliminary knowledge regarding TRSs. Chapter 3 deals with the problem of calculating the non-ndg positions of a TRS. These are later encoded in our procedure, presented in Chapter 4, which also includes a proof of its usefulness. Chapter 5 introduces some new techniques to improve the overall encoding and its analysis by limiting the introduction of non-terminating rules. Chapter 5 serves as a summary and a look at future work.

2. Preliminaries

In this chapter is introduced some of the necessary groundwork on term rewrite systems with accompanying examples.

For now a term can be viewed as some object. Rewriting is then simply a transformation of one term into another. This rewriting is guided by rules, which describe what the input and output of the transformation is. Rules are also associated with some cost, which is considered when analyzing runtime complexity. Unless stated otherwise all rules have a cost of *one*. Rules can also have conditions besides what input they take, however such conditional rules are not in the focus of this thesis. A term rewrite system is a set of rules.

Example 1 (Addition)**.**

$$\begin{aligned}\alpha_1 &: \mathsf{add}(\mathsf{x},\mathsf{0}) &\to\ & \mathsf{x}\\ \alpha_2 &: \mathsf{add}(\mathsf{x},\mathsf{s}(\mathsf{y})) &\to\ & \mathsf{add}(\mathsf{s}(\mathsf{x}),\mathsf{y})\end{aligned}$$

Example 1. shows the simple TRS $\mathcal{R}_1$ for the calculation of addition on natural numbers. Numbers here are represented by the so-called successor function and the symbol 0, i.e. 1 would be $\mathsf{s}(\mathsf{0})$, 2 would be $\mathsf{s}(\mathsf{s}(\mathsf{0}))$ and so on. To enhance readability, it is denoted $\mathsf{f}^\mathsf{n}(\mathsf{x})$, when some function f is applied n-many times to an input x.

The base case would be represented in α_1, where $x + 0 = x$. In α_2 the TRS recursively subtracts 1 from the second position and adds 1 to the first position. This repeats until the second position reaches 0 and thus the first position is output. One could intuitively imagine this as follows:

$$\begin{array}{ccccccc} 3+2 & = & 4+1 & = & 5+0 & = & 5 \\ \mathsf{add}(\mathsf{s}^3(\mathsf{0}),\mathsf{s}^2(\mathsf{0})) & \to_{\mathcal{R}_1} & \mathsf{add}(\mathsf{s}^4(\mathsf{0}),\mathsf{s}(\mathsf{0})) & \to_{\mathcal{R}_1} & \mathsf{add}(\mathsf{s}^5(\mathsf{0}),\mathsf{0}) & \to_{\mathcal{R}_1} & \mathsf{s}^5(\mathsf{0}) \end{array}$$

While not explicitly stated in the example, x and y are variables which can be instantiated with other terms. Without a proper definition, one could assume they are some constants like $\mathsf{0}$. Therefore each TRS is associated with a set, which holds all function symbols.

Definition 2 (Signature [6])**.**

A **signature** Σ is a set of function symbols. The number of inputs n of each function symbol is referred to as its arity and it cannot be negative. The subset $\Sigma^{(n)} \subseteq \Sigma$ holds all symbols with arity n. Symbols with arity 0 are called constant.

The signature of $\mathcal{R}_1$ is therefore $\{\mathsf{add},\mathsf{s},\mathsf{0}\}$ with the accompanying $\Sigma^{(2)} = \{\mathsf{add}\}$, $\Sigma^{(1)} = \{\mathsf{s}\}$ and $\Sigma^{(0)} = \{\mathsf{0}\}$, whereas x and y are variables.

Definition 3 (Terms [6])**.**

Let Σ be a signature and $\mathcal{V}$ a set of variables such that $\Sigma \cap \mathcal{V} = \emptyset$. The set $\mathcal{T}(\Sigma, \mathcal{V})$ of all **terms** over Σ and $\mathcal{V}$ is the smallest set such that:

- $\mathcal{V} \subseteq \mathcal{T}(\Sigma, \mathcal{V})$
- for all $n \geq 0$, all $f \in \Sigma^{(n)}$ and all $t_1, ..., t_n \in \mathcal{T}(\Sigma, \mathcal{V})$, we have $f(t_1, ..., t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$

The first item states that each variable on its own is a term and the second item specifies that for arbitrary n many terms $t_1, ..., t_n$ and a function symbol f with arity n, $f(t_1, ..., t_n)$ is also a term. $\mathcal{T}(\Sigma, \mathcal{V})$ is denoted as $\mathcal{T}$, if Σ and $\mathcal{V}$ are irrelevant or clear from the context [4] .

Given the signature $\Sigma = \{\mathsf{add}, \mathsf{s}, \mathsf{0}\}$ of $\mathcal{R}_1$, we can construct terms like $\mathsf{s}(\mathsf{0})$ or $\mathsf{add}(\mathsf{x}, \mathsf{y})$ or more complex ones like $\mathsf{add}(\mathsf{add}(\mathsf{x}, \mathsf{s}(\mathsf{s}(\mathsf{0}))), \mathsf{s}(\mathsf{0}))$.

Definition 4 (Term positions [6, 4])**.**

1. Let Σ be a signature, $\mathcal{V}$ be a set of variables with $\Sigma \cap \mathcal{V} = \emptyset$ and s some term in $\mathcal{T}(\Sigma, \mathcal{V})$. The set of **positions** of s contains sequences of natural numbers, denoted $pos(s)$ and inductively defined as follows:

   - If $s = x \in \mathcal{V}$, then $pos(s) := \{\varepsilon\}$
   - If $s = f(s_1, ..., s_n)$, then

   $$pos(s) := \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{i.\pi \mid \pi \in pos(s_i)\}$$

   The symbol ε is used to point to the **root position** of a term. The function symbol at that position is called the **root symbol** denoted by $root(s)$. Positions can be compared to each other

   $$\pi \leq \tau, \text{ iff there exists } \pi' \text{ such that } \pi.\pi' = \tau$$

   Positions for which neither $\pi \leq \tau$, nor $\pi \geq \tau$ hold, are called **parallel positions** denoted by $\pi \parallel \tau$.

2. The **size** of a term s is defined as $|s| := |pos(s)|$.

3. For $\pi \in pos(s)$, the **subterm of** s **at position** π denoted by $s|_\pi$ is defined as

   $$\begin{aligned} s|_\varepsilon &:= s \\ f(s_1, ..., s_n)|_{i.\pi} &:= s_i|_\pi \end{aligned}$$

   If $\pi \neq \varepsilon$, then $s|_\pi$ is a **proper subterm** of s.

4. For $\pi \in pos(s)$, the **replacement in** s **at position** π **with term** t denoted by $s[t]_\pi$ is defined as

   $$\begin{aligned} s[t]_\varepsilon &:= t \\ f(s_1, ..., s_n)[t]_{i.\pi} &:= f(s_1, ..., s_i[t]_\pi, ..., s_n) \end{aligned}$$

5. The set of **variables occurring in** s, denoted $\mathcal{V}ar(s)$ is defined as

   $$\mathcal{V}ar(s) := \{x \in \mathcal{V} \mid \exists \pi \in pos(s) \text{ such that } s|_\pi = x\}$$

Consider the term $t = \mathsf{add}(\mathsf{x}, \mathsf{s}(\mathsf{y}))$. The set of position is $pos(t) = \{\varepsilon, 1, 2, 2.1\}$. Some statements that hold for example are $\varepsilon \leq 2 \leq 2.1$ and also $1 \parallel 2$. Some expressions to illustrate are $t|_2 = \mathsf{s}(\mathsf{y})$ and $t[s(0)]_1 = \mathsf{add}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{y}))$.

Definition 5 (Substitution [6])**.**

Let Σ be a signature and $\mathcal{V}$ a finite set of variables. A **substitution** is a function $\sigma : \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V})$ such that $\sigma(x) \neq x$ holds for only finitely many xs. It can be written as

$$\sigma = \{x_1 \mapsto \sigma(x_1)\ ,\ \cdots\ ,\ x_n \mapsto \sigma(x_n)\}\ .$$

- The term resulting from $\sigma(t)$ is called an **instance** of t. We also write $t\sigma$ instead of $\sigma(t)$.

For example let $\sigma(x) = a$ and $t = \mathsf{f}(x)$, then $t\sigma = \mathsf{f}(a)$.

Definition 6 (Term rewrite systems [4])**.**

A **rewrite rule** is a pair of terms $\ell \to r$ such that ℓ is not a variable and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell)$. It is referred to ℓ as the left-hand side(lhs) and r as the right-hand side(rhs). A **term rewrite system** is a finite set of such rewrite rules combined with a signature Σ.

1. Given the signature Σ of a TRS $\mathcal{R}$ we have
   - the **set of defined symbols** $\Sigma_d := \{root(\ell) \mid \ell \to r \in \mathcal{R}\}$
   - the **set of constructor symbols** $\Sigma_c := \Sigma \setminus \Sigma_d$
2. A term $f(s_1, ..., s_n)$ is **basic**, if $f \in \Sigma_d$ and $s_1, ..., s_n \in \mathcal{T}(\Sigma_c, \mathcal{V})$. The **set of all basic terms** over Σ and $\mathcal{V}$ is denoted by $\mathcal{T}_B(\Sigma, \mathcal{V})$.
3. The **number of occurrences of** x **in term** t is denoted by $\#_x(t)$.
4. A **redex** (**red**ucible **ex**pression) is an instance of ℓ of some rule $\ell \to r$.
   - A term t is an **innermost redex**, if none of its proper subterms is a redex.
   - A term t is in **normal form**, if none of its subterms is a redex.

For $\mathcal{R}_1$ in example 1. we have $\Sigma_d = \{\mathsf{add}\}$ and $\Sigma_c = \{\mathsf{0}, \mathsf{s}\}$. Consider the term $s = \mathsf{add}(\mathsf{add}(0,0),0)$. With a substitution $\sigma = \{\mathsf{x} \mapsto \mathsf{add}(0,0)\}$ it holds that s is an instance of the left-hand side of rule α_1 and therefore s is a redex. But it is not an innermost one as the subterm $s|_1 = \mathsf{add}(0,0)$ can be reduced to $\mathsf{0}$. Since none of the subterms of $s|_1$ can be reduced, it is an innermost redex.

Definition 7 (Rewrite step [4])**.**

A **rewrite step** is a reduction(or evaluation) of a term s to t by applying a rule $\ell \to r$ at position π, denoted by $s \to_{\ell \to r, \pi} t$ and means that for some substitution σ, $s|_\pi = \sigma(\ell)$ and $t = s[\sigma(r)]_\pi$ holds.

1. Rule and position in the subscript can be omitted, if they are irrelevant. We denote $s \to_{\mathcal{R}} t$, if it holds for some rule in $\mathcal{R}$, and in order to distinguish between rules and rewrite steps.
2. A sequence of rewrite steps(or rewrite sequence) $s = t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \cdots \to_{\mathcal{R}} t_m = t$ is denoted by $s \to^m_{\mathcal{R}} t$.
3. Only rules with cost 1 contribute to the length of a rewrite sequence, i.e. for $s \to^m_{\mathcal{R}} t$ it holds that $m = 0$, if it is a single rewrite step that uses a rule with 0 cost.
4. A rewrite step is called **innermost**, if $s|_\pi$ is an innermost redex, and is denoted by $s \overset{i}{\to}_\pi t$

The term $s = \mathsf{add}(\mathsf{add}(0,0),0)$ mentioned above can be used to create the following sequence in $\mathcal{R}_1$

$$\mathsf{add}(\mathsf{add}(0,0),0) \to_{\mathcal{R}_1} \mathsf{add}(0,0) \to_{\mathcal{R}_1} \mathsf{0}$$

Definition 8 (Derivation height [4]).

The **derivation height** $dh : \mathcal{T} \times 2^{\mathcal{T} \times \mathcal{T}} \to \mathbb{N} \cup \{\omega\}$ is a function with two inputs: a term t and a binary relation on terms, in this case $\to$. It defines the length of the longest sequence of rewrite steps starting with term t, i.e.

$$dh(t, \to) = \sup\{m \mid \exists t' \in \mathcal{T} \ , \ t \to^m t'\}$$

In the case of an infinitely long rewrite sequence starting with term s, it is denoted as $dh(s, \to) = \omega$.

Definition 9 ((Innermost) Runtime complexity [4]).

The **runtime complexity**(rc) of a TRS $\mathcal{R}$ maps any $n \in \mathbb{N}$ to the length of the longest $\to$-sequence (rewrite sequence) starting with a basic term t with $|t| \leq n$. The innermost runtime complexity(irc) is defined analogously, but it only considers innermost rewrite steps. More precisely $\text{rc}_{\mathcal{R}} : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ and $\text{irc}_{\mathcal{R}} : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$, defined as

$$\text{rc}_{\mathcal{R}}(n) = \sup\{dh(t, \to_{\mathcal{R}}) \mid t \in \mathcal{T}_B, |t| \leq n\}$$
$$\text{irc}_{\mathcal{R}}(n) = \sup\{dh(t, \overset{i}{\to}_{\mathcal{R}}) \mid t \in \mathcal{T}_B, |t| \leq n\}$$

Since every $\overset{i}{\to}$-sequence can be viewed as a $\to$-sequence, it is clear that $\text{irc}_{\mathcal{R}}(n) \leq \text{rc}_{\mathcal{R}}(n)$. Therefore, an upper bound for $\text{rc}_{\mathcal{R}}$ infers an upper bound for $\text{irc}_{\mathcal{R}}$ and a lower bound for $\text{irc}_{\mathcal{R}}$ infers a lower bound for $\text{rc}_{\mathcal{R}}$.

For now we can say that the length of rewrite sequences in $\mathcal{R}_1$ depends solely on the second argument of the starting basic term. The size of this subterm is decremented by one after each rewrite until it reaches $\mathsf{0}$, terminating after one more rewrite. $\mathcal{R}_1$ also belongs to a class of system, for which $\text{rc}_{\mathcal{R}} = \text{irc}_{\mathcal{R}}$. This class was the focus of [4] and we aim to go beyond it in this thesis.

Definition 10 (Non-dup-generalized (ndg) rewrite step [4, 7]).

The rewrite step $s \to_{\ell \to r, \pi} t$ with the matching substitution σ is called **non-dup-generalized**(ndg), if

- For all variables x with $\#_x(r) > 1$, $\sigma(x)$ is in normal form, and
- For all $\tau \in pos(\ell) \backslash \{\varepsilon\}$ with $root(\ell|_\tau) \in \Sigma_d$, it holds that $\sigma(\ell|_\tau)$ is in normal form.

A TRS $\mathcal{R}$ is called ndg, if every rewrite sequence starting with a basic term consists of only ndg rewrite steps. A **non-ndg** rewrite step does not fulfill one of the criteria and a TRS is non-ndg, if it produces a sequence with a non-ndg rewrite step.

The first condition regards the duplication of redexes, which obviously affects the runtime complexity and is by definition unreachable via an innermost evaluation strategy. The second condition regards nested redexes whose root symbol is matched. This means a function call is ignored, which once again can affect the complexity in unexpected ways.

One can now easily observe that the TRS $\mathcal{R}_1$ is indeed ndg, since none of its rules duplicate variables on the rhs or have nested defined symbols on the lhs. It is impossible for a rewrite sequence starting with a basic term to reach a non-ndg rewrite step. Therefore it holds that $\text{rc}_{\mathcal{R}_1} = \text{irc}_{\mathcal{R}_1}$. However, we are interested in non-ndg TRSs, so let us take a look at one.

Example 11.

