Optimizing Term Rewriting with Creeper Trace Transducers

Rick Erkens
Eindhoven University of Technology
The Netherlands
r.j.a.erkens@tue.nl

In the context of functional programming/term normalization algorithms we discuss the optimization problem of constructing the result of a sequence of rewrite steps, without computing all the intermediate terms. From a rewrite system we construct a so-called creeper trace transducer, which reads a creeper trace while producing the desired answer. The transducer skips overlap between each pair of subsequent rules, and in some cases a part of the trace can be disregarded altogether.

1 Introduction

In previous work [4] we described how a subterm pattern matching can be used for efficient term rewriting. This work gave rise to another optimization problem, best introduced by example. Consider the rewrite rule $R_* : *(x,0) \to 0$ and the following sequence of rewrite steps:

$$*(u_1,*(u_2,*(u_3,0))) \to *(u_1,*(u_2,0)) \to *(u_1,0) \to 0\,.$$

In our context it is possible to discover which rules have to be applied at which positions, without computing all intermediate terms. In this example, we can obtain an encoding of the steps given by $\varepsilon \cdot R_* \cdot 2 \cdot R_* \cdot 2.2 \cdot R_*$ without rewriting any term. This *creeper trace* consists of pairs of positions and rules, where the left-hand side of each rule overlaps the right-hand side of the next rule and the positions creep up to the root of the term. If we are only interested in the final result, and not in the intermediates, it is more efficient to construct the result with the information in the creeper trace. In this example many steps can be skipped, since we can immediately observe from $\varepsilon \cdot R_*$ that the result of the steps is 0. Another example is found in the rule $R_+ : +(x,s(y)) \to s(+(x,y))$ and the following reductions:

$$+(u_1,+(u_2,s(u_3))) \to +(u_1,s(+(u_2,u_3))) \to s(+(u_1,+(u_2,u_3)))\,.$$

The successor symbol in the second term demonstrates overlap that can be skipped by only using the creeper trace $\varepsilon \cdot R_+ \cdot 2 \cdot R_+$.

In this paper we address this optimization problem in general. From an arbitrary rewrite system, we construct a *creeper trace transducer* that reads the rules and the positions of a creeper trace, while producing the result from top to bottom. The transducer can be used on many traces and many terms, making transducer construction an excellent operation to perform at compile-time. In theory constructing the result with the transducer is efficient since all duplicate work between subsequent overlapping rules is skipped. Especially in rewrite systems with zero-element rules like $*(x,0) \to 0$ and $or(x,true) \to true$, a lot of work can be skipped. Moreover, practical implementations of rewriting that support maximally shared terms need to check whether a newly created term already exists in the term storage. Since the transducers construct fewer intermediate terms, we also economize on such queries to the term storage.

The focus of this paper is on the explanation of the approach and the mathematical details. The creeper trace transducer approach is designed to be used in a rewrite engine based on a subterm matching

Submitted to:
WPTE 2023

algorithm [4]. In this setting, one rewrite step can infer a creeper trace of length n in $O(n)$ time if the structure of the right-hand sides is embedded in the pattern matching automaton. The details of creeper trace construction exceed the scope of this paper, as they require a thorough understanding of the subterm pattern matching algorithm in [8] in order to expand upon it.

Related work Backward overlap is a well-known efficiency problem in functional programming. A lot of research has been conducted in the context of Haskell [13] on deforestation [9, 10, 15, 16]: a program transformation technique that yields a more efficient program by eliminating tree data structures. The approach in this paper addresses the same problem in a completely different way, since no program transformation is involved. Creeper trace transducers can be used in functional programming as well.

In the Spineless Tagless G-Machine [12] pattern matching with the '`case (f x) of`' construction is performed one operator at a time, which makes it feasible to economize and not write the constructors of `(f x)`. In [14] the GHC compiler is extended to support user-defined rewrite rules, thereby allowing one to circumvent the overlap problem manually.

Origin tracking [7] shares some formalities with this paper since the formal goal can be seen as dual. Instead of constructing the result, the subterms of the result are related to their origin in the start term.

Transducers are a popular topic in computer science [5]. This paper may be related to tree transducer theory, in particular to streaming string/tree transducers [1, 2]. The existence of a clear connection is unknown as of yet due to the specific nature of the algorithmic problem discussed in this paper.

2 Preliminaries

We recap some preliminaries on term rewriting [3]. Given a ranked alphabet $\mathbb{F}$ and a set of variables $\mathbb{V}$, the set of *terms over* $\mathbb{F}$ *with variables in* $\mathbb{V}$ is denoted by $\mathbb{T}(\mathbb{F},\mathbb{V})$. The variables of a term t are given by $\mathsf{vars}(t)$. A *ground term* is an element of $\mathbb{T}(\mathbb{F},\emptyset)$. A *substitution* is a mapping $\sigma : \mathbb{V} \to \mathbb{T}(\mathbb{F},\mathbb{V})$. The application of σ to a term t is denoted by t^σ.

A *position* is a list of positive natural numbers. We use $\mathbb{P}$ to denote the set of all positions and we use ε to denote the empty list; it is referred to as the *root position*. Given two positions p, q their concatenation is denoted by $p.q$. Position q is deeper than position p, denoted $p \leq q$, iff there is a position r such that $p.r = q$. Two positions p,q are disjoint iff $p \not\leq q$ and $q \not\leq p$. On non-disjoint positions (say p and $p.q$) we define their difference by $p.q - p = q$. Every term has a *domain*: a set of positions at which it has function symbols or variables, assigned to it by the mapping $\mathscr{D} : \mathbb{T}(\mathbb{F},\mathbb{V}) \to \mathscr{P}(\mathbb{P})$. Similarly we define the *edge* of a term $\mathscr{E}(t)$ to be the set of positions where it has variables. Given a position $p \in \mathscr{D}(t)$, the subterm of t at position p is denoted by $t|_p$. By $t[u]_p$ we denote the term obtained by replacing the subterm of t at position p by term u. A *pattern* is a term $\ell \in \mathbb{T}(\mathbb{F},\mathbb{V}) \setminus \mathbb{V}$. We say that t *matches* u iff there is a substitution σ such that $t = u^\sigma$. If, additionally, u is a pattern, we say that t *overlaps* u.

