Minimal Partial Languages and Automata[⋆]

F. Blanchet-Sadri[1], K. Goldner[2], and A. Shackleton[3]

[1] Department of Computer Science, University of North Carolina,
P.O. Box 26170, Greensboro, NC 27402–6170, USA
`blanchet@uncg.edu`
[2] Department of Mathematics, Oberlin College,
King 205, 10 N. Professor St, Oberlin, OH 44074, USA
[3] Department of Computer Science, Swarthmore College,
500 College Ave, Swarthmore, PA 19081, USA

Abstract. Partial words are sequences of characters from an alphabet in which some positions may be marked with a "hole" symbol, $\diamond$. We can create a $\diamond$-substitution mapping this symbol to a subset of the alphabet, so that applying such a substitution to a partial word results in a set of full words (ones without holes). This setup allows us to compress regular languages into smaller partial languages. Deterministic finite automata for such partial languages, referred to as $\diamond$-DFAs, employ a limited non-determinism that can allow them to have lower state complexity than the minimal DFAs for the corresponding full languages. Our paper focuses on algorithms for the construction of minimal partial languages, associated with some $\diamond$-substitution, as well as approximation algorithms for the construction of minimal $\diamond$-DFAs.

1 Introduction

Words over some finite alphabet Σ are sequences of characters from Σ and the set of all such sequences is denoted by Σ^* (we also refer to elements of Σ^* as *full words*). The *empty word* ε is the unique sequence of length zero. A *language* over Σ is a subset of Σ^*. The *regular languages* are those that can be recognized by *finite automata*. A *deterministic finite automaton*, or DFA, is a tuple $M = (Q, \Sigma, \delta, s, F)$: a set of states, an input alphabet, a transition function $\delta : Q \times \Sigma \to Q$, a start state, and a set of accept or final states. The machine M accepts w if and only if the state reached from s after reading w is in F. In a DFA, δ is defined for all state-symbol pairs, so there is exactly one computation for any word. In contrast, a *non-deterministic finite automaton*, or NFA, is a tuple $N = (Q, \Sigma, \Delta, s, F)$, where $\Delta : Q \times \Sigma \to 2^Q$ is the transition function that maps state-symbol pairs to zero or more states, and consequently may have zero or more computations on a given word. Additionally, N accepts a word w if *any* computation on w ends in an accept state. Two automata are *equivalent* if they recognize the same language, so every NFA has an equivalent DFA. In general,

[⋆] This material is based upon work supported by the National Science Foundation under Grant No. DMS–1060775.

NFAs allow for a more compact representation of a given language. The *state complexity* of an automaton with state set Q is $|Q|$. If a given NFA has state complexity n, the smallest equivalent DFA may require as many as 2^n states.

Partial words over Σ are sequences of characters from $\Sigma_\diamond = \Sigma \cup \{\diamond\}$, where $\diamond \notin \Sigma$ is a "hole" symbol representing an "undefined" position. A *partial language* over Σ is a subset of $\Sigma_\diamond^*$, the set of all partial words over Σ. A partial language, subset of $\Sigma_\diamond^*$, is associated with a full language, subset of Σ^*, through a $\diamond$-*substitution* $\sigma : \Sigma_\diamond^* \to 2^{\Sigma^*}$, defined such that $\sigma(a) = \{a\}$ for all $a \in \Sigma$, $\sigma(\diamond) \subseteq \Sigma$, $\sigma(uv) = \sigma(u)\sigma(v)$ for all $u, v \in \Sigma_\diamond^*$, and $\sigma(L) = \bigcup_{w \in L} \sigma(w)$ for all $L \subseteq \Sigma_\diamond^*$. A $\diamond$-substitution, then, maps a partial language to a full language and is completely defined by $\sigma(\diamond)$; e.g., if $\sigma(\diamond) = \{a, b\}$ and $L = \{\diamond a, b\diamond c\}$ then $\sigma(L) = \{aa, ba, bac, bbc\}$. By reversing this process, we can compress full languages into partial languages. We can easily extend regular languages to *regular partial languages* as the subsets of $\Sigma_\diamond^*$ that are regular when treating $\diamond$ as a character in the input alphabet. We can recognize them using *partial word DFAs*, introduced by Dassow et al. [4]. A $\diamond$-DFA $M_\sigma = (Q, \Sigma_\diamond, \delta, s, F)$, associated with some σ, is defined as a DFA that recognizes a partial language L, but that is also associated with the full language $\sigma(L)$. Balkanski et al. [1] proved that given a $\diamond$-DFA with state complexity n associated with some σ and recognizing L, the smallest DFA recognizing $\sigma(L)$ may require as many as $2^n - 1$ states.

Given classes of automata $\mathcal{A}, \mathcal{B}$ and a finite automaton A from $\mathcal{A}$, the problem $\mathcal{A} \to \mathcal{B}$-Minimization asks for an automaton B from $\mathcal{B}$ that has the lowest state complexity possible while maintaining $L(A) = L(B)$, i.e., the language that A accepts is the language that B accepts. We will abbreviate $\mathcal{A} \to \mathcal{A}$-Minimization by $\mathcal{A}$-Minimization. Now, let $\mathcal{DFA}$, $\mathcal{NFA}$, and $\diamond$-$\mathcal{DFA}$ be the class of all DFAs, NFAs, and $\diamond$-DFAs, respectively. It is known that $\mathcal{DFA}$-Minimization can be done in $O(n \log n)$ time [7], where n is the number of states in the input DFA, and that $\mathcal{DFA} \to \mathcal{NFA}$-Minimization is $\mathcal{PSPACE}$-complete [8]. Looking at $\diamond$-DFAs as DFAs over the extended alphabet $\Sigma_\diamond$ makes the minimization step easy ($\diamond$-DFAs are DFAs, so $\diamond$-$\mathcal{DFA}$-Minimization is $\mathcal{DFA}$-Minimization), and in general, $\mathcal{A} \to \diamond$-$\mathcal{DFA}$-Minimization and $\diamond$-$\mathcal{DFA} \to \mathcal{A}$-Minimization are not defined because $\diamond$-DFAs accept partial languages (not full languages). We thus define a slightly different problem for $\diamond$-DFAs: given a DFA M, Minimal-$\diamond$-$\mathcal{DFA}$ asks for the smallest $\diamond$-DFA (over all possible $\diamond$-substitutions) associated with $L(M)$. Using the methods from Björklund and Martens [2], it is a simple exercise to show that Minimal-$\diamond$-$\mathcal{DFA}$ is $\mathcal{NP}$-hard, so we discuss an approach to approximating minimal $\diamond$-DFAs. Note also that Holzer et al. [6] have recently further studied the computational complexity of partial word automata problems and have shown that many problems are $\mathcal{PSPACE}$-complete, among them is Minimal-$\diamond$-$\mathcal{DFA}$.