Consider the non-ndg TRS $\mathcal{Q}$, which simply counts down given some input:

$$\begin{array}{ll}
\alpha_1 : \mathsf{f}(x) \to \mathsf{g}(\mathsf{a}(x)) & \alpha_2 : \mathsf{a}(x) \to \mathsf{b}(\mathsf{x}) \\
\alpha_3 : \mathsf{g}(\mathsf{b}(x)) \to \mathsf{lin}(x) & \alpha_4 : \mathsf{g}(\mathsf{a}(x)) \to \mathsf{quad}(x) \\
\alpha_5 : \mathsf{lin}(\mathsf{s}(x)) \to \mathsf{lin}(x) & \alpha_6 : \mathsf{quad}(\mathsf{s}(x)) \to \mathsf{c}(\mathsf{lin}(x), \mathsf{quad}(x))
\end{array}$$

For now we can assume that the starting term of the longest rewrite sequences have the root symbol f. Innermost evaluation would force the subterm $\mathsf{a}(x)$ to be reduced to $\mathsf{b}(x)$ and thus create a rewrite sequence with length in $\mathcal{O}(n)$. Otherwise, the term can be matched to the lhs of α_4 and in a rewrite step that is non-ndg, create a rewrite sequence with length in $\mathcal{O}(n^2)$. In other words, $\mathrm{irc}_{\mathcal{Q}} \in \mathcal{O}(n)$, while $\mathrm{rc}_{\mathcal{Q}} \in \mathcal{O}(n^2)$

While a non-ndg TRS $\mathcal{R}$ fails to satisfy $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}}$, we show there exists a function Φ, which encodes $\mathcal{R}$ into $\Phi(\mathcal{R})$. This encoded TRS has a very specific property: the innermost runtime complexity of $\Phi(\mathcal{R})$, constrained to the basic terms of $\mathcal{R}$, is equal to the full runtime complexity of $\mathcal{R}$. In short, $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}'_{\Phi(\mathcal{R})}$, where irc' is the modified irc as described above.

3. Foundations of the Encoding

COPY COPY COPY COPY COPY COPY COPY COPY COPY COPY COPY

By definition every TRS $\mathcal{R}$ has finitely many rules and therefore finitely many positions, which can be encoded. However, for practical purposes encodings that encode every possible position are not useful. Generally speaking, encodings like this introduce new rules, which can produce loops and create infinitely long rewrite sequences albeit using rules with 0 cost. The focus of this chapter lies in the optimization of which positions to encode (Section 3.1.) and the limitation of non-terminating rules (Section 3.2.). To this effort the goal of the encoding is to ensure each innermost rewrite sequence in $\Phi(\mathcal{R})$ has an equally long full rewrite sequence in $\mathcal{R}$ and vice versa, which then serves as proof of $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}'_{\mathcal{R}'}$.

Consider again the TRS $\mathcal{Q}$ from example 11. and the rule $\alpha_4 : \mathsf{g}(\mathsf{a}(x)) \to \mathsf{quad}(x)$. We already pointed out that evaluating with this rule creates a non-ndg rewrite step, due to the fact the subterm $\mathsf{a}(x)$ is a redex for any instantiation. However, replacing the defined root symbol with some fresh constructor symbol, makes every rewrite step using that rule innermost. Doing that everywhere in the TRS for a specifically will allow for innermost rewrite sequences with length in $\mathcal{O}(n^2)$. So let us take a look at the eventual encoding $\Phi(\mathcal{Q})$:

$$\begin{array}{ll}
\alpha'_1 : \mathsf{f}(x) \to \mathsf{g}(\mathsf{i}(l_{\mathsf{a}}(x))) & \alpha'_2 : \mathsf{a}(x) \to \mathsf{b}(\mathsf{x}) \\
\alpha'_3 : \mathsf{g}(\mathsf{b}(x)) \to \mathsf{lin}(x) & \alpha'_4 : \mathsf{g}(l_{\mathsf{a}}(x)) \to \mathsf{quad}(\mathsf{i}(x)) \\
\alpha'_5 : \mathsf{lin}(\mathsf{s}(x)) \to \mathsf{lin}(x) & \alpha'_6 : \mathsf{quad}(\mathsf{s}(x)) \to \mathsf{c}(\mathsf{lin}(\mathsf{i}(x)), \mathsf{quad}(\mathsf{i}(x))) \\
\beta_1 : \mathsf{i}(x) \xrightarrow{0} x & \beta_2 : \mathsf{i}(l_{\mathsf{a}}(x)) \xrightarrow{0} \mathsf{i}(\mathsf{a}(x)) \\
 & \beta_3 : \mathsf{i}(l_{\mathsf{a}}(x)) \xrightarrow{0} \mathsf{i}(l_{\mathsf{a}}(\mathsf{i}(x)))
\end{array}$$

While everything we discussed is as we expected, there are certainly more differences than we mentioned. Besides the addition of the fresh constructor symbol for a, there is also a new defined symbol i, which is at the root of two relative rules, one of which restoring the defined symbol a in the place of l_{a}.

As mentioned previously, the goal is not to alter the rules of the TRS so that its longest rewrite sequence is innermost, but to simulate it and every other rewrite sequence via an innermost strategy. The newly introduced rules combined with the replacing of certain defined symbols with their constructor counterpart, and their subsequent encapsulation by an i-symbol allow us to do so. Consider the innermost rewrite sequence

$$\mathsf{f}(0) \quad \to_{\alpha'_1} \quad \mathsf{g}(\mathsf{i}(l_{\mathsf{a}}(0))) \quad \xrightarrow{0}_{\beta_1} \quad \mathsf{g}(\mathsf{a}(0)) \quad \to_{\alpha'_4} \quad \mathsf{quad}(\mathsf{i}(0))$$

We can say that it is a simulation of the rewrite sequence in $\mathsf{f}(0) \to_{\mathcal{Q}} \mathsf{g}(\mathsf{a}(0)) \to_{\mathcal{Q}} \mathsf{quad}(0)$. Both sequences have the same length and for now can be concluded that this holds for any other rewrite sequence in $\Phi(\mathcal{Q})$, that start with a basic term in $\mathcal{Q}$. Let us see why we need this restriction on starting terms. Consider the rewrite sequence with a starting term that is basic in $\Phi(\mathcal{Q})$:

$$\mathsf{f}(l_{\mathsf{a}}(0)) \quad \to_{\alpha'_1} \quad \mathsf{g}(\mathsf{i}(l_{\mathsf{a}}(l_{\mathsf{a}}(0)))) \quad \xrightarrow{0}{}^{3}_{\Phi(\mathcal{Q})} \quad \mathsf{g}(\mathsf{i}(l_{\mathsf{a}}(\mathsf{a}(0)))) \quad \to_{\alpha'_4} \quad \mathsf{g}(\mathsf{i}(l_{\mathsf{a}}(\mathsf{b}(0)))) \quad \to_{\Phi(\mathcal{Q})} \cdots$$

This is obviously a rewrite sequence that cannot be mapped to a rewrite sequence in $\mathcal{Q}$, i.e. it is a rewrite sequence that we are not interested in. Therefore, if we want useful results from the encoding, we must limit the starting terms to the ones from $\mathcal{Q}$.

But that is not all that we can notice by observing this encoded TRS. There is a duplication of variables in rule α_6, which are now also encapsulated by an i-symbol on the rhs. The reason is that the encoding has discovered the potential of a redex to flow into these duplicated variables, which as we remind is also a condition to be avoided for ndg rewriting. We expand on the issue of how these positions in the TRS are determined to be encoded and how in the next section.

3.1 Non-ndg locations

Before we establish how we mark non-ndg positions, we need to introduce two more definitions to help us better define this flow. The first of these are the so-called locations, which refer to positions in a TRS, so we can more easily tell them apart from position in terms.

Definition 13 (Location)**.**

Let $\mathcal{R}$ be a TRS. A **location** in $\mathcal{R}$ is a triple $(\alpha, \mathtt{X}, \pi)$, where $\alpha \in \mathcal{R}$, $\mathtt{X} \in \{\mathtt{L}, \mathtt{R}\}$ and π a position on the left or right hand side of α, depending on $\mathtt{X}$. The set of all **locations** of $\mathcal{R}$ is defined as

$$\mathcal{L}_{\mathcal{R}} := \{(\alpha, \mathtt{L}, \pi) \mid \alpha = \ell \to r \in \mathcal{R},\ \pi \in \mathrm{pos}(\ell)\} \cup \{(\alpha, \mathtt{R}, \pi) \mid \alpha = \ell \to r \in \mathcal{R},\ \pi \in \mathrm{pos}(r)\}$$

- The set of all **sub-locations** of a location λ is defined as
  $\mathrm{loc}_{\mathcal{R}}(\lambda) := \{(\alpha, \mathtt{X}, \pi) \mid \lambda = (\alpha, \mathtt{X}, \tau),\ \tau \leq \pi,\ (\alpha, \mathtt{X}, \pi) \in \mathcal{L}_{\mathcal{R}}\}$,
- For all $\ell \to r \in \mathcal{R}$ we write $\mathcal{R}|_{(\ell \to r, \mathtt{L}, \pi)} = \ell|_\pi$ and $\mathcal{R}|_{(\ell \to r, \mathtt{R}, \pi)} = r|_\pi$.

The matching of nested redexes does create some problems, which we will explore further later. One solution to this is temporarily replacing them with fresh variables, while the matching occurs. We do this by using the CAP-function.

Definition 14 ($\mathrm{CAP}_{\mathcal{R}}$ [8])**.**

For a given TRS $\mathcal{R}$ and some term q we define $\mathrm{CAP}_{\mathcal{R}} : \mathcal{T} \to \mathcal{T}$ as follows:

$$\mathrm{CAP}_{\mathcal{R}}(q) = \begin{cases} q & \text{if} \quad q \in \mathcal{V} \\ f(\mathrm{CAP}_{\mathcal{R}}(t_1), \dots, \mathrm{CAP}_{\mathcal{R}}(t_n)) & \text{else if } q = f(t_1, \dots t_n), f \in \Sigma_c \\ x & \text{else if } x \text{ is a fresh variable} \end{cases}$$

In other words, in the resulting term $\mathrm{CAP}_{\mathcal{R}}(q)$ all proper subterms of a term q, which have a defined root symbol, are replaced with different fresh variables. Multiple occurrences of the same subterm are replaced by pairwise different variables.

Consider the TRS $\mathcal{Q}$ from example 11. and the term $q = \mathsf{g}(\mathsf{a}(x))$, which corresponds to the rhs of rule α_1. As we will discover later, the locations of the a-symbol flows into a non-ndg locations in rule α_4. We discover this flow by matching $\mathrm{CAP}_{\mathcal{R}}(q) = g(x)$ to the lhs of α_4. While it would have matched regardless of the application of the $\mathrm{CAP}_{\mathcal{R}}$-function, there are cases where this is not quite as obvious.

As we continue to the algorithm for marking non-ndg locations its is important to say that while some work on data flow analysis exists [9], it was deemed not precise enough to use for our encoding. Instead it was preferred to create a separate algorithm, that would return all to-be-encoded locations in a given TRS. The following set contains all locations marked as non-ndg.

Definition 15.

Let $\mathcal{R}$ be a TRS.

1. The **set of base non-ndg locations** $\mathcal{X}'_{\mathcal{R}}$ is the smallest set such that:
   - $\{(\alpha, \mathtt{L}, \tau) \mid \alpha = \ell \to r \in \mathcal{R},\quad \tau \in pos(\ell) \backslash \{\varepsilon\},\quad root(\ell|_\tau) \in \Sigma_d\} \subseteq \mathcal{X}'_{\mathcal{R}}$ $(S1)$
   - $\{(\alpha, \mathtt{R}, \tau) \mid \alpha = \ell \to r \in \mathcal{R}, \tau, \pi \in pos(r), \tau \neq \pi, r|_\tau = r|_\pi \in \mathcal{V}\} \subseteq \mathcal{X}'_{\mathcal{R}}$ $(S2)$

   The locations in $\mathcal{X}'_{\mathcal{R}}$ are the ones where a defined symbol is nested on the lhs of a rule $(S1)$, and where variables are duplicated on the rhs of a rule $(S2)$.
2. The **set of over-approximated non-ndg locations** $\mathcal{X}_{\mathcal{R}}$ is the smallest set such that:
   - $\mathcal{X}'_{\mathcal{R}} \subseteq \mathcal{X}_{\mathcal{R}}$

- $\left\{(\alpha, \mathtt{L}, \tau) \;\middle|\; \begin{array}{l} \alpha = \ell \to r \in \mathcal{R}, \tau \in pos(\ell), \pi \in pos(r), \\ \ell|_\tau = r|_\pi \in \mathcal{V} \end{array} \right\} \subseteq \mathcal{X}_{\mathcal{R}}$, if $(\alpha, \mathtt{R}, \pi) \in \mathcal{X}_{\mathcal{R}}$ $(S3)$
- $\left\{(\alpha, \mathtt{R}, \tau) \;\middle|\; \begin{array}{l} \alpha = \ell \to r \in \mathcal{R}, \tau \in pos(r), \tau = \pi.\upsilon, \upsilon \neq \varepsilon \\ \beta = s \to t \in \mathcal{R}, \omega \in pos(s), \upsilon \not\parallel \omega, \\ \exists \text{ MGU for } \mathrm{CAP}_{\mathcal{R}}(r|_\pi) \text{ and } s \end{array} \right\} \subseteq \mathcal{X}_{\mathcal{R}}$, if $(\beta, \mathtt{L}, \omega) \in \mathcal{X}_{\mathcal{R}}$ $(S4)$
- $\left\{(\alpha, \mathtt{R}, \tau) \;\middle|\; \begin{array}{l} \alpha = \ell \to r \in \mathcal{R}, \tau \in pos(r), \\ \beta = s \to t \in \mathcal{R}, \pi \in pos(t), \\ \exists \text{ MGU for } \mathrm{CAP}_{\mathcal{R}}(t|_\pi) \text{ and } \ell \end{array} \right\} \subseteq \mathcal{X}_{\mathcal{R}}$, if $(\beta, \mathtt{R}, \pi) \in \mathcal{X}_{\mathcal{R}}$ $(S5)$

$\mathcal{X}_{\mathcal{R}}$ includes all base non-ndg locations and then some more based on what flows into these. The first set $(S3)$ includes the locations of variables on the lhs of rules, which flow directly into marked variable locations on the rhs of the same rules. Simply put, the marking of a location of variable x on the rhs, marks all locations of x on the lhs. The second set $(S4)$ tracks the flow of locations from rhs of rules to lhs of rules. And finally $(S5)$ contains all over-approximated reachable return locations of marked defined symbols at rhs of rules. Return locations refer to all locations on the rhs of a rule, whose lhs root symbol is some specified symbol.

We can now go into further detail why each of these locations are marked as non-ndg. Since our encoding will be analyzed via its irc, all marked location of defined symbols on the lhs of any rule will be replaced by a corresponding constructor in the encoded TRS. The locations of such subterms are contained in $(S1)$.

Next to consider is the duplication of variables. Or the more relevant - duplication of redexes. This is impossible when employing an innermost evaluation strategy. That is why we must track backwards what flows into these variables and encode it, i.e. replace by corresponding constructors. The initial duplicated positions are contained in $(S2)$ and then $(S3)$ tracks back to the lhs of the rules in $(S2)$ to mark their origin. Afterwards, $(S4)$ checks if any subterm at a rhs of any rule can be matched to the marked locations in $(S1 - S2)$.

Since $(S4 - S5)$ are more complex compared to the rest, we will go into further detail for each step. In $(S4)$ the position τ is split in two parts: $\tau = \pi.\upsilon$, where $\upsilon \neq \varepsilon$. The subterm at π is what we try to match. While $\pi = \varepsilon$ is allowed, $\upsilon = \varepsilon$ is not, because this would include the ε positions of rhs of rules, which can not flow between locations. The MGU (Most General Unifier) is the matcher between the two terms and it means that a subterm reached via a rewrite step using rule α, can then potentially be evaluated using rule β.

Notably, we also use the $\mathrm{CAP}_{\mathcal{R}}$-function on the subterm we match. The reason we use it is to ease the discovery of data flows of redexes, i.e. the location of a redex may flow into a non-ndg location, but also fail to match with the term with said location. Since the redex can be rewritten, so as to actually match the term, we decided to over-approximate and basically assume it can always be rewritten to match. Due to its similarity to the Word problem, this was considered appropriate. The most obvious way to show the over-approximation caused by $\mathrm{CAP}_{\mathcal{R}}$ is via a simple example.

$$\beta_1 : \mathsf{h} \to \mathsf{g}(\mathsf{h}) \qquad \beta_2 : \mathsf{g}(\mathsf{a}) \to \mathsf{g}(\mathsf{a}) \qquad \beta_3 : \mathsf{a} \to \mathsf{a}$$

Here the h term on the rhs of β_1 does not flow into position 1 of the lhs of β_2. Using the $\mathrm{CAP}_{\mathcal{R}}$ function on $\mathsf{g}(\mathsf{h})$ returns $\mathsf{g}(x)$, which can be matched.

