Efficient Construction of Reversible Transducers from Regular Transducer Expressions

Paul Gastin, Luc Dartois, R. Govind, Shankara Narayanan Krishna

Efficient Construction of Reversible Transducers from Regular Transducer Expressions

Luc Dartois
Univ Paris Est Creteil, LACL, F-94010 Creteil, France
luc.dartois@lacl.fr

Paul Gastin
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, 91190, Gif-sur-Yvette, France
paul.gastin@lsv.fr

R. Govind
IIT Bombay, India
govindr@cse.iitb.ac.in

Shankara Narayanan Krishna
IIT Bombay, India
krishnas@cse.iitb.ac.in

Abstract

The class of regular transformations has several equivalent characterizations such as functional MSO transductions, deterministic two-way transducers, streaming string transducers, as well as regular transducer expressions (RTE).

For algorithmic applications, it is very common and useful to transform a specification, here, an RTE, to a machine, here, a transducer. In this paper, we give an efficient construction of a two-way reversible transducer (2RFT) equivalent to a given RTE. 2RFTs are a well behaved class of transducers which are deterministic and co-deterministic (hence allows evaluation in linear time w.r.t. the input word), and where composition has only polynomial complexity.

We show that, for full RTE, the constructed 2RFT has size doubly exponential in the size of the expression, while, if the RTE does not use Hadamard product or chained-star, the constructed 2RFT has size exponential in the size of the RTE.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases transducers, regular expressions, parser, evaluation

Funding Supported by IRL ReLaX

1 Introduction

One of the most celebrated results in theoretical computer science is the robust characterization of languages using machines, expressions and logic. For regular languages, these three dimensions are given by finite state automata, regular expressions as well as monadic second-order logic, while for aperiodic languages, the respective three pillars are counter-free automata, star-free expressions and first-order logic. The Büchi-Elgot-Trakhtenbrot theorem was generalized by Engelfreit and Hoogeboom [10], where regular transformations were defined using two-way transducers (2DFTs) as well as by the MSO transductions of Courcelle [5]. The analogue of Kleene's theorem for transformations was proposed by [2] and [8], while [7] proved the analogue of Schützenberger's theorem [13] for transformations. In another related work, [3, 4] proposes a translation from unambiguous two-way transducers to regular function expressions extending the Brzozowski and McCluskey algorithm. All these papers propose declarative languages which are expressive enough to capture all regular (respectively, aperiodic) transformations.

Our starting point in this paper is the combinator expressions presented in the declarative language RTE of [8, 9]. Like classical regular expressions, these expressions provide a robust foundation for specifying transducer patterns in a declarative manner, and can be widely used in practical applications. An important question left open in [2], [7], [8] is the complexity of the procedure that builds the transducer from the combinator expressions. Providing efficient constructions of finite state transducers equivalent to expressions is a fundamental problem, and is often the first step of algorithmic applications, such as evaluation. In this paper, we focus on this problem. First, we recall the combinators from [2], [7] and [8] in order of increasing difficulty. It is known that the combinators of [2] and [8] are equivalent (they both characterize regular transformations), even though the notations differ slightly. Our notations are closer to [8].

We begin presenting the combinators.

(i) The base combinator is $e \rhd v$ which maps any $w \in L(e)$ to the (constant) value v.

(ii) The sum combinator $f + g$ where f or g is applied to the input w depending on whether $w \in \mathsf{dom}(f)$ or $w \in \mathsf{dom}(g)$,

(iii) The Cauchy product $f \cdot g$ splits the input into two parts and outputs the concatenation of the results obtained by applying f on the first part and g on the second part, and

(iv) The star combinator $f^\star$ which splits the input into multiple parts and outputs the concatenation of evaluating f on the respective parts from left to right.

All these operators can be ambiguous and imply a relational semantics. We denote the fragment of RTE restricted to combinators (i–iv) as RTE[Rat]; this fragment captures all rational transformations (those computed by a non-deterministic one way transducer, 1NFT).

(v) The reverse concatenation combinator $f \cdot_r g$ which works like $f \cdot g$ except that the output is now the concatenation of the result of applying g on the second part of the split of the input followed by f on the first part,

(vi) The reverse star combinator $f_r^\star$, also like $f^\star$, splits the input into multiple parts but outputs the concatenation of evaluating f on each of the parts from right to left.

The fragment of RTE with combinators (i–vi) is denoted RTE[Rat, $\cdot_r, \star_r$].

Finally, we have the most involved combinators, namely

(vii) The Hadamard product $f \odot g$, which outputs the concatenation of applying f on the input followed by applying g on the input, as long as the input is in the domain of both f and g. With $\odot$ also, we have the fragment RTE[Rat, $\cdot_r, \star_r, \odot$].

($viii$) The chained k-star $[e, f]^{k\star}$ which factorizes an input w into $u_1 u_2 \cdots u_n$, each u_i belonging to the language of e, and applies f on all contiguous k blocks $u_{i+1} \cdots u_{i+k}$, $0 \leqslant i \leqslant n-k$ and finally concatenates the result.

(ix) The reverse chained k-star $[e, f]_r^{k\star}$ also factorizes an input w into $u_1 u_2 \cdots u_n$, each u_i belonging to the language of e, and applies f on all contiguous k blocks from the right to the left, and the result is concatenated.

RTE is the full class consisting of all combinators (i–ix), and its unambiguous fragment is equivalent to regular transformations (those computed by a deterministic two-way transducer, 2DFT). Note that we consider chained k-star of [7] here, even though chained 2-star suffice for expressing all regular transformations, since the idea of our construction is general.

Our Contributions. Given an RTE h, we give an efficient procedure to directly construct a reversible two-way transducer that computes h. Even though [2] and [8] construct SST/2DFT from combinator expressions,

1. they do not perform a complexity analysis,

2. the constructed machines in these papers rely on intermediate compositions, which incur an exponential blowup at each step, making them unsuitable in practice for applications,
3. translating the SST/2DFT from these papers into 2RFT results in a further exponential blow up. The emphasis on 2RFT is due to the fact that, unlike SST/2DFT, these machines incur only a polynomial complexity for composition, making them the preferred machine model for handling modular specifications.

We list our main contributions.

1. **A clean semantics**. As our first contribution, we propose a *globally* unambiguous semantics (gu-semantics for short) $[\![h]\!]^U$ for all $h \in \mathsf{RTE}$. The previous papers [2], [8] proposed a different unambiguous semantics for the product combinators $\cdot, \star, \odot$, that we refer to here as *locally* unambiguous semantics (lu-semantics for short) to distinguish from our gu-semantics (see Section 3.3 for a comparison). We now illustrate why the gu-semantics can be a preferred choice rather than the lu-semantics.

- Consider the Cauchy product $f \cdot g \cdot h$ with $\mathsf{dom}(f) = \mathsf{dom}(g) = \{a, aa\}$, $\mathsf{dom}(h) = \{b, ab\}$, and $f(a) = c$, $f(aa) = cc$, $g(a) = d$, $g(aa) = dd$, $h(b) = e$, $h(ab) = ee$. Consider $w = aaab$. Under the lu-semantics, w admits a unique factorization $(a \cdot a) \cdot ab$ for $(f \cdot g) \cdot h$ with $aa \in \mathsf{dom}(f.g)$, $ab \in \mathsf{dom}(h)$. Also, w admits a unique factorization $aa \cdot (a \cdot b)$ for $f \cdot (g \cdot h)$ with $aa \in \mathsf{dom}(f)$, $ab \in \mathsf{dom}(g \cdot h)$. Note that $a \cdot (aab)$ does not qualify as a factorization for $f \cdot (g \cdot h)$ since aab has more than one factorization for $g \cdot h$. However, $((f \cdot g) \cdot h)(w) = cdee \neq ccde = (f \cdot (g \cdot h))(w)$. For the gu-semantics, we define the unambiguous domain $\mathsf{udom}(h)$ of an expression h as the set of words which can be parsed unambiguously with respect to the *global* expression h. For the Cauchy product, $\mathsf{udom}(f \cdot g)$ is the set of words w having a unique factorization $w = u \cdot v$ with $u \in \mathsf{dom}(f)$, $v \in \mathsf{dom}(g)$, and, in addition, $u \in \mathsf{udom}(f)$, $v \in \mathsf{udom}(g)$. For the example above, we have $aaab \notin \mathsf{udom}((f \cdot g) \cdot h) = \mathsf{udom}(f \cdot (g \cdot h))$. Thus, the Cauchy product is associative under the gu-semantics. Associativity is natural for the Cauchy product, and not having this is confusing for a user working on specifications in the lu-semantics.
- The lu-semantics of the Cauchy product used in previous papers [2, 8], allows to get symmetric differences of domains, hence also complements. Consider two regular expressions e_1, e_2 over alphabet Σ and a marker $\$ \notin \Sigma$. For $i = 1, 2$ let $h_i = (\varepsilon + \$e_i) \rhd \varepsilon$ with domain $\varepsilon + \$L(e_i)$. The domain of $h_1 \cdot h_2$ is $\varepsilon + \$(L(e_1)\Delta L(e_2)) + \$L(e_1)\$L(e_2)$, where Δ denotes symmetric difference. If we intersect with $\$\Sigma^\star$, we get the symmetric difference $\$(L(e_1)\Delta L(e_2))$. If $L(e_2) = \Sigma^*$, we obtain the complement $\$(\Sigma^\star \backslash L(e_1))$. This explains that an exponential blow-up is unavoidable when dealing with the lu-semantics. Note that for a standalone expression $h_1 \cdot h_2$, the lu, gu semantics agree, however, things are different when one deals with a nested expression containing $h_1 \cdot h_2$. To illustrate this, consider the expression $h = (f_1 \cdot f_2) + f_3$. On an input w, $h(w)$ is $f_3(w)$ if $w \notin \mathsf{dom}(f_1 \cdot f_2)$. Next, to check if $w \in \mathsf{dom}(f_1 \cdot f_2)$, one has to verify if w has an unambiguous split as $u_1 u_2$ with $u_1 \in \mathsf{dom}(f_1)$, $u_2 \in \mathsf{dom}(f_2)$. This requires us to complement the set of all words having more than one split $w = u_1 u_2$. Thus, evaluating h on w requires two nested complements. In general, evaluating an expression in lu-semantics may require arbitrary nested complementation. This is required due to the "local unambiguity check" at each local nesting level in the lu-semantics, accentuating the exponential blow up problem. In contrast, under the gu-semantics, the unambiguity requirement is at a global level.

To summarize, the lu-semantics of [2], [8] may be difficult to comprehend for a user specifying with RTE given that $\cdot$ is non-associative and the same kind of unexpected behaviours arises with iterations. It allows more inputs to be in the domain, but it may not be obvious to check if a given input is in the domain or to predict which output

will be produced. Our gu-semantics on the other hand, is more intuitive, and hence easier to use. Another important point to note is that the gu-semantics does not restrict the expressiveness of RTEs. Although [2], [8] proposed the lu-semantics and showed the equivalence between RTEs and SST/2DFT, it can be seen from their equivalence proofs that the RTE h constructed there from a SST/2DFT $\mathcal{T}$ satisfies $\mathsf{udom}(h) = \mathsf{dom}(\mathcal{T})$ and $[\![h]\!]^U = [\![\mathcal{T}]\!]$.

2. **Efficient Construction of 2RFT**. The second contribution is the efficient construction of 2RFTs from RTE specifications. Given $h \in \mathsf{RTE}$, and a word w, we first "parse" w according to h using a 1NFT $\mathcal{P}_h$ called the parser (see Page 12 for a discussion why $\mathcal{P}_h$ is a one way machine). The parsing relation of h w.r.t. a word w, $P_h(w)$, can be seen as a traversal of possible parse trees of w w.r.t. h. Examples are given in Sections 4 and 6. Each possible parsing in $P_h(w)$ introduces pairs of parentheses $(_i \ \ )_i$ to bracket the factor of w matching subexpressions h_i of h. To illustrate the need for such a parsing, consider the expression $h = f_1 \cdot (f_2 \odot f_3)$. To evaluate h on some input w, one must guess the position in w where the scope of $\mathsf{dom}(f_1)$ ends, and $\mathsf{dom}(f_2 \odot f_3)$ begins. Note that we must apply f_2, f_3 on the same suffix of w, necessitating a two way behaviour. After applying f_2 on a suffix v of $w = u \cdot v$, one must come back to the beginning of v to apply f_3. It is unclear how one can do this without inserting some markers, especially if the decomposition is ambiguous.
   If w does not have an unambiguous parsing w.r.t. h, then $\mathcal{P}_h$ will non-deterministically produce the parsings of w. For each $\alpha \in [\![\mathcal{P}_h]\!](w)$, the projection of α to Σ is w. Next, we construct an evaluator which is a two-way reversible transducer (2RFT) $\mathcal{T}_h$ which takes words in $\mathcal{P}_h(w)$ as input, and produces words in $[\![h]\!]^R(w)$, where $[\![h]\!]^R$ denotes the relational semantics of h. That is, $[\![h]\!]^R = [\![\mathcal{T}_h]\!] \circ P_h$.
   2RFT for the globally unambiguous semantics. Note that the 1NFT $\mathcal{P}_h$ does not check whether $w \in \mathsf{udom}(h)$. To obtain the gu-semantics $[\![h]\!]^U$, we have to restrict to words in $\mathsf{udom}(h)$. This is achieved by proving that $\mathsf{udom}(h)$ coincides with words $w \in \mathsf{dom}(P_h)$ such that $|P_h(w)| = 1$. The unambiguity of the domain is checked by constructing an automaton that accepts the set of words having at most one parsing w.r.t. h. We construct a reversible automaton B' of size $2^{\mathcal{O}(\|\mathcal{P}_h\|^2)}$ to do this, where $\|\mathcal{P}_h\|$ denotes the size of $\mathcal{P}_h$. Next, we *uniformize* $\mathcal{P}_h$ to obtain a 2RFT $\mathcal{P}'_h$ with the same domain as $\mathcal{P}_h$ and such that $[\![\mathcal{P}'_h]\!] \subseteq [\![\mathcal{P}_h]\!]^R$: when running on $u \in \mathsf{dom}(\mathcal{P}_h)$, the 2RFT $\mathcal{P}'_h$ produces *some output* v such that $(u, v) \in [\![\mathcal{P}_h]\!]^R$. The size of $\mathcal{P}'_h$ is $2^{\mathcal{O}(\|\mathcal{P}_h\|)}$. Then, we construct a machine $\mathcal{P}^U_h$ that first runs the automaton B' without producing anything and then runs $\mathcal{P}'_h$ if B' has accepted. This transducer is reversible and computes the parsing relation on words belonging to $\mathsf{udom}(h)$. Its size is $2^{\mathcal{O}(\|\mathcal{P}_h\|^2)}$. Finally, the composition of $\mathcal{T}_h$ and $\mathcal{P}^U_h$ gives a 2RFT $\mathcal{T}^U_h$ which realizes $[\![h]\!]^U$.

Figure 1 shows all the components used in our construction, and their interconnection: the parser $\mathcal{P}_h$ (a 1NFT), the uniformizer $\mathcal{P}'_h$ of the parser $\mathcal{P}_h$ (a 2RFT), the functionality checker of the parser (NFA B and RFA B') and the final transducer $\mathcal{T}_h$ (a 2RFT).

We now discuss the sizes of the 2RFT obtained for various RTE fragments.

- $\mathsf{RTE}[\mathsf{Rat}]$. In this case, the parser $\mathcal{P}_h$ and the evaluator $\mathcal{T}_h$ have sizes $\|\mathcal{P}_h\|, \|\mathcal{T}_h\| \leqslant |h|$. Thus, the composed machine obtained from $\mathcal{P}^U_h$ and $\mathcal{T}_h$ has size $2^{\mathcal{O}(|h|^2)}$. Notice that a one-way deterministic automaton accepting the domain of h would already be of exponential size. Indeed we can use a standard construction producing a rational transducer from an expression $h \in \mathsf{RTE}[\mathsf{Rat}]$, but it would realize the relational semantics of h and not its unambiguous semantics.
- $\mathsf{RTE}[\mathsf{Rat}, ._r, \star_r]$. We have the same complexity here as for $\mathsf{RTE}[\mathsf{Rat}]$. Even if we add

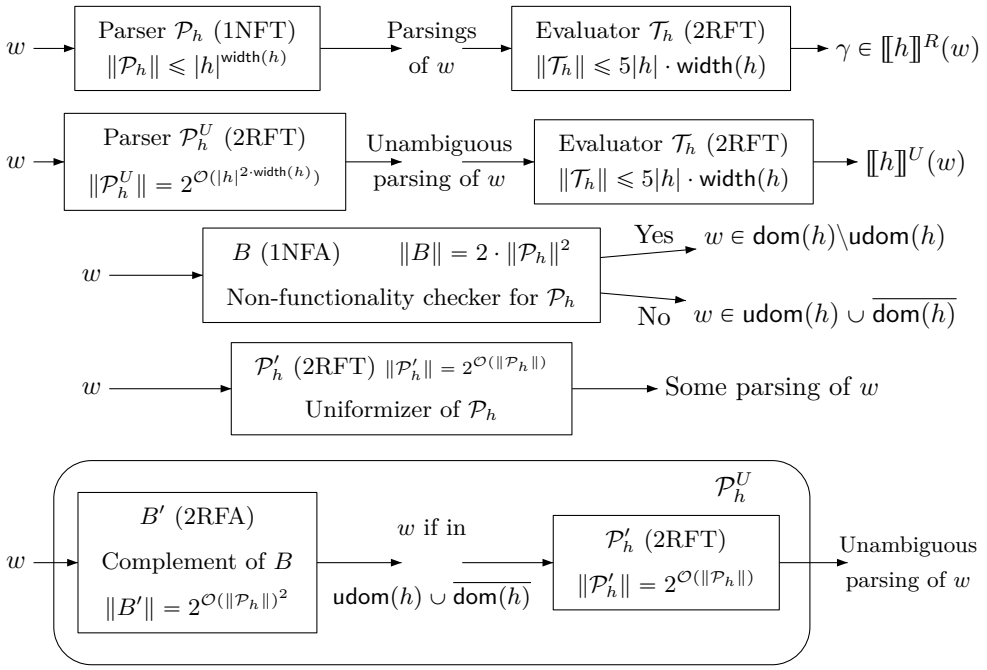


Figure 1 The topmost figure shows the parser 1NFT $\mathcal{P}_h$ which, on an input w, produces a parsing in $P_h(w)$. This is taken as input by the 2RFT $\mathcal{T}_h$, and producing a possible output γ in $h(w)$. This denotes the relational semantics of h, where $h(w)$ is not unique. The second figure from the top shows the 2RFT $\mathcal{P}_h^U$ which works only on words w having a unique parsing, and produces this unique parsing $P_h(w)$. This is then taken as input by the 2RFT $\mathcal{T}_h$, producing the output $h(w)$. Here, $w \in \mathsf{udom}(h)$, and $\mathcal{T}_h$ produces the output $h(w)$. The third figure describes the automaton B used for checking the functionality of the 1NFT $\mathcal{P}_h$: B accepts all words w which are not in $\mathsf{udom}(h)$. Note that the complement of B, a reversible automaton B' is used in $\mathcal{P}_h^U$. The fourth figure shows the uniformization of $\mathcal{P}_h$, given by the 2RFT $\mathcal{P}'_h$. This machine outputs *some* parsing of w. The last figure shows $\mathcal{P}_h^U$. This first uses the reversible automaton B' to filter words not in $\mathsf{udom}(h)$. All words accepted by B' either lie in $\mathsf{udom}(h)$, or outside $\mathsf{dom}(h)$. For words $w \in \mathsf{udom}(h)$, there is a unique parsing, and $\mathcal{P}'_h$ produces this unique parsing of w.

on to this fragment, the useful functions dup and rev which respectively duplicates and reverses the input, the complexity is still the same.

- $\mathsf{RTE}[\mathsf{Rat}, .^r, \star_r, \odot]$. Unlike the $\mathsf{Rat}, .^r, \star_r$ combinators, $\odot$ requires to read the input twice. It is noteworthy that our parser $\mathcal{P}_h$ is still a 1NFT. However, in this case, its size is $\|\mathcal{P}_h\| \leqslant |h|^{\mathsf{width}(h)}$ while $\|\mathcal{T}_h\| \leqslant 5|h|$. The *width* of an RTE h is intuitively the maximal number of times a position in w needs to be read to produce the output. Even though our parser is still a 1NFT, its size is affected by the width. Notice that the domain of a Hadamard product $h = f \odot g$ is the intersection of the domains of its arguments. Moreover, the parser $\mathcal{P}_h$ may be used to recognize the domain of h. This gives an exponential lower bound on the size of any possible parser for expressions in $\mathsf{RTE}[\mathsf{Rat}, \odot]$ (see Proposition 10).
  For expressions $h \in \mathsf{RTE}[\mathsf{Rat}, .^r, \star_r, \odot]$, the size of the final 2RFT $\mathcal{T}_h^U$ is $2^{\mathcal{O}(|h|^{2\cdot\mathsf{width}(h)})}$. Note that the fragments $\mathsf{RTE}[\mathsf{Rat}, .^r, \star_r]$, along with $\mathsf{dup}, \mathsf{rev}$ have $\mathsf{width}(h) = 1$.
- For full RTE, the parser $\mathcal{P}_h$ is still a 1NFT and the bounds are the same as $\mathsf{RTE}[\mathsf{Rat}, .^r, \star_r, \odot]$, except now, we have $\|\mathcal{T}_h\| \leqslant 5|h|\mathsf{width}(h)$.