The contents of our paper are as follows: In Section 2, we set our notation and introduce the σ-minimal partial languages given a $\diamond$-substitution σ. In Section 3, we approximate minimal finite partial languages, associated with a $\diamond$-substitution σ, by describing our *Minlang* algorithm. We then prove that running *Minlang* a polynomial number of times in the size of the input with our *Redundancy*

Check algorithm outputs the unique minimal partial language corresponding to the input language given σ. We describe our *Partial Language Check* algorithm that given a $\diamond$-DFA M_σ and a finite language L, verifies that $\sigma(L(M_\sigma)) = L$ and that M_σ is a "contender" for a minimal $\diamond$-DFA for L given σ. We also discuss the algorithms' runtime. In Section 4, we adapt *Minlang* for infinite languages. Finally in Section 5, we conclude with some open problems. Due to the 12-page restriction, some proofs have been omitted and put into an appendix.

2 Approximating Minimal $\diamond$-DFAs

The complexity of a minimal DFA may be exponentially larger than that of a minimal $\diamond$-DFA for the same language. Balkanski et al. [1] gave a construction whereby for any integer $n > 1$ there exists a $\diamond$-DFA with n states such that the minimal DFA for the same language has $2^n - 1$ states. Fig. 1 illustrates their construction for $n = 3$.

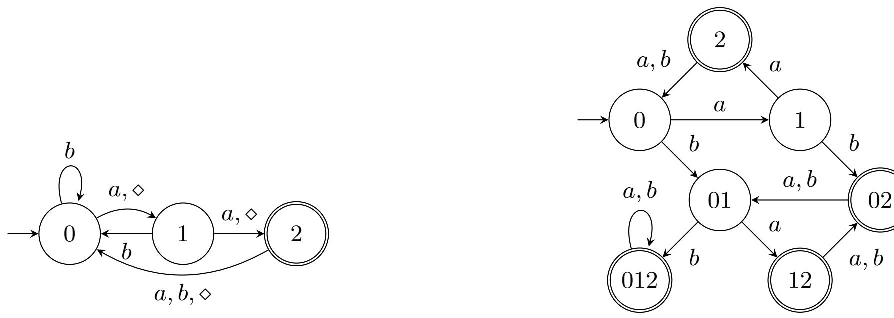

Fig. 1. Left: A 3-state $\diamond$-DFA M_σ, with $\sigma(\diamond) = \{a, b\}$. Right: The smallest DFA M satisfying $L(M) = \sigma(L(M_\sigma))$, which has $2^3 - 1 = 7$ states.

Since the problem of finding a minimal $\diamond$-DFA (over all $\diamond$-substitutions) for the full language L accepted by a given DFA is $\mathcal{NP}$-hard, we give an approximation via minimal partial languages associated with L and $\diamond$-substitutions σ, i.e., *σ-minimal partial languages*. The smallest among associated $\diamond$-DFAs thus provides an approximation for a $\diamond$-DFA with minimal state complexity for L. Before stating our definition, we recall some background material (see [3] for more information).

A partial word u is *contained* in a partial word v, denoted by $u \subset v$, if they have the same length and if a position defined in u is defined by the same letter in v (abbreviate "$u \subset v$, $u \neq v$" by "$u \sqsubset v$"). Partial word u is *compatible* with partial word v, denoted by $u \uparrow v$, if they have the same length and if a position defined in both u and v is defined by the same letter, in which case the *least upper bound* of u and v, denoted by $u \vee v$, is the partial word such that

$u \subset (u \vee v)$, $v \subset (u \vee v)$, and if $u \subset w$ and $v \subset w$ then $(u \vee v) \subset w$. For example, $aa\diamond b\diamond \uparrow a\diamond b\diamond\diamond$ and $(aa\diamond b\diamond \vee a\diamond b\diamond\diamond) = aabb\diamond$. Fixing a $\diamond$-substitution σ over Σ, a set $X \subseteq \Sigma_\diamond^*$ *covers* a partial word w if $x \uparrow w$ for all $x \in X$ and $\sigma(w) \subseteq \sigma(X)$; if X is a singleton $\{x\}$, we abbreviate "X covers w" by "x covers w".

Definition 1. *Let L be a full regular language over alphabet Σ, and let σ be a $\diamond$-substitution over Σ. The* σ-minimal partial language for L *is the unique partial language $L_{\min,\sigma}$ such that*

1. *$\sigma(L_{\min,\sigma}) = L$;*
2. *for all partial languages L' satisfying $\sigma(L') = L$, $|L_{\min,\sigma}| \leq |L'|$;*
3. *the partial words in $L_{\min,\sigma}$ are as weak as possible, i.e., for no partial word $w \in L_{\min,\sigma}$ does there exist x satisfying $\sigma(x) \subseteq L$ and $x \sqsubset w$.*

For each $\diamond$-substitution σ, there exists a partial language $L_{opt,\sigma}$ such that a minimal $\diamond$-DFA recognizing $L_{opt,\sigma}$ is identical to a minimal $\diamond$-DFA for $L = \sigma(L_{opt,\sigma})$. The σ-minimal partial language $L_{\min,\sigma}$ is "close" to $L_{opt,\sigma}$ and, as a result, a minimal $\diamond$-DFA recognizing $L_{\min,\sigma}$ is a "good" approximation for a minimal $\diamond$-DFA for L associated with σ. The more $\diamond$'s we have in our partial words, the more we are taking advantage of the non-determinism that the $\diamond$-DFAs embody.