Lastly, we require that $\upsilon \not\parallel \omega$. This part ensures that the newly included location in α actually flows into a non-ndg location. Since it may not be as obvious how we arrived at this part of the definition, we can consider an example build to show all cases.

$$\alpha_1 : \mathsf{h}(x_1, y_1, z_1) \to \mathsf{g}(x_1, y_1, \mathsf{s}(z_1)) \quad \alpha_2 : \mathsf{g}(x_2, \mathsf{s}(y_2), z_2) \to \mathsf{d}(x_2, x_2)$$

Clearly, rule α_2 is duplicating and the locations of x_2 are all marked. Let us turn our attention to α_1, where the variable x_1 on the rhs flows into x_2. Their position are not just non-parallel, they are equal, which was how we defined this set originally. However, upon further inspection, we also notice that y_1 should flow into y_2. We can not know what y_1 can be instantiated with, but since it can be for example $\mathsf{s}(t)$ for some term t, then that is what flows between the locations. Then for the position of y_1 and y_2 it holds that $2 \leq 2.1$. Analogously can be derived for the location of z_1, which flows into the location of z_2. Here the entire term $\mathsf{s}(z_1)$ is what flows between locations. Going back to their positions we have $3.1 \geq 3$. Thus we only require that the positions of the locations to be marked and the already marked location are non-parallel. If we were not to consider the relation between these positions, then a single marked location somewhere in the matched term is needed to mark any location in the matcher. As per the example, that would mean marking y_1 and z_1, which do flow into their respective locations in α_2, because x_2 is marked.

In ($S5$) are included the return locations of non-ndg marked defined symbols. The subterms on rhs of rules, which have a defined root symbol, may be redexes and thus rewritten. The newly evaluated term at the position of rewriting must inherit the non-ndg mark, otherwise said term may flow into a non-ndg location, defeating the whole purpose of the encoding.

In terms of computation, $\mathcal{X}_\mathcal{R}$ for some TRS $\mathcal{R}$ has to be continuously reevaluated until no further changes are made. Since no locations are ever removed from the set and there are finitely many locations in a TRS, we can conclude that $\mathcal{X}_\mathcal{R}$ always reaches a fix-point.

Expanding on TRS $\mathcal{Q}$ from example 11, we can now see how the newly defined set helps in establishing the set of locations to be encoded.The locations in $\mathcal{X}_\mathcal{Q}$ are underlined.

$$\begin{array}{ll}
\alpha_1 : \mathsf{f}(x) \to \mathsf{g}(\utilde{\mathsf{a}(x)}) & \alpha_2 : \mathsf{a}(\utilde{x}) \to \utilde{\mathsf{b}(x)} \\[1ex]
\alpha_3 : \mathsf{g}(\mathsf{b}(x)) \to \mathsf{lin}(x) & \alpha_4 : \mathsf{g}(\utilde{\mathsf{a}(x)}) \to \mathsf{quad}(\utilde{x}) \\[1ex]
\alpha_5 : \mathsf{lin}(\mathsf{s}(x)) \to \mathsf{lin}(x) & \alpha_6 : \mathsf{quad}(\mathsf{s}(\utilde{x})) \to \mathsf{c}(\mathsf{lin}(\utilde{x}), \mathsf{quad}(\utilde{x}))
\end{array}$$

Besides the variable duplication in α_6 and the nested redex at α_4, there are a few other locations marked as non-ndg. We previously noticed the flow from α_1 to α_4, but there is also one from α_4 to α_6. Further, the return locations of a are marked. Cross-referencing this marking with the encoding $\Phi(\mathcal{Q})$ leaves some locations not encoded. All but one are locations of variables on the lhs of rules, which will later have to be excluded, as it makes no sense to encode them too. The other one is the location of a constructor symbol and therefore not a redex, so we have no use to encode that location, as it will not affect the runtime complexity. For the same reason, we would have no use to encode locations of variable duplication, if the only thing getting duplicated are constructor terms. Assume an extension $\mathcal{Q}_1 = \mathcal{Q} \cup \{\mathsf{d}(x, y) \to \mathsf{d}(x, x)\}$. When calculating $\mathrm{rc}_{\mathcal{Q}_1}$ only basic terms are allowed as starting terms. Therefore the duplication of variable x in the extended rule will always receive some constructor term and would prove useless to encode.

At this point $\mathcal{X}_\mathcal{Q}$ is sufficient to define an encoding of $\mathcal{Q}$, but as shown in the extended example, we can improve the precision of this over-approximation. Calculating a different set of locations, which tracks the flow of defined symbols on the rhs of rules, will then allow us to define all non-ndg marked locations, which can potentially receive a redex.

Definition 16.

For some given TRS $\mathcal{R}$ and a set of locations Δ let $\mathcal{Y}^\Delta_\mathcal{R}$ be the smallest set containing Δ and all locations, which locations from Δ might flow into:

- $\Delta \subseteq \mathcal{Y}^\Delta_\mathcal{R}$ $\quad (S6)$
- $\left\{ (\alpha, \mathtt{R}, \pi) \;\middle|\; \begin{array}{l} \alpha = \ell \to r \in \mathcal{R}, \tau \in pos(\ell), \pi \in pos(r), \\ \ell|_\tau = r|_\pi \in \mathcal{V} \end{array} \right\} \subseteq \mathcal{Y}^\Delta_\mathcal{R}$, if $(\alpha, \mathtt{L}, \tau) \in \mathcal{Y}^\Delta_\mathcal{R}$ $\quad (S7)$
- $\left\{ (\beta, \mathtt{L}, \omega) \;\middle|\; \begin{array}{l} \alpha = \ell \to r \in \mathcal{R}, \tau \in pos(r), \tau = \pi.\upsilon, \upsilon \neq \varepsilon, \\ \beta = s \to t \in \mathcal{R}, \omega \in pos(s), \upsilon \nparallel \omega, \\ \exists \text{ MGU for } \mathrm{CAP}_\mathcal{R}(r|_\pi) \text{ and } s \end{array} \right\} \subseteq \mathcal{Y}^\Delta_\mathcal{R}$, if $(\alpha, \mathtt{R}, \tau) \in \mathcal{Y}^\Delta_\mathcal{R}$ $\quad (S8)$

The definitions of these subsets are similar to ($S3 - S4$). Changed is which location from a data flow is included. Applying this to TRS $\mathcal{Q}$ with $\Delta = \{(\alpha, \mathtt{R}, \tau) \mid \alpha = \ell \to r \in \mathcal{Q}, root(r|_\tau) \in \Sigma_d\}$ we get

$$\begin{array}{ll}
\alpha_1 : \mathsf{f}(x) \to \mathsf{g}(\utilde{\widehat{\mathsf{a}}(x)}) & \alpha_2 : \mathsf{a}(\utilde{x}) \to \utilde{\mathsf{b}(x)} \\[1ex]
\alpha_3 : \mathsf{g}(\mathsf{b}(x)) \to \widehat{\mathsf{lin}}(x) & \alpha_4 : \mathsf{g}(\widehat{\utilde{\mathsf{a}(x)}}) \to \widehat{\mathsf{quad}(\utilde{x})} \\[1ex]
\alpha_5 : \mathsf{lin}(\mathsf{s}(x)) \to \widehat{\mathsf{lin}}(x) & \alpha_6 : \mathsf{quad}(\mathsf{s}(\utilde{\widehat{x}})) \to \mathsf{c}(\widehat{\mathsf{lin}(\utilde{x})}, \widehat{\mathsf{quad}(\utilde{x})})
\end{array}$$

where the locations in $\mathcal{Y}_{\mathcal{Q}}^{\Delta}$ are indicated with $\widehat{\ }$ ('hat') over them. The intersection of underlined locations and locations with a 'hat', gives us the set of over-approximated non-ndg locations, in which a defined symbol flows. As can be seen this will ignore the marked constructor in α_2, as was desired. Further, even though the lin-symbols at α_3 and α_5 have a 'hat', they does not flow into a non-ndg location, therefore encoding them is not necessary. The locations in that intersection are now exactly the ones encoded in $\Phi(\mathcal{Q})$.

Definition 17($\mathrm{ENC}_{\mathcal{R}}$)**.**

For a TRS $\mathcal{R}$ and $\Delta = \{(\alpha, \mathtt{R}, \tau) \mid \alpha = \ell \to r \in \mathcal{R}, \mathit{root}(r|_\tau) \in \Sigma_d\}$ let the set of all locations to be encoded $\mathrm{ENC}_{\mathcal{R}}$ be defined as

$$\mathrm{ENC}_{\mathcal{R}} := (\mathcal{X}_{\mathcal{R}} \cap \mathcal{Y}_{\mathcal{R}}^{\Delta}) \backslash \{(\alpha, \mathtt{L}, \pi) \mid \mathcal{R}|_{(\alpha,\mathtt{L},\pi)} \in \mathcal{V}\}$$

In order to show the thought process behind the encoding, we focus on a simpler example. For the purpose of convenience we chose a TRS that only has a few rules and its runtime complexity is constant. Consider the non-ndg TRS $\mathcal{U}$, which has its locations in $\mathrm{ENC}_{\mathcal{U}}$ underlined.

$$\alpha_1 : \mathsf{h} \to \mathsf{g}(\underline{\mathsf{a}}, \underline{\mathsf{a}}, \mathsf{a}) \qquad \alpha_2 : \mathsf{g}(x, \underline{\mathsf{a}}, \mathsf{y}) \to \mathsf{f}(\underline{x}, \underline{x}) \qquad \alpha_3 : \mathsf{a} \to \mathsf{b}$$

Since we strive to analyze its irc, the nested a-symbol in α_2 can not stay as is and must be changed to some fresh constructor symbol. We choose l and in order to differentiate between other such constructor symbols, we indicate the original symbol being replaced in the subscript, giving us l_{a}.

$$\alpha_1 : \mathsf{h} \to \mathsf{g}(\underline{\mathsf{a}}, \underline{\mathsf{a}}, \mathsf{a}) \qquad \alpha_2 : \mathsf{g}(x, \underline{l_{\mathsf{a}}}, \mathsf{y}) \to \mathsf{f}(\underline{x}, \underline{x}) \qquad \alpha_3 : \mathsf{a} \to \mathsf{b}$$

However, this change eliminates the flow from α_1 to α_2. If we want to preserve it, we can also encode in the same fashion the a-symbols in α_1.

$$\alpha_1 : \mathsf{h} \to \mathsf{g}(\underline{l_{\mathsf{a}}}, \underline{l_{\mathsf{a}}}, \mathsf{a}) \qquad \alpha_2 : \mathsf{g}(x, \underline{l_{\mathsf{a}}}, \mathsf{y}) \to \mathsf{f}(\underline{x}, \underline{x}) \qquad \alpha_3 : \mathsf{a} \to \mathsf{b}$$

Replacing defined symbols with constructor creates a new problem. There are rewrite sequences which evaluate these exact term, that we have now practically removed. In our example after rewriting with α_1, only one a term in the left position can be evaluated, in comparison to the three a terms before the change. If we can restore the defined symbol at these positions, then that would not be an issue. Ideally, we do this via a rule with 0 cost, so as to not affect time complexity. This means we need to introduce a new defined symbol and new rules.

$$\alpha_1 : \mathsf{h} \to \mathsf{g}(\underline{\mathsf{i}(l_a)}, \underline{\mathsf{i}(l_a)}, \mathsf{a}) \qquad \alpha_2 : \mathsf{g}(x, \underline{l_{\mathsf{a}}}, \mathsf{y}) \to \mathsf{f}(\underline{x}, \underline{x}) \qquad \alpha_3 : \mathsf{a} \to \mathsf{b}$$

$$\alpha_4 : \mathsf{i}(l_{\mathsf{a}}) \xrightarrow{0} \mathsf{a}$$

Again, we run into a problem of matching rhs of α_1 to lhs of α_2. While position 1 can be matched, the rewrite step would not be innermost. And even ignoring that, position 2 would fail to match anyways. In situations like this we would like to remove the i-symbols from the redex, so the rewrite is innermost.

$$\alpha_1 : \mathsf{h} \to \mathsf{g}(\underline{\mathsf{i}(l_a)}, \underline{\mathsf{i}(l_a)}, \mathsf{a}) \qquad \alpha_2 : \mathsf{g}(x, \underline{l_{\mathsf{a}}}, \mathsf{y}) \to \mathsf{f}(\underline{x}, \underline{x}) \qquad \alpha_3 : \mathsf{a} \to \mathsf{b}$$

$$\alpha_4 : \mathsf{i}(l_{\mathsf{a}}) \xrightarrow{0} \mathsf{i}(\mathsf{a}) \qquad \alpha_5 : \mathsf{i}(x) \xrightarrow{0} x$$

After all these changes, the duplication in α_2 has remain unaltered. We will show now why the marked x's should be encapsulated by i, similar to the rhs of α_1. Consider the following rewrite sequence using the rules $\alpha_1 - \alpha_5$ directly above.

$$\mathsf{h} \quad \to_{\alpha_1} \quad \mathsf{g}(\mathsf{i}(l_{\mathsf{a}}), \mathsf{i}(l_{\mathsf{a}}), \mathsf{a}) \quad \xrightarrow{0}_{\alpha_5} \quad \mathsf{g}(l_{\mathsf{a}}, \mathsf{i}(l_{\mathsf{a}}), \mathsf{a}) \quad \xrightarrow{0}_{\alpha_5} \quad \mathsf{g}(l_{\mathsf{a}}, l_{\mathsf{a}}, \mathsf{a}) \quad \to_{\alpha_2} \quad \mathsf{f}(l_{\mathsf{a}}, l_{\mathsf{a}})$$

The last term $\mathsf{f}(l_{\mathsf{a}}, l_{\mathsf{a}})$ cannot be further evaluated, since there is no way to restore the a-symbols from these constructors. We can assume from the concept of the encoding so far that it is possible that redexes flow into these marked variable positions. Therefore an i-symbol should be placed there when rewriting with α_2.

$$\alpha_1 : \mathsf{h} \to \mathsf{g}(\mathsf{i}(l_\mathsf{a}), \mathsf{i}(l_\mathsf{a}), \mathsf{a}) \qquad \alpha_2 : \mathsf{g}(x, l_\mathsf{a}, y) \to \mathsf{f}(\mathsf{i}(x), \mathsf{i}(x)) \qquad \alpha_3 : \mathsf{a} \to \mathsf{b}$$

$$\alpha_4 : \mathsf{i}(l_\mathsf{a}) \xrightarrow{0} \mathsf{i}(\mathsf{a}) \qquad \alpha_5 : \mathsf{i}(x) \xrightarrow{0} x$$

We get the final version of the TRS, which corresponds to the eventual encoding $\Phi(\mathcal{U})$. So far we discussed and formalized what locations in a TRS are to be encoded. The example above has shown that only changing the rules is not sufficient, and new rules should be added. The additional rules exclusively evaluate redexes with the newly introduced defined symbol i at their root and can be split into three categories.

- $\mathsf{i}(l_\mathsf{a}) \xrightarrow{0} \mathsf{i}(\mathsf{a})$

  The *executing rules* restore the defined symbol, whose constructor version is at position 1 on the lhs. Shorthand notation for this rule is $\mathtt{EXE}_\mathsf{a}$, where a stands for the defined symbol being restored.

- $\mathsf{i}(x) \xrightarrow{0} x$

  The *omission rule* is mandatory in all encodings, as was shown in the previous section. Shorthand notation for this rule is `OMIT`.

- $\mathsf{i}(l_\mathsf{g}(x, y, z)) \xrightarrow{0} \mathsf{i}(l_\mathsf{g}(\mathsf{i}(x), \mathsf{i}(y), \mathsf{i}(z)))$

  The *propagation rule* encapsulates with an i-symbol all subterms directly bellow the root of the term matched at position 1. This type of rule was was missing from the example above, because it was indeed not needed. We know this because g was the only symbol that had non-zero arity, but was not encoded. Shorthand notation for this rule is $\mathtt{PROP}_\mathsf{g}$, where g is the symbol at which the propagation occurs.