Related Work. A paper which has looked at the evaluation of transducer expressions

is [1]. Here, the authors investigate the complexity of evaluation of a DReX program on a given input word. A DReX program is a combinator expression [2] and works with the lu-semantics. A major difference between [1] and our paper is that [1] does not construct a machine equivalent to a DReX program, citing complexity considerations and the difficulty in coming up with an automaton for the lu-semantics. Instead, [1] directly solve the evaluation problem using dynamic programming.

To the best of our knowledge, our paper is the first one to efficiently construct a 2RFT from an RTE. This 2RFT may be used to solve algorithmic problems on transformations specified by transducer expressions. One such problem is indeed the evaluation of any number of input words w; we can simply run our constructed 2RFT on w in time linear in $|w|$. Note that [1] also evaluates with the same linear bound, under what they call the "consistent" semantics, a restriction of the lu-semantics. The consistent semantics is also more restrictive than our gu-semantics. To mention an instance, the $\texttt{combine}(f, g)$ combinator in [1] is analogous to the Hadamard product $f \odot g$, with the added restriction that that $\mathsf{dom}(f) = \mathsf{dom}(g)$. Our gu-semantics for $f \odot g$ only requires that the input is in $\mathsf{dom}(f) \cap \mathsf{dom}(g)$.

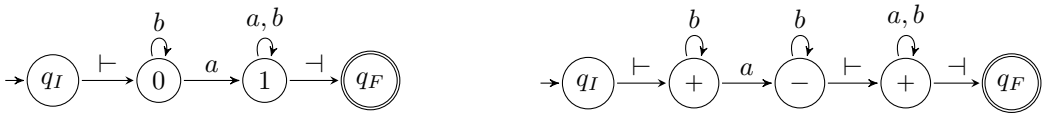
Structure of the paper. Section 2 introduces our models of automata and transducers while Section 3 defines the Regular Transducers Expressions, as well as the relational semantics and the unambiguous semantics considered throughout the paper. It also states our results. The following sections are devoted to the constructions of transducers and the proofs of our main results, in an incremental fashion: Section 4 treats the case of Rational relations, Section 5 handles some simple extensions, and Sections 6 and 7 treats the Hadamard product and k-star operators respectively. Finally, Section 8 shows how to compute a reversible transducer for the unambiguous semantics.

2 Automata and Transducers

Automata. Let Σ be an *alphabet*, i.e., a finite set of letters. A word u over Σ is a possibly empty sequence of letters. The set of words is denoted Σ^*, with ε denoting the empty word. Given an alphabet Σ, we denote by $\Sigma_{\vdash\dashv}$ the set $\Sigma \uplus \{\vdash, \dashv\}$, where $\vdash$ and $\dashv$ are two fresh symbols called the left and right endmarkers. A *two-way finite state automaton* (2NFA) is a tuple $\mathcal{A} = (\Sigma, Q, q_I, F, \Delta)$, where Σ is a finite alphabet, Q is a finite set of states partitioned into the set of forward states Q^+ and the set of backward states Q^-, $q_I \in Q^+$ is the initial state, $F \subseteq Q^+$ is the set of final states, $\Delta \subseteq Q \times \Sigma_{\vdash\dashv} \times Q$ is the state transition relation. By convention, q_I and $q_F \in F$ are the only forward states verifying $(q_I, \vdash, q) \in \Delta$ and $(q, \dashv, q_F) \in \Delta$ for some $q \in Q$. However, for any backward state $p^- \in Q^-$, Δ might contain transitions $(p^-, \vdash, q)$ and $(q, \dashv, p^-)$, for some $q \in Q$.

Before defining the semantics of our two-way automata, let us remark that we choose one of several equivalently expressive semantics of two-way. The particularity of the one we chose, which is the one in [6], is that (1) the reading head is put between positions rather than on, and (2) the set of states is divided into + states and − states. The advantage of this semantics is that the sign of a state defines what position the head reads both before and after this state in a valid run. A + state (resp. − state) reads the position to its right (resp. to its left) and the previous position read was on its left (resp. on its right). Intuitively, in a transition (p, a, q) both states move the reading head half a position, either to the right for + states or to the left for − states. Hence if p and q are of different signs, the reading head does not move, but the position read will be different.

We now formally define the semantics. A configuration $u.p.u'$ of $\mathcal{A}$ is composed of two words u, u' such that $uu' \in \vdash\Sigma^*\dashv$ and a state $p \in Q$. The configuration $u.p.u'$ admits a set



(a) A 1DFA $\mathcal{A}_1$ **(b)** A 2RFT $\mathcal{A}_2$

Figure 2 Two automata recognizing the same language $\Sigma^* a \Sigma^*$.

of successor configurations, defined as follows. If $p \in Q^+$, the input head currently reads the first letter of the suffix $u' = a'v'$. The successor of $u.p.u'$ after a transition $(p, a', q) \in \Delta$ is either $ua'.q.v'$ if $q \in Q^+$, or $u.q.u'$ if $q \in Q^-$. Conversely, if $p \in Q^-$, the input head currently reads the last letter of the prefix $u = va$. The successor of $u.p.u'$ after $(p, a, q) \in \Delta$ is $u.q.u'$ if $q \in Q^+$, or $v.q.au'$ if $q \in Q^-$. A *run* of $\mathcal{A}$ on a word $u \in \vdash\Sigma^*\dashv$ is a sequence of successive configurations $\varrho = u_0.q_0.u'_0, \ldots, u_m.q_m.u'_m$ such that for every $0 \leqslant i \leqslant m$, $u_i u'_i = u$. The run ϱ is called *initial* if it starts in configuration $q_I.u$, *final* if it ends in configuration $u.q$ with $q \in F$, *accepting* if it is both initial and final. The language $\mathcal{L}_\mathcal{A}$ *recognized* by $\mathcal{A}$ is the set of words $u \in \Sigma^*$ such that $\vdash u \dashv$ admits an accepting run. The automaton $\mathcal{A}$ is called

- a *one-way finite state automaton* (1NFA) if the set $Q^- = \varnothing$,
- *deterministic* (2DFA) if for all $(p, a) \in Q \times \Sigma_{\vdash\dashv}$, there is at most one $q \in Q$ verifying $(p, a, q) \in \Delta$,
- *co-deterministic* if for all $(q, a) \in Q \times \Sigma_{\vdash\dashv}$, there is at most one $p \in Q$ verifying $(p, a, q) \in \Delta$ and $F = \{q_F\}$.
- *reversible* (2RFA) if it is both deterministic and co-deterministic.

Example. Let us consider the language $\mathcal{L}_a \subseteq \{a, b\}^*$ composed of the words that contains at least one a symbol. This language is recognized by the deterministic one-way automaton $\mathcal{A}_1$ and represented in Figure 2a, and by the reversible two-way automaton $\mathcal{A}_2$, represented in Figure 2b. Note that $\mathcal{A}_1$ is not co-deterministic in state 1 reading an a. In fact, this language is not recognizable by a one-way reversible automaton because reading an a from state 1 cannot lead to state 0, and adding a new state simply moves the problem forward. The reversible two-way transducer solves this problem by using the left endmarker.

Transducers. A *two-way finite state transducer* (2NFT) is a tuple $\mathcal{T} = (\Sigma, \Gamma, Q, q_I, F, \Delta, \mu)$, where Γ is a finite alphabet; $\mathcal{A}_\mathcal{T} = (\Sigma, Q, q_I, F, \Delta)$ is a 2NFA, called the *underlying automaton* of $\mathcal{T}$; and $\mu : \Delta \to \Gamma^*$ is the output function. A run of $\mathcal{T}$ is a run of its underlying automaton, and the language $\mathcal{L}_\mathcal{T}$ recognized by $\mathcal{T}$ is the language $\mathcal{L}_{\mathcal{A}_\mathcal{T}} \subseteq \Sigma^*$ recognized by its underlying automaton. Given a run ϱ of $\mathcal{T}$, we set $\mu(\varrho) \in \Gamma^*$ as the concatenation of the images by μ of the transitions of $\mathcal{T}$ occurring along ϱ. The transduction $\mathcal{R}_\mathcal{T} \subseteq \Sigma^* \times \Gamma^*$ defined by $\mathcal{T}$ is the set of pairs (u, v) such that $u \in \mathcal{L}_\mathcal{T}$ and $\mu(\varrho) = v$ for an accepting run ϱ of $\mathcal{A}_\mathcal{T}$ on $\vdash u \dashv$. Two transducers are called *equivalent* if they define the same transduction. A transducer $\mathcal{T}$ is respectively called one-way (1NFT), deterministic (2DFT), co-deterministic or reversible (2RFT), if its underlying automaton has the corresponding property.

Note that while a generic transducer defines a relation over words, a deterministic, co-deterministic or a reversible one defines a (partial) function from the input words to the output words since any input word has at most one accepting run, and hence at most one image. Extracting a maximal function from a relation is called a *uniformization* of a relation. Formally, given a relation on words $R \subseteq \Sigma^* \times \Gamma^*$, a *uniformization* of R is a function f such that:

- $\mathsf{dom}(f) = \mathsf{dom}(R) = \{u \in \Sigma^* \mid \exists v \in \Gamma^*, (u, v) \in R\}$
- $\forall u \in \mathsf{dom}(f), (u, f(u)) \in R$.

Intuitively, a uniformization chooses, for each left component of R, a unique right component. If R is already a function, then it is its only possible uniformization.

Reversible transducers can be composed easily, and the composition can be done with a single machine having a polynomial number of states. Hence, when dealing with two-way machines, it is always beneficial to handle reversible machines. To this end, we specialize results from [6] and [11] respectively.

▸ **Lemma 1.** *Let $\mathcal{T}$ be a 1NFT with n states. Then we can construct a reversible 2RFT $\mathcal{T}'$ such that $[\![\mathcal{T}']\!]$ is a uniformization of $[\![\mathcal{T}]\!]$ and $\mathcal{T}'$ has at most $144n^2 2^{2n}$ states.*

Proof. Let $\mathcal{T}$ be a 1NFT and n its number of states. We write $\mathcal{T}'$ as the composition $D \circ C$ where C is a co-deterministic one-way transducer and D is a deterministic one. The co-deterministic transducer C is a classical powerset construction that computes and adds to the input the set of co-reachable states of $\mathcal{T}$. Its number of states is at most 2^n. The set of states of the deterministic transducer D is the same as $\mathcal{T}$, and at each step, if it is in a state q, it uses the information given by C to select a successor of q that is also co-reachable. It can be made deterministic by using an arbitrary global order on the set of states of $\mathcal{T}$. Its number of states is then n.

We conclude on the size of $\mathcal{T}'$ using two theorems from [6], stating that C can be made into a reversible two-way C' with $4m^2$ states with m the number of states of C (Theorem 2) and that D can be made into a reversible D' with $36n^2$ states (Theorem 3). Finally, $\mathcal{T}'$ is defined as the composition $D' \circ C'$, whose number of states is at most $36n^2 \cdot 4(2^n)^2 = 144n^2 2^{2n}$. ◂

We will also need a more specific result for computing the complement of an automaton. We rely on Proposition 4 of [11].

▸ **Lemma 2.** *Let A be a 1NFT with n states. Then we can compute a 2RFT B such that $L(B)$ is the complement of $L(A)$, and B has at most $2^{n+1} + 6$ states.*

Proof. The proof is straightforward. First, we transform A into a deterministic automaton C by doing a classical powerset construction. Then C has 2^n states. By inverting the accepting states, we obtain C' which is deterministic and recognizes the complement of $L(C) = L(A)$. We conclude by using Proposition 4 of [11], which states that from a deterministic automaton C', we can construct a 2RFT B by adding 3 states to C' and doubling its number of states. The resulting automaton B is then reversible and its number of states is $2(2^n + 3) = 2^{n+1} + 6$. ◂

3 Transducer expressions and their semantics

In this section, we formally define RTE, and then propose the most natural relational semantics. Then we define the unambiguous domain of a relation, and propose our global unambiguous semantics (called unambiguous semantics from here on) as a restriction of the relational semantics to the unambiguous domain. As already mentioned in the introduction, this semantics refines the unambiguous semantics which has been proposed in earlier papers. Finally, we state the main results of the paper, and an overview of our results.

Regular transducer expressions (RTEs). Let Σ be the input alphabet and Γ be the output alphabet. For the combinator expressions, we use the following syntax:

$$h ::= e \rhd v \mid h + h \mid h \cdot h \mid h \cdot_r h \mid h^\star \mid h_r^\star \mid h \odot h \mid [e, h]^{k\star} \mid [e, h]_r^{k\star}$$

where e is a regular expression over Σ, $v \in \Gamma^\star$ and $k \geqslant 1$.

The semantics of the basic expression $e \rhd v$ is the partial function with (constant) value v and domain $L(e)$, the regular language denoted by e. For instance, the semantics of $\varnothing \rhd v$ is the partial function with empty domain and the semantics of $\Sigma^\star \rhd v$ is the *total* constant function with value v. We use $\bot$ and v as macros to respectively denote $\varnothing \rhd \varepsilon$ and $\Sigma^* \rhd v$.

Since our goal is to construct "small" transducers from RTEs, we have to define formally the *size* of expressions. We use the classical syntax for regular expressions over Σ:

$$e ::= \varnothing \mid \varepsilon \mid a \mid e + e \mid e \cdot e \mid e^\star$$

where $a \in \Sigma$. We also define inductively the number of literals occurring in a regular expression e, denoted by $\mathsf{nl}(e)$: $\mathsf{nl}(\varepsilon) = \mathsf{nl}(\varnothing) = 0$, $\mathsf{nl}(a) = 1$ for $a \in \Sigma$, $\mathsf{nl}(e_1 + e_2) = \mathsf{nl}(e_1 \cdot e_2) = \mathsf{nl}(e_1) + \mathsf{nl}(e_2)$ and $\mathsf{nl}(e_1^\star) = \mathsf{nl}(e_1)$. Notice that we have $\mathsf{nl}(e) \leqslant |e|$ for all regular expressions e, where $|e|$ denotes the standard size of expressions. Actually, if e is not a single letter $a \in \Sigma$, we even have $1 + \mathsf{nl}(e) \leqslant |e|$.

Now, we define the size $|h|$ of a regular transducer expression h. For the base case, we define $|e \rhd v| = 1 + (1 + \mathsf{nl}(e)) + \max(1, |v|)$. Note that when $v = \varepsilon$ it still contributes 1 to the size of $e \rhd \varepsilon$ since it appears as a symbol. Also, we have chosen that the regular expression e contributes to $1 + \mathsf{nl}(e)$ in this size. This is because the number of states of the Glushkov automaton associated with e (which will be used in our construction) is $1 + \mathsf{nl}(e)$. As discussed above, unless e is a single letter from Σ, we have $1 + \mathsf{nl}(e) \leqslant |e|$ (and otherwise $\mathsf{nl}(e) = 1 = |e|$). For the inductive cases, we let $|f+g| = |f \cdot g| = |f \cdot_r g| = |f \odot g| = 1+|f|+|g|$, $|f^\star| = |f_r^\star| = 1 + |f|$, and $|[e, f]^{k\star}| = |[e, f]_r^{k\star}| = 1 + \mathsf{nl}(e) + |f| + k + 1$.

3.1 Relational semantics

In general, the semantics of a regular transducer expression h is a relation $[\![h]\!]^R \subseteq \Sigma^\star \times \Gamma^\star$. This is due to the fact that input words may be parsed in several ways according to a given expression. For instance, when applying a Cauchy product $h = f \cdot g$ to an input word $w \in \Sigma^\star$, we split $w = uv$ and we output the concatenation of f applied to u and g applied to v. There might be several decompositions $w = uv$ with $u \in \mathsf{dom}(f)$ and $v \in \mathsf{dom}(g)$, in which case, the parsing is ambiguous and h applied to w may result in several outputs.

We define inductively for an RTE h, the domain $\mathsf{dom}(h) \subseteq \Sigma^\star$ and the relational semantics $[\![h]\!]^R \subseteq \Sigma^\star \times \Gamma^\star$. As usual, for $u \in \Sigma^\star$, we let $[\![h]\!]^R(u) = \{v \in \Gamma^\star \mid (u, v) \in [\![h]\!]^R\}$.

We also define simultaneously the *unambiguous domain* $\mathsf{udom}(h) \subseteq \mathsf{dom}(h)$ which is the set of words $w \in \mathsf{dom}(h)$ such that parsing w according to h is unambiguous. This is used in the next subsection to define a functional semantics.

- $h = e \rhd v$: As already discussed, we set $[\![e \rhd v]\!]^R = \{(u, v) \mid u \in L(e)\}$ and $\mathsf{udom}(h) = \mathsf{dom}(h) = L(e)$.
- $h = f + g$: We have $\mathsf{dom}(f + g) = \mathsf{dom}(f) \cup \mathsf{dom}(g)$, $[\![f + g]\!]^R = [\![f]\!]^R \cup [\![g]\!]^R$ and $\mathsf{udom}(f + g) = (\mathsf{udom}(f) \backslash \mathsf{dom}(g)) \cup (\mathsf{udom}(g) \backslash \mathsf{dom}(f))$.
- $h = f \cdot g$ (Cauchy product): We have $\mathsf{dom}(f \cdot g) = \mathsf{dom}(f) \cdot \mathsf{dom}(g)$, for $w \in \Sigma^*$ we let $[\![f \cdot g]\!]^R(w) = \bigcup_{w=uv} [\![f]\!]^R(u) \cdot [\![g]\!]^R(v)$ and a word w is in the unambiguous domain of $f \cdot g$ if there is a *unique* factorization $w = uv$ with $u \in \mathsf{dom}(f)$ and $v \in \mathsf{dom}(g)$ and moreover this factorization satisfies $u \in \mathsf{udom}(f)$ and $v \in \mathsf{udom}(g)$: $\mathsf{udom}(f \cdot g) = (\mathsf{udom}(f) \cdot \mathsf{udom}(g)) \backslash \{uvw \mid v \neq \varepsilon \text{ and } u, uv \in \mathsf{dom}(f) \text{ and } vw, w \in \mathsf{dom}(g)\}$.
- $h = f \cdot_r g$ (reverse Cauchy product): We have $\mathsf{dom}(f \cdot_r g) = \mathsf{dom}(f \cdot g)$, $\mathsf{udom}(f \cdot_r g) = \mathsf{udom}(f \cdot g)$, and for $w \in \Sigma^*$ we let $[\![f \cdot_r g]\!]^R(w) = \bigcup_{w=uv} [\![g]\!]^R(v) \cdot [\![f]\!]^R(u)$.
- $h = f^\star$ (Kleene star): We have $\mathsf{dom}(h) = \mathsf{dom}(f)^\star$, and for $w \in \Sigma^*$ we let $[\![h]\!]^R(w) = \bigcup_{w=u_1 \cdots u_n} [\![f]\!]^R(u_1) \cdots [\![f]\!]^R(u_n)$. Notice that if $[\![f]\!]^R$ is *proper*, i.e., if $\varepsilon \notin \mathsf{dom}(f)$, then we may restrict n to at most $|w|$ in the union above. Otherwise, the union will have

infinitely many nonempty terms and $[\![h]\!]^R(w)$ may be an infinite language. Finally, $\mathsf{udom}(h)$ is the set of words $w \in \Sigma^\star$ which have a unique factorization $w = u_1 \cdots u_n$ with $n \geqslant 0$ and $u_i \in \mathsf{dom}(f)$ for all $1 \leqslant i \leqslant n$, and moreover, for this factorization, we have $u_i \in \mathsf{udom}(f)$ for all $1 \leqslant i \leqslant n$. Notice that if $\varepsilon \in \mathsf{dom}(f)$, then $\mathsf{udom}(h) = \varnothing$.

- $h = f_r^\star$ (reverse Kleene star): We have $\mathsf{dom}(f_r^\star) = \mathsf{dom}(f^\star)$, $\mathsf{udom}(f_r^\star) = \mathsf{udom}(f^\star)$, and for $w \in \Sigma^*$ we let $[\![f_r^\star]\!]^R(w) = \bigcup_{w=u_1\cdots u_n} [\![f]\!]^R(u_n) \cdots [\![f]\!]^R(u_1)$.
- $h = f \odot g$ (Hadamard product): We have $\mathsf{dom}(f \odot g) = \mathsf{dom}(f) \cap \mathsf{dom}(g)$, for $w \in \Sigma^*$ we let $[\![f \odot g]\!]^R(w) = [\![f]\!]^R(w) \cdot [\![g]\!]^R(w)$ and $\mathsf{udom}(f \odot g) = \mathsf{udom}(f) \cap \mathsf{udom}(g)$.
- $h = [e, f]^{k\star}$ (k-star): The domain of h is the set of words $w \in \Sigma^\star$ which have a factorization $w = u_1 \cdots u_n$ satisfying (†) $n \geqslant 0$, $u_i \in L(e)$ for $1 \leqslant i \leqslant n$, and $u_{i+1} \cdots u_{i+k} \in \mathsf{dom}(f)$ for $0 \leqslant i \leqslant n-k$. For $w \in \Sigma^*$ we let

$$[\![[e, f]^{k\star}]\!]^R(w) = \bigcup_{\substack{w=u_1\cdots u_n \\ u_1,\ldots,u_n \in L(e)}} \prod_{i=0}^{n-k} [\![f]\!]^R(u_{i+1} \cdots u_{i+k}) \,.$$

  Notice that a factorization $w = u_1 \cdots u_n$ with $n < k$ and $u_i \in L(e)$ for all $1 \leqslant i \leqslant n$, automatically satisfies (†). Hence, $\bigcup_{n<k} L(e)^n \subseteq \mathsf{dom}(h)$. Moreover, when $n < k$, the empty product in the definition above evaluates to $\{\varepsilon\}$ which is the unit for concatenation of languages. The *unambiguous* domain of h is the set of words $w \in \mathsf{dom}(h)$ which have a *unique* factorization $w = u_1 \cdots u_n$ satisfying (†) and moreover, for this factorization, we have $u_{i+1} \cdots u_{i+k} \in \mathsf{udom}(f)$ for all $0 \leqslant i \leqslant n-k$.
- $h = [e, f]_r^{k\star}$ (reverse k-star): We have $\mathsf{dom}([e, f]_r^{k\star}) = \mathsf{dom}([e, f]^{k\star})$, $\mathsf{udom}([e, f]_r^{k\star}) = \mathsf{udom}([e, f]^{k\star})$, and for $w \in \Sigma^*$ we let

$$[\![[e, f]_r^{k\star}]\!]^R(w) = \bigcup_{\substack{w=u_1\cdots u_n \\ u_1,\ldots,u_n \in L(e)}} [\![f]\!]^R(u_{n-k+1} \cdots u_n) \cdots [\![f]\!]^R(u_1 \cdots u_k) \,.$$

We show that the inductive definitions of $\mathsf{dom}(h)$ and $\mathsf{udom}(h)$ indeed give the *domain* of the relation $[\![h]\!]^R$ and ensure functionality. The proof is an easy structural induction.