For convenience of notation, when referring to a particular $\diamond$-substitution σ, we replace σ with $\sigma(\diamond)$, e.g., $\{a\diamond, \diamond b\}$ is an $\{a, b\}$-minimal partial language for $\{aa, ab, bb\}$. Note that $a\diamond$ covers both aa and ab, and $\diamond b$ covers both ab and bb.

3 Computing Minimal $\diamond$-DFAs

We describe our algorithms for approximating minimal $\diamond$-DFAs. The input and output finite languages are represented by listing their words.

3.1 Our *Minlang* algorithm

Algorithm 1, referred to as *Minlang*, is an efficient algorithm for approximating $L_{\min,\sigma}$. Pseudocode is given below, as well as an example of its execution on the full language $L = \{aaa, aab, aac, aba, abb, aca, acb, bac, cac\}$ with $\sigma(\diamond) = \{a, b, c\}$ (see Figs. 2–3). Note that the output L_σ of *Minlang* is not necessarily σ-minimal. In our example, $L_{\min,\sigma} = \{a\diamond a, a\diamond b, \diamond ac\}$ and $L_\sigma = \{aa\diamond, a\diamond a, a\diamond b, \diamond ac\}$, the partial word $aa\diamond$ being redundant. However, *Minlang* is useful as both an approximation and as a stepping stone toward the minimal partial language in the sense of Definition 1 (see Section 3.2).

For any finite language L and $\diamond$-substitution σ, the output of *Minlang* on input L and σ is a tree that can easily be converted to a $\diamond$-DFA with the start state represented by the root node, the accept states by the terminal nodes, and the transitions by the edges. Running standard DFA minimization algorithms on this $\diamond$-DFA results in an approximation of a minimal $\diamond$-DFA for L.

Proposition 1. *The language L_σ output by* Minlang *satisfies $\sigma(L_\sigma) = L$.*

Algorithm 1 *Minlang* Given as input a finite language L over Σ and a $\diamond$-substitution σ, computes a partial language L_σ that approximates $L_{\min,\sigma}$

```
1: put $L$ into a prefix tree with leaf nodes marked as terminals
2: for each node $n$ in a reverse level-order traversal of the tree do
3:   if parent$(n) = u$ has children on every branch of $\sigma(\diamond)$ then
4:     order children of $u$ by non-decreasing height, $c_1, \ldots, c_k$
5:     initialize $\mathcal{C} = \{$all terminal paths from $c_1$ (including $\varepsilon$ if $c_1$ is terminal node)$\}$
6:     for $m \in \{c_2, \ldots, c_k\}$ do
7:       for all $w \in \mathcal{C}$ do
8:         remove $w$ from $\mathcal{C}$
9:         if there is a terminal path from $m$ to $mx$ such that $x \subset w$ then
10:          add $w$ to $\mathcal{C}$
11:        if there is a terminal path from $m$ to $mx$ such that $w \subset x$ then
12:          add $x$ to $\mathcal{C}$ for all such $x$
13:      if $\mathcal{C}$ is empty then
14:        break
15:    if $\mathcal{C}$ is non-empty then
16:      add a $\diamond$-transition from node $u$ to a new node $u\diamond$ and a terminal path from
         node $u\diamond$ to a new node $u\diamond w$, for each $w \in \mathcal{C}$
17:      for each pair $a \in \sigma(\diamond), w \in \mathcal{C}$, start from $uaw$, unmark $uaw$ as a terminal
         node and move upwards, deleting the path until a node is found that has
         more than one child or is terminal
```

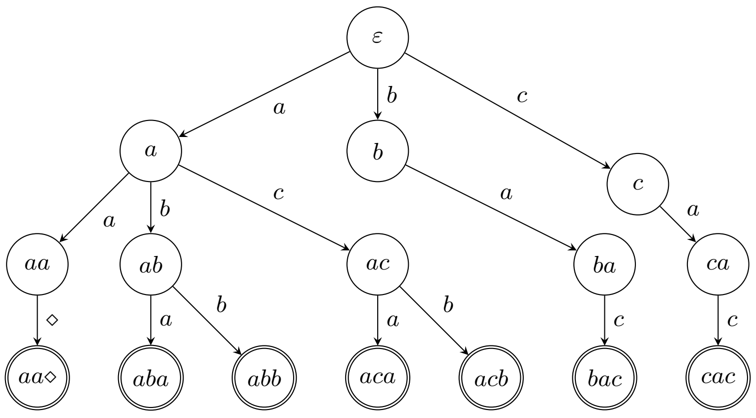

Fig. 2. Starting with the prefix tree for $L = \{aaa, aab, aac, aba, abb, aca, acb, bac, cac\}$, *Minlang* begins at the leaf nodes, compiling $\mathcal{C}$ when a node's parent has children for every letter in $\sigma(\diamond) = \{a, b, c\}$. This consolidates aa's children into a single child, $aa\diamond$. Then *Minlang* examines nodes at reverse depth 1 and their parents, finding children for every letter in $\sigma(\diamond)$ at the node a. Then $\mathcal{C} = \{a, b\}$, adding a transition from a to $a\diamond$ and from $a\diamond$ to children $a\diamond a$ and $a\diamond b$, removing the b and c branches from a, but leaving the a branch as it does not contain any words in $\mathcal{C}$.

