Let $\mathcal{A} = (A, +, \cdot)$ be a semiring. Every mapping $\varphi : T \to A$ for some $T \subseteq T_\Sigma(V)$ is a *tree series*. We denote the set of those by $\mathcal{A}\langle\langle T\rangle\rangle$. We usually write the *coefficient* $\varphi(t)$ *of* t *in* φ *as* (φ, t). Moreover, we write φ as the formal sum $\sum_{t\in T}(\varphi, t)t$. We extend both operations of $\mathcal{A}$ component wise to tree series, *i.e.*, $(\varphi + \psi, t) = (\varphi, t) + (\psi, t)$ for every $\varphi, \psi \in \mathcal{A}\langle\langle T\rangle\rangle$ and $t \in T$. The *support of* φ is $supp(\varphi) = \{t \mid (\varphi, t) \neq 0\}$. The set of tree series with finite support is denoted by $\mathcal{A}\langle T\rangle$. For every $a \in A$, the tree series $\tilde{a}$ is such that $(\tilde{a}, t) = a$ for every $t \in T$. The tree series φ is *nondeleting* (resp., *linear*) in V, if every $t \in supp(\varphi)$ is nondeletinng (resp., linear) in V. We use $var(\varphi)$ as a shorthand for $\bigcup_{t\in supp(\varphi)} var(t)$.

Let $\varphi \in \mathcal{A}\langle T_\Delta(Z)\rangle$ and $\psi_1, \ldots, \psi_k \in \mathcal{A}\langle T_\Delta(Z)\rangle$. The *pure substitution* [19,2] of $(\psi_1, \ldots, \psi_k)$ into φ is defined by

$$\varphi \leftarrow (\psi_1, \ldots, \psi_k) = \sum_{t,t_1,\ldots,t_k\in T_\Delta(Z)} (\varphi, t)(\psi_1, t_1)\cdots(\psi_k, t_k)t[t_1, \ldots, t_k]. \qquad (1)$$

Let $\mathcal{A}$ be a semiring, $\sum$ and Δ be ranked alphabets, and Q a finite set. A *(polynomial) representation* [2] is a family $\mu = (\mu_k \mid k \in \mathbb{N})$ of $\mu_k : \sum_k \to \mathcal{A}\langle T_\Delta(Z)\rangle^{Q\times(Q\times X_k)^*}$ such that for every $\sigma \in \sum_k$ and $q \in Q$

(i). $\mu_k(\sigma)_{q,w} \in \mathcal{A}\langle T_\Delta(Z_{|w|})\rangle$ for every $w \in (Q \times X_k)^*$ and

(ii). $\mu_k(\sigma)_{q,w} = \tilde{0}$ for almost all $w \in (Q \times X_k)^*$.

A *(polynomial) tree series transducer* [1,2] is a tuple $(Q, \Sigma, \Delta, \mathcal{A}, I, \mu)$ such that μ is a representation and $I \subseteq Q$. It is *top-down* (resp., *bottom-up*) [2] if $\mu_k(\sigma)_{q,w}$ is nondeleting and linear in $Z_{|w|}$ [resp., if there exist $q_1, \ldots, q_k \in Q$ such that $w = (q_1, x_1)\cdots(q_k, x_k)$] for every $\sigma \in \Sigma_k, q \in Q$, and $w \in (Q \times X_k)^*$ such that $\mu_k(\sigma)_{q.w} \neq \tilde{0}$. Let $h_\mu : T_\Sigma \to \mathcal{A}\langle\langle T_\Delta\rangle\rangle^Q$ be defined for every $\sigma \in \Sigma_k$, $t_1, \ldots, t_k \in T_\Sigma$, and $q \in Q$ by

$$h_\mu\big(\sigma(t_1, \ldots, t_k)\big)_q = \sum_{w\in(Q\times X_k)^*, w=(q_1,x_{i_1})\cdots(q_n,x_{i_n})} \mu_k(\sigma)_{q,w} \leftarrow \big(h_\mu(t_{i_1})_{q_1}, \ldots, h_\mu(t_{i_n})_{q_n}\big) \qquad (2)$$