A *rewrite rule* is pair of terms ℓ,r denoted by $\ell \to r$, such that ℓ is a pattern and $\mathsf{vars}(r) \subseteq \mathsf{vars}(\ell)$. A *term rewrite system* (TRS) is a finite, non-empty set of rewrite rules. A *redex* of t is a rule $\ell \to r$ and a position $p \in \mathscr{D}(t)$ such that $t|_p$ matches ℓ. Such a redex is denoted by $(\ell \to r)@p$. Term t rewrites to t' by redex $(\ell \to r)@p$, denoted by $t' \xleftarrow{(\ell\to r)@p} t$, if and only if there is a term u with $p \in \mathscr{D}(u)$, and a substitution σ such that $t = u[\ell^\sigma]_p$ and $t' = u[r^\sigma]_p$. The notation $t' \xleftarrow{(\ell\to r)@p} t$ is unconventional, but for most of the paper it is more intuitive since we will read sequences of rewrite steps from left to right.

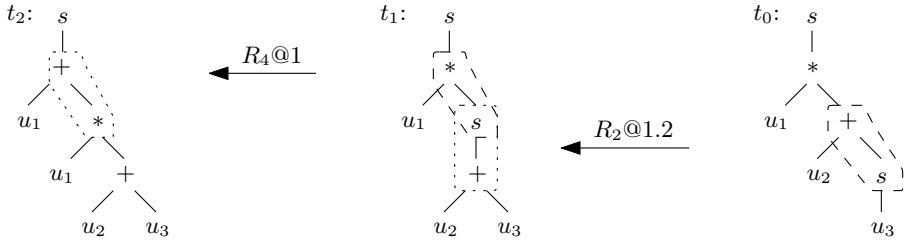

Figure 1: Reducing t_0 in two overlapping steps. The matching left-hand sides are surrounded by dashed lines; the corresponding right-hand sides are surrounded by dots.

3 Creeper traces in rewriting

We discuss the notion of traces of overlapping rules by starting with the textbook example of addition and multiplication on natural numbers: the TRS $\mathscr{R}_{nat}$ consisting of the following rules.

$$R_1 : +(x,0) \to x \qquad R_2 : +(x,s(y)) \to s(+(x,y))$$
$$R_3 : *(x,0) \to 0 \qquad R_4 : *(x,s(y)) \to +(x,*(x,y))$$

Consider for example $t_0 = s(*(u_1,+(u_2,s(u_3))))$ and the sequence of reductions in Figure 1. Rewriting t_0 with redex $R_2@1.2$ yields $t_1 = s(*(u_1,s(+(u_2,u_3))))$. The successor symbol that was 'pushed up' creates a new redex $R_4@1$. It can be used to obtain $t_2 = s(+(u_1,*(u_1,+(u_2,u_3))))$. Observe that the successor symbol in the intermediate term t_1 at position 1.2 merely serves as a conduit by enabling an additional step to t_2. A matching algorithm can combine the new right-hand side with the context of the redex (the multiplication symbol) and immediately yield another match, *without computing the intermediate term*.

In this paper we assume that such a matching algorithm provides us with a *trace*: a sequence of rules and positions that can be applied to a starting term. In this example we have the trace $1 \cdot R_4 \cdot 1.2 \cdot R_2$ of t_0. We describe a transducer that can read this trace while producing the output. When rules overlap as in the previous example, this method always saves some work. We save a lot of work in more extreme examples such as the term $+(u_1,*(u_2,*(u_3,0)))$ and the trace $\varepsilon \cdot R_2 \cdot 2 \cdot R_4 \cdot 2.2 \cdot R_4$. Computing the result naively requires three steps:

$$u_1 \xleftarrow{R_2@\varepsilon} +(u_1,0) \xleftarrow{R_4@2} +(u_1,*(u_2,0)) \xleftarrow{R_4@2.2} +(u_1,*(u_2,*(u_3,0)))\,.$$

By carefully performing the steps in the reverse order we can see that after reading the position ε followed by the rule R_2, the result should be u_1. The rest of the trace is irrelevant.

Let t_0 be a term. A *trace of* t_0 is a finite, non-empty word of alternating positions and rules $p_n \cdot R_n \cdot p_{n-1} \cdot R_{n-1} \cdot \ \ldots\ \cdot p_1 \cdot R_1 \in (\mathbb{P} \cdot \mathscr{R})^+$ such that $t_n \xleftarrow{R_n@p_n} \ldots \xleftarrow{R_1@p_1} t_0$ for some sequence of terms $t_n,\ldots,t_1$. We call t_0 the *start* and t_n the *result* of said trace. We present the start t_0 on the right and the result t_n on the left, since we will read the trace from left to right.

Currently the method is limited to a special kind of overlapping traces [6, 11]. Let $\ell \to r$ be a rule and let $p \in \mathscr{D}(\ell)$. The *backward overlap set* of $\ell \to r$ with respect to p is the set of rules whose right-hand side overlaps $\ell|_p$:

$$\mathsf{BOS}(\ell \to r, p) = \{\ell' \to r' \in \mathscr{R} \mid r' \text{ overlaps } \ell|_p\}\,.$$

A trace $p_n \cdot R_n \cdot \ \ldots\ \cdot p_1 \cdot R_1 \in (\mathbb{P} \cdot \mathscr{R})^+$ is called a *creeper trace* iff for all $i < n$ we have $p_i \leq p_{i-1}$ and $R_{i-1} \in \mathsf{BOS}(R_i, p_{i-1} - p_i)$. That is, the positions are of increasing depth and every right-hand side overlaps the previous left-hand side at the difference in positions.