  The omission rules have two variations:

  – *Terminating* $\quad \mathsf{i}(l_\mathsf{g}(x, y, z)) \xrightarrow{0} l_\mathsf{g}(\mathsf{i}(x), \mathsf{i}(y), \mathsf{i}(z))$

  – *Non-terminating* $\quad \mathsf{i}(l_\mathsf{g}(x, y, z)) \xrightarrow{0} \mathsf{i}(l_\mathsf{g}(\mathsf{i}(x), \mathsf{i}(y), \mathsf{i}(z)))$

  We make this distinction due to the consequences of adding both the omission rule and the propagation rule for any symbol. The non-terminating rule above in combination with $\mathsf{i}(x) \to x$ for some encoded TRS $\Phi(\mathcal{R})$ creates non-terminating rewrite sequences like the following:

$$\cdots \quad \to_{\mathcal{R}'} \quad \mathsf{i}(l_\mathsf{g}(\mathsf{b}, \mathsf{b}, \mathsf{b})) \quad \xrightarrow{0}_{\mathcal{R}'} \quad \mathsf{i}(l_\mathsf{g}(\mathsf{i}(\mathsf{b}), \mathsf{i}(\mathsf{b}), \mathsf{i}(\mathsf{b}))) \quad \xrightarrow{0}{}^{*}_{\mathcal{R}'} \quad \mathsf{i}(l_\mathsf{g}(\mathsf{b}, \mathsf{b}, \mathsf{b})) \quad \xrightarrow{0}_{\mathcal{R}'} \quad \cdots$$

  Such sequences could throttle the automatic complexity analysis and therefore it is best to avoid adding non-terminating rules whenever possible. An analysis on the rules for which function symbols it would be viable to add the terminating version of the rule, is provided in this section.

For each of these shorthand notation, we use a superscript L or R to indicate lhs or rhs of the respective rule. For example $\mathtt{OMIT}^L = \mathsf{i}(x)$.

The propagation rules are required for function symbols, which are encoded and have a non-zero arity. Without further analysis, the encoding would have to add the non-terminating propagation rules for these symbols. As discussed, this is not ideal and we should focus on eliminating as many of these non-terminating rules as possible, which will be covered in the next chapter.

3.2 Encoding

In this section, we introduce the definition of the encoding, which so far has only been partially conceptualized. As a reminder, the goal of the encoding is to encode a TRS $\mathcal{R}$ to $\Phi(\mathcal{R})$, whose innermost runtime complexity is the same as the full runtime complexity of $\mathcal{R}$. Thus allowing us to use the much more powerful techniques for irc analysis to derive results for rc, mainly an upper-bound on rc.

Definition 23.

Let $\mathcal{R}$ be a TRS. We define **the encoding** as a function Φ :

$$\Phi(\mathcal{R}) = \mathcal{R}' \cup \mathcal{S}, \text{ such that}$$

$$\begin{aligned}
\mathcal{R}' &= \{\psi(\alpha) \mid \alpha \in \mathcal{R}\} \\
\mathcal{S} &= \{\mathsf{i}(x) \xrightarrow{0} x\} \\
&\cup \{\, \mathsf{i}(l_f\,(x_1,\dots,x_n)) \xrightarrow{0} \mathsf{i}(l_f(\mathsf{i}(x_1),\dots,\mathsf{i}(x_n))) \mid f = \mathit{root}(\mathcal{R}|_\lambda) \in \Sigma_d^n, \lambda \in \mathrm{ENC}_\mathcal{R}\} \\
&\cup \{\, \mathsf{i}(l_f\,(x_1,\dots,x_n)) \xrightarrow{0} \mathsf{i}(f(x_1,\dots,x_n)) \qquad \mid f = \mathit{root}(\mathcal{R}|_\lambda) \in \Sigma_d^n, \lambda \in \mathrm{ENC}_\mathcal{R}\} \\
&\cup \{\, \mathsf{i}(d\,(x_1,\dots,x_n)) \xrightarrow{0} d(\mathsf{i}(x_1),\dots,\mathsf{i}(x_n)) \qquad \mid d \in \Sigma_c^n\}
\end{aligned}$$

ψ encodes the rules in $\mathcal{R}$ and uses φ to encode the individual sides of each rule. We define the ψ-function as follows:

$$\psi(\alpha) = \varphi(\mathcal{R}, (\alpha, \mathtt{L}, \varepsilon)) \to \varphi(\mathcal{R}, (\alpha, \mathtt{R}, \varepsilon)),\ \ \alpha \in \mathcal{R}$$

For some location $\lambda = (\alpha, \mathtt{X}, \pi)$, $\mathtt{X} \in \{\mathtt{L}, \mathtt{R}\}$ let $\lambda^{(k)} = (\alpha, \mathtt{X}, \pi.k)$, and $f = \mathit{root}(\mathcal{R}|_\lambda) \in \Sigma^n$ or $f \in \mathcal{V}$.

1. $$\varphi(\mathcal{R}, \lambda) = \begin{cases} f(\ \varphi(\mathcal{R}, \lambda^{(1)}), \dots,\ \varphi(\mathcal{R}, \lambda^{(n)})) & \text{if } \quad f \in \Sigma_c \text{ or } \lambda \notin \mathrm{ENC}_\mathcal{R} \\ l_f(\ \varphi(\mathcal{R}, \lambda^{(1)}), \dots,\ \varphi(\mathcal{R}, \lambda^{(n)})) & \text{else if } \mathtt{X} = \mathtt{L} \\ \mathsf{i}\ (l_f(\varphi_\mathcal{N}(\mathcal{R}, \lambda^{(1)}), \dots, \varphi_\mathcal{N}(\mathcal{R}, \lambda^{(n)}))) & \text{else if } f \notin \mathcal{V} \\ \mathsf{i}^k(f) & \text{else} \end{cases}$$

2. $$\varphi_\mathcal{N}(\mathcal{R}, \lambda) = \begin{cases} f(\varphi(\mathcal{R}, \lambda^{(1)}), \dots, \varphi(\mathcal{R}, \lambda^{(n)})) & \text{if } \quad f \in \Sigma_c \\ l_f(\varphi(\mathcal{R}, \lambda^{(1)}), \dots, \varphi(\mathcal{R}, \lambda^{(n)})) & \text{else if } f \in \Sigma_d \\ f & \text{else} \end{cases}$$

We can now see precisely how a TRS gets encoded via Φ. so an example that incorporates different aspects of the encoding is presented.

Example 24.

Consider the following TRS $\mathcal{P}$ and all the relevant information about its encoding. The locations in $\mathrm{ENC}_\mathcal{P}$ are underlined.

$$\begin{array}{lll}
\alpha_1 : \mathsf{h} \to \mathsf{a}(\underline{\mathsf{f}(\mathsf{c})}) & & \alpha_2 : \mathsf{a}(x) \to \mathsf{a}(\underline{\mathsf{a}(x)}) \\
\alpha_3 : \mathsf{f}(\underline{\mathsf{c}}) \to \underline{\mathsf{f}(\mathsf{c})} & \alpha_4 : \mathsf{a}(x) \to \mathsf{b}(\underline{x}, \underline{x}) & \alpha_5 : \mathsf{c} \to 0
\end{array}$$

We are now going take a look at how $\Phi(\mathcal{P})$ is calculated by going through each step. Terms with zero non-ndg marked locations are skipped, since the first condition of φ clearly ignores these and changes nothing in the process. First we encode each rule to give us $\mathcal{P}'$:

- $\psi(\alpha_1) = \mathsf{h} \to \varphi(\mathsf{a}(\mathsf{f}(\mathsf{c}))) \quad = \mathsf{h} \to \mathsf{a}(\varphi(\mathsf{f}(\mathsf{c}))) \quad = \mathsf{h} \to \mathsf{a}(\mathsf{i}(l_\mathsf{f}(\varphi_\mathcal{N}(\mathsf{c})))) \quad = \mathsf{h} \to \mathsf{a}(\mathsf{i}(l_\mathsf{f}(l_\mathsf{c}))$
- $\psi(\alpha_2) = \mathsf{a}(x) \to \varphi(\mathsf{a}(\mathsf{a}(x))) = \mathsf{a}(x) \to \mathsf{a}(\varphi(\mathsf{a}(x))) = \mathsf{a}(x) \to \mathsf{a}(\mathsf{i}(l_\mathsf{a}(\varphi_\mathcal{N}(x)))) = \mathsf{a}(x) \to \mathsf{a}(\mathsf{i}(l_\mathsf{a}(x)))$
- $\psi(\alpha_3) = \varphi(\mathsf{f}(\mathsf{c})) \to \varphi(\mathsf{f}(\mathsf{c})) \ = \mathsf{f}(\varphi(\mathsf{c})) \to \mathsf{i}(l_\mathsf{f}(\varphi_\mathcal{N}(\mathsf{c}))) = \mathsf{f}(l_\mathsf{c}) \to \mathsf{i}(l_\mathsf{f}(l_\mathsf{c}))$
- $\psi(\alpha_4) = \mathsf{a}(x) \to \varphi(\mathsf{b}(x, x)) \ = \mathsf{a}(x) \to \mathsf{b}(\varphi(x), \varphi(x)) \quad = \mathsf{a}(x) \to \mathsf{b}(\mathsf{i}(x), \mathsf{i}(x))$
- $\psi(\alpha_5) = \alpha_5$

Describing every step in detail is pointless, but we can summarize the process of encoding the rules of $\mathcal{P}$. The subterm $\mathsf{f}(\mathsf{c})$ on the rhs of α_1 flows into the duplication of α_4 and thus is encapsulated by an i-symbol, analogous to the subterm $\mathsf{a}(x)$ in α_2. The only case of nested defined symbol on a lhs in TRS $\mathcal{P}$ is located in α_3. It is only replaced by a fresh constructor symbol and the entire rhs of α_3 is encoded, since these are return locations of f, marked non-ndg in α_1. The rhs of α_4 contains the duplicated variables, also encoded to $\mathsf{i}(x)$. Next come the additional rules. Instead of grouping them by category, where all `PROP` rules and all `EXE` rules are defined in one set definition, it is much more readable and easy to understand if we group them by function symbols. We start with the `OMIT` rule, which is not tied to any function symbol. Next we iterate over the symbols which are underlined above, i.e. which get encoded. Let the first symbol be f. The encoding adds $\mathtt{EXE}_\mathsf{f}$ and the non-terminating $\mathtt{PROP}_\mathsf{f}$.

- $\mathcal{S} \supseteq \{\mathsf{i}(x) \xrightarrow{\mathsf{0}} x\} \cup \{\mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{f}(x)),\ \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(l_\mathsf{f}(\mathsf{i}(x)))\ \}$

Next symbol we pick is c. It has an arity of 0, therefore no $\mathtt{PROP}_\mathsf{c}$ rule can be added. Thus only $\mathtt{EXE}_\mathsf{c}$ is included in $\mathcal{S}$.

- $\mathcal{S} \supseteq \{\mathsf{i}(x) \xrightarrow{\mathsf{0}} x\} \cup \{\mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{f}(x)),\ \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(l_\mathsf{f}(\mathsf{i}(x)))\ \} \cup \{\mathsf{i}(l_\mathsf{c}) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{c})\}$

The last function symbol to consider is a. with a step similar to the one for f.

- $\mathcal{S} \supseteq \{\mathsf{i}(x) \xrightarrow{\mathsf{0}} x\} \cup \{\mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{f}(x)),\ \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(l_\mathsf{f}(\mathsf{i}(x)))\ \} \cup \{\mathsf{i}(l_\mathsf{c}) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{c})\}$
  $\cup \{\mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{a}(x)),\ \mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(l_\mathsf{a}(\mathsf{i}(x)))\}$

By definition we should also add propagation rules for constructor symbols, but that was not precisely demonstrated previously and only mentioned in passing. In TRS $\mathcal{P}$ there is only one constructor symbol with non-zero arity - b. It is not marked as non-ndg, since locations with constructor root symbols were excluded from $\mathrm{ENC}_\mathcal{P}$. Sparing any further analysis on the TRS, we can simply add the terminating rules for all constructor symbols, with which $\mathcal{S}$ is finally complete.

- $\mathcal{S} = \{\mathsf{i}(x) \xrightarrow{\mathsf{0}} x\} \cup \{\mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{f}(x)),\ \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{l}_\mathsf{f}(\mathsf{i}(x)))\ \} \cup \{\mathsf{i}(l_\mathsf{c}) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{c})\}$
  $\cup \{\mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{a}(x)),\ \mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(l_\mathsf{a}(\mathsf{i}(x)))\} \cup \{\mathsf{i}(\mathsf{b}(x,x) \xrightarrow{\mathsf{0}} \mathsf{b}(\mathsf{i}(x),\mathsf{i}(x))\}$

With these results we can construct the full encoding $\Phi(\mathcal{P})$

$$\begin{array}{lll}
\alpha'_1 : \mathsf{h} \to \mathsf{a}(\mathsf{i}(l_\mathsf{f}(l_\mathsf{c})) & & \alpha'_2 : \mathsf{a}(x) \to \mathsf{a}(\mathsf{i}(l_\mathsf{a}(x))) \\
\alpha'_3 : \mathsf{f}(l_\mathsf{c}) \to \mathsf{i}(l_\mathsf{f}(l_\mathsf{c})) & \alpha'_4 : \mathsf{a}(x) \to \mathsf{b}(\mathsf{i}(x),\mathsf{i}(x)) & \alpha'_5 : \mathsf{c} \to \mathsf{0} \\
 & & \\
\beta_1 : \mathsf{i}(x) \xrightarrow{\mathsf{0}} x & \beta_2 : \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{f}(x)) & \beta_3 : \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(l_\mathsf{f}(\mathsf{i}(x))) \\
 & \beta_5 : \mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{a}(x)) & \beta_6 : \mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{\mathsf{0}} \mathsf{i}(l_\mathsf{a}(\mathsf{i}(x))) \\
 & \beta_4 : \mathsf{i}(l_\mathsf{c}) \xrightarrow{\mathsf{0}} \mathsf{i}(\mathsf{c}) & \beta_7 : \mathsf{i}(\mathsf{b}(x,x) \xrightarrow{\mathsf{0}} \mathsf{b}(\mathsf{i}(x),\mathsf{i}(x))
\end{array}$$

One thing that is important to clarify is that the result of the encoding is not ndg. When we talk about expanding the technique from [4], we do not mean transforming the TRS into an equivalent ndg TRS, for which irc = rc holds. We can show this via a simple rewrite sequence:

$$\mathsf{h} \quad \to_{\alpha'_1} \quad \mathsf{a}(\mathsf{i}(l_\mathsf{f}(l_\mathsf{c}))) \quad \to_{\alpha'_4} \quad \mathsf{b}(\mathsf{i}^2(l_\mathsf{f}(l_\mathsf{c})),\mathsf{i}^2(l_\mathsf{f}(l_\mathsf{c})))$$

What we see in the second rewrite step is a duplication of a redex, namely the one at position 1. Therefore $\Phi(\mathcal{P})$ is not an ndg TRS, but that is not a problem, since the encoding has not been designed to eliminate the duplication of redexes. Instead, it was designed so that it can work around the duplication of redexes.

So far it was stated multiple times that the encoding over-approximates. It would be good to also look into an example of a TRS that is ndg, but whose rules still end up getting encoded.

Example 25.

Consider the following TRS $\mathcal{M}$ with locations in $\mathrm{ENC}_\mathcal{M}$ underlined.

$$\alpha_1 : \mathsf{f} \to \mathsf{f}(\underline{\mathsf{g}(\mathsf{a})}) \qquad \alpha_2 : \mathsf{a} \to \mathsf{s}(\mathsf{0}) \qquad \alpha_3 : \mathsf{g}(\mathsf{s}(x)) \to \mathsf{c}(\underline{x},\underline{x})$$

In this example, the reason why the location of a in α_1 is included in $\mathrm{ENC}_{\mathcal{M}}$ is because before matching to see if a flow occurs, the a-symbol is replaced by a variable through the $\mathrm{CAP}_{\mathcal{M}}$-function. In that case the flow is over-approximated and leads to an encoding of an ndg TRS.

Let us now show that it is indeed ndg. Firstly, there are no nested defined symbols on a lhs of any rule, so one of the conditions of ndg can never be met. The second is the duplication of redexes. This can only occur when evaluating with α_3 and specifically if a redex flows into the variable x. For that matter, we need to consider all other occurrences of g in $\mathcal{M}$. The only other one is in α_1, where the only proper subterm is a redexes. As mentioned previously $\mathsf{g}(\mathsf{a})$ can not be matched with the lhs of α_3. However, we should continue by exploring the evaluations of a, of which there is only one and is obviously in normal form. Therefore, all rewrite steps in $\mathcal{M}$ are ndg, making $\mathcal{M}$ an ndg TRS, which can be analyzed via the technique in [4] without the usage of the encoding. Regardless, the encoding Φ will not preserve the rules and will consider this a non-ndg TRS.