▸ **Lemma 3.** *Let h be an RTE and $w \in \Sigma^\star$. Then,*

1. *$w \in \mathsf{dom}(h)$ if and only if $[\![h]\!]^R(w) \neq \varnothing$.*
2. *If $w \in \mathsf{udom}(h)$, then $[\![h]\!]^R(w)$ is a singleton.*

3.2 Functional semantics

Our goal is now to define *functions* with regular transducer expressions h. This can be achieved by a restriction of the relational semantics $[\![h]\!]^R$ to a suitable subset of the domain $\mathsf{dom}(h)$.

The first natural idea is to restrict to the set of input words w on which h is functional. Formally, let $\mathsf{fdom}(h) = \{w \in \mathsf{dom}(h) \mid |[\![h]\!]^R(w)| = 1\}$ be the functional domain of h. A functional semantics is obtained by restricting the relational semantics $[\![h]\!]^R$ to the functional domain $\mathsf{fdom}(h)$. The unacceptable problem with this approach is that $\mathsf{fdom}(h)$ need not be regular. For instance, consider $h = f + g$ where $f = (a \rhd c + b \rhd \varepsilon)^\star$ and $g = (a \rhd \varepsilon + b \rhd c)^\star$. We have $\mathsf{dom}(h) = \mathsf{dom}(f) = \mathsf{dom}(g) = \{a, b\}^\star$. Both f and g are functional: for $w \in \{a, b\}^\star$ we have $[\![f]\!]^R(w) = \{c^{|w|_a}\}$ and $[\![g]\!]^R(w) = \{c^{|w|_b}\}$. We deduce that $\mathsf{fdom}(h)$ is the set of words with same number of a's and b's, which is not a regular set.

We adopt the next natural idea, which is to restrict the relational semantics $[\![h]\!]^R$ to its *unambiguous* domain $\mathsf{udom}(h)$ which ensures functionality by Lemma 3. It is not hard to check by structural induction that both $\mathsf{dom}(h)$ and $\mathsf{udom}(h)$ are regular languages over Σ.

We define the *unambiguous* semantics as the restriction of the relational semantics $[\![h]\!]^R$ to the unambiguous domain $\mathsf{udom}(h)$. Formally, this is a partial function $[\![h]\!]^U : \Sigma^\star \to \Gamma^\star$ defined for $w \in \mathsf{udom}(h)$ by the equation $[\![h]\!]^R(w) = \{[\![h]\!]^U(w)\}$.

3.3 Comparing the unambiguous semantics of [2], [8] with ours

1. L/v of [2], $L\,?\,v : \bot$ of [8], our $e \rhd v$.
   The base function L/v in [2] corresponds to our simple expression $e \rhd v$ when $L(e) = L$. This can be written as a if-then-else $L\,?\,v : \bot$ of [8]. The semantics of if-then-else $K\,?\,f : g$ checks if w is in the regular language K or not, and appropriately produces $f(w)$ or $g(w)$.
2. $f \rhd g$ of [2], $\mathsf{dom}(f)\,?\,f : g$ of [8], our $f + g$.
   The *conditional choice* combinator $f \rhd g$ of [2] maps an input w to $f(w)$ if it is in $dom(f)$, and otherwise it maps it to $g(w)$. This can again be written as the $\mathsf{dom}(f)\,?\,f : g$ of [8]. Our analogue of this is $f + g$. Note that when $\mathsf{dom}(f) \cap \mathsf{dom}(g) = \varnothing$, all these combinators coincide.
3. $f \oplus g$ of [2], $f \boxdot g$ of [8], our $h = f \cdot g$.
   The *split-sum* combinator $f \oplus g$ of [2] is the Cauchy product $f \boxdot g$ of [8]. The semantics of $f \boxdot g$, when applied on $w \in \Sigma^*$ produces $f(u) \cdot g(v)$ if there is a unique factorization $w = u \cdot v$ with $u \in \mathsf{dom}(f)$ and $v \in \mathsf{dom}(g)$. Our analogue is $h = f \cdot g$. As mentioned in the introduction, $f \oplus g$ and $f \boxdot g$ are different from our Cauchy product $h = f \cdot g$ which works on $\mathsf{udom}(h)$. While our definition preserves associativity $f \cdot (g \cdot h) = (f \cdot g) \cdot h$, the notions $\oplus, \boxdot$ from [2], [8] do not.
4. Σf of [2], $f^{\boxplus}$ of [8], our $f^\star$.
   The *iterated sum* Σf of [2] is the Kleene-plus $f^{\boxplus}$ of [8] which, when applied to $w \in \Sigma^*$ produces $f(u_1) \cdots f(u_n)$ if $w = u_1 \cdots u_n$ is an unambiguous factorization of w, with each $u_i \in \mathsf{dom}(f)$. This has the same problems as the Cauchy product compared to our $f^\star$.
5. $f + g$ of [2], $f \odot g$ of [8], our $f \odot g$.
   The *sum* $f + g$ of two functions in [2] is the Hadamard product $f \odot g$ of [8], which when applied to w produces $f(w) \cdot g(w)$, provided $w \in \mathsf{dom}(f) \cap \mathsf{dom}(g)$. This agrees with our notion of Hadamard product.
6. $\Sigma(f, L)$ of [2], $[L, f]^{2\boxplus}$ of [8], our $[L, f]^{2\star}$.
   Finally, the *chained sum* $\Sigma(f, L)$ of [2] is the two-chained Kleene-plus $[L, f]^{2\boxplus}$ of [8], which, when applied to w having a unique factorization $w = u_1 \cdot u_2 \cdots u_n$ with $n \geqslant 1$ and $u_i \in L$ for all $1 \leqslant i \leqslant n$ produces $[L, f]^{2\boxplus}(w) = f(u_1u_2) \cdot f(u_2u_3) \cdots f(u_{n-1}u_n)$. We consider $[L, f]^{k\star}$ instead of $[L, f]^{2\star}$ in this paper, and we additionally check if the blocks $u_{i+1} \cdots u_{i+k} \in \mathsf{udom}(f)$ for all $0 \leqslant i \leqslant n - k$.

To summarize, our notion of unambiguity is a global one, compared to the notion in [2], [8], which checks it only at a local level, thereby leading to the undesirable properties as pointed out already for the Cauchy, Kleene-star operators.

3.4 Main results

The goal of the paper is to construct efficiently, a two-way reversible transducer equivalent to a given RTE under the unambiguous semantics. Consider an RTE h and some word $w \in \mathsf{dom}(h)$. We first parse w by adding some marker symbols $(_f,)_f$ inside w, signifying the scope of the subexpressions f in h. If $w \notin \mathsf{udom}(h)$, then there can be many ways of parsing w. We build a non-deterministic one-way transducer $\mathcal{P}_h$ which produces all possible parsings of w. Figure 1 helps to get an overview of the construction. We check if w has at

most one parsing using a 2RFA B'; and if so, apply the uniformized parser transducer $\mathcal{P}'_h$ on w to obtain the unique parsing of w. $\mathcal{P}_h^U$ represents the sequential composition of B' and $\mathcal{P}'_h$. Finally, the parsing of w, $\mathcal{P}_h^U(w)$ is taken as input by a 2RFT, the evaluator transducer $\mathcal{T}_h$, and produces the output of w according to h, making use of the markers.

Why not a 2-way machine for the parser?. A natural question to ask is whether we can have a two-way transducer for $\mathcal{P}_h$ or even directly construct a two-way machine that evaluates h. We discuss some difficulties in this direction. Consider for instance $h = f \cdot g$. We could have a non-deterministic two-way transducer (2NFT) which guesses the point where the scope of $\mathsf{dom}(f)$ ends in the input w and where $\mathsf{dom}(g)$ begins; if $w \notin \mathsf{udom}(h)$, it is unclear if in the backward sweep, the machine can go back to this correct point so as to apply f, g. On another note, if we design a 2NFT which first inserts a marker where the scope of $\mathsf{dom}(f)$ ends and $\mathsf{dom}(g)$ begins, and a second 2NFT which processes this, we will require the composition of these two machines. It is unclear how we can go about composition of two 2NFTs. Irrespective of these difficulties, a 1NFT is easier to use anytime than a 2NFT, if one can construct one, justifying our choice.

In Section 4, we define the parsing relation and construct the corresponding parser and evaluator for $\mathsf{RTE}[\mathsf{Rat}]$, and then extend to $\mathsf{RTE}[\mathsf{Rat}, ._r, \star_r]$ in Section 5. For these fragments, both the parser and the evaluator have size linear in the given expression. In Section 6, we extend the constructions to handle Hadamard product. There, we show that the size of the parser for h is at most exponential in a new parameter, called the *width* of h. Section 7 concludes by showing how to handle the k-star operators.

We define the *width* of an RTE h, denoted $\mathsf{width}(h)$, intuitively as the maximum number of times a position in w needs to be read to output $h(w)$: $\mathsf{width}(e \rhd v) = 1$, $\mathsf{width}(f + g) = \mathsf{width}(f \cdot g) = \mathsf{width}(f \cdot_r g) = \max(\mathsf{width}(f), \mathsf{width}(g))$, $\mathsf{width}(f^\star) = \mathsf{width}(f_r^\star) = \mathsf{width}(f)$, $\mathsf{width}(f \odot g) = \mathsf{width}(f) + \mathsf{width}(g)$ and $\mathsf{width}([e, f]^{k\star}) = \mathsf{width}([e, f]_r^{k\star}) = 2 + k \times \mathsf{width}(f)$. Note that, for k-star and reverse k-star, we define the width as $2 + k \times \mathsf{width}(f)$ instead of $1 + k \times \mathsf{width}(f)$ simply to get a uniform expression for the complexity bounds.

We are ready to state our main theorems, which are proven for each fragment in the corresponding sections.

▸ **Theorem 4.** *Let h be a regular transducer expression. We can define a parsing relation P_h, and construct an evaluator $\mathcal{T}_h$ and a parser $\mathcal{P}_h$ such that*

1. *We have $\mathsf{dom}(P_h) = \mathsf{dom}(h)$ and $\mathsf{udom}(h) = \mathsf{fdom}(P_h)$.*
   Moreover, for each $w \in \mathsf{dom}(h)$ and $\alpha \in P_h(w)$, the projection of α on Σ is $\pi_\Sigma(\alpha) = w$.
2. *The evaluator $\mathcal{T}_h$ is a 2RFT and, when composed with the parsing relation P_h, it computes the relational semantics of h: $[\![h]\!]^R = [\![\mathcal{T}_h]\!] \circ P_h$.*
   Moreover, the number of states of the evaluator is $\|\mathcal{T}_h\| \leqslant 5|h|\mathsf{width}(h)$.
   If h does not use k-star or reverse k-star, then $\|\mathcal{T}_h\| \leqslant 5|h|$.
3. *The parser $\mathcal{P}_h$ is a 1NFT which computes the parsing relation $[\![\mathcal{P}_h]\!]^R = P_h$.*
   The number of states of the parser is $\|\mathcal{P}_h\| \leqslant |h|^{\mathsf{width}(h)}$.
   If h does not use Hadamard product or k-star or reverse k-star then $\|\mathcal{P}_h\| \leqslant |h|$.

▸ **Theorem 5.** *Let h be a regular transducer expression. We can construct a 2RFT $\mathcal{T}_h^U$ which computes the unambiguous semantics of h: $[\![h]\!]^U = [\![\mathcal{T}_h^U]\!]$. The number of states of $\mathcal{T}_h^U$ is $\|\mathcal{T}_h^U\| \leqslant 2^{\mathcal{O}(|h|^{2 \cdot \mathsf{width}(h)})}$. Moreover, if h does not use Hadamard product, k-star or reverse k-star, the number of states of $\mathcal{T}_h^U$ is $2^{\mathcal{O}(|h|^2)}$.*

Theorem 4 is proved in the following sections. The statements for rational functions are proved in Lemma 6. Section 5 shows that the result still hold when the rational functions are enriched with reverse products. Lemmas 7, 8 and 9 respectively show that items 1,2 and

3 still hold when the Hadamard product is present, and finally Lemmas 12, 13 and 14 extend this to the full RTEs. Theorem 5 is proved in Section 8.

4 Rational functions

In this section, we deal with *rational* transducer expressions which consist of the fragment of *regular* transducer expressions defined by the syntax:

$$h ::= e \rhd v \mid h + h \mid h \cdot h \mid h^\star$$

where e is a regular expression over the input alphabet Σ and $v \in \Gamma^\star$. The semantics $[\![h]\!]^R$ and $[\![h]\!]^U$ are inherited from RTEs (Section 3).

Our goal is to parse an input word $w \in \Sigma^\star$ according to a given rational transducer expression h and to insert markers resulting in parsed words $\alpha \in P_h(w)$. From these marked words, it will be easier to compute the value defined by the expression, especially in the forthcoming sections where we also deal with Hadamard product, reverse Cauchy product, reverse Kleene star, and (reverse) k-star.

We construct a 1-way non-deterministic transducer (1NFT) $\mathcal{P}_h$ which computes the parsing relation P_h. We also construct a 2-way reversible transducer (2RFT) $\mathcal{T}_h$, called the *evaluator*, such that $[\![h]\!]^R = [\![\mathcal{T}_h]\!] \circ P_h$.

In this section, for a *rational* transducer expression h, both $\mathcal{P}_h$ and $\mathcal{T}_h$ will have a number of states linear in the size of h. We denote by $\|\mathcal{T}\|$ the number of states of a transducer $\mathcal{T}$, sometimes also called the size of $\mathcal{T}$. The evaluator $\mathcal{T}_h$ will actually be 1-way, i.e., it is a 1RFT.

Parsing and Evaluation

The parser of an expression h will not try to check that a given input word w can be unambiguously parsed according to h. Instead, it will compute the set $P_h(w)$ of all possible ways to parse w w.r.t. h. Hence, we define a parsing relation P_h.

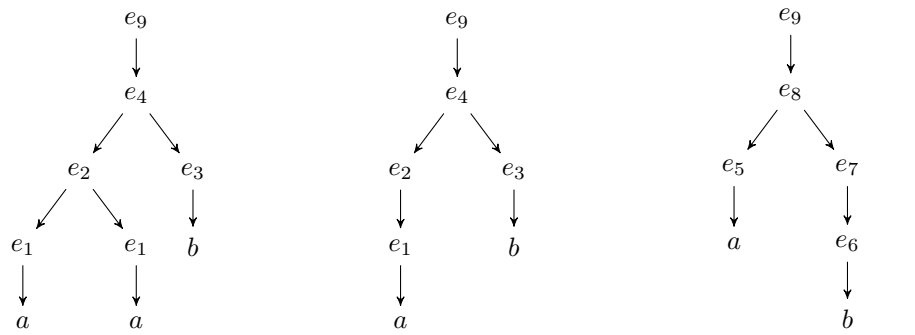
We start with an example on a classical regular expression $e = a^\star \cdot b + a \cdot b^\star$. For each occurrence of a subexpression e_i, we introduce a pair of parentheses $(_i \;)_i$ which is used to bracket the factor of the input word matching e_i. The above expression e has 9 occurrences of subexpressions: $e_1 = a$, $e_2 = e_1^\star$, $e_3 = b$, $e_4 = e_2 \cdot e_3$, $e_5 = a$, $e_6 = b$, $e_7 = e_6^\star$, $e_8 = e_5 \cdot e_7$ and $e_9 = e_4 + e_8 = e$. Hence, we use 9 pairs of parentheses $(_i \;)_i$ for $1 \leqslant i \leqslant 9$. The input word aab can be unambiguously parsed according to e, whereas the input word ab admits two parsings:

$$P_e(aab) = \{(_9\,(_4\,(_2\,(_1\,a\,)_1)(_1\,a\,)_1\,)_2)(_3\,b\,)_3\,)_4\,)_9\}$$
$$P_e(ab) = \{(_9\,(_4\,(_2\,(_1\,a\,)_1\,)_2)(_3\,b\,)_3\,)_4\,)_9\;\;,\;(_9\,(_8\,(_5\,a\,)_5)(_7\,(_6\,b\,)_6\,)_7\,)_8\,)_9\}$$

Observe that the parsing of a word w.r.t. an expression e can be viewed as the traversal of the parse tree of w w.r.t. e. For instance, in Figure 3 we depict the unique parse tree of the word aab w.r.t. e given above. We also give the two parse trees of the word ab w.r.t. e.

Below, we define inductively the parsing relation P_h for a rational transducer expression h. As in the examples above, when $\alpha \in P_h(w)$ with $w \in \Sigma^*$, then the projection of α on $\Sigma^\star$ is w. We simultaneously define the 1NFT parser $\mathcal{P}_h$ which implements P_h and the 2RFT evaluator $\mathcal{T}_h$. The *Parser* $\mathcal{P}_h$ is a 1NFT satisfying the following invariants:

1. $\mathcal{P}_h$ has a unique initial state q_0^h, which has no incoming transitions,
2. $\mathcal{P}_h$ has a unique final state q_F^h, which has no outgoing transitions,



(a) The unique parse tree of aab **(b)** A parse tree for ab **(c)** Another parse tree for ab

Figure 3 Parse trees for aab and ab w.r.t. $e = a^\star \cdot b + a \cdot b^\star$

3. a transition of $\mathcal{P}_h$ either reads a visible letter $a \neq \varepsilon$ and outputs a, or it reads ε and outputs $(_f$ or $_f)$, where f is some subexpression of h.

The *Evaluator* $\mathcal{T}_h$ is a reversible transducer that computes h when composed with $P_h(w)$. It will be of the form given in Figure 4, where $(_h \ \ _h)$ is the pair of parentheses associated with (this occurrence of the transducer expression) h. It always starts on the left of its input, and ends on the right. This is trivial in this section as $\mathcal{T}_h$ is one-way, but is a useful invariant in the following sections. Note that an output denoted $-$ on a transition stands for any word $v \in \Gamma^\star$. In all our figures, we will often use $\mathcal{P}_f$ and $\mathcal{T}_f$ for f a subexpression of h. Everytime, we separate the initial and final states of these machines from the main part which will be represented by a rectangle. This choice allows us to highlight the particular role of these states which are the unique entry and exit points. Nevertheless, they are still states of $\mathcal{P}_f$ and $\mathcal{T}_f$ when counting the number of states.

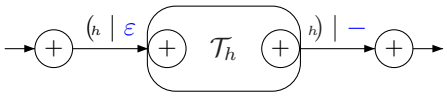


Figure 4 Format of the evaluator transducer

- Let $h = e \rhd v$ be a basic expression. The parsing relation is defined by $P_h(w) = \{(_h\, w\, _h) \mid w \in L(e)\}$. Here, P_h is actually functional and we have $\mathsf{dom}(P_h) = L(e) = \mathsf{fdom}(P_h)$. Notice that $L(e) = \mathsf{dom}(h) = \mathsf{udom}(h)$.
  Let $\mathcal{A}_e$ be the non-deterministic Glushkov automaton that recognizes $L(e)$. Recall that $\mathcal{A}_e$ has a unique initial state q_0^e with no incoming transitions. Also, the number of states of $\mathcal{A}_e$ is $1 + \mathsf{nl}(e)$ where $\mathsf{nl}(e)$ is the number of literals in the regular expression e which denotes the language $L(e)$, [12].
  Then, let $\mathcal{A}'_e$ be the transducer with $\mathcal{A}_e$ as the underlying input automaton and such that each transition simply copies the input letter to the output. The 1NFT $\mathcal{A}'_e$ realizes the *identity* function restricted to the domain $L(e)$. The parser $\mathcal{P}_h$ is then $\mathcal{A}'_e$ enriched with an initial and a final states q_0^h and q_F^h, and transitions $q_0^h \xrightarrow{\varepsilon|(_h} q_0^e$ and $q \xrightarrow{\varepsilon|_h)} q_F^h$ for $q \in F_{\mathcal{A}'_e}$, as given on Figure 5. The number of states in $\mathcal{P}_h$ is $\|\mathcal{P}_h\| = \mathsf{nl}(e) + 3 \leqslant |h|$.
  Notice that $\mathcal{P}_h$ is possibly non-deterministic due to the Glushkov automaton $\mathcal{A}_e$. But it is *functional* and realizes the parsing relation: $[\![\mathcal{P}_h]\!]^R = P_h$.
  The evaluator $\mathcal{T}_h$ for $h = e \rhd v$, as given in Figure 6, simply reads $(_h\, \Sigma^*\, _h)$ and outputs v while reading $_h)$, satisfying the format described in Figure 4.