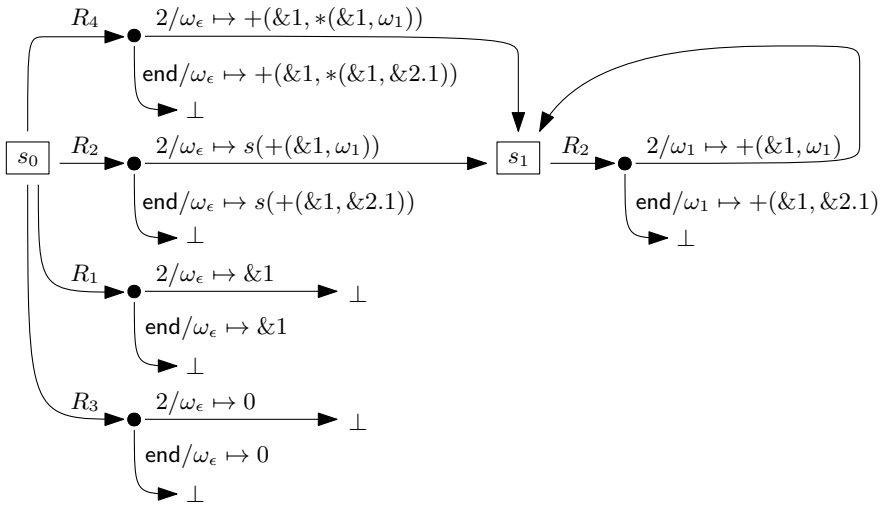

Figure 2: The creeper trace transducer for $\mathscr{R}_{nat}$.

In Figure 1 we have that $1 \cdot R_4 \cdot 1.2 \cdot R_2$ is a creeper trace of t_0, since $1 \leq 1.2$ and the right-hand side $s(+(x,y))$ of R_2 overlaps the left-hand side $*(x,s(y))$ of R_4 at position $1.2 - 1 = 2$. In Section 8 we discuss some non-examples.

4 An example creeper trace transducer

In Figure 2 there is a transducer for $\mathscr{R}_{nat}$. We explain its construction informally and give an example run before proceeding with the details in the next sections. One step in the transducer consists of reading a rule followed by a position or end. After a step it writes a term given by the substitution after the position. From s_0 we expect to read a rule and a position that has no overlap with the previously read rule/position pair. From state s_1 we expect to read a rule whose right-hand side overlaps the left-hand side of rule R_2 or R_4 at position 2. This can only be R_2, so s_1 only has specified behaviour for this rule.

We interpret the transducer on $t_0 = s(*(u_1, +(u_2, s(u_3))))$ and the trace $tr = 1 \cdot R_4 \cdot 1.2 \cdot R_2$. To this end we maintain a *configuration* that consists of four pieces of information.

- A subterm of t_0, initially $t_0|_1 = *(u_1, +(u_2, s(u_3)))$. Some transitions copy a part of t_0. Carefully keeping track of the subterm guarantees that the correct part of t_0 is copied along the run.
- The state, initially s_0.
- The result under construction, called a *writable term*, initially $t_0[\omega_\varepsilon]_1 = s(\omega_\varepsilon)$. Until we finish the run, this term contains *write variables* of the form ω_p. Every transition yields a substitution on all of the variables in this term.
- The trace with relativized positions, initially $R_4 \cdot 2 \cdot R_2 \cdot \mathsf{end}$. A finite state machine cannot have one transition for every position that can occur in a trace, so we use a relativized (or normalized) version where every position p_i is replaced by $p_i - p_{i+1}$. In this case $\mathsf{relativize}(1 \cdot R_4 \cdot 1.2 \cdot R_2) = R_4 \cdot 2 \cdot R_2 \cdot \mathsf{end}$. Three things changed: the initial position is removed, position 1.2 is replaced by $1.2 - 1 = 2$, and there is an end symbol after the last rule.

So initially we have a configuration $\langle t, s, u, \tau \rangle$ where:

$$t = *(u_1, +(u_2, s(u_3))) \qquad s = s_0 \qquad u = s(\omega_\varepsilon) \qquad \tau = R_4 \cdot 2 \cdot R_2 \cdot \mathsf{end}\,.$$

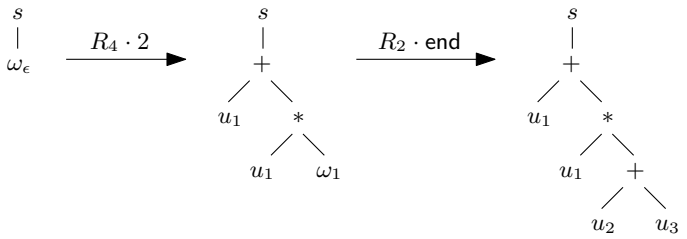

Figure 3: Constructing the result from top to bottom with the transducer for $\mathscr{R}_{nat}$.