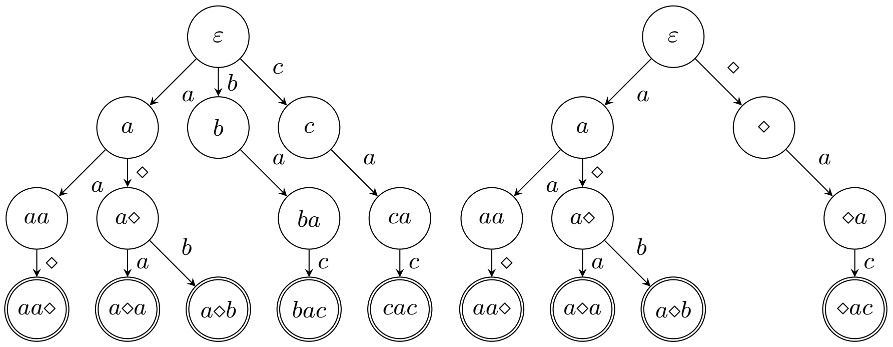

Fig. 3. Left: *Minlang* examines nodes at reverse depth 2 and their parents, finding children for every letter in $\sigma(\diamond)$ at the node ε. Then $\mathcal{C} = \{ac\}$, adding a transition from ε to $\diamond$ and from $\diamond$ a terminal path to $\diamond ac$, removing the b and c branches from ε, but leaving the a branch as it does not contain any words in $\mathcal{C}$. Right: the final tree output by *Minlang*. The words in L_σ are the labels of the terminal nodes in the tree.

Proof. First, a node u in the tree for L has a *representation* x in the tree for L_σ if $u \in \sigma(x)$. Now, the proof is by strong induction on the reverse depth, i.e., the height of the tree minus the depth of a given node. We show that for each $n \geq 1$, all nodes at reverse depth $n-1$ or less in the tree for L_σ have their σ-image in the tree for L implying $\sigma(L_\sigma) \subseteq L$, and all nodes at reverse depth $n-1$ or less in the tree for L have a representation in the tree for L_σ implying $L \subseteq \sigma(L_\sigma)$.

For the inductive step, consider a node u' at reverse depth n, with parent u, in the tree for L_σ. If $u' = ua$ for some $a \in \Sigma$, then by the inductive hypothesis, all nodes uav, where $v \neq \varepsilon$, in the tree for L_σ have their σ-image in the tree for L. So the σ-image of u', i.e., $\sigma(u)a$, is in the tree for L. If $u' = u\diamond$, then all nodes $u\diamond v$, where $v \neq \varepsilon$, in the tree for L_σ have their σ-image in the tree for L. So the σ-image of u', i.e., $\{\sigma(u)a \mid a \in \sigma(\diamond)\}$, is in the tree for L.

Consider a node u' at reverse depth n, with parent u, in the tree for L. First, suppose that *Minlang* finds that from u, there is a transition labeled by a in the tree for L, for each $a \in \sigma(\diamond)$. By the inductive hypothesis, each node uav, where $a \in \sigma(\diamond)$ and $v \neq \varepsilon$, in the tree for L has a representation $x\diamond y$ in the tree for L_σ. So ua, where $a \in \sigma(\diamond)$, has a representation $x\diamond$ in the tree for L_σ. Next, suppose that *Minlang* does not find such transitions from u. Set $u' = ua$ for some $a \in \Sigma$. By the inductive hypothesis, each node uav, where $v \neq \varepsilon$, in the tree for L has a representation xay in the tree for L_σ ($|x| = |u|$ and $|y| = |v|$). So, ua has a representation xa in the tree for L_σ. In either case, u' has a representation in the tree for L_σ. $\square$

The next lemma gives properties of the language output by *Minlang*.

Lemma 1. *The language L_σ output by* Minlang *satisfies the following:*

1. *For $x \in L_\sigma$, there exists some $w \in L_{\min,\sigma}$ such that $x \uparrow w$; similarly, for $w \in L_{\min,\sigma}$, there exists some $x \in L_\sigma$ such that $x \uparrow w$.*

2. *For $w \in L_{\min,\sigma}$, there is no $x \in L_\sigma \setminus L_{\min,\sigma}$ such that $x \subset w$.*
3. *For $w \in L_{\min,\sigma}$ and $x \in L_\sigma$, if $w \subset x$, then $w = x$; consequently, if $w \in L_{\min,\sigma}$, there is no $x \in L_\sigma$ such that $w \sqsubset x$.*
4. *For $w \in L_{\min,\sigma}$ and $x \in L_\sigma$, if $x \uparrow w$, then $w \in L_\sigma$.*

Proof. For Statement 1, by Definition 1 and Proposition 1, $\sigma(L_{\min,\sigma}) = \sigma(L_\sigma) = L$, and let $x \in L_\sigma$. Then $\sigma(x) \subseteq L$, and take $\hat{x} \in \sigma(x)$. Thus $\hat{x} \in L$, and so $\hat{x} \in \sigma(w)$ for some $w \in L_{\min,\sigma}$. Then $x \subset \hat{x}$ and $w \subset \hat{x}$, so by definition, $x \uparrow w$.

For Statement 2, suppose towards a contradiction that for $w \in L_{\min,\sigma}$, $x \in L_\sigma$ and $x \notin L_{\min,\sigma}$, we have $x \subset w$. Since $x \neq w$, we can write $x \sqsubset w$, and since $\sigma(L_\sigma) = L$ by Proposition 1, we know that $\sigma(x) \subseteq L$ contradicting Definition 1(3).

For Statement 3, we show by induction on k that for $w \in L_{\min,\sigma}$ and $x \in L_\sigma$, if $w \subset x$, then the suffix of length k of x equals the suffix of length k of w. For the inductive step, suppose towards a contradiction that $w = w'\diamond v$ and $x = x'av'$ where $a \in \Sigma_\diamond$ and $|v| = |v'| = k$. By the inductive hypothesis, $v = v'$. Since $w' \subset x'$, we have that $\{\sigma(x')b\sigma(v) \mid b \in \sigma(\diamond)\} \subseteq \sigma(w) \subseteq L$. Hence the node $\sigma(x')$ has children on every branch of $\sigma(\diamond)$, and each node $\sigma(x')bv$ is marked as terminal. Thus *Minlang* adds v to $\mathcal{C}$ and iterates over each child $\sigma(x')b$. It adds a $\diamond$-transition from $\sigma(x')$ to $\sigma(x')\diamond$ and a terminal path from $\sigma(x')\diamond$ to $\sigma(x')\diamond v$, and it switches the a to a $\diamond$ in the $(k+1)$th last character's index, a contradiction. Thus the suffix of length $k+1$ of x is identical to the suffix of length $k+1$ of w.