The transducer M computes the *tree-to-tree-series transformation* (t-ts transformation) $\tau_M : T_\Sigma \to \mathcal{A}\langle\langle T_\Delta\rangle\rangle$ defined by $\tau_M(t) = \sum_{q\in I} h_\mu(t)_q$ for every $t \in T_\Sigma$. Both h_μ and the t-ts transformation τ_M are well-defined. Finally, the *tree-series-to-tree-series transformation* (ts-ts transformation) computed by M is $\tau_M(\varphi) = \sum_{t\in T_\Sigma}(\varphi, t)\cdot\tau_M(t)$ for every $\varphi \in \mathcal{A}\langle\langle T_\Sigma\rangle\rangle$, whenever this sum is well-defined. We say that τ_M is well-defined whenever $\tau_M(\varphi)$ is well-defined for every $\varphi \in \mathcal{A}\langle\langle T_\Sigma\rangle\rangle$.

1 Convergence

In this section, we will explore when the ts-ts transformation of a tree series transducer $M = (Q, \Sigma, \Delta, \mathcal{A}, I, \mu)$ is well-defined. Roughly speaking, it is well-defined if every output tree $u \in T_\Delta$ can be generated [*i.e.*, $u \in supp(\tau_M(t))$] by only finitely many input trees $t \in T_\Sigma$. Note that our definition of well-definedness works in any semiring; for particular semirings like $(\mathbb{R}, +, \cdot, 0, 1)$ other notions of well-definedness (or equivalently, convergence) might be more realistic.

string in AD. By $||\text{AD}||$ we denote the sum of lengths of all words in AD. A finite automaton (FA) is a quintuple (Q, A, δ, I, F). Q is a finite set of states, A is a finite input alphabet, $F \subseteq Q$ is a set of final states. If FA is nondeterministic (NFA), then δ is a mapping $Q \times (A \bigcup \{\varepsilon\}) \mapsto \mathcal{P}(Q)$ is a set of initial states. A deterministic FA (DFA) is (Q, A, δ, q_0, F) where δ is a (partial) function $Q \times A \mapsto Q$; $q_0 \in Q$ is the only initial state. A finite transducer (FT) is $(Q, A, \Gamma, \delta, q_0, F)$, where δ is a mapping $Q \times (A \bigcup \{\varepsilon\}) \mapsto Q \times (\Gamma \bigcup \{\varepsilon\})$. A regular expression (RE) over finite alphabet A is defined as follows: $\emptyset, \varepsilon$, and a are REs, $\forall a \in A$. Let x, y be REs, then $x+y.x \cdot y, x^*$, and (x) are REs, priority of operators is $+$ (the lowest), $\cdot, *$ (the highest). Priority of evaluation is defined as the count of all symbols in the regular expression except for parentheses and concatenation operator $(\cdot)$ [2].

2 DCA Compression Method

The DCA compression method has been proposed by Crochemore *et al.* [1] in 1999. The antidictionary is a *dictionary of antiwords* - words that do not appear in the text to be coded. Let $T \in \{0,1\}^*$ be the text to be coded. The text is being read from left to right. When a symbol (a bit) is read from the input text and if the suffix of the text read so far is the longest proper prefix of an antiword, nothing is put to the output. Otherwise, the current symbol is output. The text can be decoded since the missing symbol can be inferred from the antiwords.

The coding process is based on finite transducer $B(AD) = (Q_B, \{0,1\}, \{0,1\}, \delta_B, q_{B0}, \emptyset)$. The decoding transducer is created from the encoding transducer by swapping input and output labels on all transitions. Note that an additional information about the text length is required to decode the original text properly. The decoding algorithm is shown in Algorithm 2. An example of the decoding finite transducer is given in Figure 1b).