The following steps can be tracked in Figure 3. From state s_0 we read the rule R_4 followed by relative position 2. We follow the topmost edges $s_0 \xrightarrow{R_4} \bullet \xrightarrow{2/\omega_\varepsilon \mapsto +(\&1,*(\&1,\omega_1))} s_1$. The latter edge carries a substitution, making us replace all occurrences of ω_ε in the writable term by $+(\&1,*(\&1,\omega_1))$. This is the right-hand side of the rule that we just read, but the variables are different. The variable ω_1 plays the same role as ω_ε in the sense that it indicates where we are going to write next. Furthermore there are *references*, i.e. variables of the form &1. They indicate which subterm of t we have to copy. In this case we need to substitute &1 by $t|_1 = u_1$. The new result under construction is thus $s(+(u_1,*(u_1,\omega_1)))$, the new state is s_1 and the relative trace only has two symbols left. The new subterm of t is $t|_2$ since we read position 2, landing us in the following configuration:

$$t = +(u_2, s(u_3)) \qquad s = s_1 \qquad u = s(+(u_1,*(u_1,\omega_1))) \qquad \tau = R_2 \cdot \mathsf{end}\,.$$

From state s_1 we read the remainder of the trace $R_2 \cdot \mathsf{end}$, and follow the edges $s_1 \xrightarrow{R_2} \bullet \xrightarrow{\mathsf{end}/\omega_1 \mapsto +(\&1,\&2.1)} \bot$. This step yields the substitution $\omega_1 \mapsto +(\&1,\&2.1)$. Note the difference from the previous step. Previously we wrote the entire right-hand side of the rule we just read, and now we only write a part of the right-hand side of R_2. The successor symbol is not written, because it has been used by rule R_4.

When we apply the substitution to u, we obtain $s(+(u_1,*(u_1,+(\&1,\&2.1))))$. This term no longer has any write variables. To obtain the final result we resolve the references by replacing &1 by $t|_1 = u_2$, and replacing &1.2 by $t|_{1.2} = u_3$, resulting in $s(+(u_1,*(u_1,+(u_2,u_3))))$.

Another example is the term $+(u_1,*(u_2,*(u_3,0)))$ and the creeper trace $\varepsilon \cdot R_1 \cdot 2 \cdot R_3 \cdot 2.2 \cdot R_3$. We follow the same initialization as in the previous example. The initial configuration is

$$t = +(u_1,*(u_2,*(u_3,0))) \qquad s = s_0 \qquad u = \omega_\varepsilon \qquad \tau = R_1 \cdot 2 \cdot R_3 \cdot 2 \cdot R_3\,.$$

From s_0 we read $R_1 \cdot 2$ and follow the edges $s_0 \xrightarrow{R_1} \bullet \xrightarrow{2/\omega_\varepsilon \mapsto \&1} \bot$, and do not end up in a new state. Then we apply the obtained substitution to u and get &1, which is simply a reference. Dereferencing with respect to t yields $t|_1 = u_1$, which is the desired result.

5 Formal definitions

In this section we formalize terms with references, dereferencing, writable terms, relativized traces, and creeper trace transducers. In the next section we discuss transducer construction.

Terms with references The standard definition of the single step rewrite relation uses substitutions and variable bindings. A rewrite rule $\ell \to r$ requires that every variable of r occurs in ℓ. This is necessary

to compute the result $t_c[r^\sigma]$ of a rewrite step originating from $t_c[\ell^\sigma]$. For the operational nature of this paper it is necessary to map every variable of r to a position in ℓ where this variable occurs. A *term with references* is a term where some variables are positions (i.e. an element of $\mathbb{T}(\mathbb{F},\mathbb{V}\cup\mathbb{P})$, assuming $\mathbb{V}\cap\mathbb{P}=\emptyset$). We denote the occurrence of a position as a variable by $\&p$.

With every rewrite rule $\ell\to r$ we associate an *originated right-hand side* $\mathsf{origin}(\ell\to r)=r^\sigma$, where the substitution $\sigma:\mathsf{vars}(r)\to\mathscr{E}(\ell)$ yields for every variable of r, the shortest, then leftmost position where this variable occurs in ℓ. An example that we have already seen is $\mathsf{origin}(+(x,s(y)))\to s(+(x,y)))=s(+(\&1,\&2.1))$. A more interesting example is the rule $R:f(g(x,y),x,x)\to g(y,x)$, with $\mathsf{origin}(R)=g(\&1.2,\&2)$. The variable x could be mapped to position 1.1 or 3, but we choose position 2 since it is shorter than 1.1 and it is left of 3.

Now we can operationally define the result of a specific rewrite step. Given a term with references $t_r\in\mathbb{T}(\mathbb{F},\mathbb{V}\cup\mathbb{P})$ and a ground term t, define $\mathsf{deref}(t_r,t)=t_r^\sigma$ where $\sigma(\&p)=t|_p$ for all position variables $\&p\in\mathbb{P}$ and $\sigma(x)=x$ if $x\in\mathbb{V}$. This operational definition is more convenient than using variable bindings, and closer to an implementation. The following proposition states that dereferencing an originated rule correctly characterizes a rewrite step.

Proposition 5.1. If $t'\xleftarrow{\ell\to r@p}t$ then $t'=t[\mathsf{deref}(\mathsf{origin}(\ell\to r),t|_p)]_p$.

Relativized traces One transition requires reading a rule R_i, followed by reading the difference between the position p_i of that rule and the position p_{i-1} of the next rule. Instead of letting the transducer compute this difference, we assume that we directly operate on the trace that has these positions readily available. To this end, given a creeper trace $tr=p_n\cdot R_n\cdot\dots\cdot p_1\cdot R_1$ we define the *relativized version of* tr, by $\mathsf{relativize}(tr)=R_n\cdot q_{n-1}\cdot\dots\cdot q_1\cdot R_1\cdot\mathsf{end}$ where $q_i=p_{i-1}-p_i$ for all $i<n$. Note that there is a special symbol indicating the end of the trace, and the prepending position p_n is removed. It plays a role during initialization, and is irrelevant for the rest of the computation.

Writable terms We use a reserved set of *write variables* $\Omega=\{\omega_p\mid p\in\mathbb{P}\}$ to indicate where we continue to write. A *writable term* is an element of $\mathbb{T}(\mathbb{F},\Omega)$ with at least one variable. In the interpretation of the transducer, the position subscripts merely serve as identifiers, but in the construction of the transducer they play an important role.