For Statement 4, we show the result by induction on the length of w. Assume that $w \in L_{\min,\sigma}$, $x \in L_\sigma$, and $x \uparrow w$. For the inductive step, suppose $w = cw'$ and $x = c'x'$, with $|c| = |c'| = 1$, such that $x' \uparrow w', c \uparrow c'$. First, suppose that $c \neq \diamond$. Consider the language $L' = \{t' \mid ct' \in L_{\min,\sigma}\}$ and the tree T' that results from applying *Minlang* to the tree for $\sigma(L')$. Then, clearly, $w' \in L'$, so by the inductive hypothesis, T' contains a terminal path to w'. Hence the tree for L_σ contains a terminal path from c to cw', thus $w = cw' \in L_\sigma$. Now, suppose that $c = \diamond$. Then $\sigma(\diamond w') \subseteq L$ implies that $d\sigma(w') \subseteq L$ for all $d \in \sigma(\diamond)$. Then if we consider each language L_{d_σ} constructed from taking the sub-tree of L_σ with d as the root node, we have that $w' \in L_{d_{\min,\sigma}}$, where $L_{d_{\min,\sigma}}$ is a minimal language such that $\sigma(L_{d_{\min,\sigma}}) = \sigma(L_{d_\sigma})$. By Statement 1, there exists some $t' \in L_{d_\sigma}$ such that $w' \uparrow t'$, so by the inductive hypothesis, L_{d_σ} contains a terminal path to w'. Then since every child $d \in \sigma(\diamond)$ contains a path to w', *Minlang* adds $\diamond w' = w$ to L_σ. $\square$

From Lemma 1, we can easily derive the following proposition.

Proposition 2. *The language L_σ output by* Minlang *satisfies $L_{\min,\sigma} \subseteq L_\sigma$.*

Proof. Let $w \in L_{\min,\sigma}$. Then by Lemma 1(1), there exists some $x \in L_\sigma$ such that $x \uparrow w$. By Lemma 1(4), $w \in L_\sigma$. $\square$

Fig. 4 illustrates a minimal $\diamond$-DFA for L_σ, the language output by *Minlang* that has more states than a minimal $\diamond$-DFA for the full language L as a result of redundancies in the *Minlang* approximation.

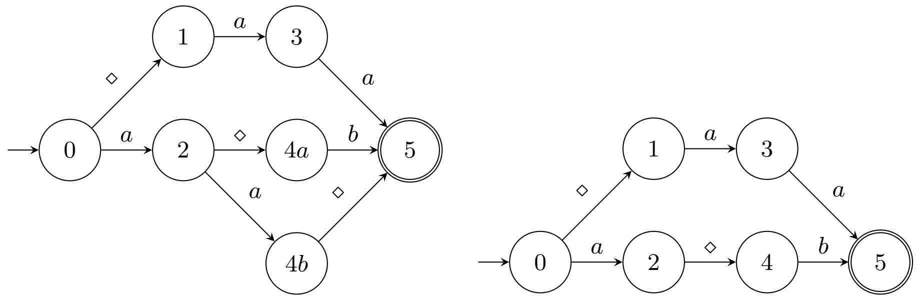

Fig. 4. Left: a minimal $\diamond$-DFA recognizing $L_\sigma = \{\diamond aa, a\diamond b, aa\diamond\}$, where $\sigma(\diamond) = \{a, b\}$, output by *Minlang* has 8 states. Right: a minimal $\diamond$-DFA recognizing $L_{\min,\sigma} = \{\diamond aa, a\diamond b\}$, which is also a minimal $\diamond$-DFA for the full language $L = \sigma(L_\sigma) = \sigma(L_{\min,\sigma})$, has 7 states. Error states are omitted.

3.2 Our *Redundancy Check* algorithm

We describe a second algorithm, referred to as *Redundancy Check*, to fine-tune the result of *Minlang*, guaranteed to output $L_{\min,\sigma}$ exactly. We prove the correctness of the algorithm and give a worst-case runtime bound. Redundancy occurs when a partial word w is already covered by some set $X \subseteq L_{\min,\sigma}$, i.e., $\sigma(w) \subseteq \sigma(X)$. In Algorithm 2, V is the set of suffixes v of partial words $u\diamond v$ in L_σ, where u is a fixed word with no holes, R_a is the set of suffixes r of partial words uar in L_σ, where $a \in \Sigma_\diamond$ and r is compatible with some element of V, and $\overline{r}$ is the part of the image $\sigma(r)$ that is left uncovered by the elements of V. Referring to Figs. 2–3, *Redundancy Check* is illustrated by Fig. 5. *Redundancy Check* maintains the relationship $L_{\min,\sigma} \subseteq L_\sigma$ while removing from L_σ any partial words not in $L_{\min,\sigma}$.

