Sparse Tiling Through Overlap Closures for Termination of String Rewriting

Alfons Geser
HTWK Leipzig, Germany

Dieter Hofbauer
ASW – Berufsakademie Saarland, Germany

Johannes Waldmann
HTWK Leipzig, Germany

Abstract

A strictly locally testable language is characterized by its set of admissible factors, prefixes and suffixes, called tiles. We over-approximate reachability sets in string rewriting by languages defined by sparse sets of tiles, containing only those that are reachable in derivations. Using the partial algebra defined by a tiling for semantic labeling, we obtain a transformational method for proving local termination. These algebras can be represented efficiently as finite automata of a certain shape. Using a known result on forward closures, and a new characterisation of overlap closures, we can automatically prove termination and relative termination, respectively. We report on experiments showing the strength of the method.

2012 ACM Subject Classification Theory of computation → Rewrite systems

Keywords and phrases relative termination, semantic labeling, locally testable language, overlap closure

Digital Object Identifier 10.4230/LIPIcs.FSCD.2019.21

1 Introduction

Methods for proving termination of rewriting (automatically) can be classified [25] into syntactical (using a precedence on letters), semantical (map each letter to a function on some domain), or transformational. Applying a transformation, one hopes to obtain an equivalent termination problem that is easier to handle.

One such transformation is *semantic labeling* [24]. This will typically increase the number of rules, sometimes drastically so. We consider here a specific semantic domain, called the k*-shift algebra*, consisting of words of length $k-1$, with the "shift left" operation.

When we use this algebra (in Section 3) for semantically labeling a string w, each labeled letter is a k-factor of w, called a *tile*.

Our implementation uses values of k from 2 to 8. *Self labeling* [17] can be seen as unrestricted shifting. The 2-shift algebra is used in *root labeling* [19] which first appeared in Termination Competitions in 2006. MultumNonMulta [12] took first place in both categories Standard and Relative SRS of the 2018 competition [13] mainly due to the use of 2-tiling.

A *sparse* tiling contains just those tiles that can occur during derivations starting in a given language. The shift algebra given by those tiles then is a *partial model* [3]. In Section 4, we present an algorithm to complete a given set of tiles with respect to a given rewriting system (i. e., to construct a minimal partial model) and provide an efficient implementation that can handle large sets of tiles.

We apply this method to derivations starting from right-hand sides of forwards closures (Section 5) and overlap closures (Section 7) since local termination on these languages implies

4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019).
Editor: Herman Geuvers; Article No. 21; pp. 21:1–21:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

global termination (a known result), and relative termination (Section 6), respectively. In all, we obtain a transformational method for proving termination and relative termination: construct a closed set of tiles, and then use it for semantic labeling. This can be combined with other methods for proving termination, e. g., weights (linear interpretations of slope 1).

We obtain yet another automated termination proof for Zantema's Problem $\{a^2b^2 \to b^3a^3\}$, which is a classical benchmark, see Example 5.6. Our implementation is part of the Matchbox termination prover, and it easily solves several termination problems from the Termination Problems Database[1] that appear hard for other approaches, e. g., Examples 8.3 and 8.4. Sparse tiling contributed to Matchbox winning the categories *SRS Standard* and *SRS Relative* of the Termination Competition 2019, see Section 9.

Our application area is string rewriting, and our implementation is tailored to that. Still, for proving correctness, we use the language of term rewriting, as this allows to re-use concepts and results.

2 Notation

Given a set of *letters* Σ, i. e., an *alphabet*, a string is a finite sequence of letters over Σ. The number of its components is the *length* of the string, and the string of length zero, the *empty string*, is denoted by ϵ. If there is no ambiguity, we denote the string with letters $a_1, \ldots, a_n$ by $a_1 \ldots a_n$. We deal, however, also with strings of strings, and then use the list notation $[a_1, \ldots, a_n]$. Let $\mathsf{alphabet}(w)$ denote the set of letters that occur in the string w. By $\mathsf{Prefix}(S)$ and $\mathsf{Suffix}(S)$ we denote the set of prefixes and suffixes resp. of strings from the set S, and $\mathsf{Prefix}_k(S)$ and $\mathsf{Suffix}_k(S)$ denotes their restriction to strings of length k.

2.1 Rewriting and Reachability

A *string rewriting system* over alphabet Σ is a set of rewrite rules. We use standard concepts and notation (see, e. g., Book and Otto [1]) with this extension: A *constrained rule* is a pair of strings l, r, together with a *constraint* $c \in \{\mathsf{factor}, \mathsf{prefix}, \mathsf{suffix}\}$ that indicates where the rule may be applied. The corresponding rewrite relations are

$$\begin{aligned}
\to_{l,r,\mathsf{factor}} &= \{(xly, xry) \mid x, y \in \Sigma^*\},\\
\to_{l,r,\mathsf{prefix}} &= \{(ly, ry) \mid y \in \Sigma^*\},\\
\to_{l,r,\mathsf{suffix}} &= \{(xl, xr) \mid x \in \Sigma^*\}.
\end{aligned}$$

A constrained rule (l, r, c) is denoted by $l \to_c r$. Standard rewriting corresponds to the factor constraint, therefore $\to$ abbreviates $\to_{\mathsf{factor}}$. For a rewrite system R, we define $\to_R$ as the union of the rewrite relations of its rules. For a relation ρ on Σ^* and a set $L \subseteq \Sigma^*$, let $\rho(L) = \{y \mid \exists x \in L, (x, y) \in \rho\}$. Hence the set of *$R$-reachable* strings from L is $\to_R^*(L)$, or $R^*(L)$ for short. A language $L \subseteq \Sigma^*$ is *closed w.r.t. R* if $\to_R(L) \subseteq L$.

▶ **Example 2.1.** For $R = \{cc \to_{\mathsf{factor}} bc, ba \to_{\mathsf{factor}} ac, c \to_{\mathsf{suffix}} bc, b \to_{\mathsf{suffix}} ac\}$, we have $bbb \to_{\mathsf{suffix}} bbac \to_{\mathsf{factor}} bacc$. The reachability set $R^*(\{bc, ac\})$ is $(a + b)b^*c$. This set is closed with respect to R.

A rewriting system R over Σ is called *terminating on* $L \subseteq \Sigma^*$, if for each $w \in L$, each R-derivation starting at w is finite, and R is called *terminating*, written $\mathsf{SN}(R)$, if it is

[1] The Termination Problems Database, Version 10.6, see `http://termination-portal.org/wiki/TPDB`.

terminating on Σ^*. A rewriting system R is called *terminating relative to* a rewriting system S *on* L, if each $(R \cup S)$-derivation starting in L has a only a finite number of R rule applications. If L is not given, we mean Σ^* and write $\mathsf{SN}(R/S)$.

2.2 Forward Closures

Given a rewrite system R over alphabet Σ, a *closure* $C = (l, r)$ of R is a pair of strings with $l \to_R^+ r$ such that each position of r is involved in some step of the derivation. In particular, we use *forward closures* [15].

The set $\mathsf{FC}(R)$ of forward closures of R is defined as the least set of pairs (l, r) of strings that contains R and satisfies

- if $(s, xuy) \in \mathsf{FC}(R)$ and $(u, v) \in \mathsf{FC}(R)$ then $(s, xvy) \in \mathsf{FC}(R)$,
- if $(s, xu) \in \mathsf{FC}(R)$ and $(uy, v) \in \mathsf{FC}(R)$ for $u \neq \epsilon \neq y$ then $(sy, xv) \in \mathsf{FC}(R)$.

The set $\mathsf{FC}(R)$ can also be characterized without recursion in the second partner, as observed by Herrmann [11] in the term rewriting case. This can be used to recursively characterize the set $\mathsf{RFC}(R) = \mathsf{rhs}(\mathsf{FC}(R))$ of right hand sides of forward closures directly [6].

$\mathsf{RFC}(R)$ can also be characterized by factor and suffix rewriting.

▶ **Proposition 2.2.** *$\mathsf{RFC}(R) = (R \cup \mathsf{forw}(R))^*(\mathsf{rhs}(R))$, where*

$$\mathsf{forw}(R) = \{l_1 \to_{\mathsf{suffix}} r \mid (l_1 l_2 \to r) \in R, l_1 \neq \epsilon \neq l_2\}.$$

They are related to termination by

▶ **Theorem 2.3** ([2]). *R is terminating on Σ^* if and only if R is terminating on $\mathsf{RFC}(R)$.*

For a self-contained proof see Section 6 in [26].

▶ **Example 2.4.** For $R = \{cc \to bc, ba \to ac\}$ we have $\mathsf{forw}(R) = \{c \to_{\mathsf{suffix}} bc, b \to_{\mathsf{suffix}} ac\}$ and $\mathsf{RFC}(R) = (a + b)b^*c$, cf. Example 2.1. As $\mathsf{RFC}(R)$ contains no R-redex, R is trivially terminating on $\mathsf{RFC}(R)$, therefore R is terminating by Theorem 2.3.

Later in the paper, we use tiled rewriting to approximate $\mathsf{RFC}(R)$, and we obtain the termination proof of Example 2.4 automatically, see Examples 3.10 and 4.3.

2.3 Partial Algebras and Partial Models

We will recall concepts and notation from [3]. A *partial* Σ*-algebra* $\mathcal{A} = (A, [\![\cdot]\!])$ consists of a non-empty set A and for each n-ary $f \in \Sigma$ a partial function $[\![f]\!] : A^n \rightharpoonup A$. Given $\mathcal{A}$ and a partial assignment of variables $\alpha : V \rightharpoonup A$, the *interpretation* $[\![t, \alpha]\!]$ of $t \in \mathsf{Term}(\Sigma, V)$ is defined as usual, noting that it will not always be defined. If t is ground, we simply write $[\![t]\!]$. A partial algebra is a *partial model* of a rewrite system R if for each rewrite rule $(l \to r) \in R$, and each assignment $\alpha : \mathsf{Var}(l) \to A$, definedness of $[\![l, \alpha]\!]$ implies $[\![l, \alpha]\!] = [\![r, \alpha]\!]$. For a partial Σ-algebra $\mathcal{A} = (A, [\![\cdot]\!])$, a term $t \in \mathsf{Term}(\Sigma, X)$, and a partial assignment $\alpha : \mathsf{Var}(t) \rightharpoonup A$, let $[\![t, \alpha]\!]^*$ denote the set of defined values of subterms of t under α, i. e., $\{[\![s, \alpha]\!] \mid s \trianglelefteq t \wedge [\![s, \alpha]\!] \text{ is defined}\}$. For $T \subseteq A$, let $\mathsf{Lang}_{\mathcal{A}}(T)$ denote the set ot ground terms that can be evaluated inside T, i. e., $\{t \in \mathsf{Term}(\Sigma) \mid [\![t]\!]^* \subseteq T\}$, and let $\mathsf{Lang}_{\mathcal{A}} = \mathsf{Lang}_{\mathcal{A}}(A)$. Note that a partial algebra *is* a deterministic (tree) automaton with set of states A, and partiality means that the automaton may be incomplete.

2.4 Languages defined by Tilings

A strictly locally testable language is specified by considering prefixes, factors, and suffixes of bounded length, called tiles. We give an equivalent definition that allows a uniform description, using end markers $\triangleleft, \triangleright \notin \Sigma$. A similar formalization is employed for two-dimensional tiling in [8].