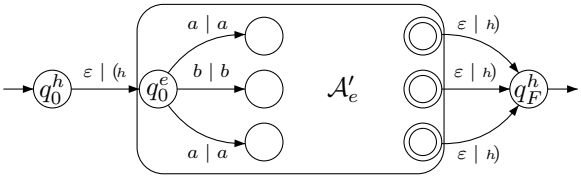


Figure 5 Parser for $h = e \rhd v$. The doubly circled states on the right are the accepting states of $\mathcal{A}_e$. Note that, if the initial state of $\mathcal{A}_e$ is also an accepting state, then there is a transition $q_0^e \xrightarrow{\varepsilon|h)} q_F^h$ in $\mathcal{P}_h$.

Remark that in this case the evaluator $\mathcal{T}_h$ is independent of e, since the filtering on the domain $L(e)$ is done by the parser $\mathcal{P}_h$ and not the evaluator $\mathcal{T}_h$. We have $\|\mathcal{T}_h\| = 3 \leqslant |h|$. Also, for all $w \in L(e)$ we have $P_h(w) = \{(_h\, w\,)_h\}$ and $[\![\mathcal{T}_h]\!]((_h\, w\,)_h) = v$. Therefore, $[\![h]\!] = [\![\mathcal{T}_h]\!] \circ P_h$.

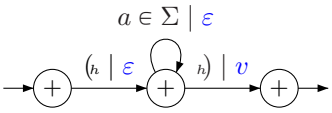


Figure 6 Evaluator for $h = e \rhd v$.

- Let $h = f + g$ and let $(_h\ \ )_h$ be the associated pair of parentheses. Then, for all $w \in \Sigma^*$, $P_h(w) = \{(_h\, \alpha\,)_h \mid \alpha \in P_f(w) \cup P_g(w)\}$. We have $\mathsf{dom}(P_h) = \mathsf{dom}(P_f) \cup \mathsf{dom}(P_g) = \mathsf{dom}(f) \cup \mathsf{dom}(g) = \mathsf{dom}(h)$. Notice that here the parsing relation is not functional when $\mathsf{dom}(f) \cap \mathsf{dom}(g) \neq \varnothing$.
  The parser $\mathcal{P}_h$ is as depicted in Figure 7 where the three pink states are merged and similarly the three blue states are merged so that the number of states of $\mathcal{P}_h$ is $\|\mathcal{P}_h\| = \|\mathcal{P}_f\| + \|\mathcal{P}_g\| \leqslant |f| + |g| < |h|$. Clearly, $[\![\mathcal{P}_h]\!]^R = P_h$.

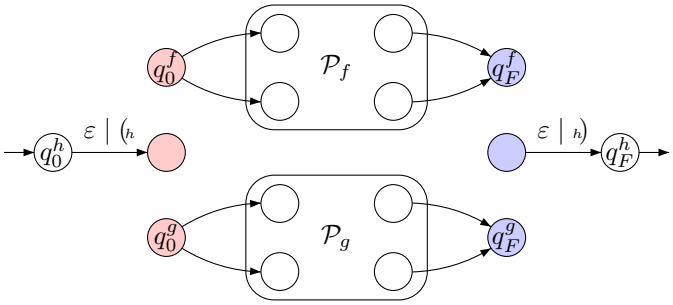


Figure 7 Parser for $h = f + g$.

The evaluator $\mathcal{T}_h$ for $h = f + g$ is as given in Figure 8. The initial states of $\mathcal{T}_f$ and $\mathcal{T}_g$ have been merged in the pink state, similarly the final states of $\mathcal{T}_f$ and $\mathcal{T}_g$ have been merged in the blue state. $\mathcal{T}_h$ first reads $(_h$ and goes to a common initial state of $\mathcal{T}_f$ and $\mathcal{T}_g$, then goes to the corresponding evaluator depending on whether it reads $(_f$ or $(_g$. When reading $)_f$ or $)_g$ it goes to a common state instead of their final one, where it go to the unique final state by $)_h$ and producing ε. We have $\|\mathcal{T}_h\| = \|\mathcal{T}_f\| + \|\mathcal{T}_g\| \leqslant |f| + |g| < |h|$.

- Let $h = f \cdot g$ and let $(_h\ \ )_h$ be the associated pair of parentheses. Then, for all $w \in \Sigma^*$, $P_h(w) = \{(_h\, \alpha\, \beta\,)_h \mid w = uv, \alpha \in P_f(u), \beta \in P_g(v)\}$. We have $\mathsf{dom}(P_h) = \mathsf{dom}(P_f) \cdot$

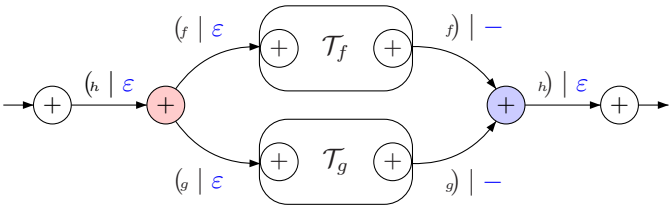


Figure 8 Evaluator for $h = f + g$.

$\mathsf{dom}(P_g) = \mathsf{dom}(f) \cdot \mathsf{dom}(g) = \mathsf{dom}(h)$. Notice again that if the product of languages $\mathsf{dom}(f) \cdot \mathsf{dom}(g)$ is ambiguous, then P_h is not functional.

The parser $\mathcal{P}_h$ is given in Figure 9 where the two pink states are merged so that the number of states of $\mathcal{P}_h$ is $\|\mathcal{P}_h\| = 1 + \|\mathcal{P}_f\| + \|\mathcal{P}_g\| \leqslant 1 + |f| + |g| = |h|$. We have, $[\![\mathcal{P}_h]\!]^R = P_h$.

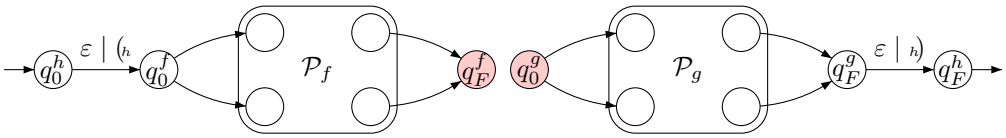


Figure 9 Parser for $h = f \cdot g$.

The evaluator $\mathcal{T}_h$ for $h = f \cdot g$ is as given in Figure 10. The final state of $\mathcal{T}_f$ is merged with the initial state of $\mathcal{T}_g$. We have $\|\mathcal{T}_h\| = 1 + \|\mathcal{T}_f\| + \|\mathcal{T}_g\| \leqslant 1 + |f| + |g| = |h|$.

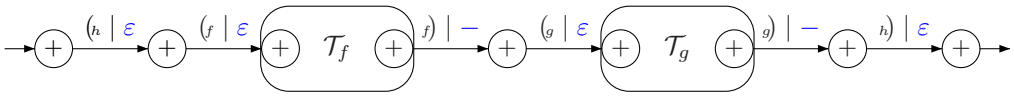


Figure 10 Evaluator for $h = f \cdot g$.

Let $h = f^\star$ and let $(_h \ \ _h)$ be the associated pair of parentheses. Then, for all $w \in \Sigma^*$, $P_h(w) = \{(_h \alpha_1 \cdots \alpha_n \,_h) \mid w = u_1 \cdots u_n \text{ and } \alpha_i \in P_f(u_i) \text{ for all } 1 \leqslant i \leqslant n\}$. We have $\mathsf{dom}(P_h) = \mathsf{dom}(P_f)^\star = \mathsf{dom}(f)^\star = \mathsf{dom}(h)$. As above, if the Kleene star of the language $\mathsf{dom}(f)$ is ambiguous, then P_h is not functional.

The parser $\mathcal{P}_h$ is as given in Figure 11 where the three pink states are merged so that the number of states of $\mathcal{P}_h$ is $\|\mathcal{P}_h\| = 1 + \|\mathcal{P}_f\| \leqslant 1 + |f| = |h|$. It is easy to see that $[\![\mathcal{P}_h]\!]^R = P_h$.

The evaluator $\mathcal{T}_h$ for $h = f^\star$ is as given in Figure 12 where the three pink states are merged so that the number of states of $\mathcal{T}_h$ is $\|\mathcal{T}_h\| = 1 + \|\mathcal{T}_f\| \leqslant 1 + |f| = |h|$.

▸ **Lemma 6.** *Theorem 4 holds for any rational transducer expression h. Moreover, in this case we have $\|\mathcal{T}_h\| \leqslant |h|$.*

Proof. We have already argued during the construction that $\mathsf{dom}(P_h) = \mathsf{dom}(h)$, $[\![\mathcal{P}_h]\!]^R = P_h$, $\|\mathcal{P}_h\| \leqslant |h|$ and $\|\mathcal{T}_h\| \leqslant |h|$. It is also easy to see that the projection on Σ of a parsed word $\alpha \in P_h(w)$ is w itself. The remaining claims are proved by structural induction.

For the base case $h = e \rhd v$ we have already seen that P_h is functional with domain $L(e) = \mathsf{udom}(h)$ and that $[\![h]\!]^R(w) = \{v\} = [\![\mathcal{T}_h]\!](P_h(w))$ for all $w \in L(e)$.

For the induction, we prove in details the case of Cauchy product. The other cases of sum and Kleene star can be proved similarly. Let $h = f \cdot g$. We first show that $[\![h]\!]^R = [\![\mathcal{T}_h]\!] \circ P_h$. Let $w \in \mathsf{dom}(h)$.

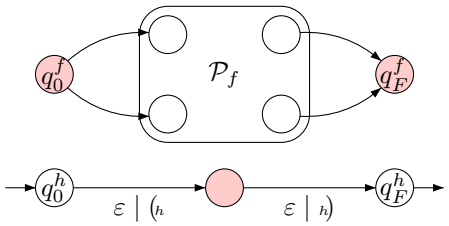


Figure 11 Parser for $h = f^\star$.

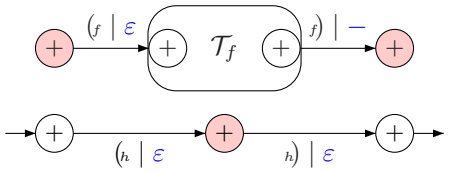


Figure 12 Evaluator for $h = f^\star$.

Let $w' \in [\![h]\!]^R(w)$. There is a factorization $w = uv$ and $u' \in [\![f]\!]^R(u)$, $v' \in [\![g]\!]^R(v)$ with $w' = u'v'$. By induction, $[\![f]\!]^R = [\![\mathcal{T}_f]\!] \circ P_f$ so we find $\alpha \in P_f(u)$ with $u' = [\![\mathcal{T}_f]\!](\alpha)$. Similarly, we find $\beta \in P_g(v)$ with $v' = [\![\mathcal{T}_g]\!](\beta)$. Let $\gamma = (_h\, \alpha\, \beta\, {}_h) \in P_h(w)$. We have $[\![\mathcal{T}_h]\!](\gamma) = [\![\mathcal{T}_f]\!](\alpha)[\![\mathcal{T}_g]\!](\beta) = u'v' = w'$. Hence, $w' \in ([\![\mathcal{T}_h]\!] \circ P_h)(w)$.

Conversely, let $w' \in ([\![\mathcal{T}_h]\!] \circ P_h)(w)$ and consider $\gamma \in P_h(w)$ such that $w' = [\![\mathcal{T}_h]\!](\gamma)$. There is a factorization $w = uv$ and $\alpha \in P_f(u)$, $\beta \in P_g(v)$ with $\gamma = (_h\, \alpha\, \beta\, {}_h)$. From the definition of the evaluator $\mathcal{T}_h$, using the form of parsed words $\alpha = (_f\, \alpha'\, {}_f)$ and $\beta = (_g\, \beta'\, {}_g)$, we deduce that $[\![\mathcal{T}_h]\!](\gamma) = [\![\mathcal{T}_f]\!](\alpha)[\![\mathcal{T}_g]\!](\beta) \in ([\![\mathcal{T}_f]\!] \circ P_f)(u) \cdot ([\![\mathcal{T}_g]\!] \circ P_g)(v) = [\![f]\!]^R(u) \cdot [\![g]\!]^R(v) \subseteq [\![h]\!]^R(w)$.

It remains to prove that $\mathsf{udom}(h) = \mathsf{fdom}(P_h)$. Recall that

$$\mathsf{udom}(h) = (\mathsf{udom}(f) \cdot \mathsf{udom}(g)) \backslash \{uvw \mid v \neq \varepsilon \text{ and } u, uv \in \mathsf{dom}(f) \text{ and } vw, w \in \mathsf{dom}(g)\}\,.$$

Let $w \in \mathsf{udom}(h)$. Then, there is a unique factorization $w = uv$ with $u \in \mathsf{dom}(f)$ and $v \in \mathsf{dom}(g)$, i.e., such that $P_f(u) \neq \varnothing$ and $P_g(v) \neq \varnothing$. We deduce that $P_h(w) = \{(_h\, \alpha\, \beta\, {}_h) \mid \alpha \in P_f(u), \beta \in P_g(v)\}$. But we also have $u \in \mathsf{udom}(f) = \mathsf{fdom}(P_f)$ and $v \in \mathsf{udom}(g) = \mathsf{fdom}(P_g)$. We deduce that $P_h(w)$ is a singleton.

Conversely, let $w \in \mathsf{fdom}(P_h)$. Assume that $w = uv = u'v'$ with $u, u' \in \mathsf{dom}(f)$ and $v, v' \in \mathsf{dom}(g)$. Let $\gamma = (_h\, (_f\, \alpha\, {}_f)(_g\, \beta\, {}_g)\, {}_h)$ with $(_f\, \alpha\, {}_f) \in P_f(u)$ and $(_g\, \beta\, {}_g) \in P_g(v)$. Similarly, let $\gamma' = (_h\, (_f\, \alpha'\, {}_f)(_g\, \beta'\, {}_g)\, {}_h)$ with $(_f\, \alpha'\, {}_f) \in P_f(u')$ and $(_g\, \beta'\, {}_g) \in P_g(v')$. We have $\gamma, \gamma' \in P_h(w)$, hence $\gamma = \gamma'$. Therefore also $\alpha = \alpha'$. The projection on Σ of α (resp. α') is u (resp. u') and we deduce that $u = u'$ and $v = v'$. Therefore, w has a unique factorization $w = uv$ with $u \in \mathsf{dom}(f)$ and $v \in \mathsf{dom}(g)$. It follows that $P_h(w) = \{(_h\, \alpha\, \beta\, {}_h) \mid \alpha \in P_f(u), \beta \in P_g(v)\}$. Since $P_h(w)$ is a singleton, we deduce that both $P_f(u)$ and $P_g(v)$ are singletons. By induction, we get $u \in \mathsf{udom}(f)$ and $v \in \mathsf{udom}(g)$. Finally, we have proved $w \in \mathsf{udom}(h)$. ◂

Consider rational transducer expressions where basic expressions are simply of the form $a \rhd v$ with $a \in \Sigma$ and $v \in \Gamma^*$ (instead of the more general $e \rhd v$). We can directly construct from such an expresion h, a transducer $\mathcal{A}_h$ which realizes the relational semantics of h. This folklore and rather simple construction gives a 1NFT of size linear in $|h|$. Notice also that the unambiguous domain of the expression h coincide with the unambiguous domain of the 1NFT $\mathcal{A}_h$. Therefore, we can even restrict to $\mathsf{udom}(h)$ and implement the unambiguous semantics by techniques similar to the constructions in Section 8.

But, since the output semiring $(2^{\Gamma^\star}, \cup, \cdot, \varnothing, \{\varepsilon\})$ is not commutative, we cannot extend this approach and construct a 1NFT for the relational semantics of regular transducer expressions using Hadamard product, reverse Cauchy product, reverse Kleene star, k-star or reverse k-star. Therefore, we decided to use in Section 4 an approach which can be extended to the two-way operators as explained in the remaining sections.

5 Rational Functions with Reverse Products

Before turning to the Hadamard product and the (reverse) k-star operators, we first focus on a set of operators which, despite expressing non-rational functions, still enjoy the linear complexity of the previous section. Intuitively, it is the set of operators that only require to process the input once. It consists of the reverse Cauchy product and the reverse Kleene star. We see at the end of this section that we may also add duplicate and reverse functions without changing the linear complexity. For each of these expressions h, we define inductively the parsing relation P_h, the parser $\mathcal{P}_h$ and the evaluator $\mathcal{T}_h$. We will show that $\|\mathcal{P}_h\| \leqslant |h|$ and that $\|\mathcal{T}_h\| \leqslant 5|h|$.

Reverse Cauchy product

For $h = f \cdot_r g$, the parsing P_h is exactly the same as that for the expression $h = f \cdot g$. As a consequence, the parser $\mathcal{P}_h$ for $h = f \cdot_r g$ is as given in Figure 9. Recall that the number of states of $\mathcal{P}_h$ is $\|\mathcal{P}_h\| = \|\mathcal{P}_f\| + \|\mathcal{P}_g\| + 1 \leqslant |f| + |g| + 1 \leqslant |h|$.

The evaluator $\mathcal{T}_h$ for $h = f \cdot_r g$ is as given in Figure 13. Notice that $\mathcal{T}_h$ is a 2RFT and its number of states is $\|\mathcal{T}_h\| = \|\mathcal{T}_f\| + \|\mathcal{T}_g\| + 3 \leqslant 5|f| + 5|g| + 3 \leqslant 5|h|$.

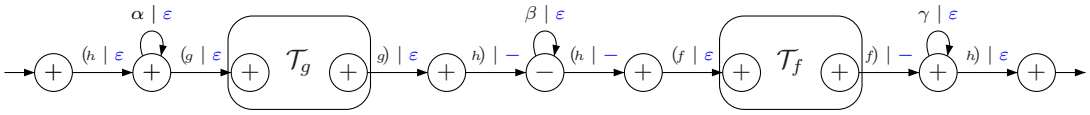


Figure 13 Evaluator for $h = f \cdot_r g$, with α any letter different from $(_h, (_g$, β any letter different from $(_h, {}_h)$, and γ any letter different from ${}_f), {}_h)$.

Reverse Kleene star

For $h = f_r^\star$, the parsing P_h is exactly the same as that for the expression $h = f^\star$. Therefore, the parser $\mathcal{P}_h$ for $h = f_r^\star$ is as given in Figure 11. Recall that the number of states of $\mathcal{P}_h$, $\|\mathcal{P}_h\| = \|\mathcal{P}_f\| + 1 \leqslant |f| + 1 = |h|$.

The evaluator $\mathcal{T}_h$ is depicted in Figure 14. Note that $\mathcal{T}_h$ is a 2RFT and its number of states is $\|\mathcal{T}_h\| = \|\mathcal{T}_f\| + 5 \leqslant 5|f| + 5 = 5|h|$.

Duplicate and reverse

The function rev simply reverses its input: for $w = a_1 \cdots a_n$ with $a_i \in \Sigma$ we have $[\![\mathsf{rev}]\!](w) = a_n \cdots a_1$. We can express it with a reverse Kleene star: $\mathsf{rev} = (\mathsf{copy})_r^\star$ where $\mathsf{copy} = \sum_{a \in \Sigma} a \rhd a$ simply copies an input letter. Using constructions above, we obtain a parser and an evaluator of size $\mathcal{O}(|\Sigma|)$. We construct below even simpler parser and evaluator for rev.

The duplicate function $\mathsf{dup}_\#$ is parametrized by a separator symbol $\#$, its semantics is given for $w \in \Sigma^*$ by $[\![\mathsf{dup}_\#]\!](w) = w\#w$. The function $\mathsf{dup}_\#$ has to read its input twice and cannot be expressed with the combinator considered so far. It may be expressed with the

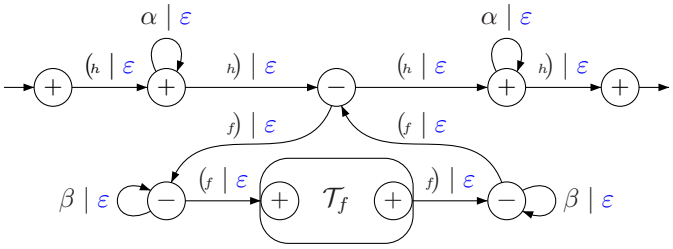


Figure 14 Evaluator for $h = f_r^\star$, with α any letter different from $(_h$, $_h)$, and β any letter different from $(_f$, $_f)$.

Hadamard product as follows: $\mathsf{dup}_\# = (\mathsf{copy} \cdot (\varepsilon \rhd \#)) \odot \mathsf{copy}$. But in general, when we allow Hadamard products, the size of the one-way parser is no more linear in the size of the given expression. Hence, we give below direct constructions with better complexity.

For $h = \mathsf{dup}_\#$ or $h = \mathsf{rev}$, let $(_h \ \ _h)$ be the associated pair of parentheses. Since they are basic *total* functions, the parsing is defined as $P_h(w) = \{(_h w_h) \mid w \in \Sigma^\star\}$, and the parser $\mathcal{P}_h$ is as given in Figure 15. We have $\|\mathcal{P}_h\| = 3$. We define the size $|\mathsf{dup}_\#| = 3 = |\mathsf{rev}|$ in order to get $\|\mathcal{P}_h\| \leqslant |h|$ for these basic functions as well. The evaluator $\mathcal{T}_h$ is given in Figure 16 when $h = \mathsf{dup}_\#$ and in Figure 17 when $h = \mathsf{rev}$. In both cases, we have $\|\mathcal{T}_h\| = 5 \leqslant 3|h|$.

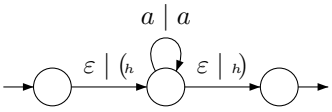


Figure 15 Parser for duplicate and reverse functions, with $a \in \Sigma$.