Algorithm 2 *Redundancy Check* Given as input the output L_σ of *Minlang*, computes $L_{\min,\sigma}$

```
1: for all $u\diamond, u \in \Sigma^*$ do
2:     $V = \{v \mid u\diamond v \in L_\sigma\}$
3:     for all children $ua$ of $u$ for $a \in \sigma(\diamond)$ do
4:         compile $R_a = \{r \mid r \uparrow v$ for some $v \in V, uar \in L_\sigma\}$
5:         for all $r \in R_a$ do
6:             let $\overline{r} = \sigma(r) \setminus \{r \vee v \mid r \uparrow v$ for $v \in V\}$
7:             if for every $e \in \overline{r}$ there is a path $uae' \in L_\sigma$ such that $e' \subset e$ then
8:                 delete $r$
9: compile $L_\sigma$ from the tree (every root-to-terminal path)
10: return $L_\sigma$
```

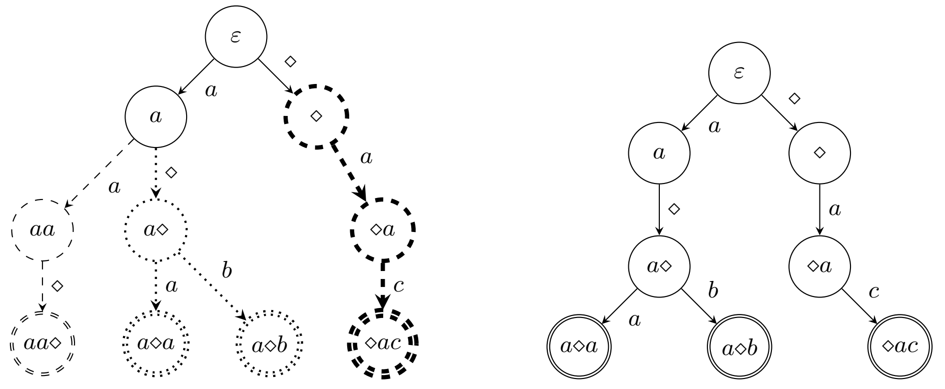

Fig. 5. Left: for $u = \varepsilon$, the thick dashed path gives the set $V = \{ac\}$ from the pseudocode of *Redundancy Check*. The thin dashed path is the only element $r = a\diamond \in R_a$. Then $\overline{r} = \{aa, ab, ac\} \setminus \{ac\} = \{aa, ab\}$. The dotted paths from a labeled $\diamond a, \diamond b$ represent $e' \subset e \in \overline{r}$, hence we delete r. Right: the final language tree for the $\{a, b, c\}$-minimal partial language for L, $L_{\min,\sigma} = \{a\diamond a, a\diamond b, \diamond ac\}$.

Theorem 1. *Given as input a finite language L over Σ and a $\diamond$-substitution σ,* Minlang *followed by* Redundancy Check *returns $L_{\min,\sigma}$. The runtime is polynomial in the size of the input.*

Proof. Recall by Definition 1 that $L_{\min,\sigma}$ is a minimal partial language with its words of the weakest form such that $\sigma(L_{\min,\sigma}) = L$. We claim that *Redundancy Check* removes the elements of the output of *Minlang*, L_σ, that are redundant. It follows directly from Proposition 2 and our claim that $L_\sigma = L_{\min,\sigma}$.

To prove our claim, consider some element x that is removed by our *Redundancy Check*. Thus $x = uar$ for some $u \in \Sigma^*$, $r \in \Sigma_\diamond^*$, and $a \in \sigma(\diamond)$, and there exists $w = u\diamond y \in L_{\min,\sigma}$ such that $y \in \Sigma_\diamond^*$ and $y \uparrow r$. Then for x to be removed, $r \in R_a$, which means that $r \uparrow v$ for some $v \in V$, i.e., $u\diamond v \in L_\sigma$, which is the case since $y \in V$. Then for every $e \in \overline{r}$, there must be some path $uae' \in L_\sigma$ such that $e' \subset e$. But if this is the case, then $uae' \subset uae$ for all such e'. This means precisely that $\sigma(x) \subseteq \sigma(L_\sigma \setminus \{x\})$, and hence $x \notin L_{\min,\sigma}$, so we remove x.

Similarly, if $x \notin L_{\min,\sigma}$ and $x \in L_\sigma$, then there must be some minimal set $X \subseteq L_{\min,\sigma} \subseteq L_\sigma$ such that $\sigma(x) \subseteq \sigma(X)$. If we take an element of X with a hole in the leftmost position of any partial word in X, say $u\diamond v$, we have that $x = uar$ where $a \in \Sigma_\diamond$, and $r \uparrow v$ since $x \uparrow w$ for every $w \in X$. Then clearly $v \in V$ and $r \in R_a$. Any intersection between $\sigma(r)$ and $\{r \vee v\}$ is removed from $\sigma(r)$. Removing from $\sigma(r)$ all elements $r \vee z$ with $r \uparrow z$ and $z \in V$ yields the set $\overline{r}$, so every path in $\overline{r}$ contains a weaker path, and hence x is removed from L_σ. ☐

3.3 Our *Partial Language Check* algorithm

We describe a third algorithm, referred to as *Partial Language Check*, that verifies if $\sigma(L(M_\sigma)) = L$ when given as input a $\diamond$-DFA $M_\sigma = (Q, \Sigma_\diamond, \delta, s, F)$, associated with a $\diamond$-substitution σ, and a finite language L over Σ. A *contender* for a minimal $\diamond$-DFA for a finite language L is a $\diamond$-DFA M_σ such that $|L(M_\sigma)| \leq |L|$.

Algorithm 3 *Partial Language Check* Given a $\diamond$-substitution σ, a $\diamond$-DFA $M_\sigma = (Q, \Sigma_\diamond, \delta, s, F)$, and a finite language L over Σ, checks whether $\sigma(L(M_\sigma)) = L$

```
1: run standard DFA minimization on $M_\sigma$
2: compile the list $P$ of all paths from $s$ to any $f \in F$ (if any grows longer than $\ell$, the
   length of the longest word in $L$, terminate and return false)
3: compile $L'$ from $P$ (if $|L'|$ grows larger than $|L|$, terminate and return false)
4: let $\ell'$ be the length of the longest word in $L'$
5: if $\ell' \neq \ell$ then return false
6: let $L'_\sigma$ be the result of running Minlang on $L'$ and $\sigma$
7: while $L'_\sigma$ continues to change with each pass (at most $\ell$ times) do
8:     run Minlang on $L'_\sigma$
9: run Redundancy Check on $L'_\sigma$
10: run Minlang with Redundancy Check on $L$, creating $L_{\min,\sigma}$
11: for each $w \in L_{\min,\sigma}$ do
12:     if $w \in L'_\sigma$ then delete $w$ from $L'_\sigma$
13:     else return false
14: if $L'_\sigma$ is non-empty then return false
15: else return true
```

Referring to the notation used in the pseudocode of Algorithm 3, for a word $w \in L_{\min,\sigma}$, the *weakest covering set* X for w is the set of those words $x \in L'_\sigma$ such that $x \uparrow w$, L'_σ contains no element z satisfying $z \sqsubset x$, and $\sigma(w) \subseteq \sigma(X)$.