▶ **Definition 2.5.** *For an alphabet Γ, $k \geq 1$ and $a_i \in \Gamma$, the k-*tiled version *of a string $a_1 \ldots a_n$ is the string over Γ^k of all k-tiles, i. e., factors of length k:*

$$\mathit{tiled}_k(a_1 \ldots a_n) = [a_1 \ldots a_k, a_2 \ldots a_{k+1}, \ldots, a_{n-k+1} \ldots a_n]$$

This string is empty in case $n < k$. Let $\mathit{tiles}_k(w)$ denote $\mathit{alphabet}(\mathit{tiled}_k(w))$.

▶ **Definition 2.6.** *For $k \geq 0$ and $w \in \Sigma^*$, the k-*bordered version *of w is $\mathit{bord}_k(w) = \triangleleft^k w \triangleright^k$ over $\Sigma \cup \{\triangleleft, \triangleright\}$. By $\mathit{btiled}_k(w)$ we abbreviate $\mathit{tiled}_k(\mathit{bord}_{k-1}(w))$, and $\mathit{btiles}_k(w)$ stands for $\mathit{alphabet}(\mathit{btiled}_k(w))$.*

▶ **Example 2.7.** $\mathsf{btiled}_2(abbb) = \mathsf{tiled}_2(\mathsf{bord}_1(abbb)) = \mathsf{tiled}_2(\triangleleft abbb\triangleright) = [\triangleleft a, ab, bb, bb, b\triangleright]$, thus $\mathsf{btiles}_2(abbb) = \{\triangleleft a, ab, bb, b\triangleright\}$. Further, $\mathsf{btiles}_2(\epsilon) = \{\triangleleft\triangleright\}$, $\mathsf{btiles}_2(a) = \{\triangleleft a, a\triangleright\}$, and $\mathsf{btiled}_3(a) = [\triangleleft\triangleleft a, \triangleleft a\triangleright, a\triangleright\triangleright]$.

▶ **Definition 2.8.** *For $k \geq 1$, the language defined by a set of tiles $T \subseteq \mathit{btiles}_k(\Sigma^*)$ is*

$$\mathit{Lang}(T) = \{w \in \Sigma^* \mid \mathit{btiles}_k(w) \subseteq T\}.$$

This is a characterization of the class of strictly locally k-testable languages [16, 23], a subclass of regular languages.

▶ **Example 2.9.** For $k = 2$ and $T = \{\triangleleft a, ab, ba, a\triangleright\}$ we obtain $\mathsf{Lang}(T) = a(ba)^*$.

3 Tiled Rewrite Systems and Shift Algebras

We apply the method of semantic labelling w.r.t. a partial model to transform a local termination problem to a global one. Our contribution is to use the *k-shift algebra*. We obtain Algorithm 4.1 that over-approximates reachability sets w.r.t. rewriting, and is guaranteed to halt.

One application is to approximate right-hand sides of forward closures, to prove global termination (Algorithm 5.1). Later, we approximate right-hand sides of overlap closures to prove relative termination (Algorithm 8.1).

Our intended application area is string rewriting. For proving correctness we want to use concepts and results from local termination [3], so we need a translation to term rewriting. We view strings as terms with unary symbols, and a nullary symbol (representing ϵ), where the rightmost (!) position in the string is the topmost position in the term. As in [3], we choose this order (left to right in the string means bottom to top in the term) since we later use deterministic automata, working from left to right on the string, realising evaluation in the algebra, which goes bottom to top. This choice also has the notational consequence that a string rewriting rule, e. g., $ab \to baa$, is translated to a term rewriting rule $((z)a)b \to (((z)b)a)a$, where z is a variable, which we abbreviate to $(z)ab \to (z)baa$. This is just postfix notation for function application, recommended also by Sakarovitch [18], p. 12.

▶ **Definition 3.1** (The *k-shift algebra*). *For $T \subseteq \mathit{btiles}_k(\Sigma^*)$, the partial algebra $\mathit{Shift}_k(T)$ over signature $\Sigma \cup \{\epsilon, \triangleright\}$ has domain $\mathit{tiles}_{k-1}(\triangleleft^*\Sigma^*\triangleright^*)$, the interpretation of ϵ is $\triangleleft^{k-1}$, and each letter (unary symbol) $c \in \Sigma \cup \{\triangleright\}$ is interpreted by the unary function that maps p to $\mathit{Suffix}_{k-1}(pc)$ if $pc \in T$, and is undefined otherwise.*

We have the following obvious connection (modulo the translation between words and terms) between the language of the algebra (i. e., all terms that have a defined value) and the language of the set of tiles (i. e., all words that can be covered):

▶ **Proposition 3.2.** *For any set of k-tiles T, $\mathsf{Lang}_{\mathsf{Shift}_k(T)} = \mathsf{Prefix}(\mathsf{Lang}(T) \cdot \triangleright^{k-1})$.*

We need the prefix closure since a language of a partial algebra always is subterm-closed, according to the definition from [3], a feature that had already been criticised in [4].

To apply semantic labelling, we need a partial algebra that is a partial model. A k-shift algebra is a model for a rewrite system R only if R does not change the $k-1$ topmost symbols. This property can be guaranteed by the following closure operation that also translates our notion of constrained string rewriting to the standard notion of term rewriting:

▶ **Definition 3.3.** *For a constrained string rewriting system R over Σ define its* context closure*, the term rewriting system $\mathsf{CC}_k(R)$ over $\Sigma \cup \{\epsilon, \triangleright\}$, where ϵ is a constant, all other symbols are unary, and z is a variable symbol, as follows. Note that the second subset consists of ground rules.*

$$\begin{aligned}\mathsf{CC}_k(R) = {} & \{(z)ly \to (z)ry \mid (l \to_{\mathsf{factor}} r) \in R, y \in \mathsf{tiles}_{k-1}(\Sigma^* \triangleright^*)\} \cup \\ & \{(\epsilon)ly \to (\epsilon)ry \mid (l \to_{\mathsf{prefix}} r) \in R, y \in \mathsf{tiles}_{k-1}(\Sigma^* \triangleright^*)\} \cup \\ & \{(z)ly \to (z)ry \mid (l \to_{\mathsf{suffix}} r) \in R, y = \triangleright^{k-1}\}\end{aligned}$$

Constrained rewrite steps of R on Σ^* are directly related to term rewrite steps of the context closure of R on (the set of terms corresponding to) $\Sigma^* \triangleright^{k-1}$:

▶ **Proposition 3.4.** *$s \to_R t$ iff $(\epsilon)s\triangleright^{k-1} \to_{\mathsf{CC}_k(R)} (\epsilon)t\triangleright^{k-1}$.*

Since $\mathsf{CC}_k(R)$ does keep the $k-1$ topmost (rightmost) symbols intact, the shift algebra of T is a partial model provided it contains a sufficiently large set of tiles:

▶ **Proposition 3.5.** *For a set of k-tiles T and a rewriting system R, if $\mathsf{Lang}_{\mathsf{Shift}_k(T)}$ is closed with respect to R, then $\mathsf{Shift}_k(T)$ is a partial model for $\mathsf{CC}_k(R)$.*

In Section 4 we provide an algorithm for constructing such a closed set T.

Given a partial model, we use it for semantic labeling. The labeling of $\mathsf{CC}_k(R)$ with respect to $\mathsf{Shift}_k(T)$ (see [3], Def. 6.3) produces a term rewriting system that can be re-transformed to a string rewriting system by replacing each function symbol c, that is labelled with an element p from the algebra, to the string (the tile) pc. The following definition avoids the round-trip, and shows how to label the string rewriting system directly.

▶ **Definition 3.6.** *For a rule $l \to_c r$ over signature Σ with $c \in \{\mathsf{factor}, \mathsf{prefix}, \mathsf{suffix}\}$, we define a set of rules over signature $\mathsf{btiles}_k(\Sigma^*)$ by*

$$\begin{aligned}\mathsf{btiled}_k(l \to_{\mathsf{factor}} r) &= \{\mathsf{tiled}_k(xly) \to_{\mathsf{factor}} \mathsf{tiled}_k(xry) \mid x \in T^{\triangleleft}, y \in T^{\triangleright}\} \\ \mathsf{btiled}_k(l \to_{\mathsf{prefix}} r) &= \{\mathsf{tiled}_k(xly) \to_{\mathsf{prefix}} \mathsf{tiled}_k(xry) \mid x = \triangleleft^{k-1}, y \in T^{\triangleright}\} \\ \mathsf{btiled}_k(l \to_{\mathsf{suffix}} r) &= \{\mathsf{tiled}_k(xly) \to_{\mathsf{suffix}} \mathsf{tiled}_k(xry) \mid x \in T^{\triangleleft}, y = \triangleright^{k-1}\}\end{aligned}$$

where $T^{\triangleleft} = \mathsf{tiles}_{k-1}(\triangleleft^* \Sigma^*)$, $T^{\triangleright} = \mathsf{tiles}_{k-1}(\Sigma^* \triangleright^*)$, and for a set of tiles $T \subseteq \mathsf{btiles}_k(\Sigma^*)$ let

$$\mathsf{btiled}_T(l \to_c r) = \mathsf{btiled}_k(l \to_c r) \cap T^* \times T^* \times \{c\},$$

the set of tiled rules that use tiles from T only. Both btiled_k and btiled_T are extended to sets of rules. Note that $\{\triangleleft^{k-1}\} = \mathsf{tiles}_{k-1}(\triangleleft^*)$ and $\{\triangleright^{k-1}\} = \mathsf{tiles}_{k-1}(\triangleright^*)$.

▶ **Example 3.7.** The set $\mathsf{btiled}_2(ba \to_{\mathsf{factor}} ac)$ contains 16 rules, among them $[\lhd b, ba, a\rhd] \to [\lhd a, ac, c\rhd]$, $[\lhd b, ba, aa] \to [\lhd a, ac, ca], \ldots, [ab, ba, a\rhd] \to [aa, ac, c\rhd], \ldots, [cb, ba, ac] \to [ca, ac, cc]$, and $\mathsf{btiled}_2(b \to_{\mathsf{suffix}} ac) = \{[\lhd b, b\rhd] \to [\lhd a, ac, c\rhd], [ab, b\rhd] \to [aa, ac, c\rhd], [bb, b\rhd] \to [ba, ac, c\rhd], [cb, b\rhd] \to [ca, ac, c\rhd]\}$. For $S = \{ac, ba, bb, cc\}$ we get $\mathsf{btiled}_S(ba \to_{\mathsf{factor}} ac) = \{[bb, ba, ac] \to [ba, ac, cc]\}$ and for any strict subset T of S, $\mathsf{btiled}_T(ba \to_{\mathsf{factor}} ac) = \emptyset$.

This translation is faithful, in the following sense:

▶ **Proposition 3.8.** *$\mathsf{btiled}_T(R)$ is exactly the (string rewriting translation of the) labeling of $\mathsf{CC}_k(R)$ with respect to $\mathsf{Shift}_k(T)$.*

To actually enumerate $\mathsf{btiled}_T(R)$ in an implementation, we will fuse both parts of Definition 3.6 by restricting contexts x and y to be elements of T^* right from the beginning.