Creeper trace transducers Let $\mathbb{T}(\mathbb{F},X)^\mathbb{V}$ denote the set of substitutions from $\mathbb{V}$ to $\mathbb{T}(\mathbb{F},X)$. A *creeper trace transducer for a TRS* $\mathscr{R}$ and write variables Ω is a 5-tuple $(S,s_0,\varphi,\psi,\delta)$ where:

- S is a set of states, including the initial state s_0.
- $\varphi:S\times\mathscr{R}\rightharpoonup\mathbb{T}(\mathbb{F},\mathbb{P})^\Omega$ is a partial *termination function*, yielding for every state and every rule that can be read next in a trace, the terms that must be written on the write variables when reaching the end of the trace. Instead of $\varphi(s,R)=\alpha$ we write $s\xrightarrow{R}\bullet\xrightarrow{\mathsf{end}/\alpha}\bot$.
- $\psi:S\times\mathscr{R}\times\mathbb{P}\rightharpoonup\mathbb{T}(\mathbb{F},\mathbb{P})^\Omega$ is a partial *discard function*, indicating that terms must be written, but the end of the trace is not reached yet. Instead of $\psi(s,R,p)=\alpha$ we write $s\xrightarrow{R}\bullet\xrightarrow{p/\alpha}\bot$.
- $\delta:S\times\mathscr{R}\times\mathbb{P}\rightharpoonup S\times\mathbb{T}(\mathbb{F},\mathbb{P}\cup\Omega)^\Omega$ is a partial *transition function*. Instead of $\delta(s,R,p)=(s',\alpha)$ we write $s\xrightarrow{R}\bullet\xrightarrow{p/\alpha}s'$.

Recall Figure 2. For a clean graphical notation we use the black dot to group the edges per rule. The termination function is displayed by edges accompanied by end. The edges $s_0\xrightarrow{R_1}\bullet\xrightarrow{2/\omega_\varepsilon\mapsto\&1}\bot$ and

$s_0 \xrightarrow{R_3} \bullet \xrightarrow{2/\omega_\varepsilon \mapsto 0} \bot$ are discards. All the other edges specify transitions. Note that φ, ψ and δ are partial. We only deal with creeper traces, so not every state can meaningfully support every rule.

Interpreting a transducer The *configurations* of a transducer are 4-tuples of the form $\langle t, s, u, \tau \rangle$ where t is a term, s is a state, u is a writable term, and τ is a relativized creeper trace. Consider a start term t_0 and a creeper trace $tr = p_n \cdot R_n \cdot \ \ldots \ \cdot p_1 \cdot R_1$. The initial configuration is obtained by descending t_0 to arrive at the starting position p_n, creating a writable term from t_0 by replacing its subterm at position p_n by the write variable ω_ε, and relativizing the trace. Formally the *initial configuration* is given by

$$\langle t_0|_{p_n}, s_0, t_0[\omega_\varepsilon]_{p_n}, \mathsf{relativize}(tr) \rangle .$$

Note the role of the prepending p_n. It is removed upon relativization, and only used to obtain the correct subterm of t_0 and to create the writable term. We define the interpretation of a transducer by a step relation $\rightsquigarrow$ on configurations, with the possibility to terminate in a term:

$$\langle t, s, u, R \cdot \mathsf{end} \rangle \rightsquigarrow \mathsf{deref}(u^\alpha, t) \qquad \text{if } s \xrightarrow{R} \bullet \xrightarrow{\mathsf{end}/\alpha} \bot$$

$$\langle t, s, u, R \cdot p \cdot \tau \rangle \rightsquigarrow \mathsf{deref}(u^\alpha, t) \qquad \text{if } s \xrightarrow{R} \bullet \xrightarrow{p/\alpha} \bot$$

$$\langle t, s, u, R \cdot p \cdot \tau \rangle \rightsquigarrow \langle t|_p, s', \mathsf{deref}(u^\alpha, t), \tau \rangle \qquad \text{if } s \xrightarrow{R} \bullet \xrightarrow{p/\alpha} s'$$

After every step the position variables are dereferenced locally with respect to t, after which t is descended to position p to preserve correct dereferencing.

6 Construction of a creeper trace transducer

We define a creeper trace transducer for an arbitrary rewrite system $\mathscr{R}$. We define the state space formally by identifying every state s by a subset of the TRS denoted by $\rho(s)$, and a set of *offset positions* $\pi(s)$. These determine how φ, ψ and δ are defined on s. For every rule $R \in \rho(s)$ we define $\varphi(s, R)$ and for every position q that has a non-empty backward overlap set w.r.t. R, we define either $\psi(s, R, q)$ or $\delta(s, R, q)$. The offsets define which part of the next right-hand side we still have to write. They tell us which write variables are in a result term under construction. For the initial state we fix $\rho(s_0) = \mathscr{R}$ and $\pi(s_0) = \{\varepsilon\}$. A creeper trace can start with any rule, and initially the only write variable is ω_ε.

6.1 Terminations, discards and transitions from a state

We define the termination, discard, and transition functions on an arbitrary state s with $\pi(s) = \{p_1, \ldots, p_n\}$, for every rule $R \in \rho(s)$.

Termination function The termination function determines what should be done when the end of a trace is reached by specifying what should be written on the remaining write variables. The resulting substitution is defined by

$$\varphi(s, R) = [\omega_{p_1} \mapsto \mathsf{origin}(R)|_{p_1}, \ldots, \omega_{p_n} \mapsto \mathsf{origin}(R)|_{p_n}] .$$

The positions $p_1, \ldots, p_n$ are disjoint, so the remainder of $\mathsf{origin}(R)$ is divided over the write variables.