Theorem 2. *Let L be a language over alphabet Σ and σ be a $\diamond$-substitution over Σ. If L' is a partial language such that $\sigma(L') = L$, the language L'_σ produced by running* Minlang *at most ℓ times on L' and then running* Redundancy Check *is equal to $L_{\min,\sigma}$, where ℓ denotes the length of the longest word in L.*

Proof. First, we claim that for any partial language L' such that $\sigma(L') = L$, if L'_σ is the language produced by running *Minlang* as many times as necessary on L', then $L_{\min,\sigma} \subseteq L'_\sigma$. To prove our claim, let $w \in L_{\min,\sigma}$. Since L'_σ is a partial language associated with L, L'_σ contains some weakest covering set X for w. We show that for all $x \in X$ and any factorizations $w = uv$ and $x = u'v'$ where $|v| = |v'|$, we have that $u' \uparrow u$ and $v' \subset v$. We do this by induction on $|v|$. For the inductive step, consider the factorizations $w = uv = uay$ and $x = u'v' = u'a'y'$ where $|a| = |a'| = 1$ and $|y| = |y'|$. By the inductive hypothesis, $u'a' \uparrow ua$ and $y' \subset y$, so $u' \uparrow u$. Suppose $a \neq \diamond$ or $a' = \diamond$. To have $u'a' \uparrow ua$, we must have $a' \subset a$, hence $v' = a'y' \subset ay = v$. Otherwise, since $X \subseteq L'_\sigma$ covers w, $u'by' \in L'_\sigma$ for all $b \in \sigma(\diamond)$, and a pass of *Minlang* clearly results in $u'\diamond y' \in L'_\sigma$. This implies

$u'\diamond y' \sqsubset u'a'y'$, contradicting the fact that X is a weakest covering set for w. Thus, for every $w \in L_{\min,\sigma}$, we have $x \subset w$ for all $x \in X$. By Lemma 1(3), $x = w$, so $w \in L'_\sigma$. Hence $L_{\min,\sigma} \subseteq L'_\sigma$.

Next, we claim that if ℓ is the length of the longest word in L, no more than ℓ passes of *Minlang* are required for our first claim to hold. To see this, if L' is a partial language for L with $\diamond$-substitution σ, the language tree for L' is of height ℓ. Likewise, the tree for L'_σ, the language produced by running *Minlang* on L', is of height ℓ. The only case where we require an additional pass of *Minlang* on L'_σ is when in the previous pass of *Minlang*, some partial word $u\diamond xay$, where $u, x, y \in \Sigma_\diamond^*$ and $a \in \sigma(\diamond)$, is added to L'_σ such that it is then possible to add $u\diamond x\diamond y'$ to L'_σ for some $y' \in \Sigma_\diamond^*$ with $y \subset y'$. As this newly-available addition can only occur at a strictly lower level of the tree than the previous addition, the tree is correctly minimized to depth k by the kth pass. Then, minimization is complete after at most ℓ passes.

By our two claims, $L_{\min,\sigma} \subseteq L'_\sigma$ after at most ℓ passes of *Minlang*. By Theorem 1, we have that *Redundancy Check* on L'_σ removes all redundant elements of L'_σ, resulting in simply $L_{\min,\sigma}$. □

We finally prove *Partial Language Check*'s correctness and runtime.

Theorem 3. *Given as input a finite language L, a $\diamond$-substitution σ, and a $\diamond$-DFA M_σ,* Partial Language Check *runs in polynomial time in the size of the input. It properly verifies that $\sigma(L(M_\sigma)) = L$ and that M_σ is a contender for a minimal $\diamond$-DFA for L given σ.*

Proof. To see that *Partial Language Check* properly verifies that $\sigma(L(M_\sigma)) = L$ and that M_σ is a contender for a minimal $\diamond$-DFA for L given σ, it first minimizes M_σ to optimize runtime. It compiles all paths from the start state s to all accept states $f \in F$ and consolidates the paths into a partial language $L' = L(M_\sigma)$. However, a contender for a minimal $\diamond$-DFA for L never accepts a language larger than L, so if it finds that $|L'|$ has grown larger than $|L|$ at any given point in the compiling of L', it immediately terminates and returns false, as this $\diamond$-DFA is no longer a contender for a minimal $\diamond$-DFA for L.

If the length ℓ' of the longest word in L' is not the length ℓ of the longest word in L, then clearly L' is not a partial language for L, so it suffices to terminate and return false.

The next step is to run *Minlang* followed by *Redundancy Check* on L and *Minlang* at most ℓ times on L' followed by *Redundancy Check*. By Theorem 1 and Theorem 2, this produces the unique σ-minimal partial language for L and for $\sigma(L')$. Hence if $\sigma(L') = L$, then $L_{\min,\sigma} = L'_\sigma$. It then checks if the two languages $L_{\min,\sigma}$ and L'_σ are equal. If not, it returns false, and if so, it returns true (Lines 14–15). Hence it returns true if and only if $\sigma(L(M_\sigma)) = L$ and M_σ is a contender for a minimal $\diamond$-DFA for L given σ. □

4 Adapting *Minlang* for Infinite Languages

We can extend regular expressions to partial words by adding $\diamond$ to the basic regular expressions. This leads naturally to the concept of regular partial languages as the sets of partial words that match partial regular expressions. It is possible to run a slightly modified version of *Minlang* on an infinite language L using the following process. In place of a complete list of the words in L, we use a regular expression for L along with a given $\diamond$-substitution σ.