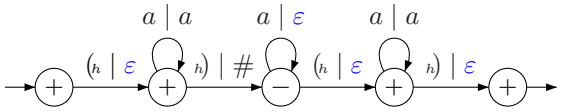


Figure 16 Evaluator for $h = \mathsf{dup}_\#$, with $a \in \Sigma$.

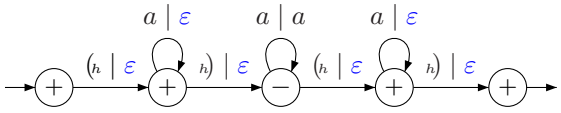


Figure 17 Evaluator for $h = \mathsf{rev}$

To conclude this section, let us remark that, having rational functions, duplicate, reverse as well as the composition operator gives the expressive power of the full RTEs.

6 Hadamard product

In Section 4 and Section 5, we consider rational functions with sum, Cauchy product and Kleene star as well as rational-reverse functions. In this section, we extend this fragment with the Hadamard product.

Just as for the rational functions, we first motivate the parsing of a word w.r.t. a Hadamard product expression e as the traversal of the parse *dag* of w w.r.t. e. As an example, consider

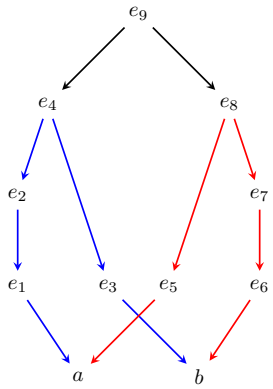


Figure 18 Parse dag for ab w.r.t. the Hadamard product $f \odot g$, where $\mathsf{dom}(f) = a^*b$ and $\mathsf{dom}(g) = ab^*$.

$e = (a^\star \cdot b) \odot (a \cdot b^\star)$. For each occurrence of a subexpression e_i, we introduce a pair of parentheses $(_i \ )_i$ which is used to bracket the factor of the input word matching e_i. The above expression e has 9 occurrences of subexpressions: $e_1 = a$, $e_2 = e_1^\star$, $e_3 = b$, $e_4 = e_2 \cdot e_3$, $e_5 = a$, $e_6 = b$, $e_7 = e_6^\star$, $e_8 = e_5 \cdot e_7$ and $e_9 = e_4 \odot e_8 = e$. Hence, we use 9 pairs of parentheses $(_i \ )_i$ for $1 \leqslant i \leqslant 9$. The input word ab can be unambiguously parsed according to e as follows

$$P_e(ab) = \{(_9(_4(_2(_1(_8(_5a_1)_2)_3(_5)_7(_6b_3)_4)_6)_7)_8)_9\}\,.$$

In Figure 18 we depict the unique parse dag of the word ab w.r.t. e given above. Consider a traversal of this parse dag with two heads. The first head carries out the traversal according to the parse tree of e_4 given in blue, while the second head carries out the traversal according to the parse tree of e_8 given in red. We also fix an ordering on the movement of the two heads as follows. Between the points of the traversal which reach a leaf, the first head carries out its traversal, and then the second head carries out its traversal. This means that from the root, the first head moves first, and continues its traversal until it reaches a leaf of the parse tree, after which the second head starts its traversal and continues until it reaches the same leaf. Then, the first head continues its traversal until the next leaf is reached, and then the second head continues its traversal until seeing the same leaf, and so on. Then, the parsing of w w.r.t. e can be viewed as such a traversal of the parse tree of w w.r.t. e.

Parsing relation for $h = f \odot g$.

For $w \in \Sigma^*$ we let $P_h(w)$ be the set of words $(_h \, \alpha \,)_h$ such that

1. $\pi_{\text{-}g}(\alpha) \in P_f(w)$ and $\pi_{\text{-}f}(\alpha) \in P_g(w)$. Here, $\pi_{\text{-}g}(\alpha)$ denotes the projection which erases parentheses which are indexed by subexpressions of g. $\pi_{\text{-}f}(\alpha)$ is defined similarly.
2. α does not contain a parenthesis indexed by a subexpression of g immediately followed by a parenthesis indexed by a subexpression of f.

Observe that there are several ways to satisfy the first condition above, even when we fix the projections $\pi_{\text{-}g}(\alpha)$ and $\pi_{\text{-}f}(\alpha)$. This is because parentheses indexed with subexpressions of f can be shuffled arbitrarily with parentheses indexed with subexpressions of g. This would break the equality $\mathsf{udom}(h) = \mathsf{fdom}(P_h)$. Hence, we fix a specific order by giving priority to parentheses w.r.t. first argument.

▸ **Lemma 7.** *Let h be an RTE not using k-star or reverse k-star. We have $\mathsf{dom}(P_h) = \mathsf{dom}(h)$ and $\mathsf{fdom}(P_h) = \mathsf{udom}(h)$.*

Proof. The proof is by structural induction. The only new case is when $h = f \odot g$ is an Hadamard product. Using the induction hypothesis and the definitions, it is sufficient to prove that $\mathsf{dom}(P_h) = \mathsf{dom}(P_f) \cap \mathsf{dom}(P_g)$ and $\mathsf{fdom}(P_h) = \mathsf{fdom}(P_f) \cap \mathsf{fdom}(P_g)$.

Let $w \in \mathsf{dom}(P_f) \cap \mathsf{dom}(P_g)$. We find $\alpha_1 \in P_f(w)$ and $\alpha_2 \in P_g(w)$. Let α be the (unique) word satisfying $\pi_{\text{-}g}(\alpha) = \alpha_1$, $\pi_{\text{-}f}(\alpha) = \alpha_2$ and condition 2 on the order of parentheses in P_h. We get $(_h\, \alpha\,)_h \in P_h(w) \neq \varnothing$, hence we have $w \in \mathsf{dom}(P_h)$. The converse inclusion $\mathsf{dom}(P_h) \subseteq \mathsf{dom}(P_f) \cap \mathsf{dom}(P_g)$ is even easier to show.

Let $w \in \mathsf{fdom}(P_f) \cap \mathsf{fdom}(P_g)$. It is easy to see that there is a unique α satisfying conditions 1 and 2 of the definition of $P_h(w)$. Therefore, $w \in \mathsf{fdom}(P_h)$. Conversely, let $w \in \mathsf{fdom}(P_h)$ and assume that $w \notin \mathsf{fdom}(P_f) \cap \mathsf{fdom}(P_g)$. For instance, we have $w \in (\mathsf{dom}(P_f) \setminus \mathsf{fdom}(P_f)) \cap \mathsf{dom}(P_g)$. Then, we find $\alpha_1, \alpha_1' \in P_f(w)$ with $\alpha_1 \neq \alpha_1'$ and $\alpha_2 \in P_g(w)$. There is a unique α (resp. α') with $\pi_{\text{-}g}(\alpha) = \alpha_1$ (resp. $\pi_{\text{-}g}(\alpha') = \alpha_1'$), $\pi_{\text{-}f}(\alpha) = \alpha_2$ and condition 2 on the order of parentheses in P_h. We get $\alpha \neq \alpha'$ and $(_h\, \alpha\,)_h, (_h\, \alpha'\,)_h \in P_h(w)$, which contradicts $w \in \mathsf{fdom}(P_h)$. ◂

Evaluator for $h = f \odot g$

Let $h = f \odot g$ and let $(_h\ \ )_h$ be the associated pair of parentheses. Then, recall that for all $w \in \Sigma^*$, $P_h(w) \subseteq \{(_h\, \alpha\,)_h \mid \pi_{\text{-}g}(\alpha) \in P_f(w) \text{ and } \pi_{\text{-}f}(\alpha) \in P_g(w)\}$. From the 2RFT $\mathcal{T}_f$, we construct the 2RFT $\mathcal{T}_f^{+g}$ by adding self-loops to all states labelled with all parentheses associated with subexpressions of g, the output of these new transitions is ε. When $\pi_{\text{-}g}(\alpha) \in P_f(w)$ then the 2RFT $\mathcal{T}_f^{+g}$ behaves on α as $\mathcal{T}_f$ would on $\pi_{\text{-}g}(\alpha)$. Similarly, we construct $\mathcal{T}_g^{+f}$ from $\mathcal{T}_g$. Finally, the evaluator $\mathcal{T}_h$ is depicted in Figure 19. Recall the generic form of an evaluator given in Figure 4 where we decided to draw the initial state and the final state outside the box to underline the fact that there are no transitions going to the initial state and no transitions starting from the final state. The number of states of $\mathcal{T}_h$ is therefore $\|\mathcal{T}_h\| = \|\mathcal{T}_f\| + \|\mathcal{T}_g\| + 3$.

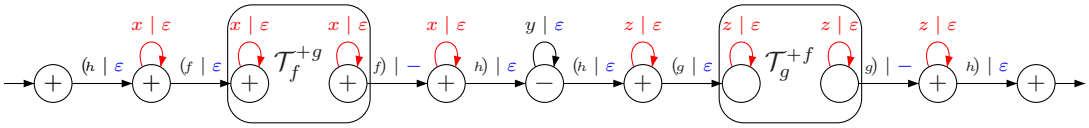


Figure 19 Evaluator for $h = f \odot g$ where x (resp. z) is any parenthesis from a subexpression of g (resp. f) and y is any letter different from $(_h\, ,)_h$.

▸ **Lemma 8.** *Let h be an RTE not using k-star or reverse k-star. The translator $\mathcal{T}_h$ composed with the parsing relation P_h implements the relational semantics: $[\![h]\!]^R = [\![\mathcal{T}_h]\!] \circ P_h$. Moreover, the number of states of the translator is $\|\mathcal{T}_h\| \leqslant 5|h|$.*

Proof. The proof is by structural induction. We have already seen that the statements hold when h does not use a Hadamard product. The only new case is when $h = f \odot g$ is a Hadamard product. We have $\|\mathcal{T}_h\| = \|\mathcal{T}_f\| + \|\mathcal{T}_g\| + 3 \leqslant 5(|f| + |g| + 1) = 5|h|$. We turn to the proof of $[\![h]\!]^R = [\![\mathcal{T}_h]\!] \circ P_h$. Let $w \in \mathsf{dom}(h) = \mathsf{dom}(f) \cap \mathsf{dom}(g)$.

Let $w' \in [\![h]\!]^R(w) = [\![f]\!]^R(w) \cdot [\![g]\!]^R(w)$. We write $w' = w_1 w_2$ with $w_1 \in [\![f]\!]^R(w)$ and $w_2 \in [\![g]\!]^R(w)$. Since $[\![f]\!]^R = [\![\mathcal{T}_f]\!] \circ P_f$, we find $\alpha_1 \in P_f(w)$ such that $[\![\mathcal{T}_f]\!](\alpha_1) = w_1$. Similarly, we find $\alpha_2 \in P_g(w)$ such that $[\![\mathcal{T}_g]\!](\alpha_2) = w_2$. Let α be the unique word satisfying $\pi_{\text{-}g}(\alpha) = \alpha_1$, $\pi_{\text{-}f}(\alpha) = \alpha_2$ and condition 2 on the order of parentheses in P_h. We have $(_h\, \alpha\,)_h \in P_h(w)$. It is easy to check that

$$[\![\mathcal{T}_h]\!]((_h\, \alpha\,)_h) = [\![\mathcal{T}_f^{+g}]\!](\alpha) \cdot [\![\mathcal{T}_g^{+f}]\!](\alpha) = [\![\mathcal{T}_f]\!](\alpha_1) \cdot [\![\mathcal{T}_g]\!](\alpha_2) = w_1 w_2 = w\,.$$

Conversely, let $(_h\,\alpha\,_h) \in P_h(w)$. We have

$$\begin{aligned}[\![\mathcal{T}_h]\!]((_h\,\alpha\,_h)) &= [\![\mathcal{T}_f^{+g}]\!](\alpha) \cdot [\![\mathcal{T}_g^{+f}]\!](\alpha) = [\![\mathcal{T}_f]\!](\pi_{\text{-}g}(\alpha)) \cdot [\![\mathcal{T}_g]\!](\pi_{\text{-}f}(\alpha)) \\ &\in ([\![\mathcal{T}_f]\!] \circ P_f)(w) \cdot ([\![\mathcal{T}_g]\!] \circ P_g)(w) = [\![f]\!]^R(w) \cdot [\![g]\!]^R(w) = [\![h]\!]^R(w)\,. \end{aligned}$$

◂

One-way parser for $h = f \odot g$

For the Hadamard product $h = f \odot g$, we carry out the parsing of f and g in parallel by shuffling the parentheses, giving priority to the left argument.

The 1-way parser $\mathcal{P}_h$ is depicted in Figure 20 where $\mathcal{P}_f \otimes \mathcal{P}_g$ is a product defined below of the parsers for expressions f and g. The number of states of $\mathcal{P}_h$ is given by $\|\mathcal{P}_h\| = 2 + 2 \times \|\mathcal{P}_f\| \times \|\mathcal{P}_g\|$.

Let $\mathcal{P}_f = (Q_f, \Sigma, B, q_I^f, q_F^f, \Delta_f, \mu_f)$ and $\mathcal{P}_g = (Q_g, \Sigma, C, q_I^g, q_F^g, \Delta_g, \mu_g)$ be the 1-way parsers for expressions f and g, respectively.

The set of states of $\mathcal{P}_f \otimes \mathcal{P}_g$ is $Q_f \times Q_g \times \{0,1\}$. The input alphabet is Σ and the output alphabet is $B \cup C$. The initial state is $(q_I^f, q_I^g, 0)$ and the accepting state is $(q_F^f, q_F^g, 1)$. The transition function of $\mathcal{P}_f \otimes \mathcal{P}_g$ is defined as follows, with $(s,t) \in Q_f \times Q_g$ and $\nu \in \{0,1\}$:

- if $s \xrightarrow{\varepsilon|x} s'$ in $\mathcal{P}_f$, then $(s,t,0) \xrightarrow{\varepsilon|x} (s',t,0)$ in $\mathcal{P}_f \otimes \mathcal{P}_g$,
- if $t \xrightarrow{\varepsilon|x} t'$ in $\mathcal{P}_g$, then $(s,t,\nu) \xrightarrow{\varepsilon|x} (s,t',1)$ in $\mathcal{P}_f \otimes \mathcal{P}_g$,
- if $s \xrightarrow{a|a} s'$ in $\mathcal{P}_f$ and $t \xrightarrow{a|a} t'$ in $\mathcal{P}_g$, then $(s,t,\nu) \xrightarrow{a|a} (s',t',0)$ in $\mathcal{P}_f \otimes \mathcal{P}_g$.

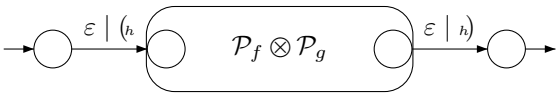


Figure 20 Parser for $h = f \odot g$

▸ **Lemma 9.** *Let h be an RTE not using k-star or reverse k-star. The parser $\mathcal{P}_h$ computes the parsing relation P_h. Moreover, the number of states of the parser is $\|\mathcal{P}_h\| \leqslant |h|^{\mathsf{width}(h)}$.*

Proof. We first prove that $[\![\mathcal{P}_h]\!]^R = P_h$ by structural induction. The only new case is when $h = f \odot g$ is a Hadamard product. Let $w \in \Sigma^\star$.

We first show that $P_h(w) \subseteq [\![\mathcal{P}_h]\!]^R(w)$. Let $(_h\,\alpha\,_h) \in P_h(w)$. We have to show that α is accepted by $\mathcal{P}_f \otimes \mathcal{P}_g$. Consider an accepting run ϱ_f of $\mathcal{P}_f$ (resp. ϱ_g of $\mathcal{P}_g$) reading the input word w and producing the projection $\pi_{\text{-}g}(\alpha) \in P_f(w)$ (resp. $\pi_{\text{-}f}(\alpha) \in P_g(w)$). We construct an accepting run ϱ of $\mathcal{P}_f \otimes \mathcal{P}_g$ reading w and producing α by shuffling ϱ_f and ϱ_g. The transitions reading an input letter $a \in \Sigma$ are synchronized and between two such synchronized transitions we execute first the epsilon moves of $\mathcal{P}_f$ (the extra bit of the state being 0) and then the epsilon moves of $\mathcal{P}_g$ (extra bit being 1). Notice that α starts with $(_f$ hence ϱ starts from the initial state $(q_I^f, q_I^g, 0)$ and α ends with $_g)$ so ϱ ends in the final state $(q_F^f, q_F^g, 1)$.

Conversely, we show that $[\![\mathcal{P}_h]\!]^R(w) \subseteq P_h(w)$. Let $(_h\,\alpha\,_h) \in [\![\mathcal{P}_h]\!]^R(w)$. There is an accepting run ϱ of $\mathcal{P}_f \otimes \mathcal{P}_g$ reading w and producing α. Let ϱ_f be the projection on the first component of the run ϱ after removing transitions of the form $(s,t,\nu) \xrightarrow{\varepsilon|x} (s,t',1)$ coming from transitions of $\mathcal{P}_g$. It is easy to see that ϱ_f is an accepting run of $\mathcal{P}_f$ reading w and producing the projection $\pi_{\text{-}g}(\alpha)$. Therefore, $\pi_{\text{-}g}(\alpha) \in P_f(w)$. Similarly, the projection on the second component of ϱ after removing transitions of the form $(s,t,0) \xrightarrow{\varepsilon|x} (s',t,0)$ is an accepting run ϱ_g of $\mathcal{P}_g$ reading w and producing the projection $\pi_{\text{-}f}(\alpha)$. We get $\pi_{\text{-}f}(\alpha) \in P_g(w)$. Finally, the definition of $\mathcal{P}_f \otimes \mathcal{P}_g$ ensures that α does not contain a parenthesis indexed by a

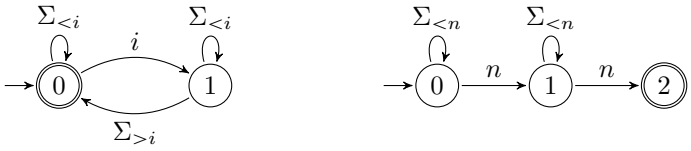


Figure 21 On the left, the automaton A_i, for $i < n$. On the right, the automaton A_n.

subexpression of g (produced by a transition of the form $(s,t,\nu) \xrightarrow{\varepsilon|x} (s,t',1)$) immediately followed by a parenthesis indexed by a subexpression of f (produced by a transition of the form $(s,t,0) \xrightarrow{\varepsilon|x} (s',t,0)$).

We prove now that $\|\mathcal{P}_h\| \leqslant |h|^{\mathsf{width}(h)}$. Again, the proof is by structural induction on the expression h. We have already seen that, when h does not use Hadamard products (in particular for the base cases), we have $\|\mathcal{P}_h\| \leqslant |h| = |h|^{\mathsf{width}(h)}$.

Consider the Hadamard product $h = f \odot g$. We know that $\|\mathcal{P}_h\| = 2 \times \|\mathcal{P}_f\| \times \|\mathcal{P}_g\| + 2$. We also know that $|h| = |f| + |g| + 1$ and $\mathsf{width}(h) = \mathsf{width}(f) + \mathsf{width}(g)) \geqslant 2$. By induction hypothesis, we have $\|\mathcal{P}_f\| \leqslant |f|^{\mathsf{width}(f)}$ and $\|\mathcal{P}_g\| \leqslant |g|^{\mathsf{width}(g)}$. Then, we get

$$\begin{aligned}\|\mathcal{P}_h\| = 2 \times \|\mathcal{P}_f\| \times \|\mathcal{P}_g\| + 2 &\leqslant 2 \times |f|^{\mathsf{width}(f)} \times |g|^{\mathsf{width}(g)} + 2 \\ &\leqslant (|f| + |g|)^{\mathsf{width}(f)+\mathsf{width}(g)} \leqslant |h|^{\mathsf{width}(h)}\,.\end{aligned}$$

The other cases are easy. When $h = f + g$ or $h = f \cdot g$ or $h = f \cdot_r g$, then we have

$$\begin{aligned}\|\mathcal{P}_h\| \leqslant 1 + \|\mathcal{P}_f\| + \|\mathcal{P}_g\| &\leqslant 1 + |f|^{\mathsf{width}(f)} + |g|^{\mathsf{width}(g)} \\ &\leqslant 1 + |f|^{\mathsf{width}(h)} + |g|^{\mathsf{width}(h)} \leqslant |h|^{\mathsf{width}(h)}\,.\end{aligned}$$

When h is $f^\star$ or $f_r^\star$ then $\|\mathcal{P}_h\| = 1 + \|\mathcal{P}_f\| \leqslant 1 + |f|^{\mathsf{width}(f)} = 1 + |f|^{\mathsf{width}(h)} \leqslant |h|^{\mathsf{width}(h)}$. ◂

Finally, the next proposition shows that this exponential blow-up in the width of the expression is unavoidable.

▸ **Proposition 10.** *For all $n > 0$, there exists an RTE C_n of size $O(n)$ such that $\mathsf{width}(C_n) = n$ and any parser of C_n is of size $\Omega(2^n)$.*

Proof. The key argument is that any parser of an RTE has to at least recognize its domain. As the Hadamard product restrict the domain to the intersection of its subexpressions, we can construct an RTE C_n which is the Hadamard product of n subexpressions of fixed size, and whose intersection is a single word of size 2^n. Consequently, any parser of C_n is of size at least 2^n.