▶ **Theorem 3.9.** *For $k \geq 1$ and $T \subseteq \mathsf{btiles}_k(\Sigma^*)$, if $\mathsf{Lang}(T)$ is closed with respect to R, then R is terminating on $\mathsf{Lang}(T)$ if and only if $\mathsf{btiled}_T(R)$ is terminating.*

Proof. By Proposition 3.8 and Theorem 6.4 from [3], applicable due to Proposition 3.5. ◀

▶ **Example 3.10** (Example 2.4 continued)**.** Let $R = \{cc \to bc, ba \to ac\}$. Then $\mathsf{RFC}(R) = \mathsf{Lang}(T)$ for the set of tiles $T = \{\lhd a, \lhd b, ab, ac, bb, bc, c\rhd\}$. The set $\mathsf{RFC}(R)$ is closed w.r.t. R by definition and $\mathsf{tiled}_T(R)$ is empty, therefore terminating. By Theorem 3.9, R is terminating on $\mathsf{RFC}(R)$, thus by Theorem 2.3, R is terminating. See Example 4.3 for a computation that produces T from R.

4 Completion in Shift Algebras

To apply Theorem 3.9, we need an R-closed set T of tiles. The following algorithm computes such a set by starting from an initial set S, and successively adding tiles that become reachable via R-steps. This is similar to other algorithms that produce rewrite-closed automata [5]. Due to the algebra we use, we have the stronger property of guaranteed termination.

▶ **Algorithm 4.1.**

- *Specification:*
  - *Input: A term rewriting system R over Σ, a finite partial Σ-algebra $\mathcal{A} = (A, [\![\cdot]\!])$, a set $S \subseteq A$.*
  - *Output: A minimal set $T \subseteq A$ such that $S \subseteq T$ and $\mathsf{Lang}_{\mathcal{A}}(T)$ is closed w.r.t. R.*
- *Implementation: Let $T = \bigcup_i T_i$ for the sequence $S = T_0 \subseteq T_1 \subseteq \cdots$ where*

$$T_{i+1} = T_i \cup \bigcup \{[\![r, \alpha]\!]^* \mid (l \to r) \in R, \alpha : \mathsf{Var}(l) \to T_i, [\![l, \alpha]\!]^* \subseteq T_i\}$$

where it is sufficient to compute a finite prefix.

Proof. This algorithm terminates, since the sequence T_i is increasing, and bounded from above by A, so it is eventually constant. The result is R-closed by construction. ◀

This algorithm will be applied to $\mathsf{CC}_k(R)$, and we construct the context closure on the fly: in each step, we use only those contexts that are accessible in $\mathsf{Shift}_k(T_i)$.

Let us first specify the representation of $\mathsf{Shift}_k(T)$. This algebra is a deterministic automaton, possibly incomplete.

▶ **Definition 4.2.** *For $k \geq 1$, a finite automaton over alphabet $\Sigma \cup \{\triangleleft, \triangleright\}$ is a k*-shift automaton *if its states are in* $\mathsf{tiles}_{k-1}(\triangleleft^*\Sigma^*\triangleright^*)$*, its initial state is $\triangleleft^{k-1}$, its final state is $\triangleright^{k-1}$, and for each transition $p \xrightarrow{c} q$, state q is the suffix of length $k-1$ of pc. Such an automaton A represents the set of tiles (of length k)* $\mathsf{tiles}(A) = \{pc \mid p \xrightarrow{c}_A q\}$.

Condition $[\![l, \alpha]\!]^* \subseteq T_i$ of Algorithm 4.1 is equivalent to the existence of a path in the automaton T_i that starts at state $p = \alpha(z)$ and is labelled l. We call this a *redex path* $p \xrightarrow{l} q$. Adding tiles then corresponds to adding edges and states. Whenever we add edges for some reduct path $p \xrightarrow{r} q'$, corresponding to $[\![r, \alpha]\!]^* \subseteq T_i$, the target state of each transition is determined by the shift property of the automaton. This is in contrast to other completion methods where there is a choice of adding fresh states, or re-using existing states.

The set of states could be defined to be $\mathsf{btiles}_{k-1}(\Sigma^*)$ in advance, but for efficiency, we only store accessible states, and add states as soon as they become accessible.

With the automata representation, we implement $\mathsf{btiled}_T(R)$ as follows: To determine xly in Definition 3.6, we compute all pairs p, q of states with $p \xrightarrow{l} q$. This can be done by starting at each p, but our implementation uses the product-of-relations method of [22]. Note that p, the state where the redex path starts, *is* actually x, the left context.

From state q, we follow all paths of length $k-1$ to determine the set of y (right contexts). For each such pair (x, y), we add the path starting at x labeled ry. Note that this path (for the context-closed reduct) meets the path for ly (the context-closed redex) in the end, since the automaton is a shift automaton. The tree search for possible y can be cut short if we detect that these paths meet earlier.

The following example demonstrates completion only. For examples that use the completed automaton for semantic labeling, see Section 5.

▶ **Example 4.3** (Example 3.10 continued)**.** In order to illustrate the use of shift automata for implementing Algorithm 4.1, consider again $R = \{cc \to bc, ba \to ac\}$ with $\mathsf{forw}(R) = \{c \to_{\mathsf{suffix}} bc, b \to_{\mathsf{suffix}} ac\}$. We choose $k = 2$ and represent $\mathsf{btiled}_2(\mathsf{rhs}(R)) = \{[\triangleleft b, bc, c\triangleright], [\triangleleft a, ac, c\triangleright]\}$ by the left automaton in Figure 1. Here, completion refers to the set of rules $C = \mathsf{CC}_2(R \cup \mathsf{forw}(R)) = \{ccy \to bcy, bay \to acy, c\triangleright \to bc\triangleright, b\triangleright \to ac\triangleright \mid y \in \{a, b, c, \triangleright\}\}$. In the initial automaton we look for paths of the form $p \xrightarrow{l} q$ for some rule $l \to r \in C$. Two such paths exist, $a \xrightarrow{c\triangleright} \triangleright$ and $b \xrightarrow{c\triangleright} \triangleright$. Completion therefore adds the paths $a \xrightarrow{bc\triangleright} \triangleright$ and $b \xrightarrow{bc\triangleright} \triangleright$ for the corresponding right-hand sides, resulting in the new edges $a \xrightarrow{b} b$ and $b \xrightarrow{b} b$ (and no new nodes), depicted by the right automaton A. No further completion steps are possible, thus $\mathsf{RFC}(R) \subseteq \mathsf{Lang}(\mathsf{tiles}(A))$ with $\mathsf{tiles}(A) = \{\triangleleft a, \triangleleft b, ab, ac, bb, bc, c\triangleright\}$. Note that for this simple example, $\subseteq$ could be replaced by equality, but in general the algorithm yields an over-approximation.

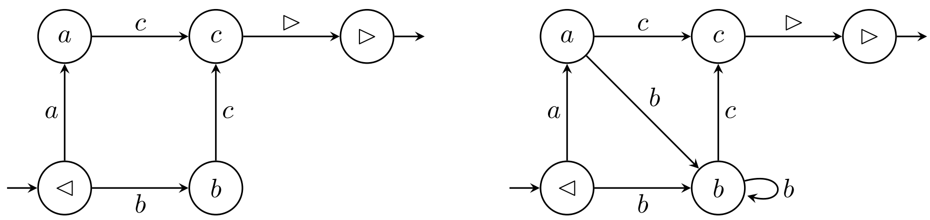


Figure 1 Constructing the shift automaton for $\mathsf{RFC}(\{cc \to bc, ba \to ac\})$ for $k = 2$.

5 Examples of Termination Proofs via Forward Closures

We transform a termination problem as follows:

▶ **Algorithm 5.1.**

- *Specification:*
  - *Input: A rewriting system R over Σ, a number k*
  - *Output: A rewriting system R' over $\mathsf{btiled}_k(\Sigma)$ such that $\mathsf{SN}(R) \iff \mathsf{SN}(R')$*
- *Implementation (and correctness): By Theorem 2.3, $\mathsf{SN}(R)$ iff $\mathsf{SN}(R)$ on $\mathsf{RFC}(R)$. By Proposition 2.2, $\mathsf{RFC}(R) = (R \cup \mathsf{forw}(R))^*(\mathsf{rhs}(R))$. By Algorithm 4.1, we construct T such that $\mathsf{Lang}(T)$ contains $\mathsf{rhs}(R)$ and is closed w.r.t. $R \cup \mathsf{forw}(R)$, that is, $\mathsf{RFC}(R) \subseteq \mathsf{Lang}(T)$. We then use the algebra $\mathsf{Shift}(T)$ as a partial model for $\mathsf{CC}_k(R)$. By Theorem 3.9 and the previous, $\mathsf{SN}(R)$ iff $\mathsf{SN}(\mathsf{btiled}_T(R))$.*

This approach had already been described in [3], Section 8, but there it was left open how to find a suitable partial algebra. An implementation used a finite-domain constraint solver, but then only small domains could be handled. In the present paper, we instead construct a suitable k-shift algebra by completion. Even if it is large, it might help solve the termination problem, cf. Example 5.6 below. We give a few smaller examples first.

▶ **Example 5.2.** We apply Algorithm 5.1 with $k = 3$ to $R = \{ab^3 \to bbaab\}$. We obtain 11 reachable tiles and 12 labeled rules. All of them can be removed by weights. We start with the automaton for $\mathsf{btiled}_3(bbaab)$ (solid edges in Figure 2).

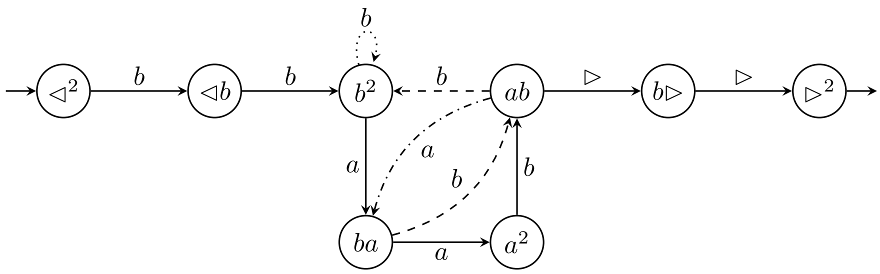


Figure 2 The 3-shift automaton for $\mathsf{RFC}(ab^3 \to bbaab)$.

It contains no R-redex. There is a $\mathsf{forw}(R)$-redex for $ab \to_{\mathsf{suffix}} bbaab$ starting at ba. We add a reduct path, starting with two fresh (dashed) edges. This creates a $\mathsf{forw}(R)$-redex for $ab \to_{\mathsf{suffix}} bbaab$ from b^2. To cover this, we add the loop at b^2 (dotted). Now we have a R-redex $ba \to a^2 \to ab \to b^2 \to b^2$. The corresponding reduct path is $ba \to ab \to b^2 \to ba \to a^2 \to ab$. The redex needs to be right-context-closed with a, and with b, as these are the possible continuations from b^2. So we context-close the reduct path as well, adding one more edge $ab \xrightarrow{a} ba$ (dash-dotted), as $ab \xrightarrow{b} b^2$ is already present. This introduces an R-redex from ab to b^2, with right extensions a and b. The extended reduct paths are already present. The automaton is now closed with respect to $R \cup \mathsf{forw}(R)$. It represents the set of tiles

$$T = \{\lhd\lhd b, \lhd bb, bba, bbb, baa, bab, aab, aba, abb, ab\rhd, b\rhd\rhd\}.$$

Absent from T are

- $\lhd\lhd\rhd, \lhd\rhd\rhd, \lhd\Sigma\rhd$ (meaning that $\mathsf{RFC}(R)$ does not contain strings of length 0 or 1),
- as well as $\lhd a\Sigma, \lhd ba, \Sigma a\rhd$ (meaning that $\mathsf{RFC}(R)$ starts with b^2 and ends with b),
- and a^3 (meaning that $\mathsf{RFC}(R)$ does not have a^3 as a factor).