At this point we can start working towards proving that $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}'_{\Phi(\mathcal{R})}$ for any TRS $\mathcal{R}$. As a reminder, we measure the $\mathrm{irc}'_{\Phi(\mathcal{R})}$ as $\mathrm{irc}_{\Phi(\mathcal{R})}$ limited to the basic terms of $\mathcal{R}$. We can start from the $\subseteq$ inclusion, by proving that for any rewrite sequence $t \to^n_{\mathcal{R}} u$ with $t \in \mathcal{T}_{\mathcal{B}}(\Sigma)$, there exists an equally long innermost rewrite sequence in $\Phi(\mathcal{R})$ starting with the same basic term t. In order to prove this statement we need to define a new type of rewrite relation. The reason is that certain full rewrite sequences in the original TRS could not be replicated by using the encoded rules in the same order, while also maintaining an innermost strategy.

Example 25.

Consider the TRS $\mathcal{W}$

$$\alpha_1 : \mathsf{g} \to \mathsf{f}(\mathsf{a}, \underline{\mathsf{a}}) \qquad \alpha_2 : \mathsf{a} \to \underline{\mathsf{a}} \qquad \alpha_3 : \mathsf{f}(x, \underline{\mathsf{a}}) \to \mathsf{f}(x, \underline{\mathsf{a}})$$

and the encoded rules in $\mathcal{W}'$:

$$\alpha'_1 : \mathsf{g} \to \mathsf{f}(\mathsf{a}, \mathsf{i}(l_{\mathsf{a}})) \qquad \alpha'_2 : \mathsf{a} \to l_{\mathsf{a}} \qquad \alpha'_3 : \mathsf{f}(x, l_{\mathsf{a}}) \to \mathsf{f}(x, \mathsf{i}(l_{\mathsf{a}}))$$

Consider the following rewrite sequence

$$\mathsf{g} \quad \to_{\alpha_1} \quad \mathsf{f}(\mathsf{a},\mathsf{a}) \quad \to_{\alpha_3} \quad \mathsf{f}(\mathsf{a},\mathsf{a}) \quad \to_{\alpha_3} \cdots$$

This rewrite sequence cannot be constructed in $\Phi(\mathcal{W})$ and be innermost. The a at position 1 of $\mathsf{f}(\mathsf{a},\mathsf{a})$ is never encoded and thus not in normal form. So when trying to reproduce this rewrite sequence in $\Phi(\mathcal{W})$, it results in $\mathsf{h} \to_{\alpha'_1} \mathsf{f}(\mathsf{a}, \mathsf{i}(l_{\mathsf{a}}))$. At this point there are not many options for innermost rewrite steps and the next non-0-cost rule to be applied can be α'_2 and not α'_3 as in the original sequence. This proves that even if the encoding, given some rewrite sequence in $\mathcal{W}$ as input, can produce innermost rewrite sequence of the same length and with the same starting term, the order of non-0-cost rule applications is not the same. Obviously, the end term is of no interest, but we need to ensure that such a rewrite sequence can be algorithmically produced.

One idea was to extend lemma 8 from [7] to TRSs with 0-cost rules, which states that for a regular TRS $\mathcal{R}$ the following holds: $t \xrightarrow{ndg}{}^{n}_{\mathcal{R}} u \Rightarrow t \xrightarrow{i}{}^{n}_{\mathcal{R}} v$. Since the set of regular TRSs is a subset of all TRSs with 0-cost rules, we can not take this statement for granted. Assuming we can prove it, we could easily show that an ndg rewrite sequence in the encoded TRS of the same length exists, while also maintaining the rule application order of the rewrite sequence in the original TRS. However, a counterexample was discovered using a non-terminating rewrite sequence. Consider the following TRS with a constructor symbol 0 with arity 0, not present in the rules

$$\gamma_1 : \mathsf{f}(x) \to \mathsf{f}(\mathsf{a}) \quad \gamma_2 : \mathsf{a} \xrightarrow{\mathsf{0}} \mathsf{a}$$

The rewrite sequence $\mathsf{f}(\mathsf{0}) \to_{\gamma_1} \mathsf{f}(\mathsf{a}) \to_{\gamma_1} \mathsf{f}(\mathsf{a}) \to_{\gamma_1} \cdots$ is clearly of infinite length, but also ndg, as no duplication or the matching of nested redexes occurs. Meanwhile, the longest innermost rewrite sequence $\mathsf{f}(\mathsf{0}) \to_{\gamma_1} \mathsf{f}(\mathsf{a}) \xrightarrow{\mathsf{0}}_{\gamma_2} \mathsf{f}(\mathsf{a}) \xrightarrow{\mathsf{0}}_{\gamma_2} \cdots$ has a length of one. Excluding non-terminating rewrite sequences, still left a difficult to prove statement even though no counter-examples were found.

The approach that was taken, is to shift the order of rule applications in the original sequence in such a way as to maintain its length, and take that as the input for an algorithm that constructs the rewrite sequence in the encoded TRS. The input is be such that the algorithm does not need to create a different order of rule applications.

This shifting of rule application is not done via an algorithm, but the existence of these rewrite sequence is defined in much the same way as in lemma 8 from [7]. Basically, we want to show that for every full rewrite sequence of length n and starting term t, there is a, what we call, *optional innermost* rewrite sequence with the same length and starting term. The new rewrite relation is based on a tagging, which works similarly to the calculation of non-ndg locations, but cuts the over-approximation.

Definition 24(Optional innermost)**.**

Given a rewrite sequence $\nabla = t_0 \to^n t_n$, a rewrite step in that sequence is **optional innermost** (oi) denoted $t_i \xrightarrow{oi}_\pi t_{i+1}$, if

$$\forall \tau \in \text{pos}(t_i) \text{ with } \tau > \pi \text{ we have either } t_i|_\tau \in \text{NF or } \tau \in \text{tag}_\nabla(i)\ .$$

In order to better illustrate where this rewrite relation stands compared to others, we have the following order: $\xrightarrow{i} \subseteq \xrightarrow{oi} \subseteq \to$. For the first inclusion we have that innermost redexes have no proper subterms not in normal form, therefore no position needs to meet the specified requirement. The second one is trivial. We can now move onto the tagging.

Definition 25.(tag_∇)

1. Given a *terminating* rewrite sequence $\nabla = t_0 \to_{\alpha_1,\pi_1} t_1 \to \cdots \to_{\alpha_n,\pi_n} t_n$ in a TRS $\mathcal{R}$ we define $\text{tag}_\nabla(i) \subseteq pos(t_i)$ as the smallest such that:
   - $\{\tau \mid \tau = \pi_i.v, \quad (\alpha_i\ , \quad \mathtt{R}, \omega) \in \mathcal{X}'_\mathcal{R}, \omega \leq v\} \ \subseteq \text{tag}_\nabla(i)$
   - $\{\tau \mid \tau = \pi_{i+1}.v, (\alpha_{i+1}, \mathtt{L}, \omega) \in \mathcal{X}'_\mathcal{R}, \omega \leq v\} \ \subseteq \text{tag}_\nabla(i)$
   - $\left\{ \tau \;\middle|\; \begin{array}{l} \alpha_{i+1} = \ell_{i+1} \to r_{i+1}, \\ \omega_L \in pos(\ell_{i+1}), \quad \omega_R \in pos(r_{i+1}), \\ x = \ell_{i+1}|_{\omega_L} = r_{i+1}|_{\omega_R} \in \mathcal{V}, \\ \pi_{i+1}.\omega_L.\tau_x = \tau, \quad \pi_{i+1}.\omega_R.\tau_x = \gamma \end{array} \right\} \subseteq \text{tag}_\nabla(i), \text{ if } \gamma \in \text{tag}_\nabla(i+1)$
   - $\{\tau \mid \tau \parallel \pi_{i+1} \qquad\qquad\qquad\qquad \} \ \subseteq \text{tag}_\nabla(i), \text{ if } \tau \in \text{tag}_\nabla(i+1)$
2. Given an infinite rewrite sequence$\nabla = t_0 \to_{\alpha_1,\pi_1} t_1 \to \cdots$, let ∇^n be the first n steps of ∇. Then

$$tag_\nabla(i) = \bigcup_{k=0}^{m} tag_{\nabla^k}(i).$$

Going back to the rewrite sequence in example 25. that showed we can not maintain rule application order in $\Phi(\mathcal{W})$, we get the following tagged rewrite sequence. Tagged positions are underlined.

$$\mathsf{g} \quad \to_{\alpha_1} \quad \mathsf{f}(\mathsf{a}, \underline{\mathsf{a}}) \quad \to_{\alpha_3} \quad \mathsf{f}(\mathsf{a}, \underline{\mathsf{a}}) \quad \to_{\alpha_3} \cdots$$

Keep in mind that the locations in the TRS considered by tag_∇ are only the base non-ndg locations. There is however an infinite oi rewrite sequence, which in our case is also innermost.

$$\mathsf{g} \quad \to_{\alpha_1} \quad \mathsf{f}(\mathsf{a}, \mathsf{a}) \quad \to_{\alpha_2} \quad \mathsf{f}(\mathsf{a}, \mathsf{a}) \quad \to_{\alpha_2} \cdots$$

Lemma 26.

For some TRS $\mathcal{R}$, if $t \to^n_\mathcal{R} v$ then $t \xrightarrow{oi}{}^n_\mathcal{R} u$ for some term u.

We claim lemma 26 without proof. Before we get to the algorithm, which produced the rewrite sequence in the encoded TRS, we must first define two helper functions. Since the encoded terms have more subterms than their decoded versions due to the added i-symbols, we need a function that translates a position from the original to the encoded version. This will be needed when we have to find at what position the next redex in the constructed sequence is located. The other function returns the position of the closest i-symbol above the input position. It will be used when an i-symbol needs to propagated inward.

Definition 25 (Position tr̲anslation)**.**

Given some term t over the signature of an encoded TRS and a position π we define the following function:

$$\mathrm{tr}(t,\pi) = \begin{cases} 1.\mathrm{tr}(t|_1,\pi) & \text{if} \quad root(t) = \mathsf{i} \\ \pi_1.\mathrm{tr}(t|_{\pi_1},\pi_2) & \text{else if} \quad \exists \pi_1 \in \mathbb{N} \text{ such that } \pi = \pi_1.\pi_2 \\ \varepsilon & \text{else} \end{cases}$$

Definition 26 (N̲ext i̲ above)**.**

Given some term t over the signature of an encoded TRS and a position π we define the following function:

$$\mathrm{n\_i}(t,\pi) = \begin{cases} \pi & \text{if} \quad root(t|_\pi) = \mathsf{i} \\ \mathrm{n\_i}(t,\pi_1) & \text{else if} \quad \exists \pi_2 \in \mathbb{N} \text{ such that } \pi = \pi_1.\pi_2 \\ \perp & \text{else} \end{cases}$$

Algorithm 31. (`CreateEncodedSequence`)

```
 1: procedure CreateEncodedSequence($\mathcal{R}$, $t_0 \xrightarrow{oi}_{\alpha_1,\pi_1} \cdots \xrightarrow{oi}_{\alpha_n,\pi_n} t_n$)
 2:     $s_{0,0} \leftarrow t_0$
 3:     $k \leftarrow 0$

 4:     while $k < n$
 5:         $j \leftarrow 0$
 6:         $\omega \leftarrow \mathrm{tr}(s_{k,j}, \pi_{k+1})$

 7:         while $s_{k,j}|_\omega$ is not innermost
 8:             Let $\omega' \geq \omega$ such that $root(s_{k,j}|_{\omega'}) = \mathsf{i}$ and $s_{k,j}|_{\omega'}$ is innermost.
 9:             $\beta_{k,j+1} \leftarrow \mathtt{OMIT}$
10:             $\tau_{k,j+1} \leftarrow \omega'$
11:             $s_{k,j} \;=\; s_{k,j}[\mathtt{OMIT}^L\sigma]_{\tau_{k,j+1}}$
12:             $s_{k,j+1} \leftarrow s_{k,j}[\mathtt{OMIT}^R\sigma]_{\tau_{k,j+1}}$
13:             $j \leftarrow j + 1$
14:         end while

15:         if $\pi_{k+1} \neq \omega$
16:             Let $\omega = \omega_1.\omega_2$ such that $\omega_2 \in \mathbb{N}$
17:             while $root(s_{k,j}|_{\omega_1}) \neq \mathsf{i}$
18:                 $q \leftarrow \mathrm{n\_i}(s_{k,j}|_{\omega_1})$
19:                 $f \leftarrow root(s_{k,j}|_{q.1})$
20:                 $\beta_{k,j+1} \leftarrow \mathtt{PROP}_f$
21:                 $\tau_{k,j+1} \leftarrow q$
22:                 $s_{k,j} \;=\; s_{k,j}[\mathtt{PROP}^L\sigma]_{\tau_{k,j+1}}$
23:                 $s_{k,j+1} \leftarrow s_{k,j}[\mathtt{PROP}^R\sigma]_{\tau_{k,j+1}}$
24:                 $\omega \leftarrow \mathrm{tr}(s_{k,j+1}, \pi_{k+1})$
25:                 Let $\omega = \omega_1.\omega_2$ such that $\omega_2 \in \mathbb{N}$
26:                 $j \leftarrow j + 1$
27:             end while

28:             $f \leftarrow root(s_{k,j}|_\omega)$
29:             $\beta_{k,j+1} \leftarrow \mathtt{EXE}_f$
30:             $\tau_{k,j+1} \leftarrow \omega_1$
31:             $s_{k,j} \;=\; s_{k,j}[\mathtt{EXE}^L\sigma]_{\tau_{k,j+1}}$
32:             $s_{k,j+1} \leftarrow s_{k,j}[\mathtt{EXE}^R\sigma]_{\tau_{k,j+1}}$
33:         end if

34:         $\alpha'_{k+1} \leftarrow \psi(\alpha_{k+1})$
35:         $\pi'_{k+1} \leftarrow \omega$
36:         $s_{k,j} \;=\; s_{k,j}[\varphi(\ell_{k+1})\sigma]_{\pi_{k+1}}$
37:         $s_{k,j+1} \leftarrow s_{k,j}[\varphi(r_{k+1})\sigma]_{\pi_{k+1}}$
38:         $k \leftarrow k + 1$
39:     end while

40:     return $s_{0,0} \xrightarrow{0}_{\beta_{0,1},\tau_{0,1}} \cdots \xrightarrow{0}_{\beta_{0,m_1},\tau_{0,m_1}} s_{0,m_1} \to_{\alpha'_1,\pi'_1} s_{1,0} \to \cdots \to_{\alpha'_n,\pi'_n} s_{n,0}$
41: end procedure
```

We can now go into further detail and better illustrate how the algorithm works. After initialization, i.e. assigning the same starting term as the one from the input sequence and setting the counter k to 0 for the big while-loop spanning lines 4 to 39. Each iteration of this loop constructs the next rewrite step of the output, which may also include some rewrite steps using the relative rules, but always ends by applying a single non-relative rule. In other words, the loop iterates over each rewrite step of the input rewrite sequence and 'simulates' it in the encoded TRS.

Let us now take a deeper look at each iteration. Lines 5 and 6 initialize a second counter j, which counts the relative rewrite steps, and a position ω, set to the translation of π_{k+1} in the current term. Position π_{k+1} refers to the position of the redex at t_k in the input rewrite sequence.

The while-loop from lines 7 to 14 removes all i-symbols, which are bellow the redex, using the `OMIT` rule. This part of the procedure ensures that subsequent rewrite steps are innermost. We will show later why considering only the i-symbols is sufficient. In the algorithm, β and τ are reserved for the details of the relative rewriting, namely the rule and position respectively.

Line 8 chooses the position ω' of a subterm with root symbol i, which is also innermost. This can be performed automatically via a depth-first search. The next lines assign the proper values to evaluate at ω' and updates the value of j.

After the redex at ω has been made innermost, the procedure checks if ω equals the position π_{k+1} from the input rewrite sequence. If that is the case, then we can directly apply the encoded version of the rule used in the rewrite step, that the algorithm is currently simulating. Otherwise, as we will show later, the position is bellow some i-symbol and is also not a redex yet. Therefore, the algorithm will propagate an i-symbol directly above the subterm at ω and restore its encoded symbol to the original defined symbol.