For the construction of the encoding transducer itself, please refer to [1].

Algorithm 1 Text compression using the DCA compression method

```
Input: Encoding transducer $E(AD) = (Q_E, \{0,1\}, \{0,1\}, \delta_E, q_{E0}, \emptyset)$.
  $q = q_{E0}$
  while not the end of the input do
      let $a$ be the next input symbol
      $(q', u) = \delta_E(q, a)$
      if $u \neq \varepsilon$ then
          print $a$ to the output
      end if
      $q = q'$
  end while
```

Algorithm 2 Pattern matching using deterministic finite automaton

```
Input: Deterministic finite automaton $M = (Q, A, \delta, q_0, F)$.
  $q = q_0$
  $i = 1$
  while not the end of the input do
      if $q \in F$ then
          mark occurrence of the pattern at index $i$
      end if
      let $a$ be the next input symbol
      $q = \delta(q, a)$
      $i = i + 1$
  end while
  if $q \in F$ then
      mark occurrence of the pattern at index $i$
  end if
```

2.1 KMP-Based Pattern Matching in DCA Coded Text

Shibata *et a*l. [4] presented a KMP based approach for pattern matching in the DCA compressed text. As a part of this method,the decoding transducer with ε-transitions is converted to a generalized transducer $G(AD) = (Q_G, 0, 1, 0, 1, \delta_G, q_{G0}, \emptyset)$ without ε-transitions. The main concept of the conversion is to concatenate sequences of transitions, consisting of a non-ε-transition and at least one ε-transition, into a single transition, see Figure **??**.

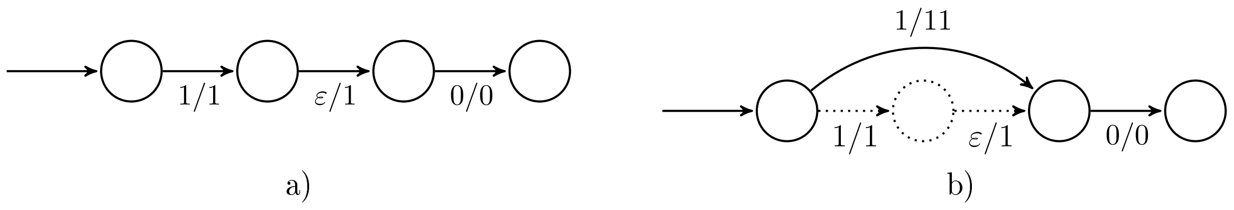

Fig. 1: Generalized decoding transducer construction. a) original decoding transducer, b) generalized decoding transducer

The original decoding transducer may contain infinite sequences of ε-transitions, which are represented by loops of ε-transitions in the decoding transducer, as shown in Figure 3. Note that the infinite ε-transitions sequence can occur only when the very last character of the coded text is being processed. As the uncoded text is of finite length, the sequence of ε-transitions will not be infinite in fact, it is called semi-infinite [4].

The infinite ε-transitions sequence handling is as follows. A special state $\perp \in Q_G$ is defined, and transitions sequences leading into a loop of ε-transitions in the original transducer are redirected into this state. The text decoded by this infinite transitions sequence is always in form of uv^* (where $u, v \in \{0,1\}^*$, u is the prefix decoded before entering the infinite loop, v is the text decoded by the infinite loop), as shown in Figure 3.