Finally, we compute $\mathsf{btiled}_T(R)$. There are three R-redex paths in the automaton, starting at b^2, ba, ab, respectively, and all ending in b^2. They will be right-context-closed by Σ^2, resulting in the following $3 \times 2^2 = 12$ tiled rules, where $x, y \in \Sigma$:

$$[bba, bab, abb, b^3, bbx, bxy] \to [b^3, b^3, bba, baa, aab, abx, bxy]$$
$$[baa, aab, abb, b^3, bbx, bxy] \to [bab, abb, bba, baa, aab, abx, bxy]$$
$$[aba, bab, abb, b^3, bbx, bxy] \to [abb, b^3, bba, baa, aab, abx, bxy]$$

With the following weights, all rules are strictly decreasing:

$$bbb \mapsto 8, bab \mapsto 4, abb \mapsto 3, bba \mapsto 3, \text{others} \mapsto 0.$$

This shows termination of $\mathsf{btiled}_T(R)$, thus, of R.

The following observation, similar to *semantic unlabeling* [20], allows to use the partial algebra for removing rules without labeling:

▶ **Proposition 5.3.** *If the set of tiles T is R-closed, and $R_0 \subseteq R$ such that $\mathsf{tiled}_T(R_0) = \emptyset$, then $\mathsf{SN}(R)$ on $\mathsf{Lang}(T)$ if and only if $\mathsf{SN}(R \setminus R_0)$ on $\mathsf{Lang}(T)$.*

Proof. Let $R_1 = R \setminus R_0$. By Theorem 3.9, each R-derivation corresponds to a $\mathsf{btiled}_T(R)$-derivation. By assumption, this is a $\mathsf{btiled}_T(R_1)$-derivation. This can be mapped back to a R_1-derivation, using the same theorem. ◀

If R_0 is nonempty, this produces a strictly smaller termination problem on the original alphabet. This results in a modification of Algorithm 5.1:

▶ **Algorithm 5.4.**
- *Specification:*
  - *Input: A rewriting system R over Σ, a number k*
  - *Output: A rewriting system R' over Σ such that $\mathsf{SN}(R) \iff \mathsf{SN}(R')$, or failure.*
- *Implementation: By Algorithm 4.1, construct T such that $\mathsf{Lang}(T)$ contains $\mathsf{rhs}(R)$ and is closed w.r.t. $R \cup \mathsf{forw}(R)$, that is, $\mathsf{RFC}(R) \subseteq \mathsf{Lang}(T)$. Let $R_0 \subseteq R$ consist of all rules $(l \to r) \in R$ with $\mathsf{btiled}_T(l \to r) = \emptyset$. If $\emptyset \neq R_0$, then output $R \setminus R_0$, else fail.*

▶ **Example 5.5.** We apply Algorithm 5.4, for $k = 2$, to $R = \{ab \to bca, bc \to cbb, ba \to acb\}$. This is `SRS/Zantema/z018` from TPDB. We construct the 2-shift automaton, see Figure 3, and we find that $\mathsf{btiled}_T(ab \to bca) = \emptyset$. The algorithm outputs $\{bc \to cbb, ba \to acb\}$. Note that

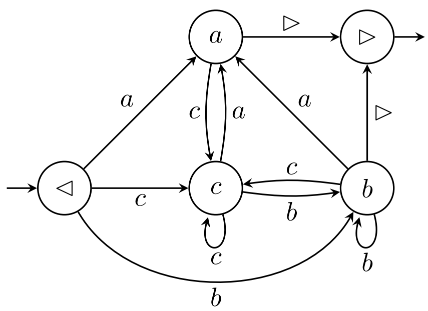


Figure 3 The 2-shift automaton for $\mathsf{RFC}(z018)$.

the automaton contains redexes for $(a \to_{\mathsf{suffix}} bca) \in \mathsf{forw}(ab \to bca)$ (from states $\lhd, c$, and b) but the criterion is the occurrence of $ab \to bca$ only. To handle the resulting termination problem, we reverse all strings in all (remaining) rules, obtaining $\{cb \to bbc, ab \to bca\}$. Again we apply Algorithm 5.4 and this time we find that ab does not occur in the automaton. This leaves $\{cb \to bbc\}$. Applying the algorithm one more time, we find that there is no cb in the 2-shift automaton for $\mathsf{RFC}(cb \to bbc)$. The algorithm outputs $\emptyset$, and we have proved termination of z018.

▶ **Example 5.6.** We prove termination of Zantema's problem $\{a^2b^2 \to b^3a^3\}$, a classical benchmark. We give an outline of the proof that consists of a chain of transformations. Each node (r, s) denotes a rewrite system with r rules on s letters. The arrows $\longrightarrow^{\mathsf{RFC}_k}_{\mathsf{All}}$ and $\longrightarrow^{\mathsf{RFC}_k}_{\mathsf{Rem}}$ denote application of Algorithm 5.1 and Algorithm 5.4 respectively, and $\overset{\mathsf{W}}{\to}$ denotes removal of rules by weights.

$$(1,2) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (4,4) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (3,4) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (12,8) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_3} (105,26) \overset{\mathsf{W}}{\to} (60,26)$$
$$\xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (37,26) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (97,44) \overset{\mathsf{W}}{\to} (65,43) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (36,43) \overset{\mathsf{W}}{\to} (28,43) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (86,68)$$
$$\overset{\mathsf{W}}{\to} (50,62) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_3} (246,128) \overset{\mathsf{W}}{\to} (42,84) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (2,44) \overset{\mathsf{W}}{\to} (0,0)$$

It is even possible to give a termination proof *without* using weights at all:

$$(1,2) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (4,4) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (3,4) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_3} (40,15) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (105,26) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (65,26) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (52,26)$$
$$\xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (37,26) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (97,44) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (37,43) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (36,43) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (110,68) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (80,64)$$
$$\xrightarrow[\mathsf{All}]{\mathsf{RFC}_2} (192,93) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (96,89) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_3} (58,79) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_5} (32,66) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_3} (0,0).$$

▶ **Example 5.7.** We show that our method can be applied as a preprocessor for other termination provers. We consider $R = \{0000 \to 1001, 0101 \to 0010\}$, which is `SRS/Gebhardt/16` from the TPDB. After the chain of transformations

$$(2,2) \xrightarrow[\mathsf{All}]{\mathsf{RFC}_3} (98,20) \overset{\mathsf{W}}{\to} (24,11) \xrightarrow[\mathsf{Rem}]{\mathsf{RFC}_2} (17,10) \overset{\mathsf{W}}{\to} (15,8),$$

the resulting problem can be solved by TTT2 [14] quickly, via KBO. TTT2 did not solve this problem in the Termination Competition 2018.

6 Overlap Closures and Relative Termination

We now apply our approach to prove relative termination. With relative termination, the RFC method does not work.

▶ **Example 6.1.** R/S may nonterminate although R/S terminates on $\mathsf{RFC}(R \cup S)$. For example, let $R = \{ab \to a\}$ and $S = \{c \to bc\}$. We have $\mathsf{RFC}(R \cup S) = a \cup b^+c$. This does not have a factor ab, therefore $\mathsf{SN}(R/S)$ on $\mathsf{RFC}(R \cup S)$. On the other hand, $\neg\mathsf{SN}(R/S)$ because of the loop $\underline{ab}c \to_R a\underline{c} \to_S abc$.

Therefore, we use overlap closures instead. To prove correctness of this approach, we use a characterization of overlap closures as derivations in which every position between letters is touched. A new left-recursive characterization of overlap closures (Corollary 7.1) allows us to enumerate $\mathsf{ROC}(R)$ by completion.

Let $\mathsf{OC}(R)$ denote the set of overlap closures [10], and let $\mathsf{ROC}(R) = \mathsf{rhs}(\mathsf{OC}(R))$. A position between letters in the starting string of a derivation is called *touched* by the derivation if it has no residual in the final string.

▶ **Example 6.2.** For the rewrite system $R = \{ab \to baa\}$ over alphabet $\{a, b\}$, all positions labelled by | in the starting string $a|a|ba|b$ are touched by the derivation $aabab \to_R abaaab \to_R baaaaab \to_R baaaabaa$. The position between b and a in the starting string has the residual position between a and b in the final string.

▶ **Lemma 6.3.** *[7, Lemma 3] The set $\mathsf{OC}(R)$ of overlap closures of R is the set of all R-derivations where all initial positions between letters are touched.*

Termination has been characterized by forward closures ([2]). In the following we obtain a characterization of relative termination by overlap closures.

▶ **Definition 6.4.** *For a finite or infinite R-derivation A, let $\mathsf{Inf}(A)$ denote the set of rules that are applied infinitely often in A. (For a finite derivation, $\mathsf{Inf}(A) = \emptyset$.)*

▶ **Proposition 6.5.** *For each R-derivation A, there are finitely many R-derivations $B_1, \ldots, B_k$ that start in $\mathsf{ROC}(R)$, and $\mathsf{Inf}(A) = \bigcup_i \mathsf{Inf}(B_i)$.*

Proof. If A is empty, then $k = 0$. If A has a finite prefix that is an OC, then $k = 1$ and B_1 is the (infinite) suffix. Else, the start of A has a position that is never touched during A. We can then split the derivation, and use induction by the length of the start of the derivation. ◀

▶ **Proposition 6.6.** *$\mathsf{SN}(R/S)$ if and only if for each $(R \cup S)$-derivation A, $\mathsf{Inf}(A) \cap R = \emptyset$.*

The following theorem says that for analysis of relative termination, we can restrict to derivations starting from right-hand sides of overlap closures.

▶ **Theorem 6.7.** *$\mathsf{SN}(R/S)$ if and only if $\mathsf{SN}(R/S)$ on $\mathsf{ROC}(R \cup S)$).*

Proof. The implication from left to right is trivial, as we consider a subset of derivations. For the other direction, let A be an $(R \cup S)$-derivation. Using Proposition 6.5 we obtain $B_1, \ldots, B_k$ for A such that

$$\mathsf{Inf}(A) \cap R = (\bigcup_i \mathsf{Inf}(B_i)) \cap R = \bigcup_i (\mathsf{Inf}(B_i) \cap R) = \bigcup_i \emptyset = \emptyset,$$

thus $\mathsf{SN}(R/S)$ by Proposition 6.6. ◀

7 Computation of Overlap Closures by Completion

We employ the following left-recursive characterisation of $\mathsf{ROC}(R)$ (proved in the Appendix) that is suitable for a completion algorithm.

▶ **Corollary 7.1.** *$\mathsf{ROC}(R)$ is the least set S such that*

1. *$\mathsf{rhs}(R) \subseteq S$,*
2. *if $tx \in S$ and $(xu, v) \in R$ for some $t, x, u \neq \epsilon$ then $tv \in S$;*
3. *if $xt \in S$ and $(ux, v) \in R$ for some $t, x, u \neq \epsilon$ then $vt \in S$;*
4. *if $tut' \in S$ and $(u, v) \in R$ then $tvt' \in S$;*
5. *if $tx \in S$ and $yv \in S$ and $(xwy, z) \in R$ for some $t, x, y, v \neq \epsilon$ then $tzv \in S$.*

Note that Item 4 is the standard (factor) rewriting relation of R, Item 2 is suffix rewriting with respect to $\mathsf{forw}(R)$, and Item 3 is prefix rewriting with respect to

$$\mathsf{backw}(R) = \{l_2 \to_{\mathsf{prefix}} r \mid (l_1 l_2 \to r) \in R, l_1 \neq \epsilon \neq l_2\}.$$

Item 5 is an inference rule with two premises, and cannot be written as a rewrite relation. As we still want to apply the partial algebra approach, we have two options: modify that approach to allow more premises, or modify our translation, as follows.