Let $\Sigma = \{1,\dots,n\}$, $\Sigma_{<i} = \{1,\dots,i-1\}$ and $\Sigma_{>i} = \{i+1,\dots,n\}$ for all i. For $n > i \geqslant 1$, we define $L_i = (\Sigma_{<i}^* i \Sigma_{<i}^* \Sigma_{>i})^*$ and $L_n = \Sigma_{<n}^* n \Sigma_{<n}^* n$. Moreover, let us define recursively the sequence of words $(u_i)_{1\leqslant i\leqslant n} \in \Sigma^*$ as follows: $u_1 = 1$, $u_i = u_{i-1} i u_{i-1}$ for $2 \leqslant i < n$ and $u_n = u_{n-1} n u_{n-1} n$. By construction, we have $|u_n| = 2^n$. Also, each L_i can be recognized by an automaton of size 2 as described in Figure 21, and $\{u_n\} = \bigcap_{i=1}^n L_i$.

Finally, let us define $C_n = \bigodot_{i=1}^n (L_i \rhd \varepsilon)$ whose size is $O(n)$ and width is n. Its domain $\mathsf{dom}(C_n)$ is the singleton $\{u_n\}$ whose size is exponential in n, and thus any parser of C_n has to be of size at least exponential in n.

Note that our definition of languages L_i depends on an alphabet of size n. Equivalently, we can use unary encodings of each integer i to define languages of size linear instead of constant, but with an alphabet of constant size. ◂

7 k-star operator and its reverse

In this section, we extend the set of RTEs discussed in Sections 4, 5 and 6 with the k-star and the reverse k-star operators.

7.1 Parsing relation for $h = [e, f]^{k\star}$ and $h = [e, f]_r^{k\star}$

We will first describe, with the help of an example, the parsing set $P_h(w)$ that we want to compute given a word w in the domain of $h = [e, f]^{k\star}$ and $h = [e, f]_r^{k\star}$. The parsing relation P_h is exactly the same for both $h = [e, f]^{k\star}$ and $h = [e, f]_r^{k\star}$. Let $L = L(e)$. To motivate the parsing relation, we will also give a brief overview of the working of the evaluator $\mathcal{T}_h$ that computes $h(w)$ from a parsing in $P_h(w)$ of the word w.

Recall that for a word w to be in the domain of h, w should have a factorization $w = u_1 \cdots u_n$ satisfying (†), i.e., $n \geqslant 0$, $u_i \in L = L(e)$ for $1 \leqslant i \leqslant n$, and $u_{i+1} \cdots u_{i+k} \in \mathsf{dom}(f)$ for $0 \leqslant i \leqslant n - k$. Given a word w and one such factorization $w = u_1 u_2 \cdots u_n$, we will refer to the factor $u_{i+1} u_{i+2} \cdots u_{i+k}$ as the ith *block* of w. While evaluating the expression $h = [e, f]^{k\star}$ on w, f is first applied on the 0th block of w, then the 1st block and so on, until the $(n - k)$th block of w. For $h = [e, f]_r^{k\star}$, the order of evaluation is reversed, i.e., f is first applied on the $(n - k)$th block of w, then the $(n - k - 1)$th block and so on, until the 0th block of w.

The parsing of w w.r.t. h should contain the information required for this evaluation. In other words, we need to add the parentheses w.r.t. f on the 0th block, the 1st block and so on, until the $(n - k)$th block of w. Since we want to construct a parser that is one-way, we need to shuffle the parentheses that arise from the application of f on different blocks. Consequently, we need some way to distinguish the parentheses that arise due to the application of f on a block from the parentheses that arise due to the application of f on other blocks. To this end, we will use parentheses indexed by $\{1, 2, \cdots, k\}$.

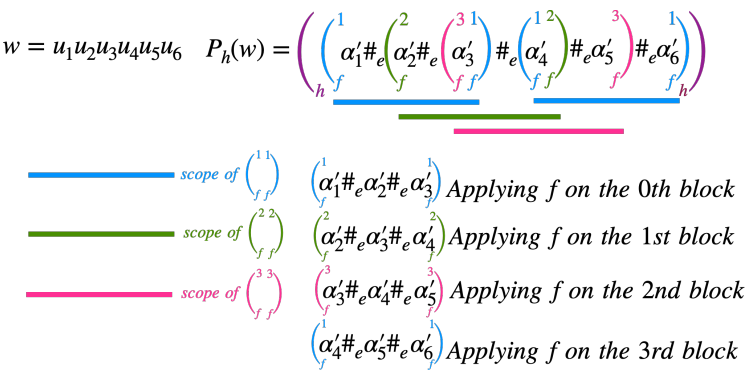


Figure 22 $P_h(w)$ for $h = [e, f]^{3\star}$ on a word w in $L(e)^6$.

In Figure 22, we illustrate a parsing in $P_h(w)$ when $h = [e, f]^{3\star}$ on a word $w = u_1 u_2 u_3 u_4 u_5 u_6$ with each $u_i \in L(e)$. As depicted in the figure, while processing the ith block $u_{i+1} u_{i+2} \cdots u_{i+k}$, $\mathcal{T}_h$ considers only the parentheses indexed by $i + 1 \bmod k$, and ignores all other parentheses. After reading the kth L-factor of the ith block, $\mathcal{T}_h$ checks if there are any more blocks to be read. If not, then $\mathcal{T}_h$ is done with its computation. Otherwise, $\mathcal{T}_h$ goes back, and repeats the same process on block $(i + 1)$, but this time considering only parentheses indexed by $i + 2 \bmod k$.

To summarise, an h-parsing of w w.r.t. a factorization $w = u_1u_2\cdots u_n$ satisfying (†) and $n \geqslant k$ is a word of the form $(_h\, \alpha_1 \#_e \alpha_2 \cdots \#_e \alpha_n\,)_h$ such that:

- It starts with an opening parenthesis $(_h$ and ends with a closing parenthesis $)_h$, and the projection of α_i on Σ is u_i.
- Each block $u_{i+1}u_{i+2}\cdots u_{i+k}$ is decorated with a parenthesisation corresponding to $P_f(u_{i+1}u_{i+2}\cdots u_{i+k})$, such that each of these parentheses is indexed by $i+1 \bmod k$.
- Between any two consecutive L-factors u_i and u_{i+1}, there is a $\#_e$. In particular, immediately after each $)_f$ and immediately before each $(_f$, there is a $\#_e$.
- Between any two letters of the ith L-factor u_i of w, the parentheses appear in non-decreasing order of their indices w.r.t. some order $\leqslant_i$ (described in detail in the formal parsing).

Note that there are several ways to satisfy just the first three conditions given above, as the parentheses indexed i could be shuffled arbitrarily with parentheses indexed by $j \neq i$. Since this would violate the requirement $\mathsf{udom}(h) = \mathsf{fdom}(P_h)$ that we crucially depend on, we have the final condition that fixes a specific order of parenthesisation.

▸ Remark 11. Note that in a factorization of $u_1\cdots u_n$ of w, it is possible that $u_i = \varepsilon$. This could lead to problems that violate the requirement $\mathsf{udom}(h) = \mathsf{fdom}(P_h)$. To address this, we have additional conditions (rule 5) in the formal definition of the parsing.

Consider a parsing $(_h\, \alpha_1\#_e\alpha_2\#_e\alpha_3\#_e\alpha_4\#_e\alpha_5\#_e\alpha_6\,)_h$ for h and w from Figure 22. Here, α_1 has only parentheses indexed by 1, α_2 has only parentheses indexed by 1 and 2, α_3 and α_4 have parentheses indexed by 1, 2 and 3, α_5 has only parentheses indexed by 1 and 3, and α_6 has only parentheses indexed by 1. In particular, α_3 for instance can be of the form $(_f{}^3 a_1 (^1\,)^2 (^2 (^3 a_2 (^2\,)^3 \cdots)^2 (^3 a_l)^3\,)_f{}^1$, where $a_i \in \Sigma$.

Formal definition of the parsing relation for $h = [e,f]^{k\star}$ and $h = [e,f]_r^{k\star}$

First let B be the set of parentheses appearing in the parsing of f. We define B_i, for $1 \leqslant i \leqslant k$, to be the set B indexed by i. We write $|^i$ for either $(^i$ or $)^i$. Additionally, for ease of notations $k \bmod k$ is set to k instead of 0 as commonly defined. Let $L = L(e)$ and w be an input word of h. The parsing set $P_h(w)$ is the set of words $(_h\, \alpha_1\#_e\alpha_2\#_e\ldots\#_e\alpha_n\,)_h$ such that there is a factorization $w = u_1\ldots u_n$ where for all $i \leqslant n$, $u_i \in L$ and $\pi_\Sigma(\alpha_i) = u_i$ and either $n < k$ and $\alpha_i = u_i$, hence the parsing is $(_h\, u_1\#_e u_2\#_e\ldots\#_e u_n\,)_h$, or $n \geqslant k$ and:

1. for all $0 \leqslant i \leqslant n-k$, $\pi_{i+1 \bmod k}(\alpha_{i+1}\ldots\alpha_{i+k}) \in P_f(u_{i+1}\ldots u_{i+k})$ where π_j is the function projecting away all parentheses $|^\ell$ for $\ell \neq j$ and erasing the exponent j,
2. for all $1 \leqslant i < j \leqslant k$, α_i does not contain any parenthesis $|^j$,
3. for all $n-k+1 \leqslant j < i \leqslant n$, α_i does not contain any parenthesis $|^{j \bmod k}$,
4. α_i ends with $)_f{}^{i+1 \bmod k}$ if $i \geqslant k$,
5. α_i starts with $(_f{}^{i \bmod k}$ if $i \leqslant n-k+1$, and if $n-k+1 < i$, then either α_i starts with a letter of Σ or $\alpha_i =)_f{}^{i+1 \bmod k}$ or $\alpha_i = \varepsilon$ (if also $i < k$),
6. for all α_i and for all j, j', if $|^j|^{j'}$ appears in α_i and $|^{j'} \neq)_f{}^{i+1 \bmod k}$, then $j \leqslant_i j'$, where $\leqslant_i$ is defined as $i+1 \bmod k \leqslant_i i+2 \bmod k \leqslant_i \cdots \leqslant_i i$.

▸ **Lemma 12.** *Let h be an RTE. We have $\mathsf{dom}(P_h) = \mathsf{dom}(h)$ and $\mathsf{fdom}(P_h) = \mathsf{udom}(h)$.*

Proof. As with the previous cases, the proof is by structural induction. Using Lemma 9, the only cases left are the k-star operator and its reverse. We only prove the result for k-star. As the reverse k-star parses the input in the same way, the proof will hold for both operators.

Let then $h = [e,f]^{k\star}$ and w be an input word of h. If $w \in \mathsf{dom}(h)$, then there exists a factorization $w = u_1\cdots u_n$ satisfying that for all $i \leqslant n$, $u_i \in L(e)$ and either $n < k$, in which

case $(_h\, u_1 \#_e u_2 \#_e \cdots \#_e u_n\, {}_h) \in P_h(w)$ and hence $w \in \mathsf{dom}(P_h)$, or $n \geqslant k$ and $u_{j+1} \cdots u_{j+k}$ belongs to $\mathsf{dom}(f)$ for all $0 \leqslant j \leqslant n-k$. We construct a parsing $(_h\, \alpha_1 \#_e \cdots \#_e \alpha_n\, {}_h)$ for this factorization $w = u_1 \cdots u_n$. Using the induction hypothesis, for all $0 \leqslant j \leqslant n-k$, there exists a word β_j such that $(u_{j+1} \cdots u_{j+k}, \beta_j) \in P_f$. Then by definition, $\pi_\Sigma(\beta_j) = u_{j+1} \cdots u_{j+k}$. Let γ_j be β_j where all parentheses are indexed by $m = j+1 \bmod k$. There is a unique factorization $\gamma_j = \gamma_j^1 \cdots \gamma_j^k$ such that $\pi_\Sigma(\gamma_j^\ell) = u_{j+\ell}$ for $1 \leqslant \ell \leqslant k$, and γ_j^ℓ starts with a letter from Σ or $\gamma_j^\ell = \varepsilon$ for $1 < \ell < k$, and γ_j^k starts with a letter from Σ or $\gamma_j^k = {}_f)^m$. Notice that γ_j^1 starts with $(_f{}^m$ and γ_j^k ends with ${}_f)^m$. Then α_i is defined by merging all words γ_j^ℓ for $j + \ell = i$, shuffling parentheses and synchronizing on letters from Σ. Notice that ℓ is comprised between 1 and k, and thus we shuffle at most k such γ_j^ℓ, with indices j between $\max(0, i-k)$ and $\min(i-1, n-k)$. This shuffling can be uniquely defined as follows:

(*i*) if $i \leqslant n-k+1$ we take all parentheses of γ_{i-1}^1 up to the first letter of Σ if any, in this case, α_i starts with $(_f{}^{i \bmod k}$,

(*ii*) if $k \leqslant i$, then γ_{i-k}^k ends with ${}_f)^{i+1 \bmod k}$ which we put at the end of α_i.

Then, we proceed from left to right, iterating the two steps below until exhaustion of all γ_j^ℓ.

(*iii*) we take the next letter of Σ, if any, which occurs in each γ_j^ℓ as they all project onto u_i,

(*iv*) we take the following parentheses on each γ_j^ℓ, with increasing indexes according to the order $\leqslant_i$, until the next letter from Σ, if any.

By construction, as α_i contains all γ_j^ℓ for $j + \ell = i$, we have $\pi_{i+1 \bmod k}(\alpha_{i+1} \dots \alpha_{i+k}) = \pi_{i+1 \bmod k}(\gamma_i^1 \dots \gamma_i^k) = \beta_i \in P_f(u_{i+1} \dots u_{i+k})$, hence condition of the parsing relation (1) is satisfied. Next, if $1 \leqslant i < j \leqslant k$, then there is no γ_{j-1}^ℓ with $j - 1 + \ell = i$, hence α_i satisfies condition (2) . Similarly, we can check that it satisfies condition (3). By Step (*ii*) (resp. (*i*)) we see that α_i satisfies condition (4) (resp. the first part of condition (5)). To prove the second part of condition (5), we first note that if $n-k+1 < i \leqslant n$ and $j + \ell = i$, then $\ell > 1$. Then, either all γ_j^ℓ with $j + \ell = i$ start with the same letter from Σ or $\alpha_i = {}_f)^{i+1 \bmod k}$ or $\alpha_i = \varepsilon$ (if $i < k$). Finally, step (*iv*) above ensures that we satisfy point (6) of the parsing relation. As a result, the word $(_h\, \alpha_i \#_e \dots \#_e \alpha_n\, {}_h)$ is a parsing of w, and thus $w \in \mathsf{dom}(P_h)$.

Conversely, let $h \in \mathsf{dom}(P_h)$. Then there exists a parsing word $(_h\, \alpha_i \#_e \dots \#_e \alpha_n\, {}_h)$ of w. As such, let $u_i = \pi_\Sigma(\alpha_i)$. By definition, $w = u_1 \dots u_n$ and for all i, $u_i \in L(e)$. If $n < k$, then $w \in \mathsf{dom}(h)$ by definition. Assume now $n \geqslant k$. Then for the words α_i we have in particular for all $0 \leqslant i \leqslant n-k$, $\pi_{i+1 \bmod k}(\alpha_{i+1} \dots \alpha_{i+k}) \in P_f(u_{i+1} \dots u_{i+k})$. Thus by induction hypothesis $u_{i+1} \dots u_{i+k}$ belongs to the domain of f and consequently $w \in \mathsf{dom}(h)$.

We now turn to the equality $\mathsf{fdom}(P_h) = \mathsf{udom}(h)$. We prove that $\mathsf{dom}(P_h) \backslash \mathsf{fdom}(P_h) = \mathsf{dom}(h) \backslash \mathsf{udom}(h)$. Together with the previous equality this gives the result. Let w be in $\mathsf{dom}(h) \backslash \mathsf{udom}(h)$. This means that either (*a*) there exists two different factorizations $w = u_1 \dots u_n$ satisfying condition (†) on page 10, i.e., such that either $n < k$ or ($n \geqslant k$ and for all $0 \leqslant i \leqslant n-k$, $u_{i+1} \dots u_{i+k}$ belongs to $\mathsf{dom}(f)$), or (*b*) there exists one such factorization with $k \leqslant n$ and at least one i such that $u_{i+1} \dots u_{i+k}$ belongs to $\mathsf{dom}(f) \backslash \mathsf{udom}(f)$.

If (*a*) holds, there exists two such factorizations, and the parsings corresponding to the two different factorization constructed by the procedure above will have the $\#_e$ symbols at different positions of the parsings, and hence we have two parsings of w and $w \notin \mathsf{fdom}(P_h)$. If (*b*) holds, then there is one factorization with an integer i such that $u_{i+1} \dots u_{i+k}$ belongs to $\mathsf{dom}(f) \backslash \mathsf{udom}(f)$. By induction hypothesis there are two different parsings of $u_{i+1} \dots u_{i+k}$ for f. Using the procedure above, we then get two different γ_i and γ_i', which means that we can construct two different $\alpha_{i+1} \dots \alpha_{i+k}$ and $\alpha'_{i+1} \dots \alpha'_{i+k}$. In the end, we get two different parsings for w, and hence $w \notin \mathsf{fdom}(P_h)$.

Conversely, let w be in $\mathsf{dom}(P_h) \backslash \mathsf{fdom}(P_h)$. Then there exist two different parsings $(_h\, \alpha_1 \#_e \dots \#_e \alpha_n\, {}_h)$ and $(_h\, \beta_1 \#_e \dots \#_e \beta_m\, {}_h)$ of w. If $\pi_\Sigma(\alpha_i) \neq \pi_\Sigma(\beta_i)$ for some i, then there

exist two valid and different factorizations of w satisfying (†), and thus w does not belong to $\mathsf{udom}(h)$. Otherwise, it means there is an integer $0 \leqslant i \leqslant n-k$ such that $\pi_{i+1 \bmod k}(\alpha_{i+1} \dots \alpha_{i+k})$ and $\pi_{i+1 \bmod k}(\beta_{i+1} \dots \beta_{i+k})$ are valid but different f-parsings of $u_{i+1} \dots u_{i+k}$. It follows that $u_{i+1} \dots u_{i+k}$ does not belong to $\mathsf{udom}(f)$ and thus w does not belong to $\mathsf{udom}(h)$. ◂

7.2 Evaluators

Here, we propose the evaluators for the k-star and the reverse k-star operators, and give an upper bound on their size.

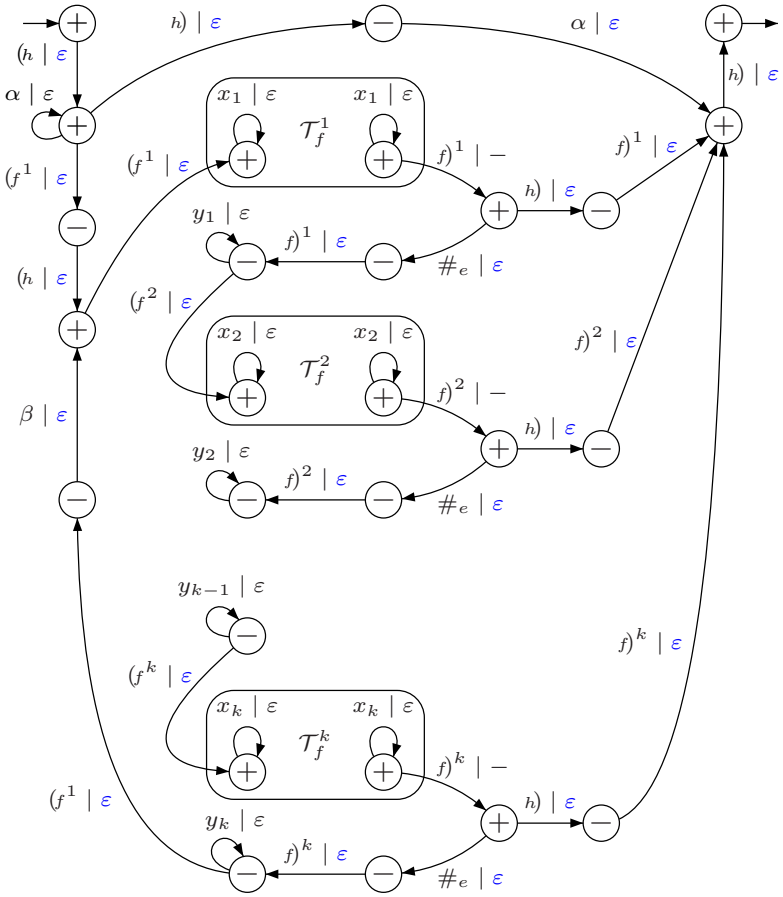


Figure 23 Evaluator for $h = [e, f]^{k\star}$. Here, $x_i \in \{|^j \mid j \neq i\} \cup \{\#_e\}$, $y_i \neq f)^i, (f^{i+1 \bmod k}$, $\alpha \in \Sigma \cup \{\#_e\}$, and $\beta \neq (h$.

Evaluator for k-star. We start with $h = [e, f]^{k\star}$ for which the evaluator $\mathcal{T}_h$ is depicted in Figure 23. It is a 2RFT that takes as input a parsing in $P_h(w)$ for a word w, and computes the output $h(w)$. More precisely, let $\mathcal{T}_f$ be the transducer that computes $f(w)$ given a parsing in $P_f(w)$. The 2RFT $\mathcal{T}_h$ has k copies of $\mathcal{T}_f$, namely $\mathcal{T}_f^1, \mathcal{T}_f^2, \cdots, \mathcal{T}_f^k$. The idea is that the copy $\mathcal{T}_f^i$ should consider only parentheses indexed by i and ignore all other parentheses. We construct $\mathcal{T}_f^i$ from $\mathcal{T}_f$ as follows: to all states of $\mathcal{T}_f$, we add self-loops labelled with all parentheses indexed by j where $j \neq i$, and $\#_e$, and the output of these new transitions is ε. Also, a transition of $\mathcal{T}_f$ reading a parenthesis $\rfloor_g$ for g a subexpression of f is relabelled with $|_g^i$. If $\mathcal{T}_f$ is a 2RFT, then so is $\mathcal{T}_f^i$ for $1 \leqslant i \leqslant k$.