Discard function Recall that we only have to deal with creeper traces. After reading rule R we can expect to see the end symbol or we can see a relative position q. In the latter case, $\mathsf{BOS}(R,q)$ must be non-empty, and we can expect to read another rule that must be in this backward overlap set.

Given a position q such that $\mathsf{BOS}(R,q) \neq \emptyset$ we either define a discard $\psi(s,R,q)$ or a transition $\delta(s,R,q)$. A rule R *discards along position* q iff $\mathsf{origin}(R)$ has no position variable $\&p$ such that $p \geq q$. Whenever a pair $R \cdot q$ occurs in a relativized trace and R discards along q, the run is finished immediately by a discard. For example, the rules $*(0,y) \to 0$ and $+(0,s(y)) \to s(y)$ both discard along position 1. As we read an overlapping trace from left to right, the result that we aim to write is independent of what we write at position 1. The rule $+(0,s(y)) \to s(y)$ does not discard along 2 since its originated version is $s(\&2.1)$.

The discard function is defined similarly to the termination function, with the exception that we also need a position as an argument. For all rules $R \in \rho(s)$ and all q such that $\mathsf{BOS}(R,q) \neq \emptyset$ and R discards along position q, we define

$$\psi(s,R,q) = [\omega_{p_1} \mapsto \mathsf{origin}(R)|_{p_1}, \ldots, \omega_{p_n} \mapsto \mathsf{origin}(R)|_{p_n}]\,.$$

Transition function If R does not discard along position q then we write the parts of $\mathsf{origin}(R)$ similarly to how we do in the termination and discard function. The main difference is that we need to substitute some position variables of $\mathsf{origin}(R)$ by write variables to indicate where we continue to write next. Since the rule does not discard along q, there is at least one position variable $\&p$ such that $p \geq q$. We convert *all* of those to write variables ω_{p-q}, and the other position variables remain the same. The state that is reached upon following this transition is identified by rule set $\mathsf{BOS}(R,q)$ and its offsets are the positions $p-q$ that accompany the variables that we converted. Formally

$$\delta(s,R,q) = (s', [\omega_{p_1} \mapsto \mathsf{origin}(R)|^{\gamma}_{p_1}, \ldots, \omega_{p_n} \mapsto \mathsf{origin}(R)|^{\gamma}_{p_n}])\,,$$

where $\gamma(\&p) = \&p$ if $p \not\geq q$ and $\gamma(\&p) = \omega_{p-q}$ if $p \geq q$, and s' is identified by $\mathsf{BOS}(R,q)$ and $\pi(s') = \{p-q \mid \&p \in \mathsf{vars}(\mathsf{origin}(R)) \wedge p \geq q\}$.

Correctness For every creeper trace of some ground term, the constructed transducer produces the correct result. The detailed proof of this claim features many pages of mostly equational reasoning. In this extended abstract we give a sketch.

Theorem 6.1. Consider a rewrite system $\mathscr{R}$ and its constructed creeper trace transducer $(S, s_0, \varphi, \psi, \delta)$. Let t_0 be a ground term and suppose that $tr = p_n \cdot R_n \cdot \ \ldots\ \cdot p_1 \cdot R_1$ is a creeper trace of t_0, inferring the sequence $t_n, \ldots, t_1$ such that $t_n \xleftarrow{R_n @ p_n} \ldots \xleftarrow{R_1 @ p_1} t_0$. Then $\langle t_0|_{p_n}, s_0, t_0[\omega_\varepsilon]_{p_n}, \mathsf{relativize}(tr)\rangle \rightsquigarrow^* t_n$.

Proof. (Sketch). By induction on the length of the trace. Suppose that the transducer handles all creeper traces of length k correctly. Now let $p_{k+1} \cdot R_{k+1} \cdot p_k \cdot R_k \cdot p_{k-1} \cdot R_{k-1} \cdot tr$ be a creeper trace of length $k+1$. If there are transitions $s_0 \xrightarrow{R_{k+1}} \bullet \xrightarrow{p_k - p_{k+1}} s \xrightarrow{R_k} \bullet \xrightarrow{p_{k-1} - p_k} s'$, then there is a transition $s_0 \xrightarrow{R_k} \bullet \xrightarrow{p_{k-1} - p_k} s'$ which is followed on the first step of the transducer run on $p_k \cdot R_k \cdot p_{k-1} \cdot R_{k-1} \cdot tr$. Then we apply the induction hypothesis and use equational reasoning to establish what is needed.

We have to treat four base cases that are not covered by the induction step. These are the cases $k = 1$, $k = 2$, and the cases where discarding happens as the first or second step. In each case we compute the result written by the transducer and identify it with the required result, which can be written in terms of origin and deref by Proposition 5.1. □

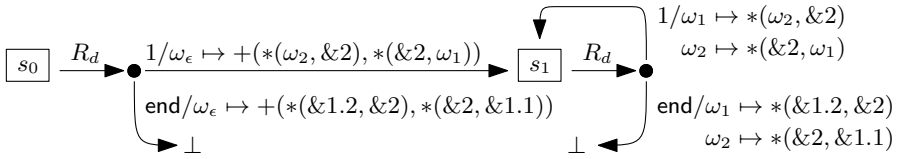

Figure 4: The creeper trace transducer for $\{R_d\}$.

7 Two more intricate examples

The reader may have already verified that the transducer in Figure 2 is the correct transducer for $\mathscr{R}_{nat}$. We discuss two more complicated examples to make some details of the construction more apparent.