Starting from the automaton constructed in Section 3, we add a path from final state $\triangleright^{k-1}$ to initial state $\triangleleft^{k-1}$, consisting of $k-1$ transitions labelled $\triangleleft$. The language of this automaton is $\mathsf{Lang}(T)\triangleright^{k-1}(\triangleleft^{k-1}\,\mathsf{Lang}(T)\triangleright^{k-1})^*$. Note that this is still a shift automaton. Then an application of Item 5 of Corollary 7.1 with $(xwy, t) \in R$ is realized by a standard rewrite step $x\triangleright^{k-1}\triangleleft^{k-1}y \to_{\mathsf{factor}} t$.

Similar to Definition 3.1, Proposition 3.2, Definition 3.3, we have

▶ **Definition 7.2.** *For $T \subseteq \mathsf{btiles}_k(\Sigma^*)$ the* looped *partial algebra $\mathsf{Shift}^o_k(T)$ has signature $\Sigma \cup \{\triangleleft, \triangleright\}$, domain $\mathsf{tiles}_{k-1}((\triangleleft^{k-1}\Sigma^*\triangleright^{k-1})^*)$, the interpretation of ϵ is $\triangleleft^{k-1}$, and each letter $c \in \Sigma \cup \{\triangleleft, \triangleright\}$ maps p to $\mathsf{Suffix}_{k-1}(pc)$, if $pc \in T \cup \mathsf{tiles}_k(\triangleright^{k-1}\triangleleft^{k-1})$, and is undefined otherwise.*

▶ **Proposition 7.3.** *For a set of tiles T, we have*

$$\mathsf{Lang}_{\mathsf{Shift}^o_k(T)} = \mathsf{Prefix}(\mathsf{Lang}(T)\triangleright^{k-1}(\triangleleft^{k-1}\,\mathsf{Lang}(T)\triangleright^{k-1})^*).$$

▶ **Definition 7.4.** *Let $\mathsf{CC}^o_k(R) = \{(z)x\triangleright^{k-1}\triangleleft^{k-1}ye \to (z)re \mid (xwy \to_{\mathsf{factor}} r) \in R, x \neq \epsilon \neq y, e \in \mathsf{tiles}_{k-1}(\Sigma^*\triangleright^*)\}$.*

The purpose of this construction is:

▶ **Proposition 7.5.** *For a set of k-tiles T and a rewriting system R, if $\mathsf{Lang}_{\mathsf{Shift}^o_k(T)}$ is closed with respect to $\mathsf{CC}_k(R \cup \mathsf{forw}(R) \cup \mathsf{backw}(R)) \cup \mathsf{CC}^o_k(R)$, then $\mathsf{ROC}(R)\triangleright^{k-1} \subseteq \mathsf{Lang}_{\mathsf{Shift}^o_k(T)}$.*

▶ **Example 7.6.** We illustrate the completion algorithm to obtain an approximation for $\mathsf{ROC}(R)$, for $R = \{a^3 \to a^2b^2a^2\}$. We take $k = 4$ and start with the automaton for $\mathsf{rhs}(R)$, and include the backwards path from $\triangleright^3$ to $\triangleleft^3$ (the solid arrows in Figure 4).

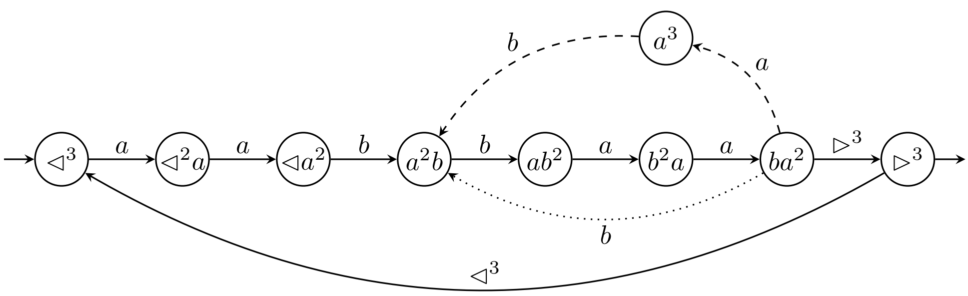


Figure 4 The 4-shift automaton for $\mathsf{ROC}(a^3 \to a^2b^2a^2)$.

We now consider rules $(a\triangleright^3\triangleleft^3 ae \to a^2b^2a^2e) \in \mathsf{CC}^o(R)$. These can only start at state b^2a, and the only choice for the right 3-context e in those rules is abb. The reduct path needs two fresh edges (dashed). For rules $(a^2\triangleright^3\triangleleft^3 ae \to a^2b^2a^2e) \in \mathsf{CC}^o(R)$, a redex must start in ab^2, and the only right 3-context e is still abb. The reduct path needs one extra edge (dotted). The automaton is now closed also with respect to the other operations. We compute $\mathsf{btiled}_T(R)$. There is just one R-redex, starting at ab^2, with just one right extension bba. This creates just one labeled rule

$$[abba, bbaa, baaa, aaab, aabb, abba] \to [abba, bbaa, baab, aabb, abba, bbaa, baab, aabb, abba].$$

Of course, the actual implementation will not explicitly represent the path labeled $\triangleleft^3$. Similar to Theorem 3.9 we have

▶ **Theorem 7.7.** *If $\mathsf{Lang}(T)$ is closed w.r.t. $R \cup S$, then $\mathsf{SN}(R/S)$ on $\mathsf{Lang}(T)$ if and only if $\mathsf{SN}(\mathsf{btiled}_T(R)/\,\mathsf{btiled}_T(S))$.*

For the proof, we need an obvious extension of [3] Thm 6.4 for relative termination, by keeping track of the origin (R or S) of labeled rules.

8 Examples of Relative Termination Proofs via Overlap Closures

We transform global relative termination $\mathsf{SN}(R/S)$ as follows:

▶ **Algorithm 8.1.**

- *Specification:*
  - *Input: rewriting systems R, S over Σ, number k*
  - *Output: rewriting systems R', S' over $\mathsf{btiled}_k(\Sigma)$ such that $\mathsf{SN}(R/S) \iff \mathsf{SN}(R'/S')$.*
- *Implementation (and correctness): By Theorem 6.7, $\mathsf{SN}(R/S)$ iff $\mathsf{SN}(R/S)$ on $\mathsf{ROC}(R \cup S)$. By Corollary 7.1, $\mathsf{ROC}(R)$ is obtained by completion. By Algorithm 4.1, we construct T such that $\mathsf{Lang}(T)$ contains $\mathsf{rhs}(R)$ and is closed w.r.t. $\mathsf{CC}(R \cup S) \cup (\mathsf{forw}(R \cup S) \cup \mathsf{backw}(R \cup S)) \cup \mathsf{CC}^o(R \cup S)$, that is, $\mathsf{ROC}(R) \subseteq \mathsf{Lang}(T)$. By Theorem 3.9 and the previous, $\mathsf{SN}(R/S)$ iff $\mathsf{SN}(\mathsf{btiled}_T(R)/\,\mathsf{btiled}_T(S))$.*

It is often the case that $\mathsf{SN}(\mathsf{btiled}_T(R)/\,\mathsf{btiled}_T(S))$ can be obtained with some easy method, e. g., weights.

Similar to Algorithm 5.4, there is a variant that removes rules in case $\mathsf{btiled}_T(R_0 \cup S_0) = \emptyset$.

▶ **Example 8.2.** $\mathsf{SN}(ababa \to \epsilon / ab \to bbaa)$ (`SRS_Relative/Waldmann_06_relative/r4` from TPDB) can be solved quickly by Algorithm 8.1 with $k = 4$. The tiled system has 270 rules on 33 tiles, and can be solved with weights. Alternatively, tiling of width 5 produces 51 reachable tiles, where the left-hand side of the strict rule is not covered, so can be removed.

In the Termination Competition 2018, AProVE [9] solved this benchmark with double root labeling, which is very similar to tiling of width 3, but this took more than 4 minutes.

▶ **Example 8.3.** The *bowls and beans* problem had been suggested by Vincent van Oostrom [21]. It asks to prove termination of this relation:

If a bowl contains two or more beans, pick any two beans in it and move one of them to the bowl on its left and the other to the bowl on its right.

In a direct model, a configuration is a function $\mathbb{Z} \to \mathbb{N}$ with finite support. In a rewriting model, this is encoded as a string. Several such models have been submitted to TPDB by Hans Zantema (`SRS_Standard/Zantema_06/beans[1..7]`). We consider here a formalisation as a relative termination problem (`SRS_Relative/Waldmann_06_relative/rbeans`).

$$\{baa \to abc, ca \to ac, cb \to ba\}/\{\epsilon \to b\}$$

Here, a is a bean, b separates adjacent bowls, and c transports a bean to the next bowl. The relative rule is used to add extra bowls at either end – although it can be applied anywhere, meaning that any bowl can be split in two, anytime, which does not hurt termination. To the best of our knowledge, this benchmark problem had never been solved in a termination competition. We can now give a termination proof via tiling of width 3, and using overlap closures. This results in a relative termination problem with 560 rules on 47 letters where 305 rules can be removed by weights, and the remaining strict rules by KBO.

The following example applies Algorithm 8.1 to a relative termination problem that comes from the dependency pairs transformation.

▶ **Example 8.4.** The system $\{ababaababa \to abaabababababaab\}$ is part of the enumeration `SRS_Standard/Wenzel_16`, and it was not solved in Termination Competitions up to 2018. In the competition of 2019, Matchbox obtained a termination proof with outline

$$\begin{aligned}(1,2) &\overset{\mathsf{DP}}{\to} (9,3) \xrightarrow[\mathsf{All}]{\mathsf{ROC}_3} (56,17) \overset{\mathsf{W}}{\to} (34,14) \overset{\mathsf{EDG}}{\to} (24,14) \xrightarrow[\mathsf{Rem}]{\mathsf{ROC}_3} (18,10) \xrightarrow[\mathsf{All}]{\mathsf{ROC}_3} (276,46)\\ &\overset{\mathsf{W}}{\to} (212,39) \overset{\mathsf{EDG}}{\to} (206,39) \xrightarrow[\mathsf{Rem}]{\mathsf{ROC}_3} (151,29) \xrightarrow[\mathsf{All}]{\mathsf{ROC}_3} (2558,138) \overset{\mathsf{W}}{\to} (1962,115)\\ &\overset{\mathsf{EDG}}{\to} (1960,115) \xrightarrow[\mathsf{Rem}]{\mathsf{ROC}_3} (1082,86) \overset{\mathsf{W}}{\to} (156,44),\end{aligned}$$

where $\longrightarrow^{\mathsf{ROC}_k}_{\mathsf{All}}$ and $\longrightarrow^{\mathsf{ROC}_k}_{\mathsf{Rem}}$ denote an application of Algorithm 8.1, or its variant for rule removal, respectively. $\overset{\mathsf{DP}}{\to}$ stands for the dependency pairs transformation, and $\overset{\mathsf{EDG}}{\to}$ denotes the restriction to a strongly connected component of the (estimated) dependency graph. The proof ends successfully with an empty graph.