Line 16 splits ω in a way as to give us the position above it, namely ω_1. Then the while-loop at lines 17 to 27 checks after each iteration if the symbol at that position is i. The first two lines of the loop initialize a position q to the position of the closest i-symbol above ω_1, and then assigns f the symbol directly bellow the discovered i-symbol. More precisely that would be the root symbol at $q.1$. The next lines define the next relative rewrite step analogously to the one seen in the previous while-loop. Since propagation of i-symbols changes the structure of the term, the algorithm must update the value of ω_1 and consequently ω.

When the algorithm reaches line 28 after exiting the previous while-loop, the exact conditions to apply `EXE` at ω_1 are met and the defined symbol is restored at ω. The goal of the if-branch from lines 15 to 33 was to create a redex at position ω, where none was there prior to it. The final non-relative rule application is therefore shared by both cases. Line 38 increments k in preparation of the next rewrite step. After n many steps, where n is the length of the input rewrite sequence, the big while-loop terminates and the procedure returns the innermost rewrite sequence in the encoded $\Phi(\mathcal{R})$.

Lemma 28 ($\mathrm{rc}_{\mathcal{R}} \subseteq \mathrm{irc}_{\mathcal{R}'}$).

Let $\mathcal{R}$ be a TRS and $\Phi(\mathcal{R})$ its encoding. Let $t \in \mathcal{T}_{\mathcal{B}}(\Sigma)$ be some basic term.

$$t \to_{\mathcal{R}}^{n} v \quad \Rightarrow \quad t \overset{i}{\to}{}_{\Phi(\mathcal{R})}^{n} u$$

4. Encoding

In this chapter, we introduce the definition of the encoding, which so far has only been partially conceptualized. As a reminder, the goal of the encoding is to encode a TRS $\mathcal{R}$ to $\Phi(\mathcal{R})$, whose innermost runtime complexity is the same as the full runtime complexity of $\mathcal{R}$. Thus allowing us to use the much more powerful techniques for irc analysis to derive results for rc, mainly an upper-bound on rc.

Definition 23.

Let $\mathbb{T}$ be the set of all TRSs. We define **the encoding** as a function $\Phi : \mathbb{T} \to \mathbb{T}$:

$$\Phi(\mathcal{R}) = \mathcal{R}'/\mathcal{S}, \text{ such that}$$

$$\begin{aligned}
\mathcal{R}' &= \{\psi(\alpha) \mid \alpha \in \mathcal{R}\} \\
\mathcal{S} &= \{\mathsf{i}(x) \overset{0}{\to} x\} \\
&\cup \{\, \mathsf{i}(l_f(\mathsf{x_1}, \dots, \mathsf{x_n})) \overset{0}{\to} \mathsf{i}(f(\mathsf{x_1}, \dots, \mathsf{x_n})) \quad \mid f \in \Sigma_d^n\} \\
&\cup \{\, \mathsf{i}(d\,(\mathsf{x_1}, \dots, \mathsf{x_n})) \overset{0}{\to} d(\mathsf{i}(\mathsf{x_1}), \dots, \mathsf{i}(\mathsf{x_n})) \quad \mid d \in \Sigma_c^n\} \\
&\cup \left\{ \mathsf{i}(l_f(\mathsf{x_1}, \dots, \mathsf{x_n})) \overset{0}{\to} \mathsf{i}(l_f(\mathsf{i}(\mathsf{x_1}), \dots, \mathsf{i}(\mathsf{x_n}))) \;\middle|\; \begin{array}{l} f = root(\mathcal{R}|_\lambda) \in \Sigma_d, \lambda \in \mathrm{ENC}_\mathcal{R}, \\ \exists \mu \in \mathrm{INF}_\mathcal{R}, root(\mathcal{R}|_\mu) = f \end{array} \right\} \\
&\cup \left\{ \mathsf{i}(l_f(\mathsf{x_1}, \dots, \mathsf{x_n})) \overset{0}{\to} l_f(\mathsf{i}(\mathsf{x_1}), \dots, \mathsf{i}(\mathsf{x_n})) \;\middle|\; \begin{array}{l} f = root(\mathcal{R}|_\lambda) \in \Sigma_d, \lambda \in \mathrm{ENC}_\mathcal{R}, \\ \forall \mu \in \mathrm{INF}_\mathcal{R}, root(\mathcal{R}|_\mu) \neq f \end{array} \right\}
\end{aligned}$$

ψ encodes the rules in $\mathcal{R}$ and uses φ to encode the individual sides of each rule. We define the ψ-function follows:

$$\psi(\alpha) = \varphi(\mathcal{R}|_{(\alpha,\mathtt{L},\varepsilon)}) \to \varphi(\mathcal{R}|_{(\alpha,\mathtt{R},\varepsilon)}), \;\; \alpha \in \mathcal{R}$$

For some location $\lambda = (\alpha, \mathtt{X}, \pi)$, $\mathtt{X} \in \{\mathtt{L}, \mathtt{R}\}$, $(\lambda, k) \in \mathrm{NST}_\mathcal{R}$ and $\mathcal{R}|_\lambda = f$, let $\lambda^{(n)} = (\alpha, \mathtt{X}, \pi.n)$. If $f \notin \mathcal{V}$, let $f \in \Sigma^n$.

1. $$\varphi(\mathcal{R}|_\lambda) = \begin{cases} f(\varphi(\mathcal{R}|_{\lambda^{(1)}}), \dots, \varphi(\mathcal{R}|_{\lambda^{(n)}})) & \text{if} \quad f \in \Sigma_c \text{ or } \lambda \notin \mathrm{ENC}_\mathcal{R} \\ l_f(\varphi(\mathcal{R}|_{\lambda^{(1)}}), \dots, \varphi(\mathcal{R}|_{\lambda^{(n)}})) & \text{else if } \mathtt{X} = \mathtt{L} \\ \mathsf{i}\;(l_f(\varphi_\mathcal{N}(\mathcal{R}|_{\lambda^{(1)}}), \dots, \varphi_\mathcal{N}(\mathcal{R}|_{\lambda^{(n)}}))) & \text{else if } \lambda \in \mathrm{INF}_\mathcal{R} \\ \mathsf{i}^k(l_f(\varphi_\mathcal{N}(\mathcal{R}|_{\lambda^{(1)}}), \dots, \varphi_\mathcal{N}(\mathcal{R}|_{\lambda^{(n)}}))) & \text{else if } f \notin \mathcal{V} \\ \mathsf{i}^k(f) & \text{else} \end{cases}$$

2. $$\varphi_\mathcal{N}(\mathcal{R}|_\lambda) = \begin{cases} f(\varphi(\mathcal{R}|_{\lambda^{(1)}}), \dots, \varphi(\mathcal{R}|_{\lambda^{(n)}})) & \text{if} \quad f \in \Sigma_c \\ l_f(\varphi(\mathcal{R}|_{\lambda^{(1)}}), \dots, \varphi(\mathcal{R}|_{\lambda^{(n)}})) & \text{else if } f \in \Sigma_d \\ f & \text{else} \end{cases}$$

We can now see precisely how a TRS gets encoded via Φ. so an example that incorporates different aspects of the encoding is presented.

Example 24.

Consider the following TRS $\mathcal{P}$ and all the relevant information about its encoding. The locations in $\mathrm{ENC}_\mathcal{P}$ are highlighted, the nesting sizes of the relevant location taken from $\mathrm{NST}_\mathcal{P}$ is listed above them, and the only location in $\mathrm{INF}_\mathcal{P}$ is underlined in rule α_3.

$$\alpha_1 : \mathsf{h} \to \mathsf{a}(\overbrace{\mathsf{f}(\mathsf{c})}^{2}) \qquad\qquad\qquad \alpha_2 : \mathsf{a}(x) \to \mathsf{a}(\overbrace{\underline{\mathsf{a}(x)}}^{1})$$

$$\alpha_3 : \mathsf{f}(\mathsf{c}) \to \overbrace{\mathsf{f}(\mathsf{c})}^{2} \qquad \alpha_4 : \mathsf{a}(x) \to \mathsf{b}(\overbrace{\ x\ }^{2}, \overbrace{\ x\ }^{2}) \qquad \alpha_5 : \mathsf{c} \to \mathsf{0}$$

We are now going take a look at how the result of $\Phi(\mathcal{P})$ is calculated by going through each step. Terms with zero non-ndg marked locations are skipped, since the first condition of φ clearly ignores these and changes nothing in the process. First we encode each rule to give us $\mathcal{P}'$:

- $\psi(\alpha_1) = \mathsf{h} \to \varphi(\mathsf{a}(\mathsf{f}(\mathsf{c}))) \quad = \mathsf{h} \to \mathsf{a}(\varphi(\mathsf{f}(\mathsf{c}))) \quad = \mathsf{h} \to \mathsf{a}(\mathsf{i}^2(l_\mathsf{f}(\varphi_\mathcal{N}(\mathsf{c})))) \quad = \mathsf{h} \to \mathsf{a}(\mathsf{i}^2(l_\mathsf{f}(l_\mathsf{c}))$
- $\psi(\alpha_2) = \mathsf{a}(x) \to \varphi(\mathsf{a}(\mathsf{a}(x))) = \mathsf{a}(x) \to \mathsf{a}(\varphi(\mathsf{a}(x))) = \mathsf{a}(x) \to \mathsf{a}(\mathsf{i}(l_\mathsf{a}(\varphi_\mathcal{N}(x)))) = \mathsf{a}(x) \to \mathsf{a}(\mathsf{i}(l_\mathsf{a}(x)))$
- $\psi(\alpha_3) = \varphi(\mathsf{f}(\mathsf{c})) \to \varphi(\mathsf{f}(\mathsf{c})) \ = \mathsf{f}(\varphi(\mathsf{c})) \to \mathsf{i}^2(l_\mathsf{f}(\varphi_\mathcal{N}(\mathsf{c}))) = \mathsf{f}(l_\mathsf{c}) \to \mathsf{i}^2(l_\mathsf{f}(l_\mathsf{c}))$
- $\psi(\alpha_4) = \mathsf{a}(x) \to \varphi(\mathsf{b}(x,x)) \ = \mathsf{a}(x) \to \mathsf{b}(\varphi(x), \varphi(x)) \quad = \mathsf{a}(x) \to \mathsf{b}(\mathsf{i}^2(x), \mathsf{i}^2(x))$
- $\psi(\alpha_5) = \alpha_5$

Describing every step in detail is pointless, but in a few words we can summarize the process of encoding the rules of $\mathcal{P}$ as follows. The subterm on the lhs of α_2 contains a constant size nest that flows into the duplication of α_4 and thus is encapsulated by two i-symbols. A repeated nesting has been discovered in the next rule α_2. Therefore, only a single i-symbol is placed in front of it. In α_3 is our only case of nested defined symbol on a lhs in TRS $\mathcal{P}$. It is only replaced by a fresh constructor symbol and the entire rhs of α_3 is encoded, since these are return locations of f, marked non-ndg in α_1. The rhs of α_4 contains the duplicated variables, which actually have a nesting size of 2 due to the nest from α_1. Next come the additional rules. Instead of grouping them by category, where all PROP rules and all EXE rules are defined in one set definition, it is much more readable and easy to understand if we group them by function symbols. We start with the OMIT rule, which is not tied to any function symbol. Next we iterate over the symbols which are highlighted above, i.e. which get encoded. Let the first symbol be f. The encoding adds $\mathtt{EXE}_\mathsf{f}$ and the terminating $\mathtt{PROP}_\mathsf{f}$, since no location in $\mathrm{INF}_\mathcal{P}$ has the same root symbol.

- $\mathcal{S} \supseteq \{\mathsf{i}(x) \xrightarrow{0} x\} \cup \{\mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{0} \mathsf{i}(\mathsf{f}(x)),\ \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{0} \quad l_\mathsf{f}(\mathsf{i}(x)))\ \}$

Next symbol we pick is c. It has an arity of 0, therefore no $\mathtt{PROP}_\mathsf{c}$ rule can be added. Thus only $\mathtt{EXE}_\mathsf{c}$ is included in $\mathcal{S}$.

- $\mathcal{S} \supseteq \{\mathsf{i}(x) \xrightarrow{0} x\} \cup \{\mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{0} \mathsf{i}(\mathsf{f}(x)),\ \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{0} \quad l_\mathsf{f}(\mathsf{i}(x))\ \} \cup \{\mathsf{i}(l_\mathsf{c}) \xrightarrow{0} \mathsf{i}(\mathsf{c})\}$

The last function symbol to consider is a. There is a location in $\mathrm{INF}_\mathcal{P}$, which has the same root symbol, so the encoding adds the non-terminating $\mathtt{PROP}_\mathsf{a}$ along with its executing rule.

- $\mathcal{S} \supseteq \{\mathsf{i}(x) \xrightarrow{0} x\} \cup \{\mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{0} \mathsf{i}(\mathsf{f}(x)),\ \ \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{0} \quad l_\mathsf{f}(\mathsf{i}(x))\ \} \cup \{\mathsf{i}(l_\mathsf{c}) \xrightarrow{0} \mathsf{i}(\mathsf{c})\}$
  $\cup \{\mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{0} \mathsf{i}(\mathsf{a}(x)),\ \mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{0} \mathsf{i}(l_\mathsf{a}(\mathsf{i}(x)))\}$

By definition we should also add propagation rules for constructor symbols, but that was not precisely demonstrated previously and only mentioned in passing. In TRS $\mathcal{P}$ there is only one constructor symbol with non-zero arity - b. It is not marked as non-ndg, since locations with constructor root symbols were excluded from $\mathrm{ENC}_\mathcal{P}$. Sparing any further analysis on the TRS, we can simply add the terminating rules for all constructor symbols, with which $\mathcal{S}$ is finally complete.

- $\mathcal{S} = \{\mathsf{i}(x) \xrightarrow{0} x\} \cup \{\mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{0} \mathsf{i}(\mathsf{f}(x)),\ \ \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{0} \quad l_\mathsf{f}(\mathsf{i}(x))\ \} \cup \{\mathsf{i}(l_\mathsf{c}) \xrightarrow{0} \mathsf{i}(\mathsf{c})\}$
  $\cup \{\mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{0} \mathsf{i}(\mathsf{a}(x)),\ \mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{0} \mathsf{i}(l_\mathsf{a}(\mathsf{i}(x)))\} \cup \{\mathsf{i}(\mathsf{b}(x,x) \xrightarrow{0} \mathsf{b}(\mathsf{i}(x),\mathsf{i}(x))\}$

With these results we can construct the full encoding $\Phi(\mathcal{P})$

$$\begin{array}{lll}
\alpha'_1 : \mathsf{h} \to \mathsf{a}(\mathsf{i}^2(l_\mathsf{f}(l_\mathsf{c})) & & \alpha'_2 : \mathsf{a}(x) \to \mathsf{a}(\mathsf{i}(l_\mathsf{a}(x))) \\
\alpha'_3 : \mathsf{f}(l_\mathsf{c}) \to \mathsf{i}^2(l_\mathsf{f}(l_\mathsf{c})) & \alpha'_4 : \mathsf{a}(x) \to \mathsf{b}(\mathsf{i}^2(x), \mathsf{i}^2(x)) & \alpha'_5 : \mathsf{c} \to 0
\end{array}$$

$$\begin{array}{lll}
\beta_1 : \mathsf{i}(\mathsf{x}) \xrightarrow{0} \mathsf{x} & \beta_2 : \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{0} \mathsf{i}(\mathsf{f}(x)) & \beta_3 : \mathsf{i}(l_\mathsf{f}(x)) \xrightarrow{0} l_\mathsf{f}(\mathsf{i}(x)) \\
\beta_4 : \mathsf{i}(l_\mathsf{c}) \xrightarrow{0} \mathsf{i}(\mathsf{c}) & \beta_5 : \mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{0} \mathsf{i}(\mathsf{a}(x)) & \beta_6 : \mathsf{i}(l_\mathsf{a}(x)) \xrightarrow{0} \mathsf{i}(l_\mathsf{a}(\mathsf{i}(x))) \\
 & \beta_7 : \mathsf{i}(\mathsf{b}(x, x) \xrightarrow{0} \mathsf{b}(\mathsf{i}(x), \mathsf{i}(x)) &
\end{array}$$

One thing that is important to clarify is that the result of the encoding is not ndg. When we talk about expanding the technique from [4], we do not mean transforming the TRS into an equivalent ndg TRS, for which irc = rc holds. We can show this via a simple rewrite sequence:

$$\mathsf{h} \quad \to_{\alpha'_1} \quad \mathsf{a}(\mathsf{i}(l_\mathsf{f}(l_\mathsf{c}))) \quad \to_{\alpha'_4} \quad \mathsf{b}(\mathsf{i}^3(l_\mathsf{f}(l_\mathsf{c})), \mathsf{i}^3(l_\mathsf{f}(l_\mathsf{c})))$$

What we see in the second rewrite step is a duplication of a redex, namely the one at position 1. Therefore $\Phi(\mathcal{P})$ is not an ndg TRS, but that is not a problem, since the encoding has not been designed to eliminate the duplication of redexes. Instead, it was designed so that it can work around the duplication of redexes.