Dancing distributivity Consider the following rewrite rule, which combines distributivity and commutativity in an intricate way:

$$R_d : *(+(x,y),z) \to +(*(y,z),*(z,x)) .$$

The right-hand side of this rule overlaps its own left-hand side at position 1. The transducer for $\{R_d\}$ is displayed in Figure 4. State s_0 is identified by rule set $\{R_d\}$ and offsets $\pi(s_0) = \{\varepsilon\}$. The transition to s_1 is constructed as follows. We have $\mathsf{origin}(R_d) = +(*(\&1.2, \&2), *(\&2, \&1.1))$ with positions 1.1 and 1.2 as variables being lower than 1. So we create a transition $s_0 \xrightarrow{R_d} \bullet \xrightarrow{1/\alpha} s_1$, not a discard, where α is a substitution that operates on ω_ε as $\alpha(\omega_\varepsilon) = +(*(\omega_{1.2-1}, \&2), *(\&2, \omega_{1.1-1})) = +(*(\omega_2, \&2), *(\&2, \omega_1))$. The state s_1 is identified by rule set $\rho(s_1) = \{R_d\}$ and offsets $\pi(s_1) = \mathsf{vars}(\alpha(\omega_\varepsilon)) = \{1, 2\}$. This means that every substitution we obtain from s_1 must operate on the two variables ω_1 and ω_2. Indeed, following the same construction with these offsets splits up the originated version of R_d in two parts. The transition $s_1 \xrightarrow{R_d} \bullet \xrightarrow{1/\beta} s_1$ is computed by setting $\beta(\omega_1) = \mathsf{origin}(R_d)|_1^{[\&1.2 \mapsto \omega_{1.2-1}]} = *(\omega_{1.2-1}, \&2) = *(\omega_2, \&2)$, and $\beta(\omega_2) = \mathsf{origin}(R_d)|_2^{[\&1.1 \mapsto \omega_{1.1-1}]} = *(\&2, \omega_{1.1-1}) = *(\&2, \omega_1)$. The addition symbol is missing due to the overlap with the rule that was followed to s_1. Note that $\beta(\omega_1)$ yields a writable term with variable ω_2 and $\beta(\omega_2)$ yields one with variable ω_1. The reader is invited to apply the transducer to the creeper trace $\varepsilon \cdot R_d \cdot 1 \cdot R_d$ of term $*(*(+(u_1, u_2), u_3), u_4)$.

Alternative addition with zero Consider a rule that defines addition on zero, but only when its right argument is headed by at least two successors.

$$R_{0ss} : +(0, s(s(y))) \to s(s(y)) .$$

Figure 5 displays the transducer for this rule. This example shows that identifying states with a set of rules is not sufficient. The right-hand side of R_{0ss} overlaps its own left-hand side on two positions, which results in three distinct states.

- State s_0 is identified by the rule set $\rho(s_0) = \{R_{0ss}\}$ and offsets $\pi(s_0) = \{\varepsilon\}$.
- State s_1 is identified by $\rho(s_1) = \mathsf{BOS}(R_{0ss}, 2) = \{R_{0ss}\}$ and $\pi(s_1) = \{1.1\}$.
- State s_2 is identified by $\rho(s_2) = \mathsf{BOS}(R_{0ss}, 2.1) = \{R_{0ss}\}$ and $\pi(s_2) = \{1\}$.

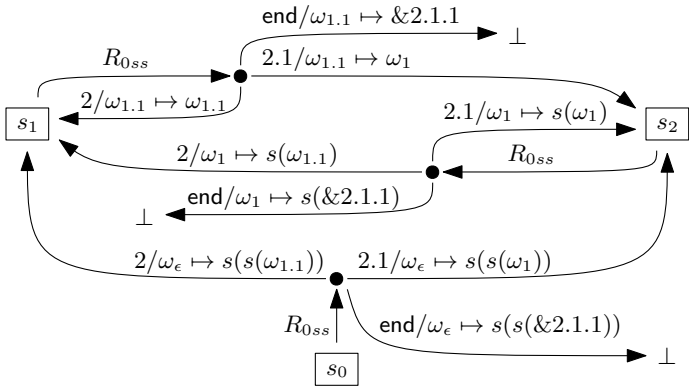

Figure 5: The creeper trace transducer for $\{R_{0ss}\}$.

8 Concluding remarks

In this paper we discussed creeper traces and a transducer that can read them so that we may exploit overlap between rules. Recall from the definition of creeper trace that we require each right-hand side to overlap the previous left-hand side. Given the rules $R_2 : +(x,s(y)) \to s(+(x,y))$, this means that $\varepsilon \cdot R_2 \cdot 2.1 \cdot R_2$ is not a creeper trace: the right-hand side $s(+(x,y))$ matches $+(x,s(y))|_{2.1}$, but it does not *overlap*, since y is a variable. It turns out that defining creeper traces and transducers with the *backward match set* $\mathsf{BMS}(\ell \to r, p) = \{\ell' \to r' \in \mathscr{R} \mid r' \text{ matches } \ell|_p\}$ is a sufficient condition for completing the correctness proof. In the transducer this results in many transitions going to the initial state.

In the future this work can be extended to support more kinds of traces. For example, an increasing, non-overlapping trace can be observed in

$$s(u_1) \xleftarrow{R_2@\varepsilon} +(s(u_1),0) \xleftarrow{R_2@1.1} +(s(+(u_1,0)),0)\,.$$

The successor symbol in the result is not part of any of the involved right-hand sides, so the transducer cannot write it. This is not a technically challenging limitation. Every increasing, non-overlapping trace consists of at least two creeper traces that can be written in separate transducer runs. Supporting these traces directly with a transducer should not be problematic.

Non-increasing traces are problematic for two reasons. First there are decreasing traces like

$$s(s(+(u_1,u_2))) \xleftarrow{R_2@1} s(+(u_1,s(u_2))) \xleftarrow{R_2@\varepsilon} +(u_1,s(s(u_2)))\,.$$

By reading from right to left it could also be possible to write the result in one sweep, but a general construction needs to be elaborated thoroughly. It may be possible to convert a set automaton [8] into a streaming tree transducer [2] so that creating the trace and writing the result is done by a single machine.