There are two more systems $\{ababaababa \to abaababababaab\}$ and $\{abaababaab \to aababaabaabab\}$ with the same status. Intermediate systems have up to 3940 rules.

9 Experimental Evaluation

Sparse tiling is implemented in the termination prover Matchbox[2] that won the categories *SRS Standard* and *SRS Relative* in the Termination Competition 2019. Matchbox employs a parallel proof search with a portfolio of algorithms, including Algorithm 8.1.

For relative termination, we use weights, matrix interpretations over the naturals, and tiling of widths 2, 3, 5, 8 (in parallel), cf. Example 8.3. For standard termination, we use RFC matchbounds, and (in parallel) the dependency pairs (DP) transformation, creating a relative termination problem, to which we apply weights, matrix interpretations over natural and arctic numbers, and tiling of width 3 (only), cf. Example 8.4.

Table 1 shows performance of variants of these strategies on SRS benchmarks of TPDB, as measured on Starexec, under the Termination profile (5 minutes wall clock, 20 minutes CPU clock, 128 GByte memory). In all experiments, we keep using weights and (for standard termination) the DP transform. The bottom right entry of each sub-table contains the result for the full strategy, used in competition.

We note a strong increase in the last column (matrices:yes) of the left sub-table. We conclude that sparse tiling is important for relative termination proofs. The right sub-table shows a very weak increase in the corresponding column. We conclude that with Matchbox' current search strategy for standard termination, other methods overshadow tiling, e. g., RFC matchbounds are used in 578 proofs, and arctic matrices in 389 proofs.

[2] `https://gitlab.imn.htwk-leipzig.de/waldmann/pure-matchbox`

Table 1 Number of termination proofs obtained by variants of Matchbox.

SRS Relative Starexec Job 33975		matrices no	matrices yes
tilling	no	1	72
	yes	176	225

SRS Standard Starexec Job 33976		RFC matchbounds, matrices none	RFC matchbounds, matrices both
tilling	no	100	1122
	yes	512	1133

For relative termination, the method of tiling, with weights, but without matrices, is already quite powerful with 176 proofs, a number between those for AProVE (163) and MultumNonMulta (192).

Table 2 shows the widths used in tiling proofs for relative SRS. The sum of the bottom row is larger than the total number of proofs (225) since one proof may use several widths.

Table 2 Number of termination proofs for relative SRS, using given width of tiling.

width	2	3	5	8
proofs	150	57	38	11

We observe that short tiles appear more often. We think the reason is that larger tiles tend to create larger systems that are more costly to handle, while resources (time and space on Starexec) are fixed. This is also the reason for using width 3 only, for standard termination.

10 Conclusion

We have presented *sparse tiling*, a method to compute a regular over-approximation of reachability sets, using sets of tiles, represented as automata, and we applied this to the analysis of termination and relative termination. The method is an instance of semantic labeling via a partial algebra. Our contribution is the choice of the k-shift algebra.

We also provide a powerful implementation in Matchbox that contributed to winning the SRS categories in the Termination Competition 2019. An exact measurement of that contribution is difficult since termination proof search (in Matchbox) depends on too many parameters.

Interesting open questions (that are independent of any implementation) are about the relation between sparse tilings of different widths, and between sparse tilings and other methods, e.g., matchbounds.

Since our focus for the present paper is string rewriting, we also leave open the question of whether sparse tiling would be useful for termination of term rewriting.

References

1 Ronald V. Book and Friedrich Otto. *String-rewriting systems.* Texts and Monographs in Computer Science. Springer, New York, 1993.

2 Nachum Dershowitz. Termination of Linear Rewriting Systems. In Shimon Even and Oded Kariv, editors, *Automata, Languages and Programming, 8th Colloquium, Acre (Akko), Israel, July 13-17, 1981, Proceedings*, volume 115 of *LNCS*, pages 448–458. Springer, 1981. `doi:10.1007/3-540-10843-2_36`.

3 Jörg Endrullis, Roel C. de Vrijer, and Johannes Waldmann. Local Termination: theory and practice. *Logical Methods in Computer Science*, 6(3), 2010. `arXiv:1006.4955`.

4 Bertram Felgenhauer and René Thiemann. Reachability, confluence, and termination analysis with state-compatible automata. *Inf. Comput.*, 253:467–483, 2017. doi:10.1016/j.ic.2016.06.011.

5 Thomas Genet. Decidable Approximations of Sets of Descendants and Sets of Normal Forms. In Tobias Nipkow, editor, *Rewriting Techniques and Applications, 9th International Conference, RTA-98, Tsukuba, Japan, March 30 - April 1, 1998, Proceedings*, volume 1379 of *LNCS*, pages 151–165. Springer, 1998. doi:10.1007/BFb0052368.

6 Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-Bounded String Rewriting Systems. *Appl. Algebra Eng. Commun. Comput.*, 15(3-4):149–171, 2004. doi:10.1007/s00200-004-0162-8.

7 Alfons Geser and Hans Zantema. Non-looping string rewriting. *ITA*, 33(3):279–302, 1999. doi:10.1051/ita:1999118.

8 Dora Giammaresi and Antonio Restivo. Two-Dimensional Languages. In Arto Salomaa and Grzegorz Rozenberg, editors, *Handbook of Formal Languages*, volume 3, pages 215–267. Springer, 1997.

9 Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing Program Termination and Complexity Automatically with AProVE. *J. Autom. Reasoning*, 58(1):3–31, 2017. doi:10.1007/s10817-016-9388-y.

10 John V. Guttag, Deepak Kapur, and David R. Musser. On Proving Uniform Termination and Restricted Termination of Rewriting Systems. *SIAM J. Comput.*, 12(1):189–214, 1983. doi:10.1137/0212012.

11 Miki Hermann. *Divergence des systèmes de réécriture et schématisation des ensembles infinis de termes*. Habilitation, Université de Nancy, France, March 1994.

12 Dieter Hofbauer. System description: MultumNonMulta. In A. Middeldorp and R. Thiemann, editors, *15th Intl. Workshop on Termination, WST 2016, Obergurgl, Austria, 2016, Proceedings*, page 90, 2016.

13 Dieter Hofbauer. MultumNonMulta at TermComp 2018. In S. Lucas, editor, *16th Intl. Workshop on Termination, WST 2016, Oxford, U. K., 2018, Proceedings*, page 80, 2018.

14 Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. In Ralf Treinen, editor, *Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings*, volume 5595 of *LNCS*, pages 295–304. Springer, 2009. doi:10.1007/978-3-642-02348-4_21.

15 Dallas S. Lankford and D. R. Musser. A finite termination criterion. Technical report, Information Sciences Institute, Univ. of Southern California, Marina-del-Rey, CA, 1978.

16 Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, 1971.

17 Aart Middeldorp, Hitoshi Ohsaki, and Hans Zantema. Transforming Termination by Self-Labelling. In Michael A. McRobbie and John K. Slaney, editors, *Automated Deduction - CADE-13, 13th International Conference on Automated Deduction, New Brunswick, NJ, USA, July 30 - August 3, 1996, Proceedings*, volume 1104 of *LNCS*, pages 373–387. Springer, 1996. doi:10.1007/3-540-61511-3_101.

18 Jacques Sakarovitch. *Éléments the théorie des automates*. Vuibert Informatique, 2003.

19 Christian Sternagel and Aart Middeldorp. Root-Labeling. In Andrei Voronkov, editor, *Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings*, volume 5117 of *LNCS*, pages 336–350. Springer, 2008. doi:10.1007/978-3-540-70590-1_23.

20 Christian Sternagel and René Thiemann. Modular and Certified Semantic Labeling and Unlabeling. In Manfred Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia*, volume 10 of *LIPIcs*, pages 329–344. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.RTA.2011.329.

21 Vincent van Oostrom. Bowls and Beans. CWI puzzle, `http://www.phil.uu.nl/~oostrom/publication/misc.html`, accessible via `https://web.archive.org/`, 2004.

22 Johannes Waldmann. Efficient Completion of Weighted Automata. In Andrea Corradini and Hans Zantema, editors, *Proceedings 9th International Workshop on Computing with Terms and Graphs, TERMGRAPH 2016, Eindhoven, The Netherlands, April 8, 2016.*, volume 225 of *EPTCS*, pages 55–62, 2016. `doi:10.4204/EPTCS.225.8`.

23 Yechezkel Zalcstein. Locally testable languages. *Journal of Computer and System Sciences*, 6(2):151–167, 1972. `doi:10.1016/S0022-0000(72)80020-5`.

24 Hans Zantema. Termination of Term Rewriting by Semantic Labelling. *Fundam. Inform.*, 24(1/2):89–105, 1995. `doi:10.3233/FI-1995-24124`.

25 Hans Zantema. Termination. In Terese, editor, *Term Rewriting Systems*, pages 181–259. Cambrigde Univ. Press, 2003.

26 Hans Zantema. Termination of String Rewriting Proved Automatically. *J. Autom. Reasoning*, 34(2):105–139, 2005. `doi:10.1007/s10817-005-6545-0`.

A Composition Trees of Overlap Closures

In this section we derive a left-recursive characterization of overlap closures in string rewriting. By left-recursive, we mean that the recursive descent takes place only in the left partners. The definition of overlap closures recurses in both arguments (we always overlap a closure with a closure):

▶ **Definition A.1** ([10]). *For a rewrite system R, the set OC is defined as the least set such that*

1. $R \subseteq \mathit{OC}$,
2. *if* $(s, tx) \in \mathit{OC}$ *and* $(xu, v) \in \mathit{OC}$ *for some* $t, x, u \neq \epsilon$ *then* $(su, tv) \in \mathit{OC}$;
2'. *if* $(s, xt) \in \mathit{OC}$ *and* $(ux, v) \in \mathit{OC}$ *for some* $t, x, u \neq \epsilon$ *then* $(us, vt) \in \mathit{OC}$;
3 *if* $(s, tut') \in \mathit{OC}$ *and* $(u, v) \in \mathit{OC}$ *then* $(s, tvt') \in \mathit{OC}$;
3'. *if* $(u, v) \in \mathit{OC}$ *and* $(svs', t) \in \mathit{OC}$ *then* $(sus', t) \in \mathit{OC}$.

The following recursive definition is left-recursive (we overlap a closure with a rule). We need an extra rule (Item 4) and drop a rule (Item 3'), the others correspond to Definition A.1.

▶ **Definition A.2.** *For a rewrite system R, the set OC' is defined as the least set such that*

1. $R \subseteq \mathit{OC}'$,
2. *if* $(s, tx) \in \mathit{OC}'$ *and* $(xu, v) \in R$ *for some* $t, x, u \neq \epsilon$ *then* $(su, tv) \in \mathit{OC}'$;
2'. *if* $(s, xt) \in \mathit{OC}'$ *and* $(ux, v) \in R$ *for some* $t, x, u \neq \epsilon$ *then* $(us, vt) \in \mathit{OC}'$;
3. *if* $(s, tut') \in \mathit{OC}'$ *and* $(u, v) \in R$ *then* $(s, tvt') \in \mathit{OC}'$;
4. *if* $(s, tx) \in \mathit{OC}'$ *and* $(u, yv) \in \mathit{OC}'$ *and* $(xwy, z) \in R$ *for some* $t, x, y, v \neq \epsilon$ *then* $(swu, tzv) \in \mathit{OC}'$.