Code Φ

variable	codeword		variable	codeword
X_1	B		X_6	AC
X_2	C		X_7	AAA
X_3	D		X_8	AD
X_4	AAD		X_9	AAB
X_5	AB		X_{10}	AAC

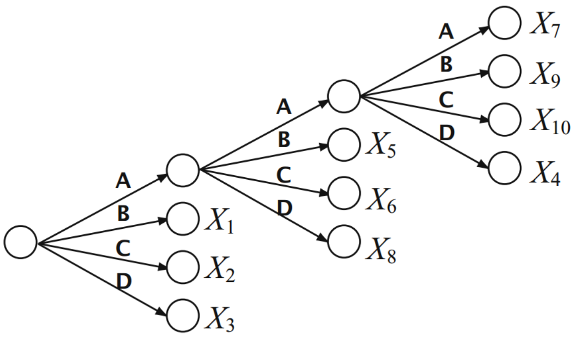

Fig. 2: A prefix code Φ that maps the variables $X_1, \ldots, X_{10}$ to string over $\Gamma = \{A, B, C, D\}$ is shown on the left, and its code tree is displayed on the right

3 Decoder-Embedded CPMA

3.1 Truncation-Free Collage Systems with $\mathcal{S}$ Encoded by Prefix Code

We encode the variables of $\mathcal{S}$ by a prefix code Φ that maps the variables defined in $\mathcal{D}$ to strings over an alphabet Γ. We illustrate our method using an example with $\Sigma = \{a, b, c, d\}$ and $\Gamma = \{A, B, C, D\}$.
Fig. 3 shows a prefix code Φ that maps variables $X_1, \ldots, X_{10}$ to strings over $\Gamma = \{A, B, C, D\}$ of Fig. 2 is encoded as $AD\ D\ AD\ AB\ AAC$. Input to our CPM problem is thus a string over Γ representing $\mathcal{S}$.

The *code tree* τ_Φ of a prefix code Φ that maps the variables defined in $\mathcal{D}$ to strings over a coding alphabet Γ is a trie representing the set of codewords of Φ such that (1) the leaves are one-to-one associated with variables defined in $\mathcal{D}$, and (2) the path from the root to the leaf associated with X_i spells out the codeword $\Phi(X_i)$. In the sequel we assume code trees are *full*, that is, every internal node has exactly $|\Gamma|$ children. Then any code tree with n internal nodes has exactly $(|\Gamma| - 1)n + 1$ leaves, and hence $|\mathcal{D}| = (|\Gamma| - 1)n + 1$.

We shall mention encoded sizes of $\mathcal{D}$ and Φ. A code tree τ_Φ with n internal nodes can be represented by the bit-string obtained during a preorder traversal over it such that every internal node is represented by '0'and every leaf is represented by '1'followed by a $\lceil \log_2 |\mathcal{D}| \rceil$-bit integer indicating the corresponding variable, and thus stored in $|\mathcal{D}|(\lceil \log_2 |\mathcal{D}| + 1) + n$ bits. A dictionary $\mathcal{D}$ can be represented as a list of the right-hand-sides of assignments of D. Suppose that c distinct symbols occurs in the original text, the first c assignments are primitive and the rest are concatenation or repetition. The primitive assignments are stored in $(c+1)\log_2 |\Sigma|$ bits. Each concatenation assignment $X_k \to X_i X_j$ takes $2\lceil \log_2 k \rceil$ bits and each repetition assignment $X_k \to (X_i)^j$ takes $\lceil \log_2 k \rceil + \ell$ bits, where ℓ is the maximum value of $\lceil \log_2 j \rceil$.

3.2 Embedding Decoder into CPMA

Given a pattern P, a dictionary $\mathcal{D}$, a prefix code Φ, and an encoded variable sequence $\Phi(\mathcal{S})$, one naive solution to the CPM problem would be to build CPMA

4 Conclusion and Open Problems

The question we address in this paper — in short, "What can be said about finitely-different automata?"— is a quite natural one. However, it has (until recently) gone unaddressed in the now half-century-long study of DFAs. In this paper, we reviewed the fundamental results in this new area, then provided a significantly improved algorithm for the central problem of hyper-minimization. We conclude with a few open problems:

1. DFA minimization is famously solvable in time $O(n * logn)$. DFCA minimization, too, was quickle reduced from $O(n^4)$ in the original paper [?] to an $O(n * logn)$ algorithm [?]. Can hyper-minimization also be achieved in $O(n * logn)$?