So far it was stated multiple times that the encoding over-approximates. It would be good to also look into an example of a TRS that is ndg, but whose rules still end up getting encoded.

Example 25.

Consider the following TRS $\mathcal{M}$ with all locations in $\mathrm{ENC}_\mathcal{M}$ highlighted.

$$\begin{array}{ll}
\alpha_1 : \mathsf{h} \to \mathsf{f}(\mathsf{dbl}(\mathsf{a}, \mathsf{a}), \mathsf{s}(0)) & \alpha_2 : \mathsf{dbl}(x, \mathsf{s}(y)) \to \mathsf{d}(y, y) \\
\alpha_3 : \mathsf{f}(x, \mathsf{s}(y)) \to \mathsf{f}(x, y) & \alpha_4 : \mathsf{a} \to \mathsf{s}(0)
\end{array}$$

In this example, the reason why the location of a in α_1 is included in $\mathrm{ENC}_\mathcal{M}$ is because before matching to see if a flow occurs, the a-symbol is replaced by a variable through the $\mathrm{CAP}_\mathcal{M}$-function. In that case the flow is over-approximated and leads to an encoding of an ndg TRS.

Let us now show that it is indeed ndg. Firstly, there are no nested defined symbols on a lhs of any rule, so one of the conditions of ndg can never be met. The second is the duplication of redexes. This can only occur when evaluating with α_2 and specifically if a redex flows into the variable y. For that matter, we need to consider all other occurrences of dbl in $\mathcal{M}$. The only other one is in α_1, where both proper subterms are redexes. As mentioned previously dbl(a, a) can not be matched with the lhs of α_2. However, we should continue by exploring the evaluations of a, of which there is only one and is obviously in NF. Therefore, all rewrite steps in $\mathcal{M}$ are ndg, making $\mathcal{M}$ an ndg TRS, which can be analyzed via the technique in [4] without the usage of the encoding,

At this point we can start working towards proving that $\mathrm{rc}_\mathcal{R} = \mathrm{irc}_{\Phi(\mathcal{R})}$ for any TRS $\mathcal{R}$. We can start from the $\subseteq$ inclusion, by proving that for any rewrite sequence $t \to^n_\mathcal{R} u$ with $t \in \mathcal{T}_\mathcal{B}(\Sigma)$, there exists an equally long rewrite sequence in $\Phi(\mathcal{R})$ starting with the same basic term t. In order to prove this statement we need to define a new type of rewrite relation. The reason is that certain full rewrite sequences in the original TRS could not be replicated by using the encoded rules in the same order, while also maintaining an innermost strategy. We can use the TRS $\mathcal{M}$ from example 25 to better illustrate that. It does not matter that the original TRS was ndg, the analysis on its encoding has to return the same results, as if it was not ndg. First, let us establish $\Phi(\mathcal{M})$.

$$\begin{array}{ll}
\alpha'_1 : \mathsf{h} \to \mathsf{f}(\mathsf{dbl}(\mathsf{a}, \mathsf{i}(l_\mathsf{a})), \mathsf{s}(0)) & \alpha'_2 : \mathsf{dbl}(x, \mathsf{s}(y)) \to \mathsf{d}(\mathsf{i}(y), \mathsf{i}(y)) \\
\alpha'_3 : \mathsf{f}(x, \mathsf{s}(y)) \to \mathsf{f}(x, y) & \alpha'_4 : \mathsf{a} \to \mathsf{s}(0) \\
\alpha'_5 : \mathsf{i}(x) \xrightarrow{0} x & \alpha'_6 : \mathsf{i}(l_\mathsf{a}) \xrightarrow{0} \mathsf{i}(\mathsf{a})
\end{array}$$

Consider the following rewrite sequence in $\mathcal{M}$:

$$\mathsf{h} \quad \to_{\alpha_1} \quad \mathsf{f}(\mathsf{dbl}(\mathsf{a}, \mathsf{a}), \mathsf{s}(0)) \quad \to_{\alpha_3} \quad \mathsf{f}(\mathsf{dbl}(\mathsf{a}, \mathsf{a}), 0)$$

This rewrite sequence cannot be constructed in $\Phi(\mathcal{M})$ and be innermost. The a at position 1 of dbl(a, a) is never encoded and thus not in NF. So when trying to reproduce this rewrite sequence in $\Phi(\mathcal{M})$ results in $\mathsf{h} \to_{\alpha'_1} \mathsf{f}(\mathsf{dbl}(\mathsf{a}, \mathsf{i}(l_\mathsf{a})), \mathsf{s}(0))$. At this point there are not many options for innermost rewrite steps and the next non-relative rule to be applied would be α'_4 and not α'_3 as in the original sequence. This proves that even if the encoding, given some rewrite sequence in $\mathcal{M}$ as input, can produce innermost rewrite sequence of the same length and with the same starting term, the order of non-relative rules applications is not the same. Obviously, the end term is of no interest, but we need to ensure that such a rewrite sequence can be algorithmically produced.

One idea was to extend Lemma 8. from [7] to relative TRSs, which states that for a non-relative TRS $\mathcal{R}$ the following holds: $t \xrightarrow{ndg}{}^{n}_{\mathcal{R}} u \Rightarrow t \xrightarrow{i}{}^{n}_{\mathcal{R}} v$. Since the set of non-relative TRSs is a subset of all relative TRSs, we cannot take this statement for granted. Assuming we can prove it, we could easily show that an ndg rewrite sequence in the encoded TRS of the same length exists, while also maintaining the rule application order of the rewrite sequence in the original TRS. However, a counterexample was discovered using a non-terminating rewrite sequence. Consider the following TRS with a constructor symbol 0 with arity 0, not present in the rules

$$\gamma_1 : \mathsf{f}(x) \to \mathsf{f}(\mathsf{a}) \quad \gamma_2 : \mathsf{a} \xrightarrow{0} \mathsf{a}$$

The rewrite sequence $\mathsf{f}(0) \to_{\gamma_1} \mathsf{f}(\mathsf{a}) \to_{\gamma_1} \mathsf{f}(\mathsf{a}) \to_{\gamma_1} \cdots$. It is clearly of infinite length, but also ndg, as no duplication or nesting of redexes on a lhs occurs. Meanwhile, the longest innermost rewrite sequence $\mathsf{f}(0) \to_{\gamma_1} \mathsf{f}(\mathsf{a}) \xrightarrow{0}_{\gamma_2} \mathsf{f}(\mathsf{a}) \xrightarrow{0}_{\gamma_2} \cdots$ has a length of one. Excluding non-terminating rewrite sequences, still left a difficult to prove statement even though no counter-examples were found.

The approach that was taken, is to shift the order of rule applications in the original sequence in such a way as to maintain its length, and take that as the input for an algorithm that constructs the rewrite sequence in the encoded TRS. The input will be such that the algorithm would not need to create a different order of rule applications.

This shifting of rule application is not done via an algorithm, but the existence of these rewrite sequence is defined in much the same way as in Lemma 8. from [7]. Basically, we want to show that for every full rewrite sequence of length n and starting term t, there is a, what we called, *optional innermost* rewrite sequence with the same length and starting term. The new rewrite relation is based on a tagging, which works similarly to the calculation of non-ndg locations, but cuts the over-approximation.

Definition 24(Optional innermost)**.**

Given a rewrite sequence $\nabla = t_0 \to^n t_n$, a rewrite step in that sequence is **optional innermost** (oi) denoted $t_i \xrightarrow{oi}_\pi t_{i+1}$, if

$$\tau \gneq \pi \text{ such that } t_i|_\tau \notin \mathrm{NF} \;\Rightarrow \tau \in \mathit{tag}_\nabla(t_i) \ .$$

In order to better illustrate where this rewrite relation stands compared to others, we have the following order: $\xrightarrow{i} \subseteq \xrightarrow{oi} \subseteq \to$. For the first inclusion we have that innermost redexes have no proper subterms not in NF, therefore no position needs to meet the specified requirement. The second one is trivial. We can now move onto the tagging.

Definition 25.(tag_∇)

1. Given a *terminating* rewrite sequence $\nabla = t_0 \to_{\alpha_1,\pi_1} t_1 \to \cdots \to_{\alpha_n,\pi_n} t_n$ in a TRS $\mathcal{R}$ we define $\mathit{tag}_\nabla(t_i) \subseteq \mathit{pos}(t_i)$ as the smallest such that:
   - $\{\tau \mid \tau = \pi_i.\upsilon, \quad (\alpha_i, \mathtt{R}, \omega) \in \mathcal{X}'_\mathcal{R}, \omega \leq \upsilon\} \ \subseteq \mathit{tag}_\nabla(t_i)$
   - $\{\tau \mid \tau = \pi_{i+1}.\upsilon, (\alpha_i, \mathtt{L}, \omega) \in \mathcal{X}'_\mathcal{R}, \omega \leq \upsilon\} \ \subseteq \mathit{tag}_\nabla(t_i)$
   - $\left\{ \tau \;\middle|\; \begin{array}{l} \alpha_{i+1} = \ell_{i+1} \to r_{i+1}, \\ \omega_L \in \mathit{pos}(\ell_{i+1}), \quad \omega_R \in \mathit{pos}(r_{i+1}), \\ x = \ell_{i+1}|_{\omega_L} = r_{i+1}|_{\omega_R} \in \mathcal{V}, \\ \pi_{i+1}.\omega_L.\tau_x = \tau, \quad \pi_{i+1}.\omega_R.\tau_x = \gamma \end{array} \right\} \subseteq \mathit{tag}_\nabla(t_i)$, if $\gamma \in \mathit{tag}_\nabla(t_{i+1})$

2. Given a *non-terminating* rewrite sequence $\nabla = t_0 \to_{\alpha_1,\pi_1} t_1 \to \cdots$, let ∇^n be the first n steps of ∇. Then there exists $m \in \mathbb{N}$ such that:

$$tag_{\nabla}(t_i) = \bigcup_{k=0}^{m} tag_{\nabla^k}(t_i).$$

Going back to the rewrite sequence in $\mathcal{M}$ that showed we can not maintain rule application order in $\Phi(\mathcal{M})$ and continuing it to termination, we get the following tagged rewrite sequence:

$$\mathsf{h} \quad \to_{\alpha_1} \quad \mathsf{f}(\mathsf{dbl}(\mathsf{a},\mathsf{a}),\mathsf{s}(0)) \quad \to_{\alpha_3} \quad \mathsf{f}(\mathsf{dbl}(\mathsf{a},\mathsf{a}),0) \quad \to_{\alpha_4} \quad \mathsf{f}(\mathsf{dbl}(\mathsf{a},\mathsf{s}(\underline{0})),0) \quad \to_{\alpha_2} \quad \mathsf{f}(\mathsf{d}(\underline{0},\underline{0}),0)$$

The underlined position indicate that they in the tag set of the respective term. We want to show now that if we take any oi rewrite sequence starting with h, it is also of at least length four. Since no redexes can get tagged in this example, it holds that $\xrightarrow{oi}_{\mathcal{M}} = \xrightarrow{i}_{\mathcal{M}}$.

$$\mathsf{h} \quad \xrightarrow{oi}_{\alpha_1} \quad \mathsf{f}(\mathsf{dbl}(\mathsf{a},\mathsf{a}),\mathsf{s}(0)) \quad \xrightarrow{oi}_{\alpha_4} \quad \mathsf{f}(\mathsf{dbl}(\mathsf{a},\mathsf{s}(0)),\mathsf{s}(0)) \quad \xrightarrow{oi}_{\alpha_4} \quad \mathsf{f}(\mathsf{dbl}(\mathsf{s}(0),\mathsf{s}(\underline{0})),\mathsf{s}(0)) \quad \xrightarrow{oi}_{\alpha_2} \quad \mathsf{f}(\mathsf{d}(\underline{0},\underline{0}),0)$$

Lemma 26.

For some TRS $\mathcal{R}$, if $t \to^n_{\mathcal{R}} v$ then $t \xrightarrow{oi}{}^{n}_{\mathcal{R}} u$ for some term u.

Before we get to the algorithm, which produced the rewrite sequence in the encoded TRS, we must first define two helper functions. Since the encoded terms have more subterms than their decoded versions due to the added i-symbols, we need a function that translates a position from the original to the encoded version. This will be needed when we have to find at what position the next redex in the constructed sequence is located. The other function returns the position of the closest i-symbol above the input position. It will be used when an i-symbol needs to propagated inward.

Definition 25 (Position $\underline{\text{tr}}$anslation)**.**

Given some term t from the signature of an encoded TRS and a position π we define the following function:

$$\mathrm{tr}(t,\pi) = \begin{cases} 1.\mathrm{tr}(t|_1,\pi) & \text{if } \ root(t) = \mathsf{i} \\ \pi_1.\mathrm{tr}(t|_{\pi_1},\pi_2) & \text{if } \ \exists \pi_1 \in \mathbb{N} \text{ such that } \pi = \pi_1.\pi_2 \\ \varepsilon & \text{else} \end{cases}$$

Definition 26 ($\underline{\text{N}}$ext $\underline{\text{i}}$ above)**.**

Given some term t from the signature of an encoded TRS and a position π we define the following function:

$$\mathrm{n\_i}(t,\pi) = \begin{cases} \pi & \text{if } \ root(t|_\pi) = \mathsf{i} \\ \mathrm{n\_i}(t,\pi_1) & \text{if } \ \exists \pi_2 \in \mathbb{N} \text{ such that } \pi = \pi_1.\pi_2 \\ \bot & \text{else} \end{cases}$$

Algorithm 31. (`CreateEncodedSequence`)

```
 1: procedure CreateEncodedSequence($\mathcal{R}$, $t_0 \xrightarrow{oi}_{\alpha_1,\pi_1} \cdots \xrightarrow{oi}_{\alpha_n,\pi_n} t_n$)
 2:     $s_{0,0} \leftarrow t_0$
 3:     $k \leftarrow 0$

 4:     while $k < n$
 5:         $j \leftarrow 0$
 6:         $\omega \leftarrow \mathrm{tr}(s_{k,j}, \pi_{k+1})$

 7:         while $s_{k,j}|_\omega$ is not innermost
 8:             Let $\omega' \geq \omega$ such that $root(s_{k,j}|_{\omega'}) = \mathsf{i}$ and $s_{k,j}|_{\omega'}$ is innermost.
 9:             $\beta_{k,j+1} \leftarrow \mathtt{OMIT}$
10:             $\tau_{k,j+1} \leftarrow \omega'$
11:             $s_{k,j} \;=\; s_{k,j}[\mathtt{OMIT}^L\sigma]_{\tau_{k,j+1}}$
12:             $s_{k,j+1} \leftarrow s_{k,j}[\mathtt{OMIT}^R\sigma]_{\tau_{k,j+1}}$
13:             $j \leftarrow j + 1$
14:         end while

15:         if $\pi_{k+1} \neq \omega$
16:             Let $\omega = \omega_1.\omega_2$ such that $\omega_2 \in \mathbb{N}$
17:             while $root(s_{k,j}|_{\omega_1}) \neq \mathsf{i}$
18:                 $q \leftarrow \mathrm{n\_i}(s_{k,j}|_{\omega_1})$
19:                 $f \leftarrow root(s_{k,j}|_{q.1})$
20:                 $\beta_{k,j+1} \leftarrow \mathtt{PROP}_f$
21:                 $\tau_{k,j+1} \leftarrow q$
22:                 $s_{k,j} \;=\; s_{k,j}[\mathtt{PROP}^L\sigma]_{\tau_{k,j+1}}$
23:                 $s_{k,j+1} \leftarrow s_{k,j}[\mathtt{PROP}^R\sigma]_{\tau_{k,j+1}}$
24:                 $\omega \leftarrow \mathrm{tr}(s_{k,j+1}, \pi_{k+1})$
25:                 Let $\omega = \omega_1.\omega_2$ such that $\omega_2 \in \mathbb{N}$
26:                 $j \leftarrow j + 1$
27:             end while

28:             $f \leftarrow root(s_{k,j}|_\omega)$
29:             $\beta_{k,j+1} \leftarrow \mathtt{EXE}_f$
30:             $\tau_{k,j+1} \leftarrow \omega_1$
31:             $s_{k,j} \;=\; s_{k,j}[\mathtt{EXE}^L\sigma]_{\tau_{k,j+1}}$
32:             $s_{k,j+1} \leftarrow s_{k,j}[\mathtt{EXE}^R\sigma]_{\tau_{k,j+1}}$
33:         end if

34:         $\alpha'_{k+1} \leftarrow \psi(\alpha_{k+1})$
35:         $\pi'_{k+1} \leftarrow \omega$
36:         $s_{k,j} \;=\; s_{k,j}[\varphi(\ell_{k+1})\sigma]_{\pi_{k+1}}$
37:         $s_{k,j+1} \leftarrow s_{k,j}[\varphi(r_{k+1})\sigma]_{\pi_{k+1}}$
38:         $k \leftarrow k + 1$
39:     end while

40:     return $s_{0,0} \xrightarrow{0}_{\beta_{0,1},\tau_{0,1}} \cdots \xrightarrow{0}_{\beta_{0,m_1},\tau_{0,m_1}} s_{0,m_1} \to_{\alpha'_1,\pi'_1} s_{1,0} \to \cdots \to_{\alpha'_n,\pi'_n} s_{n,0}$
41: end procedure
```

We can now go into further detail and better illustrate how the algorithm works. After initialization, i.e. assigning the same starting term as the one from the input sequence and setting the counter k to 0 for the big while-loop spanning lines 4 to 39. Each iteration of this loop constructs the next rewrite step of the output, which may also include some rewrite steps using the relative rules, but always ends by applying a single non-relative rule. In other words, the loop iterates over each rewrite step of the input rewrite sequence and 'simulates' it in the encoded TRS.