The main result of this Appendix is that the set OC' covers the overlap closures up to inverse rewriting of left hand sides:

▶ **Theorem A.3.** $\mathit{OC} = \{(s, t) \mid s \to_R^* s' \wedge (s', t) \in \mathit{OC}'\}$.

Since we are interested in right-hand sides of closures, these extra rewrite steps do not hurt.

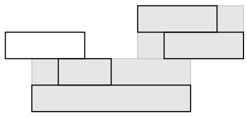
In order to prove Theorem A.3, it is useful to represent a closure by a tree that describes the way the closure is formed: the composition tree of the closure. Each node of a composition tree denotes an application of one of the inference rules of Definitions A.1 and A.2. An extra node type 3' denotes an $\to_R$-step as seen in Theorem A.3.

▶ **Definition A.4** ([7]). *Define the signature $\Omega = \{1, 2, 2', 3, 3', 4\}$, where 1 is unary, 4 is ternary, and the other symbols are binary. The set CT of* composition trees *is defined as the set of ground terms over Ω.*

▶ **Definition A.5.** *A composition tree represents a set of string pairs, as follows:*

$$\begin{aligned}
\langle 1\rangle &= \{(\ell, r) \mid (\ell \to r) \in R\},\\
\langle 2(c_1, c_2)\rangle &= \{(su, tv) \mid (s, tx) \in \langle c_1\rangle, (xu, v) \in \langle c_2\rangle, t, x, u \neq \epsilon\},\\
\langle 2'(c_1, c_2)\rangle &= \{(us, vt) \mid (s, xt) \in \langle c_1\rangle, (ux, v) \in \langle c_2\rangle, t, x, u \neq \epsilon\},\\
\langle 3(c_1, c_2)\rangle &= \{(s, tvt') \mid (s, tut') \in \langle c_1\rangle, (u, v) \in \langle c_2\rangle\},\\
\langle 3'(c_1, c_2)\rangle &= \{(sus', t) \mid (svs', t) \in \langle c_1\rangle, (u, v) \in \langle c_2\rangle\},\\
\langle 4(c_1, c_2, c_3)\rangle &= \{(swu, tzv) \mid (s, tx) \in \langle c_1\rangle, (u, yv) \in \langle c_2\rangle,\\
&\qquad (xwy, z) \in \langle c_3\rangle, t, x, y, v \neq \epsilon\} \ .
\end{aligned}$$

▶ **Example A.6.** The composition tree $4(1, 2(1, 1), 3'(1, 1))$ denotes all pairs obtained by the following overlaps of rewrite steps. Times flows from top to bottom. Each of the rectangles of height 1 is a step, corresponding to a 1 node in the tree. The grey rectangle in the top right is $2(1, 1)$, the grey rectangle in the bottom is $3'(1, 1)$.



Let CT' denote the composition trees that do not contain the function symbol 4. By construction we have:

▶ **Lemma A.7.** $\mathit{OC} = \bigcup_{c\in \mathit{CT}'}\langle c\rangle$.

Adding symbols 4 does not increase expressiveness, since $\langle 4(c_1, c_2, c_3)\rangle \subseteq \langle 2(c_1, 2'(c_2, c_3))\rangle$.

▶ **Lemma A.8.** $\mathit{OC} = \bigcup_{c\in \mathit{CT}}\langle c\rangle$.

In the remainder of this section, we give a semantics-preserving transformation from CT (arbitrary composition trees) to a subset that describes the right-hand side of Theorem A.3. Let us first characterize the goal precisely.

▶ **Definition A.9.** *The set CT_N is given by the regular tree grammar with variables T, D (top, deep), start variable T, and rules*

$$T \to 3'(1, T) \mid D, \qquad D \to 1 \mid 2(D, 1) \mid 2'(D, 1) \mid 3(D, 1) \mid 4(D, D, 1).$$

Rules for D correspond to the rules of Definition A.2, creating $(s', t) \in \mathsf{OC}'$. Rules for T correspond to the initial derivation $s \to_R^* s'$. Therefore,

▶ **Lemma A.10.** $\langle \mathit{CT}_N\rangle = \{(s, t) \mid s \to_R^* s' \wedge (s', t) \in \mathit{OC}'\}$.

We are going to construct a term rewriting system Q on Ω that has CT_N as normal forms. It must remove all non-1 symbols from the left argument of $3'$, and remove all non-1 symbols from the rightmost argument of $2, 2', 3$, and 4. Also, it must remove all $3'$ that are below some non-$3'$. These conditions already determine the set of left-hand sides of Q.

For each left-hand side l, the set of right-hand sides must cover l semantically:

$$\forall l \in \mathsf{lhs}(Q) : \langle l\rangle \subseteq \bigcup_{(l,r)\in Q} \langle r\rangle.$$

A term rewriting system Q over signature Ω with the desired properties is defined in Table 3. We bubble-up 3' symbols, e. g., $2(3'(c_1, c_2), c_3) \to 3'(c_1, 2(c_2, c_3))$ (Rule 9), and we rotate to move non-1 symbols, e. g., $2(c_1, 2(c_2, c_3)) \to 2(2(c_1, c_2), c_3)$ (Rule 1). Rotation below $3'$ goes to the left. Rules 3 and 13 show that symbol 4 cannot be avoided.

Table 3 The term rewriting system Q for composition trees.

$$2(c_1, 2(c_2, c_3)) \to 2(2(c_1, c_2), c_3) \tag{1}$$
$$2(c_1, 2(c_2, c_3)) \to 2(3(c_1, c_2), c_3) \tag{2}$$
$$2(c_1, 2'(c_2, c_3)) \to 4(c_1, c_2, c_3) \tag{3}$$
$$2(c_1, 2'(c_2, c_3)) \to 3(2(c_1, c_2), c_3) \tag{4}$$
$$2(c_1, 3(c_2, c_3)) \to 3(2(c_1, c_2), c_3) \tag{5}$$
$$2(c_1, 3'(c_2, c_3)) \to 3'(c_2, 2(c_1, c_3)) \tag{6}$$
$$2(c_1, 3'(c_2, c_3)) \to 2(2(c_1, c_2), c_3)) \tag{7}$$
$$2(c_1, 3'(c_2, c_3)) \to 2(3(c_1, c_2), c_3)) \tag{8}$$
$$2(3'(c_1, c_2), c_3) \to 3'(c_1, 2(c_2, c_3)) \tag{9}$$
$$2(c_1, 4(c_2, c_2', c_3)) \to 4(2(c_1, c_2), c_2', c_3) \tag{10}$$
$$2(c_1, 4(c_2, c_2', c_3)) \to 4(3(c_1, c_2), c_2', c_3) \tag{11}$$
$$2(c_1, 4(c_2, c_2', c_3)) \to 3(3(2(c_1, c_2'), c_2), c_3) \tag{12}$$
$$2'(c_1, 2(c_2, c_3)) \to 4(c_1, c_2, c_3) \tag{13}$$
$$2'(c_1, 2(c_2, c_3)) \to 3(2'(c_1, c_2), c_3) \tag{14}$$
$$2'(c_1, 2'(c_2, c_3)) \to 2'(2'(c_1, c_2), c_3) \tag{15}$$
$$2'(c_1, 2'(c_2, c_3)) \to 2'(3(c_1, c_2), c_3) \tag{16}$$
$$2'(c_1, 3(c_2, c_3)) \to 3(2'(c_1, c_2), c_3) \tag{17}$$
$$2'(c_1, 3'(c_2, c_3)) \to 3'(c_2, 2'(c_1, c_3)) \tag{18}$$
$$2'(c_1, 3'(c_2, c_3)) \to 2'(2'(c_1, c_2), c_3) \tag{19}$$
$$2'(c_1, 4(c_2, c_2', c_3)) \to 4(c_2, 2'(c_1, c_2'), c_3) \tag{20}$$
$$2'(c_1, 4(c_2, c_2', c_3)) \to 4(c_2, 3(c_1, c_2'), c_3) \tag{21}$$
$$2'(c_1, 4(c_2, c_2', c_3)) \to 3(3(2'(c_1, c_2), c_2'), c_3) \tag{22}$$
$$2'(3'(c_1, c_2), c_3) \to 3'(c_1, 2'(c_2, c_3)), \tag{23}$$
$$3(c_1, 2(c_2, c_3)) \to 3(3(c_1, c_2), c_3) \tag{24}$$
$$3(c_1, 2'(c_2, c_3)) \to 3(3(c_1, c_2), c_3) \tag{25}$$
$$3(c_1, 3(c_2, c_3)) \to 3(3(c_1, c_2), c_3) \tag{26}$$
$$3(c_1, 3'(c_2, c_3)) \to 3(3(c_1, c_2), c_3) \tag{27}$$
$$3(3'(c_1, c_2), c_3) \to 3'(c_1, 3(c_2, c_3)) \tag{28}$$
$$3(c_1, 4(c_2, c_2', c_3)) \to 3(3(3(c_1, c_2), c_2'), c_3) \tag{29}$$
$$3'(2(c_1, c_2), c_3) \to 3'(c_1, 3'(c_2, c_3)) \tag{30}$$
$$3'(2'(c_1, c_2), c_3) \to 3'(c_1, 3'(c_2, c_3)) \tag{31}$$
$$3'(3(c_1, c_2), c_3) \to 3'(c_1, 3'(c_2, c_3)) \tag{32}$$
$$3'(3'(c_1, c_2), c_3) \to 3'(c_1, 3'(c_2, c_3)) \tag{33}$$
$$3'(4(c_1, c_1', c_2), c_3) \to 3'(c_1, 3'(c_1', 3'(c_2, c_3))) \tag{34}$$
$$4(c_1, c_1', 2(c_2, c_3)) \to 4(2(c_1, c_2), c_1', c_3) \tag{35}$$
$$4(c_1, c_1', 2(c_2, c_3)) \to 3(4(c_1, c_1', c_2), c_3) \tag{36}$$
$$4(c_1, c_1', 2(c_2, c_3)) \to 4(3(c_1, c_2), c_1', c_3) \tag{37}$$
$$4(c_1, c_1', 2'(c_2, c_3)) \to 3(4(c_1, c_1', c_2), c_3) \tag{38}$$
$$4(c_1, c_1', 2'(c_2, c_3)) \to 4(c_1, 2(c_1', c_2), c_3) \tag{39}$$
$$4(c_1, c_1', 2'(c_2, c_3)) \to 4(c_1, 3(c_1', c_2), c_3) \tag{40}$$
$$4(c_1, c_1', 3(c_2, c_3)) \to 3(4(c_1, c_1', c_2), c_3) \tag{41}$$
$$4(c_1, c_1', 3'(c_2, c_3)) \to 3'(c_2, 4(c_1, c_1', c_3)) \tag{42}$$
$$4(c_1, c_1', 3'(c_2, c_3)) \to 4(2(c_1, c_2), c_1', c_3) \tag{43}$$
$$4(c_1, c_1', 3'(c_2, c_3)) \to 4(c_1, 2'(c_1', c_2), c_3) \tag{44}$$
$$4(c_1, c_1', 3'(c_2, c_3)) \to 3(4(c_1, c_1', c_2), c_3) \tag{45}$$
$$4(c_1, c_1', 3'(c_2, c_3)) \to 4(3(c_1, c_2), c_1', c_3) \tag{46}$$
$$4(c_1, c_1', 3'(c_2, c_3)) \to 4(c_1, 3(c_1', c_2), c_3) \tag{47}$$
$$4(3'(c_1, c_2), c_1', c_3) \to 3'(c_1, 4(c_2, c_1', c_3)) \tag{48}$$
$$4(c_1, 3'(c_1', c_2), c_3) \to 3'(c_1', 4(c_1, c_2, c_3)) \tag{49}$$
$$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 4(2(c_1, c_2), 2'(c_1', c_2'), c_3) \tag{50}$$
$$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 4(3(c_1, c_2), 2'(c_1', c_2'), c_3) \tag{51}$$
$$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 4(2(c_1, c_2), 3(c_1', c_2'), c_3) \tag{52}$$
$$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 4(3(c_1, c_2), 3(c_1', c_2'), c_3) \tag{53}$$
$$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 3(3(4(c_1, c_1', c_2'), c_2), c_3) \tag{54}$$
$$4(c_1, c_1', 4(c_2, c_2', c_3)) \to 3(3(4(c_1, c_1', c_2), c_2'), c_3) \tag{55}$$

Termination of Q follows from a lexicographic combination of an interpretation ρ that decreases under rotation, and an interpretation σ that decreases under bubbling.