Let $w = u_1u_2\cdots u_n$ be a factorisation of w with each $u_i \in L$. If $n < k$ then the parsing of w w.r.t. h is defined as $(_h\, u_1\#_e u_2\#_e \ldots \#_e u_n\,)_h$. If $k \leqslant n$, then the corresponding parsing is $(_h\, \alpha_1\#_e\alpha_2\cdots\#_e\alpha_n\,)_h$ satisfying conditions (1-6) on page 25. In particular, for $0 \leqslant i \leqslant n-k$ and $m = i+1 \bmod k$, the block $\alpha_{i+1}\cdots\alpha_{i+k}$ starts with $(_f^m$ and ends with $)_f^m$.

The working of the transducer $\mathcal{T}_h$ is based on the above observations. It starts by reading the opening parenthesis $(_h$ that indicates that domain of h is about to be read. If the next character read is an opening parenthesis $(_f^1$, then it means that the parsing of w contains $n \geqslant k$ L-factors. Otherwise, it indicates that the parsing of w contains $n < k$ L-factors.

In the case where the parsing w contains less than k L-factors, $\mathcal{T}_h$ remains in this state, where it reads letters from $\Sigma \cup \{\#_e\}$, while producing nothing until a closing parenthesis $)_h$ is read. When it reads $)_h$ at the end of w, it goes to the accepting state.

Otherwise, it reads an opening parenthesis $(_f^1$ that denotes the beginning of the first block of w. On reading $(_f^1$, $\mathcal{T}_h$ moves to the initial state of $\mathcal{T}_f^1$. We know from the construction that $\mathcal{T}_f^1$ ignores all parsing symbols that are not indexed by 1. The run of $\mathcal{T}_f^1$ goes on until we see a closing parenthesis $)_f^1$, which means that $\mathcal{T}_h$ has finished processing the first block.

Then, $\mathcal{T}_h$ should go back and read the next block, if it exists. In general, the block $\alpha_{i+1}\cdots\alpha_{i+k}$ is processed by $\mathcal{T}_f^{i+1 \bmod k}$. The decision whether to go back or not (in other words, whether the block just read is the last one) is taken depending on whether we see $)_h$ or $\#_e$ next. If the closing parenthesis $)_h$ is the next letter read, then $\mathcal{T}_h$ knows that the domain of h has been read completely, and therefore exits. Otherwise, if a $\#_e$ is the next letter read, then $\mathcal{T}_h$ knows that there are more L-factors to the right of the current position, which implies that this was not the last block. So, $\mathcal{T}_h$ will go back on the parsed word until it reads $(_f^{i+1 \bmod k}$, which signals the beginning of the next block. Then, it repeats the above process on the next block by going to the initial sate of $\mathcal{T}_f^{i+1 \bmod k}$.

In Figure 24, we illustrate the run of $\mathcal{T}_h$ on an example, where $k = 3$ and $n = 6$.

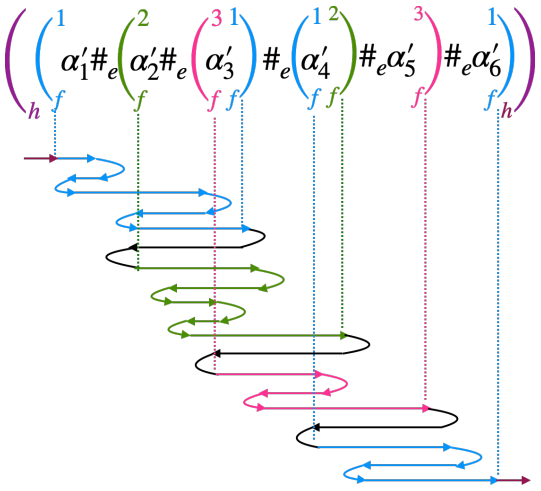


Figure 24 Run of the transducer $\mathcal{T}_h$ on $P_h(w)$, where $w = u_1u_2u_3u_4u_5u_6$, when $k = 3$. The blue zig-zag arrows ignores all paratheses not indexed by 1, and represents the run of $\mathcal{T}_f^1$, likewise, the green zig-zag arrows ignores all paratheses not indexed by 2, and represents the run of $\mathcal{T}_f^2$, the magenta zig-zag arrows ignores all paratheses not indexed by 3. The black arrow represents the backward movement of $\mathcal{T}_h$ from $)_f^i$ looking for $(_f^{i+1}$ *mod* k.

Evaluator for reverse k-star. We turn to the description of the evaluator $\mathcal{T}_h$ for $h = [e, f]_r^{k\star}$. The 2RFT $\mathcal{T}_h$ is depicted in Figure 25. We use the same k copies $\mathcal{T}_f^1, \mathcal{T}_f^2, \ldots, \mathcal{T}_f^k$

(described above) of the evaluator $\mathcal{T}_f$ for the RTE f. Recall that the parsing relation is the same for $[e,f]^{k\star}$ or $[e,f]_r^{k\star}$. Hence, we use the observations made above for a parsing $(_h\, \alpha_1 \#_e \alpha_2 \cdots \#_e \alpha_n\,)_h \in P_h(w)$.

$\mathcal{T}_h$ starts by reading the opening parenthesis $(_h$ that indicates that domain of h starts. It moves to a + state, where it scans the parsed word without producing anything, until a closing parenthesis $)_h$ is reached. On reading an $)_h$, $\mathcal{T}_h$ knows that the domain of h has been read completely and goes to a − state. $\mathcal{T}_h$ then looks at the letter on the left. If the letter is from $\Sigma \cup \{\#_e\}$, it means that the parsing of w contains strictly less than k L-factors. In this case, $\mathcal{T}_h$ reads the closing parenthesis $)_h$ and exits. Otherwise the letter is a parenthesis $)_f^i$, which means that the parsing of w has $n \geqslant k$ L-factors. Moreover, we know that the last block $\alpha_{n-k+1} \cdots \alpha_n$ starts with $(_f^i$. So, $\mathcal{T}_h$ moves to the left until it sees $(_f^i$. When $\mathcal{T}_h$ sees the $(_f^i$, it is at the beginning of the last block, and on reading $(_f^i$, it moves to the initial state of $\mathcal{T}_f^i$, which processes the block (ignoring all parsing symbols that are not indexed by i). The run goes on until we see a closing parenthesis $)_f^i$ which means that $\mathcal{T}_f^i$ has finished reading the block.

Now, $\mathcal{T}_h$ should go back and read the block on the left, if it exists. It first goes back until it sees $(_f^i$. The decision whether there is another block on the left (in other words, whether the block just read is not the leftmost one) is taken depending on whether we see on the left $\#_e$ or $(_h$. If the opening parenthesis $(_h$ is seen, then it means that the block just read is the first (leftmost) block. In this case, $\mathcal{T}_h$ knows that it has finished processing the domain of h, and therefore does a rightward run until it sees a closing parenthesis $)_h$, upon seeing which $\mathcal{T}_h$ exits. In this case, $\mathcal{T}_h$ goes on a rightward run until it sees the closing parenthesis $)_h$, and exits. Otherwise, $\mathcal{T}_h$ sees $\#_e$ and realises that there is at least one more block on the left to be processed. It moves to the beginning of this block and repeats the above process by going to the initial sate of $\mathcal{T}_f^{i-1 \bmod k}$.

▸ **Lemma 13.** *For all RTEs h, the number of states of the evaluator for h is $\|\mathcal{T}_h\| \leqslant 5|h| \cdot \mathsf{width}(h)$.*

Proof. The proof is by structural induction on the expression h. We have already seen that, when h does not use k-star or reverse k-star, then $\|\mathcal{T}_h\| \leqslant 5|h|$.

We will now consider the case where h is a k-chained Kleene-star expression ($h = [e,f]^{k\star}$) or a reverse k-chained Kleene-star expression ($h = [e,f]_r^{k\star}$). For both cases, we easily see that $\|\mathcal{T}_h\| = k\|\mathcal{T}_f\| + 3k + 8$. By induction hypothesis, we have $\|T_f\| \leqslant 5|f| \cdot \mathsf{width}(f)$. We get $\|\mathcal{T}_h\| \leqslant 5k|f| \cdot \mathsf{width}(f) + 3k + 8 \leqslant 5(|f| + k + 2)(k \cdot \mathsf{width}(f) + 1)$. Therefore, we get $\|\mathcal{T}_h\| \leqslant 5|h| \cdot \mathsf{width}(h)$. ◂

7.3 Parser for k-star and reverse k-star

In this section, we propose the parser $\mathcal{P}_h$ for $h = [e,f]^{k\star}$. The idea we use to construct $P_h(w)$ is that on the input word, whenever we finish reading an L-factor u_i, we mark this by adding a $\#_e$ indicating the end of an L-factor, and we start an instance of the transducer $\mathcal{P}_f$ in which whenever a parenthesis is output, it will be indexed by $i+1 \bmod k$. Reading an L-factor can be detected by running in parallel, the automaton $\mathcal{A}_e$ for e obtained via the Glushkov algorithm. We also employ a counter that keeps track of how many factors of L we have seen so far in the current factorization of w being considered - the counter stores $i \bmod k$ when we are reading the ith L-factor in the factorization of w. Then, whenever we reach an accepting state of $\mathcal{A}_e$ with counter value i, the parser guesses whether or not there are at least k L-factors left in the factorization of w. If the parser guesses yes, an instance of the transducer $\mathcal{P}_f$ which adds the index $i+1 \bmod k$ to its parentheses is initialized and

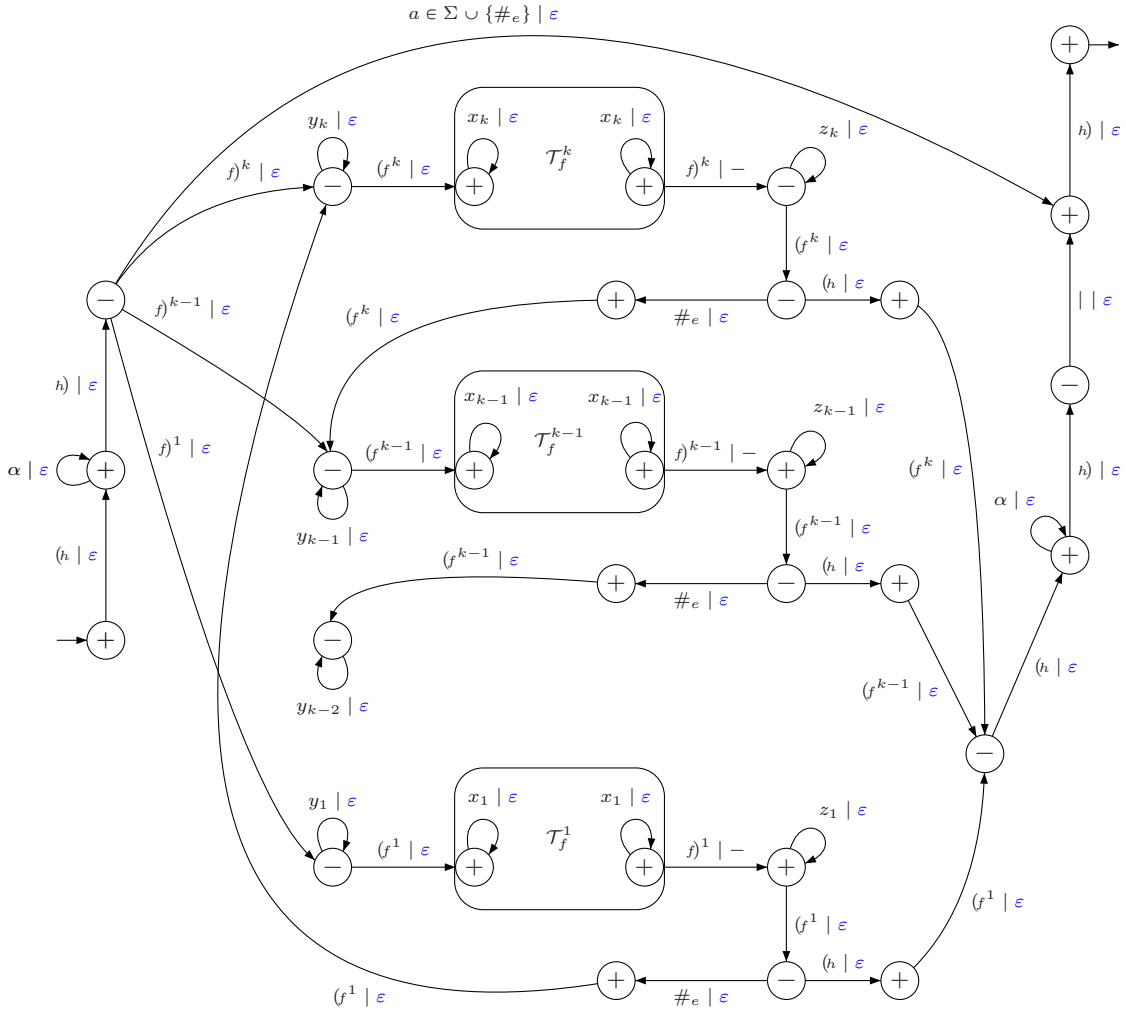


Figure 25 Evaluator for $h = [e, f]_r^{k\star}$. Here, $x_i \in \{|^j \mid j \neq i\} \cup \{\#_e\}$, $y_i \neq (f^i, f)^i, (f^{i+1 \bmod k}$, $z_i \neq f)^i, (f^i$, and $\alpha \neq (h, h)$. Note that $\mid$ denotes any parenthesis.

run on the next k L-factors of w. If the parser guesses that only fewer than k L-factors are remaining, then no new instance of the transducer $\mathcal{P}_f$ is initialized. We will show that k copies of $\mathcal{P}_f$ suffice to implement the above idea. Further, we employ a variable that ensures the order of parenthesisation required by the definition of the parsing relation.

We turn to the formal definition of the parsing transducer $\mathcal{P}_h$. Let $\mathcal{A}_e = (Q, \Sigma, q_I, F, \Delta)$ be the Glushkov automaton constructed from the regular expression e. The parser $\mathcal{P}_h$ has two main components that we define separately. The first and the more involved one $\mathcal{P}'_h$ is used in the generic case for decomposition of words having at least k factors in $L(e)$. The second one $\mathcal{P}''_h$, defined afterwards, handles the decompositions having less than k $L(e)$-factors.

Let $\mathcal{P}_f = (Q_f, \Sigma, B, q_I^f, q_F^f, \Delta_f, \mu_f)$ be the 1-way transducer that produces the parsing w.r.t. expression f. We define the 1NFT $\mathcal{P}'_h = (Q_h, \Sigma, B_h, q_I^h, F^h, \Delta_h, \mu_h)$ by

- $Q_h = \{1, 2, \cdots, k\} \times Q \times (Q_f \cup \{q_\perp\})^k \times \{1, 2, \cdots, k\}$
- The input alphabet is Σ, the same as that of $\mathcal{P}_f$ and $\mathcal{A}_e$.
- The output alphabet $B_h = B_1 \cup B_2 \cup \cdots B_k \cup \{\#_e\}$, where B_i is the output alphabet B of $\mathcal{P}_f$ where each parenthesis is indexed by i. Note that Σ is contained in each B_i.

- The initial state is $q_I^h = (1, q_I, q_I^f, q_\perp, \cdots, q_\perp, 1)$.
- The set of final states is $F^h = \{(i, q, q_1, \ldots, q_k, j) \mid q \in F, q_{i+1 \bmod k} = q_F^f$, and $q_\ell = q_\perp$ for $\ell \neq i+1 \bmod k\}$.
- The transition relation Δ_h of $\mathcal{P}'_h$ is defined below.
  Let $(i, q, q_1, \ldots, q_k, j)$ be a state of $\mathcal{P}'_h$ and let $m = i+1 \bmod k$.
  1. if $q \xrightarrow{a} q'$ in $\mathcal{A}_e$ and for $1 \leqslant \ell \leqslant k$ either $q_\ell \xrightarrow{a|a} q'_\ell$ in $\mathcal{P}_f$ or $q_\ell = q_\perp = q'_\ell$, then $(i, q, q_1, \ldots, q_k, j) \xrightarrow{a|a} (i, q', q'_1, \ldots, q'_k, m)$ in $\mathcal{P}'_h$. Note that the last component is reset to $m = i+1 \bmod k$ which is the least element w.r.t. $\leqslant_i$.
  2. if there is an ε-transition $q_\ell \xrightarrow{\varepsilon||g} q'_\ell \neq q_F^f$ in $\mathcal{P}_f$, and $j \leqslant_i \ell$ and $q_m \neq q_F^f$, then we have $(i, q, q_1, \ldots, q_k, j) \xrightarrow{\varepsilon||_g^\ell} (i, q, q_1, \ldots, q'_\ell, \ldots, q_k, \ell)$ in $\mathcal{P}'_h$. Note that the last component is set to ℓ which prevents executing next an ε-transition (case 2) in a component $\ell' <_i \ell$ since this would produce a parenthesis indexed ℓ followed by a parenthesis indexed ℓ' in the wrong order.
  3. if $q \in F$ is an accepting state of $\mathcal{A}_e$ and $q_i \neq q_I^f$ and $q_m \xrightarrow{\varepsilon|f} q'_m = q_F^f$ is a transition in $\mathcal{P}_f$, then $(i, q, q_1, \ldots, q_k, j) \xrightarrow{\varepsilon|f)^m} (i, q, q_1, \ldots, q'_m = q_F^f, \ldots, q_k, j)$ in $\mathcal{P}'_h$.
     Notice that a transition of case 2 cannot be applied after a transition of case 3 since the state of the component m is now q_F^f.
  4. if $q \in F$ and $q_i \neq q_I^f$ and $q_m \in \{q_F^f, q_\perp\}$ and $(i, q, q_1, \ldots, q_k, j) \notin F^h$, then we have $(i, q, q_1, \ldots, q_k, j) \xrightarrow{\varepsilon|\#_e} (m, q_I, q_1, \ldots, q'_m, \ldots, q_k, m)$ in $\mathcal{P}'_h$ where $q'_m = q_\perp$ if $q_i = q_\perp$ and $q'_m \in \{q_\perp, q_I^f\}$ otherwise.
     Note that the last component is set to m which is the maximal element w.r.t. $\leqslant_m$. Hence, only component m may perform ε-transitions producing parentheses indexed by m (case 2) until a letter $a \in \Sigma$ is read (case 1) or until we use again a switching transition of case 3 or case 4, which is only possible if the initial state q_I of $\mathcal{A}_e$ is also accepting ($q_I \in F$), i.e., when $\varepsilon \in L(e)$.

Note that, even if all three conditions of case 3 are satisfied, $\mathcal{P}'_h$ may choose not to execute the corresponding transition; it may still execute transitions from cases 1 or 2. Also, a transition from case 3 is either the last one in the run or it must be followed by a transition of case 4.

We will now define a transducer $\mathcal{P}''_h$ that takes care of words in $\bigcup_{n<k} L(e)^n$. Recall that the automaton $\mathcal{A}_e = (Q, \Sigma, q_I, F, \Delta)$ recognizes $L(e)$. $\mathcal{P}''_h$ is defined as the 1-way transducer whose

- set of states is $Q \times \{1, 2, \cdots, k-1\}$,
- input alphabet is Σ and output alphabet is $\Sigma \cup \{\#_e\}$,
- initial state is $(q_I, 1)$,
- set of final states is $F'' = F \times \{1, 2, \cdots, k-1\}$,
- transition relation is defined as follows:
  - $(q, i) \xrightarrow{a|a} (q', i)$ if $q \xrightarrow{a} q'$ in Δ,
  - $(q, i) \xrightarrow{\varepsilon|\#_e} (q_I, i+1)$ if $q \in F$ and $i+1 < k$.

$\mathcal{P}''_h$ just counts the number of $L(e)$-factors read upto $k-1$ and adds the corresponding separators between them. If $w \in L(e)^n$ with $n < k$, then $\mathcal{P}''_h$ has a run from the initial state $(q_I, 1)$ to a final state (q, n) where $q \in F$. The output function of $\mathcal{P}''_h$ is then the identity with $\#_e$ symbols inserted.

The 1-way transducer for $h = [e, f]^{k\star}$ is $\mathcal{P}_h$ given in Figure 26.

▸ **Lemma 14.** *Let h be an RTE. The parser $\mathcal{P}_h$ computes the parsing relation P_h. The number of states of the parser is $\|\mathcal{P}_h\| \leqslant |h|^{\mathsf{width}(h)}$.*

Proof. As before, the proof is by structural induction and the only new case is when

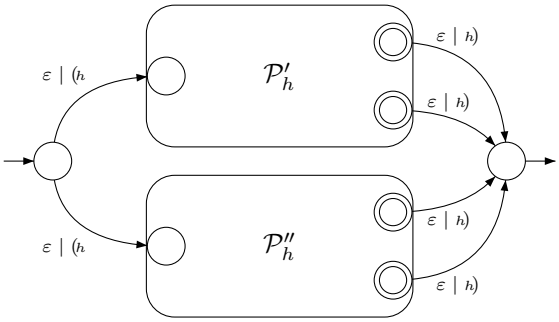


Figure 26 Parser for $h = [e, f]^{k\star}$

$h = [e, f]^{k\star}$. We first show that $[\![\mathcal{P}_h]\!]^R \subseteq P_h$.