Lastly, there are disjoint traces in e.g. the TRS $\mathscr{R}_{nat}$ such as:

$$0 \xleftarrow{R_1@\varepsilon} +(0,0) \xleftarrow{R_3@1} +(*(0,0),0) \xleftarrow{R_3@2} +(*(0,0),*(0,0))\,.$$

This is not a creeper trace since positions 1 and 2 are disjoint. We can possibly generalize the current work to *creeper trees*, for example $\varepsilon \cdot R_1 \cdot [1 \cdot R_3 \mid\mid 2 \cdot R_3]$, and create a transducer that supports parallel writing.

Acknowledgement Thanks to the reviewers for their constructive feedback and additional references.

References

[1] Rajeev Alur & Pavol Cerný (2010): *Expressiveness of streaming string transducers*. In Kamal Lodaya & Meena Mahajan, editors: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, *LIPIcs* 8, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 1–12, doi:10.4230/LIPIcs.FSTTCS.2010.1. Available at `https://doi.org/10.4230/LIPIcs.FSTTCS.2010.1`.

[2] Rajeev Alur & Loris D'Antoni (2017): *Streaming Tree Transducers*. *J. ACM* 64(5), pp. 31:1–31:55, doi:10.1145/3092842. Available at `https://doi.org/10.1145/3092842`.

[3] Franz Baader & Tobias Nipkow (1998): *Term rewriting and all that*. Cambridge University Press.

[4] Mark Bouwman & Rick Erkens (2023): *Term Rewriting Based On Set Automaton Matching*. arXiv:2202.08687.

[5] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison & M. Tommasi (2007): *Tree Automata Techniques and Applications*. Available on: `https://jacquema.gitlabpages.inria.fr/files/tata.pdf`. Release October, 12th 2007.

[6] Olivier Danvy & Jacob Johannsen (2013): *From Outermost Reduction Semantics to Abstract Machine*. In Gopal Gupta & Ricardo Peña, editors: *Logic-Based Program Synthesis and Transformation, 23rd International Symposium, LOPSTR 2013, Madrid, Spain, September 18-19, 2013, Revised Selected Papers*, *Lecture Notes in Computer Science* 8901, Springer, pp. 91–108, doi:10.1007/978-3-319-14125-1_6. Available at `https://doi.org/10.1007/978-3-319-14125-1_6`.

[7] Arie van Deursen, Paul Klint & Frank Tip (1993): *Origin Tracking*. *J. Symb. Comput.* 15(5/6), pp. 523–545, doi:10.1016/S0747-7171(06)80004-0. Available at `https://doi.org/10.1016/S0747-7171(06)80004-0`.

[8] Rick Erkens & Jan Friso Groote (2021): *A Set Automaton to Locate All Pattern Matches in a Term*. In Antonio Cerone & Peter Csaba Ölveczky, editors: *Theoretical Aspects of Computing - ICTAC 2021 - 18th International Colloquium, Virtual Event, Nur-Sultan, Kazakhstan, September 8-10, 2021, Proceedings*, *Lecture Notes in Computer Science* 12819, Springer, pp. 67–85, doi:10.1007/978-3-030-85315-0_5. Available at `https://doi.org/10.1007/978-3-030-85315-0_5`.

[9] Alex Ferguson & Philip Wadler (1988): *When will deforestation stop*. In: *Proc. of 1988 Glasgow Workshop on Functional Programming*, Citeseer, pp. 39–56.

[10] Andrew John Gill, John Launchbury & Simon Peyton Jones (1993): *A Short Cut to Deforestation*. In John Williams, editor: *Proceedings of the conference on Functional programming languages and computer architecture, FPCA 1993, Copenhagen, Denmark, June 9-11, 1993*, ACM, pp. 223–232, doi:10.1145/165180.165214. Available at `https://doi.org/10.1145/165180.165214`.

[11] John V. Guttag, Deepak Kapur & David R. Musser (1983): *On Proving Uniform Termination and Restricted Termination of Rewriting Systems*. *SIAM J. Comput.* 12(1), pp. 189–214, doi:10.1137/0212012. Available at `https://doi.org/10.1137/0212012`.

[12] Simon Peyton Jones (1992): *Implementing Lazy Functional Languages on Stock Hardware: The Spineless Tagless G-Machine*. *J. Funct. Program.* 2(2), pp. 127–202, doi:10.1017/S0956796800000319. Available at `https://doi.org/10.1017/S0956796800000319`.

[13] Simon Peyton Jones (2003): *Haskell 98 language and libraries: the revised report*. Cambridge University Press.

[14] Simon Peyton Jones, Andrew Tolmach & Tony Hoare (2001): *Playing by the rules: rewriting as a practical optimisation technique in GHC*. In: *Haskell workshop*, 1, pp. 203–233.

[15] Simon Marlow & Philip Wadler (1992): *Deforestation for Higher-Order Functions*. In John Launchbury & Patrick M. Sansom, editors: *Functional Programming, Glasgow 1992, Proceedings of the 1992 Glasgow Workshop on Functional Programming, Ayr, Scotland, UK, 6-8 July 1992*, Workshops in Computing, Springer, pp. 154–165, doi:10.1007/978-1-4471-3215-8_14. Available at `https://doi.org/10.1007/978-1-4471-3215-8_14`.

[16] Philip Wadler (1990): *Deforestation: Transforming Programs to Eliminate Trees*. *Theor. Comput. Sci.* 73(2), pp. 231–248, doi:10.1016/0304-3975(90)90147-A. Available at `https://doi.org/10.1016/0304-3975(90)90147-A`.