▶ **Lemma A.11.** *Q terminates.*

Proof. Let the two interpretations ρ and σ on natural numbers be defined by

$$\begin{aligned}
\rho(1) &= 2, \\
\rho(2(c_1, c_2)) = \rho(2'(c_1, c_2)) = \rho(3(c_1, c_2)) &= \rho(c_1) + 2\rho(c_2), \\
\rho(3'(c_1, c_2)) &= 2\rho(c_1) + \rho(c_2), \\
\rho(4(c_1, c_2, c_3)) &= \rho(c_1) + \rho(c_2) + 2\rho(c_3),
\end{aligned}$$

$$\begin{aligned}
\sigma(1) &= 2,\\
\sigma(2(c_1,c_2)) &= \sigma(2'(c_1,c_2)) = \sigma(3(c_1,c_2)) = \sigma(c_1)\cdot\sigma(c_2),\\
\sigma(3'(c_1,c_2)) &= \sigma(c_1)\cdot\sigma(c_2)+1,\\
\sigma(4(c_1,c_2,c_3)) &= \sigma(c_1)\cdot\sigma(c_2)\cdot\sigma(c_3)\ .
\end{aligned}$$

The order $>$ on terms defined by $s > t$ if $\rho(s) > \rho(t)$ or $\rho(s) = \rho(t)$ and $\sigma(s) > \sigma(t)$ is a reduction order. With this, the rules $\ell \to r$ in 9, 28, 48, and 49 satisfy $\rho(\ell) = \rho(r)$ and $\sigma(\ell) > \sigma(r)$. For instance, Rule 49 satisfies $\rho(\ell) = \rho(r) = \rho(c_1) + \rho(c_1') + \rho(c_2) + 2\rho(c_3)$ and $\sigma(\ell) = (\sigma(c_1)\sigma(c_2)+1)\sigma(c_1')\sigma(c_3) > \sigma(c_1)\sigma(c_2)\sigma(c_1')\sigma(c_3)+1 = \sigma(r)$. All other rules $\ell \to r$ in Q satisfy $\rho(\ell) > \rho(r)$. For instance, Rule 54 satisfies $\rho(\ell) = \rho(c_1) + \rho(c_1') + 2\rho(c_2) + 2\rho(c_2') + 4\rho(c_3) > \rho(c_1) + \rho(c_1') + 2\rho(c_2) + 2\rho(c_2') + 2\rho(c_3) = \rho(r)$. So Q is ordered by the reduction order $>$, and so Q terminates. ◀

▶ **Lemma A.12.** *For every composition tree c that admits a Q rewrite step, and for every $(s,t) \in \langle c\rangle$ there is a composition tree c' such that both $c \to_Q c'$ and $(s,t) \in \langle c'\rangle$.*

Proof. The proof is done by a case analysis over all left hand sides of Q. We show only one particularly complex case; the other cases work similarly.

Let $c = 4(c_1, c_1', 4(c_2, c_2', c_3))$. By definition of $\langle\cdot\rangle$, we get $s = \hat{s}wu$, $t = \hat{t}zv$, $(\hat{s}, \hat{t}x) \in \langle c_1\rangle$, $(u, yv) \in \langle c_1'\rangle$, $(xwy, z) \in \langle 4(c_2, c_2', c_3)\rangle$ for some $\hat{t}, x, y, v \neq \epsilon$. Again, we get $xwy = s'w'u'$, $z = t'z'v'$, $(s', t'x') \in \langle c_2\rangle$, $(u', y'v') \in \langle c_2'\rangle$, $(x'w'y', z') \in \langle c_3\rangle$ for some $t', x', y', v' \neq \epsilon$. We distinguish cases according to the overlaps:

1. $x \in \mathsf{Prefix}(s')$, $y \in \mathsf{Suffix}(u')$. Then $(\hat{s}s'', \hat{t}t'x') \in \langle 2(c_1, c_2)\rangle$ where $s'' \neq \epsilon$ is defined by $s' = xs''$. Next, $(u''u, y'v'v) \in \langle 2'(c_1', c_2')\rangle$ where $u'' \neq \epsilon$ is defined by $u' = u''y$. Finally, $(s,t) = (\hat{s}s''w'u''u, \hat{t}t'z'v'v) \in \langle 4(2(c_1,c_2), 2'(c_1',c_2'), c_3)\rangle$, and we choose $c \to c' = 4(2(c_1,c_2), 2'(c_1',c_2'), c_3)$ by Rule 50.
2. s' is a prefix of x, $x \in \mathsf{Prefix}(s'w')$, $y \in \mathsf{Suffix}(u')$. Then $(\hat{s}, \hat{t}t'x'') \in \langle 3(c_1,c_2)\rangle$ where x'' is defined by $x = s'x''$. Next, $(u''u, y'v'v) \in \langle 2'(c_1', c_2')\rangle$ where $u'' \neq \epsilon$ is defined by $u' = u''y$. Finally, $(s,t) = (\hat{s}w''u''u, \hat{t}t'z'v'v) \in \langle 4(3(c_1,c_2), 2'(c_1',c_2'), c_3)\rangle$, where w'' is defined by $w' = x''w''$, and we choose $c \to c' = 4(3(c_1,c_2), 2'(c_1',c_2'), c_3)$ by Rule 51.
3. $x \in \mathsf{Prefix}(s')$, u' is a suffix of y, $y \in \mathsf{Suffix}(w'u')$. This case is symmetric to Case 2. We use Rule 52.
4. s' is a prefix of x, u' is a suffix of y. Then $(\hat{s}, \hat{t}t'x'') \in \langle 3(c_1,c_2)\rangle$ where x'' is defined by $x = s'x''$. Next, $(u, y''v'v) \in \langle 3(c_1', c_2')\rangle$ where y'' is defined by $y = y''u'$. Finally, $(s,t) = (\hat{s}w''u, \hat{t}t'z'v'v) \in \langle 4(3(c_1,c_2), 3(c_1',c_2'), c_3)\rangle$, where w'' is defined by $w' = x''w''y''$, and we choose $c \to c' = 4(3(c_1,c_2), 3(c_1',c_2'), c_3)$ by Rule 53.
5. $s'w'$ is a prefix of x, $y \in \mathsf{Suffix}(u')$. Then $(\hat{s}u''u, \hat{t}y'v') \in \langle 4(c_1, c_2, c_2')\rangle$ where x'' is defined by $x = s'w'x''$, and u'' is defined by $u' = x''u''$. Next, $(\hat{s}u''u, \hat{t}t'x'w'y'v'v) \in \langle 3(4(c_1,c_2,c_2'), c_1')\rangle$. Finally, $(s,t) = (\hat{s}u''u, \hat{t}t'z'v'v) \in \langle 3(3(4(c_1,c_2,c_2'), c_1'), c_3)\rangle$, by Rule 54.
6. $x \in \mathsf{Prefix}(s')$, $w'u'$ is a suffix of y. This case is symmetric to Case 5. We use Rule 55. ◀

▶ **Lemma A.13.** *For every composition tree c that is in Q-normal form and does not contain any $3'$ symbols, we have $\langle c\rangle \subseteq \mathsf{OC}'$.*

Proof. The claim is proven by induction on $|c|$ as follows. If $c = 1$ then $\langle c\rangle = R \subseteq \mathsf{OC}'$. If $c = 2(c_1, c_2)$ then $c_2 = 1$ because c is in Q-normal form. From the inductive hypothesis for c_1 we get $\langle c_1\rangle \subseteq \mathsf{OC}'$; so $\langle c\rangle \subseteq \mathsf{OC}'$. The same argument applies when $c = 2'(c_1, c_2)$ or $c = 3(c_1, c_2)$. If $c = 4(c_1, c_2, c_3)$ then $c_3 = 1$ because c is in Q-normal form. From the inductive hypothesis for c_1 we get $\langle c_1\rangle \subseteq \mathsf{OC}'$. From the inductive hypothesis for c_2 we get $\langle c_2\rangle \subseteq \mathsf{OC}'$. So $\langle c\rangle \subseteq \mathsf{OC}'$. ◀

Now we are ready to prove Theorem A.3.

Proof of Theorem A.3. We prove that $c \in \mathsf{OC}$ and $(s,t) \in \langle c \rangle$ implies $s \to_R^* s'$ and $(s',t) \in \mathsf{OC}'$ for some s'. We do so by induction on c, ordered by $>$. If c admits a Q rewrite step then by Lemma A.12 there is a composition tree c' such that both $c \to_Q c'$ and $(s,t) \in \langle c' \rangle$. Because $c > c'$, the claim follows by inductive hypothesis for c'. Now suppose that c is in Q-normal form. If c does not contain any $3'$ symbol then $(s,t) \in \mathsf{OC}'$ by Lemma A.13, and we choose $s' = s$. Else, because c is in Q-normal form, $c = 3'(1, c_2)$ for some c_2. Let $(s'', t) \in \langle c_2 \rangle$ and $s \to_R s''$. From the inductive hypothesis for c_2 we get $s'' \to_R^* s'$ and $(s',t) \in \mathsf{OC}'$ for some s'. So $s \to_R s'' \to_R^* s'$ and the claim holds. ◀

From Theorem A.3, we immediately get:

▶ **Corollary A.14.** *$\mathit{rhs}(OC) = \mathit{rhs}(OC')$.*

Because OC' is left-recursive, we can derive a recursive characterization of the set of right hand sides of overlap closures:

▶ **Corollary A.15** (This is Corollary 7.1)**.** *$\mathit{rhs}(OC)$ is the least set S such that*

1. *$\mathit{rhs}(R) \subseteq S$,*
2. *if $tx \in S$ and $(xu, v) \in R$ for some $t, x, u \neq \epsilon$ then $tv \in S$;*
3. *if $xt \in S$ and $(ux, v) \in R$ for some $t, x, u \neq \epsilon$ then $vt \in S$;*
4. *if $tut' \in S$ and $(u, v) \in R$ then $tvt' \in S$;*
5. *if $tx \in S$ and $yv \in S$ and $(xwy, z) \in R$ for some $t, x, y, v \neq \epsilon$ then $tzv \in S$.*