Let us now take a deeper look at each iteration. Lines 5 and 6 initialize a second counter j, which counts the relative rewrite steps, and a position ω, set to the translation of π_{k+1} in the current term. Position π_{k+1} refers to the position of the redex at t_k in the input rewrite sequence.

The while-loop from lines 7 to 14 removes all i-symbols, which are bellow the redex, using the `OMIT` rule. This part of the procedure ensures that subsequent rewrite steps are innermost. We will show later why considering only the i-symbols is sufficient. In the algorithm, β and τ are reserved for the details of the relative rewriting, namely the rule and position respectively.

Line 8 chooses the position ω' of a subterm with root symbol i, which is also innermost. This can be performed automatically via a depth-first search. The next lines assign the proper values to evaluate at ω' and updates the value of j.

After the redex at ω has been made innermost, the procedure checks if ω equals the position π_{k+1} from the input rewrite sequence. If that is the case, then we can directly apply the encoded version of the rule used in the rewrite step, that the algorithm is currently simulating. Otherwise, as we will show later, the position is bellow some i-symbol and is also not a redex yet. Therefore, the algorithm will propagate an i-symbol directly above the subterm at ω and restore its encoded symbol to the original defined symbol.

Line 16 splits ω in a way as to give us the position above it, namely ω_1. Then the while-loop at lines 17 to 27 checks after each iteration if the symbol at that position is i. The first two lines of the loop initialize a position q to the position of the closest i-symbol above ω_1, and then assigns f the symbol directly bellow the discovered i-symbol. More precisely that would be the root symbol at $q.1$. The next lines define the next relative rewrite step analogously to the one seen in the previous while-loop. Since propagation of i-symbols changes the structure of the term, the algorithm must update the value of ω_1 and consequently ω.

When the algorithm reaches line 28 after exiting the previous while-loop, the exact conditions to apply `EXE` at ω_1 are met and the defined symbol is restored at ω. The goal of the if-branch from lines 15 to 33 was to create a redex at position ω, where none was there prior to it. The final non-relative rule application is therefore shared by both cases. Line 38 increments k in preparation of the next rewrite step. After n many steps, where n is the length of the input rewrite sequence, the big while-loop terminates and the procedure returns the innermost rewrite sequence in the encoded $\Phi(\mathcal{R})$.

Lemma 28 ($\mathrm{rc}_{\mathcal{R}} \subseteq \mathrm{irc}_{\mathcal{R}'}$)**.**

Let $\mathcal{R}$ be a TRS and $\Phi(\mathcal{R})$ its encoding. Let $t \in \mathcal{T}_{\mathcal{B}}(\Sigma)$ be some basic term.

$$t \to_{\mathcal{R}}^{n} v \quad \Rightarrow \quad t \xrightarrow{i}{}^{n}_{\Phi(\mathcal{R})} u$$

Proof. We apply Lemma 26. to $t \to_{\mathcal{R}}^{n} v$ and obtain $\nabla = t \xrightarrow{oi}{}^{n}_{\mathcal{R}} v'$ for some term v'. The procedure $\textsc{CreateEncodedSequence}(\mathcal{R}, \nabla)$ returns a rewrite sequence ∇' in $\Phi(\mathcal{R})$. Showing that the procedure terminates, that ∇' exists in $\Phi(\mathcal{R})$, that ∇' is innermost and of the same length, proves Lemma 28.

1. ∇' exists in $\Phi(\mathcal{R})$

The starting term of ∇ is defined as basic and the procedure copies it for the starting term of its output. For the first iteration of the big while-loop, we know that $\omega = \pi_1 = \epsilon$ and that $s_{0,0}$ is already innermost. The procedure then goes directly to line 34 and applies the encoded $\psi(\alpha_1)$. We know that $\varphi(\ell_1) = \ell_1$, since the only time φ changes the lhs of a rule is when there is a nested defined symbol. This is not the case here, since the starting term is basic and therefore cannot be matched to such lhs of any rule. Afterwards, k is incremented and the loop goes into the next iteration.

The loop has reached term $s_{1,0}$. The reasoning behind every rewrite step going forward is analogous, therefore we show how the procedure operates for terms $s_{k,0}$, where $1 \le k < n$.

After the first rule application, its possible that the next produced term no longer equals its counterpart in the input rewrite sequence. That would be due to encoding adding i-symbols at the over-approximated non-ndg locations of the TRS. That is why the procedure has to translate the position of the next redex from the input.

Regardless of the translation, we want the algorithm to produce an innermost rewrite sequence. Since the corresponding rewrite step in the input is oi, we can conclude that the positions of all subterms bellow π_{k+1} not in NF are in $tag(t_k)$. The encoding Φ over-approximates these locations in $\mathcal{R}$ and subsequently it holds that in each rewrite sequence, subterms at positions in $tag(t_k)$ are encapsulated by i-symbols (1). It also holds that the root symbols of any subterm bellow these i-symbols is either another i-symbol or a constructor (2). Therefore, removing all i-symbols bellow the translated ω position, ensures that the subterm at it is not non-innermost. Each `OMIT` rewrite step is also innermost as per the definition of ω' at line 8. (3)

The next part of the procedure is to check if the translated position is equal to π_{k+1}. This case distinction is needed for the situations when the redex is over-approximated to flow into a non-ndg location, i.e. the subterm at ω is encapsulated by an i-symbol.

Case I. ($\pi_{k+1} \neq \omega$) From (2) we also know that the root symbol at ω is a constructor, which by the definition of the translation function can be restored to some defined symbol. Since it is not necessary that the root symbol at ω_1, i.e. the position directly above ω, is an i-symbol, the procedure has to propagate one to it from the closest non-parallel position containing an i-symbol, which is conveniently returned from the function n_i.

This is done by the while-loop from lines 17 to 27, which terminates once an i-symbol is directly above ω. Each iteration calculates the new closest non-parallel position with root symbol i and propagates at it. These `PROP` rewrite steps are innermost, since all other i-symbols bellow ω have been removed and because it follows from (2) that the subterm at the position calculated by n_i is innermost (4).When this while-loop terminates, the `EXE` rewrite step restores the defined symbol at the now updated ω.

Case II. ($\pi_{k+1} = \omega$) In this case we know that the subterm at ω is already an innermost redex. For subterm that were not in NF the while-loop from lines 7 to 14 removed all nested i-symbols and from (2) we know all that is left at these positions are constructors.

Convergence of both cases. At this point we have some term $s_{k,j}$ and an innermost redex at position ω. We now have to show that the redex can be matched to ℓ_{k+1}. It follows from the procedure so far that $dec(s_{k,j}) = t_k$. If there are nested defined symbols in ℓ_{k+1}, then these are encoded by Φ to their constructor versions. These locations always tag positions in t_k and from (1) we know that the constructor version of said symbol is also present at the same position in $s_{k,j}$. Therefore, the encoded rule $\psi(\alpha_{k+1})$ can be applied ω and we are done. If there are no nested defined symbols in ℓ_{k+1}, then the matching is trivial and we are done.

2. ∇' is an innermost rewrite sequence

Most of this has already been shown in the part above, namely statements $(3-4)$ cover the removal and propagation of i-symbols. And since the `PROP` rewrite steps are innermost, then it follows the `EXE` step is also innermost, since it evaluates at a position where an i-symbol was lastly propagated. Consequently the last non-relative step per iteration is also innermost.

3. ∇' is has a length of n

The length of rewrite sequences is only affected by non-relative rewrite steps. The big while-loop in the procedure produces a single non-relative rewrite step per iteration and it iterates n many times, where n is the length of the input rewrite sequence.

4. The procedure CreateEncodedSequence always terminates

There are three while-loops in the procedure. The big while-loop that contains the other two, terminates trivially since the parameter k is incremented at the end of each iteration and is not changed elsewhere.

The while-loop from lines 7 to 14 terminates once all i-symbols bellow position ω are removed. Since no other i-symbols are introduced in the process and there are finitely many of them at the start, we can conclude this loop also always terminates.

The last while-loop to consider is from lines 17 to 27. Before entering this loop it is ensured that there is a position above ω which contains a subterm with root symbol i. The process of propagation does not remove any i-symbols in the process and therefore will eventually terminate.

The two other function calls in the procedure, namely tr and n_i, do calculate recursively, but they obviously always terminate.

The proof of these statements suffices to prove Lemma 2_

□

Definition 27 (Term decoding).

Let Σ and Σ' be the signatures of $\mathcal{R}$ and $\Phi(\mathcal{R})$ respectively. Let $t \in \mathcal{T}(\Sigma', \mathcal{V})$ with $t = f(t_0, \ldots, t_n)$. We define the function

$$dec(t) = \begin{cases} dec(t|_1) & \text{if} \quad f = \mathsf{i} \\ g(dec(t_1), \ldots, dec(t_n)) & \text{else if} \quad f = l_g \in \Sigma' \\ f(dec(t_1), \ldots, dec(t_n)) & \text{else} \end{cases}$$

Lemma 28.

Let $\mathcal{R}$ be a TRS and $\Phi(\mathcal{R}) = \mathcal{R}' \uplus \mathcal{S}$ its encoding, where $\mathcal{S}$ holds all the relative rules.

$$t \xrightarrow{i}_{\Phi(\mathcal{R})} v \quad \Rightarrow \quad dec(t) \rightarrow_{\mathcal{R}} dec(v)$$

Proof. Represent $t \xrightarrow{i}_{\Phi(\mathcal{R})} v$ as

$$t = t_0 \xrightarrow{i}_{\mathcal{S}} \cdots \xrightarrow{i}_{\mathcal{S}} t_n \xrightarrow{i}_{\mathcal{R}'} v_0 \xrightarrow{i}_{\mathcal{S}} \cdots \xrightarrow{i}_{\mathcal{S}} v_m = v$$

It holds that $dec(t_j) = dec(t_k), \forall j, k$. Since the decoder removes all i-symbols and restores the original defined symbol of it encounters its constructor version, no amount of rewriting with the rules in $\mathcal{S}$ can change the decoding of the term. Analogously, $dec(v_j) = dec(v_k), \forall j, k$.
Therefore, proving $t_n \xrightarrow{i}_{\mathcal{R}'} v_0 \Rightarrow dec(t_n) \rightarrow dec(v_0)$ also shows $dec(t) \rightarrow dec(v)$.Let $t_n \rightarrow_{\psi(\alpha),\pi} v_0$. We define τ via $\pi = tr(t_n, \tau)$. Let $t_n = C[\varphi(\ell)\sigma]$. Define context $D = dec(C)$ with $D|_\tau = \Box$. We have

$$dec(C[\varphi(\ell)\sigma]) = D[dec(\varphi(\ell)\sigma)] = D[\ell\sigma'] = dec(t_n),$$

where $\sigma = \{x_0 \backslash q_0,\ x_1 \backslash q_1,\ \ldots,\ x_p \backslash q_p\}$ and $\sigma' = \{x_0 \backslash dec(q_0),\ x_1 \backslash dec(q_1),\ \ldots,\ x_p \backslash dec(q_p)\}$. Therefore

$$dec(C[\varphi(r)\sigma]) = D[dec(\varphi(r)\sigma)] = D[r\sigma'] = dec(v_0).$$

This gives us the rewrite step

$$D[\ell\sigma'] = dec(t_n) \rightarrow_{\alpha,\tau} dec(v_0) = D[r\sigma']$$

and we are done.

□

Lemma 29. ($\mathrm{rc}_{\mathcal{R}} \supseteq \mathrm{irc}_{\Phi(\mathcal{R})}$)

Let $\mathcal{R}$ be a TRS with signature Σ and $\Phi(\mathcal{R})$ its encoding. Let $t \in \mathcal{T}(\Sigma)$ be some basic term.

$$t \xrightarrow{i}{}^{n}_{\Phi(\mathcal{R})} v \quad \Rightarrow \quad t \rightarrow^{n}_{\mathcal{R}} dec(v)$$

Proof. For $n = 0$ it is trivial. For $n > 0$ apply the induction hypothesis on the first $n-1$ steps.

$$t \xrightarrow{i}{}^{n-1}_{\Phi(\mathcal{R})} u \xrightarrow{i}_{\Phi(\mathcal{R})} v \quad \Rightarrow \quad t \rightarrow^{n-1}_{\mathcal{R}} dec(u)$$

We apply Lemma 1. on the last $u \xrightarrow{i}_{\Phi(\mathcal{R})} v$ step to get $dec(u) \rightarrow_{\mathcal{R}} dec(v)$. We now have the sequence

$$t \rightarrow^{n}_{\mathcal{R}} dec(v)$$

and we are done.

□

Corollary 30. ($\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\Phi(\mathcal{R})}$)

Let $\mathcal{R}$ be a TRS and $\Phi(\mathcal{R})$ its encoding. Then $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\Phi(\mathcal{R})}$.

5. Results

Bibliography

[1] G. Moser, “Proof theory at work: Complexity analysis of term rewrite systems,” 2009.

[2] M. Avanzini, G. Moser, and M. Schaper, “TcT: Tyrolean Complexity Tool,” in *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, vol. 9636. Springer Verlag Heidelberg, 2016, pp. 407–423.

[3] F. Frohn, J. Giesl, J. Hensel, C. Aschermann, and T. Ströder, “Lower bounds for runtime complexity of term rewriting,” *Journal of Automated Reasoning*, vol. 59, pp. 1–43, 06 2017.

[4] F. Frohn and J. Giesl, “Analyzing runtime complexity via innermost runtime complexity,” in *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning.* EasyChair, 2017.

[5] Termination competition. [Online]. Available: https://termination-portal.org/wiki

[6] F. Baader and T. Nipkow, *Term Rewriting and All That.* Cambridge University Press, 1998. [Online]. Available: https://www.cambridge.org/core/books/term-rewriting-and-all-that/71768055278D0DEF4FFC74722DE0D707

[7] J. Pol, van de and H. Zantema, “Generalized innermost rewriting,” in *Rewriting Techniques and Applications (Proceedings 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005)*, ser. Lecture Notes in Computer Science, J. Giesl, Ed. Germany: Springer, 2005, pp. 2–16.

[8] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke, “Mechanizing and improving dependency pairs,” *J. Autom. Reasoning*, vol. 37, pp. 155–203, 10 2006.

[9] S. Lechner, “Data flow analysis for integer term rewrite systems,” *RWTH*, 2022.