2. There are numerous open problems surrounding finite-factored automata. For example: does the method presented here always result in the smallest total number of states? If not, are some hyper-minimized automata in the same equivalence class better than others?

3. Starting with a minimized DFA, first change the acceptance values of selected preamble states, then minimize the DFA again. It is clear that for some automata, this process can produce a hyper-minimized result. For exactly which automata is this true?

References

1. Badr, A., Geffert, V., Shipman, I.C.: Hyper-minimizing minimized deterministic finite state automata. RAIRO - Theoretical Informatics and Applications (to appear), `http://www.rairoita.org/articles/ita/abs/first/ita07033/ita07033.html`

2. Câmpeanu, C., Santean, N., Yu, S.: Minimal cover-automata for finite languages. In: Workshop on Implementing Automata, pp. 43–56 (1998)

3. Körner, H.: A time and space efficient algorithm for minimizing cover automata for finite languages. International Journal of Foundations of Computer Science 14(2), 1071–1086 (2003)

4. Câmpeanu, C., Paun, A., Smith, J.R.: Incremental construction of minimal deterministic finite cover automata. Theor. Comput. Sci. 363(2), 135-148 (2006)

5. Sipser, M.: Introduction to the Theory of Computation. International Thomson Publishing (1996)

6. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages and Computability. Addison-Wesley Longman Publishinh Co., Inc., Boston (2000)

7. Watson, B.W.: A taxonomy of finite automata minimization algorithms. Technical report (1994)

8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT Press, Cambridge (2001)

Our methodology using higher-order recursive definitions of functional programming language leads to formal proofs amenable to a complete formalization using higher-order logic. Such a formal development is available in the companion paper [?] using the Coq proof assistant [?]. Remark that this methodology leads to the same programs as shown here, by Coqś extraction mechanism from the formal development.

The enumeration of solutions implemented by the above algorithms uses a relatively naive lexicographic ordering. It is easy to refine these algorithms with more complex strategies, yielding weighted automata and stochastic methods such as *hidden Markov chains*. Such experiments will guide the design of the specification language for Eilenberg machines using regular expressions and compilation techniques such as presented by Allauzen and Mohri [?].

Acknowledgments

Gérard Huet and Jean-Baptiste Tristan helped in the elaboration of this paper; I thank them for their participation.

References

1. The Coq proof assistant. Software and documentation (1995–2008),http://coq.inria.fr/
2. Allauzen, C., Mohri, M.: A unified construction of the Glushkov, Follow, and Antimirov automata. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 110–121. Springer, Heidelberg (2006)
3. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Commun. ACM 22(8), 465–476 (1979)
4. Eilenberg, S.: Automata, Languages and Machines, vol. A. Academic Press, London (1974)
5. Huet, G.: A functional toolkit for morphological and phonological processing, applictation to a Sanskrit tagger. J. Functional programming 15 (2005)
6. Huet, G., Razet, B.: The reactive engine for modular transducers. In: Jouannaud, J.-P., Futatsugi, K., Meseguer, J. (eds.) Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 355–374. Springer, Heidelberg (2006)
7. Leroy, X., Doligez, D., Garrigue, J., Vouillon, J.: The Objective Caml system. Software and documentation (1996–2008), http://caml.inria.fr/
8. Theory of X-machines, http://www.x-machines.com
9. Razet, B.: Simulating Eilenberg machines with a reactive engine: Formal specification, proof and program extraction. INRIA Research Report (2008), http://hal.inria.fr/inria-00257352/en/
10. Roche, E., Schabes, Y.: Finite-state language processing. MIT press, Cambridge (1997)
11. Sakarovitch, J.: Eléments de théorie des automates. Vuibert, Paris (2003)