It is easy to see that the relation defined by $\mathcal{P}''_h$ is the set of pairs (w, α) where $w = u_1 u_2 \cdots u_n$ with $n < k$ and $u_i \in L(e)$, and $\alpha = u_1 \#_e u_2 \cdots \#_e u_n$. We get $(_h\, \alpha\,)_h \in P_h(w)$.

Let ϱ be an accepting run of $\mathcal{P}'_h$ reading an input word $w \in \Sigma^\star$. We factorize the run as $\varrho = \varrho_1 \delta_1 \varrho_2 \cdots \delta_{n-1} \varrho_n$ where δ_i are the transitions of case 4, i.e., labelled $\varepsilon \mid \#_e$. Let u_i and α_i be respectively the input read by ϱ_i and the output produced by ϱ_i. We have $w = u_1 \cdots u_n$ and the output produced by ϱ is $\alpha = \alpha_1 \#_e \alpha_2 \cdots \#_e \alpha_n$. We will show that $(_h\, \alpha\,)_h \in P_h(w)$.

The counter modulo k which is the first component of states of $\mathcal{P}'_h$ is incremented only by transitions of case 4. Hence, during ϱ_i the counter is constantly $i \bmod k$ and the transition δ_i increments it to $i + 1 \bmod k$. From the condition $q \in F$ in case 4 or by definition of F_h, we deduce that the projection of ϱ_i on the second component is an accepting run of $\mathcal{A}_e$ for the input word u_i. Hence $u_i \in L(e)$. It is also easy to see that $\pi_\Sigma(\alpha_i) = u_i$. Before ϱ can reach an accepting state of F_h, its third component which started initially with q_I^f has to reach q_F^f which is possible only by a transition of case 3 when the first component which counts modulo k has the value k. We deduce that $n \geqslant k$.

We show below that conditions (1-6) on page 25 defining the parsing relation are satisfied. We deduce that $(_h\, \alpha\,)_h \in P_h(w)$ as desired.

1. Let $0 \leqslant i \leqslant n - k$ and $m = i + 1 \bmod k$. We consider the projection ϱ' on the component $2 + m$ of the subrun $\varrho_{i+1} \cdots \varrho_{i+k}$ reading $u_{i+1} \cdots u_{i+k}$ and producing $\alpha_{i+1} \#_e \cdots \#_e \alpha_{i+k}$. We can check that ϱ' is an accepting run of $\mathcal{P}_f$ reading $u_{i+1} \cdots u_{i+k}$ and producing the projection of $\alpha_{i+1} \cdots \alpha_{i+k}$ on $\Sigma \cup B_m$. We deduce that $\pi_m(\alpha_{i+1} \cdots \alpha_{i+k}) \in P_f(u_{i+1} \cdots u_{i+k})$ and condition (1) holds.
2. Let $1 \leqslant i < j \leqslant k$. During the run ϱ_i, the component $2 + j$ of the states is constantly $q_\perp$ and we deduce that α_i does not contain a parenthesis $|^j$.
3. Similarly, if $n - k + 1 \leqslant i < j \leqslant n$, then during the run ϱ_i, the component $2 + (j \bmod k)$ of the states is constantly $q_\perp$ and we deduce that α_i does not contain a parenthesis $|^{j \bmod k}$.
4. Let $i \geqslant k$ and $m = i + 1 \bmod k$. As above, consider the projection ϱ' on the component $2 + m$ of the subrun $\varrho_{i-k+1} \cdots \varrho_i$ reading $u_{i-k+1} \cdots u_i$ and producing $\alpha_{i-k+1} \#_e \cdots \#_e \alpha_i$. Since ϱ' is an accepting run of $\mathcal{P}_f$, it ends with some $q_{2+m} \xrightarrow{\varepsilon \mid)^m} q_F^f$, which must be the projection of a transition of case 3. Now, a transition of case 3 may only be followed by a transition of case 4. Hence, this is the last transition of ϱ_i and we deduce that α_i ends with $)^m$.
5. Let $i \leqslant n - k + 1$ and $m = i \bmod k$. We can see that ϱ_i starts from some state $(m, q_I, q_1, \ldots, q_k, m)$ with $q_m = q_I^f$. We have a transition $q_m = q_I^f \xrightarrow{\varepsilon \mid (} q'_m$ in $\mathcal{P}_f$. Hence the first transition of ϱ_i must be with case 2 and since m is the maximal element w.r.t.

the order $\leqslant_m$ it must be induced by the transition $q_m = q_I^f \xrightarrow{\varepsilon \mid (_f} q'_m$ of $\mathcal{P}_f$. We deduce that it is labelled $\varepsilon \mid (_f^m$ and that α_i starts with $(_f^m$.
Let $i > n-k+1$ and $m = i \bmod k$. We can see that ϱ_i starts from some state $(m, q_I, q_1, \ldots, q_k, m)$ with $q_m = q_\perp$. Since m is the maximal element w.r.t. the order $\leqslant_m$, the first transition of ϱ_2 cannot be from case 2. Either it is from case 1 and α_i starts with a letter from Σ, or it is from case 3 and $\alpha_i =)_f^{i+1 \bmod k}$, or it is from case 4 and $\alpha_i = \varepsilon$.

6. Assume that α_i has two consecutive parentheses $|^j|^\ell$ with $|^\ell \neq)_f^{i+1 \bmod k}$. The two parentheses have been produced by consecutive transitions of case 2. We get $j \leqslant_i \ell$.

Conversely, we prove that $P_h \subseteq [\![\mathcal{P}_h]\!]^R$. Let $(_h \alpha)_h \in P_h(w)$. We write $w = u_1 u_2 \cdots u_n$ with $n \geqslant 0$, $u_i \in L(e)$ for $1 \leqslant i \leqslant n$, and $\alpha = \alpha_1 \#_e \alpha_2 \cdots \#_e \alpha_n$ such that either $n < k$ and $\alpha_i = u_i$, or $n \geqslant k$ and conditions (1-6) on page 25 defining the parsing relation are satisfied. In the first case, it is clear that (w, α) is in $[\![\mathcal{P}''_h]\!]^R$. So we assume that $n \geqslant k$ and we will show that (w, α) is in $[\![\mathcal{P}'_h]\!]^R$.

For each $0 \leqslant i \leqslant n-k$, with $m = i+1 \bmod k$, condition (1) implies that we have $\pi_m(\alpha_{i+1} \cdots \alpha_{i+k}) \in P_f(u_{i+1} \cdots u_{i+k})$. We choose a corresponding accepting run σ_i of $\mathcal{P}_f$. We write $\sigma_i = \sigma_i^1 \cdots \sigma_i^k$ where σ_i^ℓ reads $u_{i+\ell}$ and produces $\pi_m(\alpha_{i+\ell})$ (this factorization is unique since each transition of $\mathcal{P}_f$ produces a single symbol).

Now, let $1 \leqslant j \leqslant n$, and consider an accepting run ϱ'_j for u_j in $\mathcal{A}_e$. The output word α_j determines a unique way to order transitions of ϱ'_j and transitions of the runs σ_i^ℓ with $i + \ell = j$, synchronizing transitions reading letters from Σ and interleaving transitions reading ε and producing parentheses. Using conditions (4,5,6) on α_j, we can check that following the above order we obtain the run ϱ_j using transitions of types (1,2,3) and such that the projection of ϱ_j on the second component (resp. on component $2 + (j - \ell + 1 \bmod k)$) is ϱ'_j (resp. $\sigma_{j-\ell}^\ell$). Notice that, during the run ϱ_j, the first component is constantly j and the last component starts with j and is then deterministically determined by each transition.

We conclude the proof by showing the upper bound on the number of states of $\mathcal{P}_h$. By Lemma 9 we already know that the upper bound is valid when the expression does not use k-star or reverse k-star. So, again, the only new cases to consider in the induction is when $h = [e, f]^{k\star}$ or $h = [e, f]_r^{k\star}$. In both cases, the parser is the same and its number of states is

$$\|\mathcal{P}_h\| = k^2(\mathsf{nl}(e)+1)(\|\mathcal{P}_f\|+1)^k + (k-1)(\mathsf{nl}(e)+1) + 2\,.$$

Recall that, in both cases, $|h| = 1 + \mathsf{nl}(e) + |f| + k + 1$ and $\mathsf{width}(h) = k \times \mathsf{width}(f) + 2$. Using induction hypothesis $\|\mathcal{P}_f\| \leqslant |f|^{\mathsf{width}(f)}$ and the fact $(a^b + 1) \leqslant (a+1)^b$ when $a, b > 0$, we get

$$\begin{aligned}
\|\mathcal{P}_h\| &= k^2(\mathsf{nl}(e)+1)(\|\mathcal{P}_f\|+1)^k + (\mathsf{nl}(e)+1)(k-1) + 2 \\
&\leqslant k^2(\mathsf{nl}(e)+1)(|f|^{\mathsf{width}(f)}+1)^k + (\mathsf{nl}(e)+1)(k-1) + 2 \\
&\leqslant k^2(\mathsf{nl}(e)+1)(|f|+1)^{k\cdot\mathsf{width}(f)} + (\mathsf{nl}(e)+1)(k-1) + 2\,.
\end{aligned}$$

On the other hand, considering only three terms of the binomial expansion for the first inequality, we have

$$\begin{aligned}
|h|^{\mathsf{width}(h)} &= (k + (\mathsf{nl}(e)+1) + (|f|+1))^{k\cdot\mathsf{width}(f)+2} \\
&\geqslant (k\cdot\mathsf{width}(f)+2)\cdot k \cdot (\mathsf{nl}(e)+1)\cdot(|f|+1)^{k\cdot\mathsf{width}(f)} + 1 + 1 \\
&\geqslant k^2(\mathsf{nl}(e)+1)(|f|^{\mathsf{width}(f)}+1)^k + (\mathsf{nl}(e)+1)(k-1) + 2\,.
\end{aligned}$$

We deduce that $\|\mathcal{P}_h\| \leqslant |h|^{\mathsf{width}(h)}$. ◂

8 Reversible transducer for the unambiguous semantics of RTEs

In this section, we will discuss how to check if a word is in the unambiguous domain of an RTE. As already discussed in Section 3.2, the unambiguous domain $\mathsf{udom}(h)$ of an RTE h is defined as the set of words $w \in \mathsf{dom}(h)$ such that parsing w according to h is unambiguous. Further, from Theorem 4, we know that $\mathsf{udom}(h)$ coincides with $\mathsf{fdom}(\mathcal{P}_h)$, which is the set of words w such that $[\![\mathcal{P}_h]\!]^R(w)$ is a singleton. Making use of this observation, we will check if a word w is in the unambiguous domain of h by checking whether $\mathcal{P}_h$ is functional on w.

Let h be an RTE. Let T_h^o denote the automaton obtained from the parser $\mathcal{P}_h$ by erasing the inputs on transitions and reading the output instead. Recall that from Theorem 4, for each parsing α of w w.r.t. h, the projection of α on Σ is w. Now, we claim that in order to check for functionality of $\mathcal{P}_h$ on a word w, it is sufficient to check whether T_h^o accepts two words $\alpha \neq \beta$ having the same projection w on Σ. In the rest of this section, we will give a construction that checks this and show its correctness. Specifically, we will compute an automaton B_h from T_h^o, such that B_h accepts the language $\mathsf{dom}(h)\backslash\mathsf{udom}(h) = \mathsf{dom}(\mathcal{P}_h)\backslash\mathsf{fdom}(\mathcal{P}_h)$.

Let $\mathcal{P}_h = (Q_h, \Sigma, A, q_I^h, q_F^h, \Delta_h, \mu_h)$ be the 1-way transducer that produces the parsing w.r.t. the expression h. Then, $B = (Q, \Sigma, q_I, F, \Delta)$ where $Q = Q_h \times Q_h \times \{0,1\}$, $q_I = (q_I^h, q_I^h, 0)$. A state (p, q, ν) is accepting, i.e., $(p, q, \nu) \in F$, if we find two runs $p \xrightarrow[+]{\varepsilon|x} q_F^h$ and $q \xrightarrow[+]{\varepsilon|y} q_F^h$ in $\mathcal{P}_h$ with $\Pi_\Sigma(x) = \Pi_\Sigma(y) = \varepsilon$, and in addition, $x \neq y$ if $\nu = 0$. The transition relation Δ is given by the following rules, where $p \xrightarrow[+]{a|xa} p'$ in the premises denotes that there is a sequence of transitions in $\mathcal{P}_h$ that reads a and produces xa.

$$\frac{p \xrightarrow[+]{a|xa} p' \quad q \xrightarrow[+]{a|xa} q' \quad \Pi_\Sigma(x) = \varepsilon}{(p, q, 0) \xrightarrow{a} (p', q', 0)} \ (1)$$

$$\frac{p \xrightarrow[+]{a|xa} p' \quad q \xrightarrow[+]{a|ya} q' \quad x \neq y \quad \Pi_\Sigma(x) = \Pi_\Sigma(y) = \varepsilon}{(p, q, 0) \xrightarrow{a} (p', q', 1)} \ (2)$$

$$\frac{p \xrightarrow[+]{a|xa} p' \quad q \xrightarrow[+]{a|ya} q' \quad \Pi_\Sigma(x) = \Pi_\Sigma(y) = \varepsilon}{(p, q, 1) \xrightarrow{a} (p', q', 1)} \ (3)$$

We will now show that B accepts precisely the set of words that have multiple parsings in $\mathcal{P}_h$.

▸ **Lemma 15.** *$w \in \mathsf{dom}(h)\backslash\mathsf{udom}(h)$ iff B has an accepting run on w.*

Proof. Suppose that B has an accepting run on w. Then, by our construction, T_h^o has two accepting runs α and β, such that $\alpha \neq \beta$ and $\Pi_\Sigma(\alpha) = \Pi_\Sigma(\beta) = w$. This in turn means that $\mathcal{P}_h$ has two runs reading w and producing α and β respectively. Therefore, $\alpha, \beta \in P_h(w)$ and we get $w \in \mathsf{dom}(P_h)\backslash\mathsf{fdom}(P_h) = \mathsf{dom}(h)\backslash\mathsf{udom}(h)$.

Conversely, suppose that $w \in \mathsf{dom}(h)\backslash\mathsf{udom}(h) = \mathsf{dom}(P_h)\backslash\mathsf{fdom}(P_h)$. Let $\alpha, \beta \in P_h(w)$ with $\alpha \neq \beta$. Then, $\mathcal{P}_h$ has two runs reading w and producing α and β respectively. By Theorem 4 we have $\Pi_\Sigma(\alpha) = \Pi_\Sigma(\beta) = w$. Hence, we can write $w = a_1 a_2 \cdots a_n$, $\alpha = u_0 a_1 u_1 \cdots a_n u_n$, and $\beta = v_0 a_1 v_1 \cdots a_n v_n$, where $\Pi_\Sigma(u_i) = \Pi_\Sigma(v_i) = \varepsilon$, for $0 \leqslant i \leqslant n$. Further, let i be the least index such that $u_i \neq v_i$. From the construction, we know that after reading $a_1 \cdots a_i$, the automaton B reaches a state $(p, q, 0)$ (by using rule (1) repeatedly). If

$i = n$, then $(p, q, 0) \in F$ and B accepts w. If $i < n$, then B uses the rule (2) to go to a state $(p', q', 1)$ when reading a_{i+1}. Then, we can use repeatedly rule (3) to read $a_{i+2} \cdots a_n$ and reach a state in F. In both cases, B has an accepting run on w. ◂

Note that the number of states of B is $2\|\mathcal{P}_h\|^2$, where $\|\mathcal{P}_h\|$ denotes the number of states of $\mathcal{P}_h$. From Theorem 4, we know that $\|\mathcal{P}_h\| \leqslant |h|^{\mathsf{width}(h)}$ for general RTEs and $\|\mathcal{P}_h\| \leqslant |h|$ when h does not use Hadamard product, k-star or reverse k-star. Thus, the number of states of B is at most $2|h|^{2\cdot\mathsf{width}(h)}$ in general, and $2|h|^2$ when h does not use Hadamard product, k-star or reverse k-star. Moreover, the construction takes time $\mathsf{poly}(\|\mathcal{P}_h\|)$, which is $\mathsf{poly}(|h|^{\mathsf{width}(h)})$ for general RTEs and $\mathsf{poly}(|h|)$ when h does not use Hadamard product, k-star or reverse k-star.

Using Lemma 2 on B, we obtain a reversible automaton B' with number of states $2 \cdot 2^{|h|^{2\cdot\mathsf{width}(h)}} + 6$ that accepts the complement of $L(B)$, i.e., B' accepts the set of words having at most one parsing w.r.t. h. Using Lemma 1 on $\mathcal{P}_h$, we obtain a uniformizing 2RFT $\mathcal{P}'_h$ of size $2^{\mathcal{O}(|h|^{\mathsf{width}(h)})}$.

Let $\mathcal{P}^U_h$ be defined as the 2RFT that first runs the automaton B' without producing anything, then runs $\mathcal{P}'_h$ if B' has accepted. This transducer is reversible since it is the sequential composition of reversible transducers, and computes the parsing relation on words belonging to the intersection of the domain of the two machines, i.e., $\mathsf{udom}(h)$ the set of words having exactly one parsing. Its number of states is the sum of the number of states of the two machines (+1 dummy state making the transition), hence $\|\mathcal{P}^U_h\| = 2^{\mathcal{O}(|h|^{2\cdot\mathsf{width}(h)})}$.

Thanks to Theorem 4, for any given RTE h, we have the evaluator 2RFT $\mathcal{T}_h$ which, when composed with the parsing relation P_h, computes the relational semantics of h: $[\![h]\!]^R = [\![\mathcal{T}_h]\!] \circ P_h$. Let $\mathcal{T}^U_h$ be the 2RFT obtained by the composition of $\mathcal{T}_h$ and $\mathcal{P}^U_h$ (which restricts the parser to the unambiguous domain of h), i.e., $\mathcal{T}^U_h = \mathcal{T}_h \circ \mathcal{P}^U_h$. Then, $\mathcal{T}^U_h$ computes the unambiguous semantics of h: $[\![h]\!]^U = [\![\mathcal{T}^U_h]\!]$.

Recall that the number of states of the evaluator is $\|\mathcal{T}_h\| = \mathcal{O}(|h| \cdot \mathsf{width}(h))$. Then, as the composition of 2RFTs can be done with polynomial blowup, we get $\mathcal{T}^U_h$ whose number of states is $2^{\mathcal{O}(|h|^{2\cdot\mathsf{width}(h)})}$. Note that, if h does not use Hadamard, k-star or reverse k-star, then both $\|\mathcal{P}_h\|$ and $\|\mathcal{T}_h\|$ are linear in $|h|$. As a consequence, for an RTE h belonging to this fragment, we get $\mathcal{T}^U_h$ whose number of states is $2^{\mathcal{O}(|h|^2)}$.

9 Conclusion

We conclude with some interesting avenues for future work. An immediate future work is to adapt our parser-evaluator construction to work for SDRTE. We believe that this can be done with some effort, preserving our complexity bounds. Note that in this paper, we have already done the work to handle $[e, f]^{k\star}$ even though 2-star was sufficient for RTE; it remains to ensure the aperiodicity of the parser-evaluator, preserving our complexity bounds, to make our construction work for SDRTE. Another interesting question is to see if our approach and construction can be made amenable for extended RTE, which also allows function composition as an operator of the RTE syntax. Note that this does not increase expressiveness, but is a useful shorthand. Already in this paper, we have considered some useful functions like duplicate and reverse along with the fragment RTE [Rat, $\cdot_r, \star_r$]; allowing function composition makes the use of these shorthands meaningful. As we construct reversible machines, composing them is effective and hence having composition as the topmost operator is straight-forward. However, using composition as an operator would require the parsing of intermediate outputs which we leave as an open problem.

References

1. Rajeev Alur, Loris D'Antoni, and Mukund Raghothaman. DReX: A declarative language for efficiently evaluating regular string transformations. In *POPL*, pages 125–137. ACM, 2015.
2. Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In *CSL-LICS*, pages 9:1–9:10. ACM, 2014.
3. Nicolas Baudru and Pierre-Alain Reynier. From two-way transducers to regular function expressions. In *DLT*, volume 11088 of *Lecture Notes in Computer Science*, pages 96–108. Springer, 2018.
4. Nicolas Baudru and Pierre-Alain Reynier. From two-way transducers to regular function expressions. *Int. J. Found. Comput. Sci.*, 31(6):843–873, 2020.
5. Bruno Courcelle. Monadic second-order definable graph transductions: A survey. *Theor. Comput. Sci.*, 126(1):53–75, 1994.
6. Luc Dartois, Paulin Fournier, Ismaël Jecker, and Nathan Lhote. On reversible transducers. In *ICALP*, volume 80 of *LIPIcs*, pages 113:1–113:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
7. Luc Dartois, Paul Gastin, and Shankara Narayanan Krishna. SD-regular transducer expressions for aperiodic transformations. In *LICS*, pages 1–13. IEEE, 2021.
8. Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna. Regular transducer expressions for regular transformations. In *LICS*, pages 315–324. ACM, 2018.
9. Vrunda Dave, Paul Gastin, and Shankara Narayanan Krishna. Regular transducer expressions for regular transformations. *Inf. Comput.*, 282:104655, 2022.
10. Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001.
11. Attila Kondacs and John Watrous. On the power of quantum finite state automata. In *FOCS*, pages 66–75. IEEE Computer Society, 1997.
12. Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.
13. Marcel-Paul Schützenberger. Sur certaines opérations de fermeture dans les langages rationnels. In *Symposia Mathematica, Vol. XV (Convegno di Informatica Teorica, INDAM, Roma, 1973)*, pages 245–253. Academic Press, 1975.