First, convert a regular expression for our language L into a slightly modified but equivalent form: distribute out unions whenever possible and separate the expression into a list of words that are unioned together at the outer most level. Call this list L, as it is evidently equivalent. Note that the only remaining unions must be inside a Kleene star block. Denote the start of a Kleene star block with "[" and the end of it with "]".

Then, put L into a prefix tree. However, whenever we start a Kleene star block, each element that is unioned together is the child of the "[" character. The end of each element in the Kleene star block has a child to the same joint "]" node that continues on with the suffix of the block.

Next, perform the same algorithm as *Minlang* with respect to the given $\diamond$-substitution σ, except when deleting redundant paths for some word w, if $w = u\,]\,v$ and the "]" node has multiple parents, only delete the nodes relating to u according to the algorithm's requirements and break the tie from the "]" node to its parent in the path of u.

Hence *Minlang* finds all possible $\diamond$'s and removes redundancies in this tree. Then a trained traversal of the tree that matches every "[" node with its descendant balanced "]" node and unions all paths from the same "[" node to the joined "]" node yields all regular expressions with the maximum number of $\diamond$'s in place. We call the resulting list of regular expressions represented, L_σ. The modified algorithm runs in polynomial time of the input regular expression.

This modification of *Minlang* does not produce an equivalent minimal partial language L_σ for the infinite language L. First, such a definition does not make sense, as we cannot produce a language of minimal size, since any partial language for L is infinite. Thus we focus on the equivalent of Definition 1(3): for no partial word $w \in L_\sigma$ does there exist x satisfying $\sigma(x) \subseteq L$ and $x \sqsubset w$. We cannot guarantee that L_σ meets this criterion, as L_σ is dependent on the regular expression used for L and not on the infinite language that the regular expression represents. The problem lies in the representation of a Kleene star block. While $ab(cb)^*b \equiv a(bc)^*bb$, the regular expression $ab(cb)^*b + a(bc)^*ab + a(bc)^*cb$, that uses the former form, finds no $\diamond$'s when the modified *Minlang* is run on it. However, the regular expression $a(bc)^*bb + a(bc)^*ab + a(bc)^*cb$, that uses the latter form, finds $a(bc)^*\diamond b$.

Checking all possible configurations of a loop for every loop used in a regular expression for the language is intractable. We could use a standardized configuration of a loop, such as the unambiguous form from [5]. For a regular expression composed of regular expressions x, y, define $x(yx)^*$ to be the unambiguous form, as a DFA is easily constructed from it. This is opposed to any

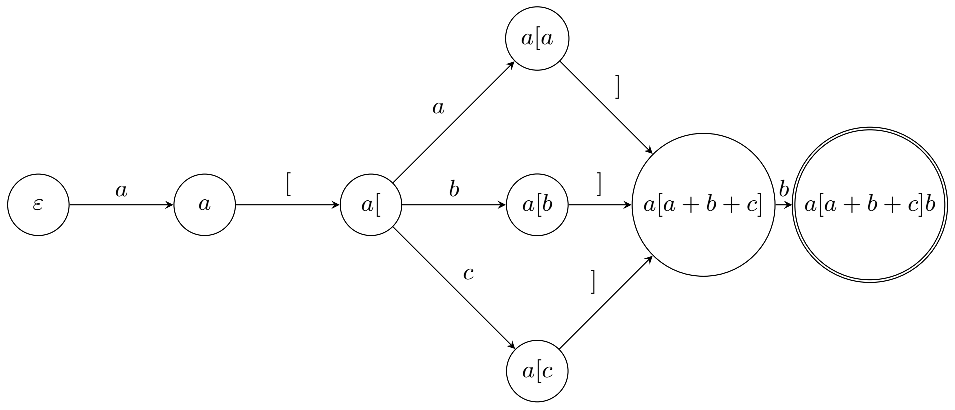

Fig. 6. A language tree representing the regular expression $a(a+b+c)^*b$. Running the modified *Minlang* produces the desired regular expression $a\diamond^*b$.

$u(vyu)^*v$ where $uv = x$. However, even with a standardized unambiguous form, $a(ba)^* + (ab)^*b + (ab)^*c$ finds no $\diamond$'s, while $(ab)^*a + (ab)^*b + (ab)^*c$ finds $(ab)^*\diamond$.

5 Conclusion and Open Problems

The choice of a $\diamond$-substitution σ can vastly change the state complexity of a minimal $\diamond$-DFA, associated with σ, for a given DFA. Fig. 7 illustrates different $\diamond$-substitutions resulting in different state complexities for minimal $\diamond$-DFAs, associated with them. An open problem is to develop computational techniques for selecting an *optimal* $\diamond$-substitution σ for a given DFA M, that is, optimality occurs when a minimal $\diamond$-DFA for $L(M)$, associated with σ, has the same state complexity as a minimal $\diamond$-DFA for $L(M)$ over all possible $\diamond$-substitutions. Because a solution to the σ-CHOICE problem is defined in terms of a solution to the MINIMAL-$\diamond$-$\mathcal{DFA}$ problem, which is $\mathcal{NP}$-hard, it does not make sense to define or attempt to solve the σ-CHOICE problem separately from the MINIMAL-$\diamond$-$\mathcal{DFA}$ problem.

Another open problem is the one of extending $\diamond$-DFAs. In light of the understanding that $\diamond$-DFAs are weakly non-deterministic, it makes sense to ask whether meaningful extensions of the class $\diamond$-$\mathcal{DFA}$ exist, and what properties those extensions might have. In particular, what would happen if we created additional $\diamond$-like symbols, say $\diamond_1, \ldots, \diamond_k$?

A World Wide Web server interface has been established at

`www.uncg.edu/cmp/research/planguages2`

for automated use of a program that given a $\diamond$-substitution σ and a full language L, computes the σ-minimal partial language for L. This is our own implementation, we do not use any known automata library.

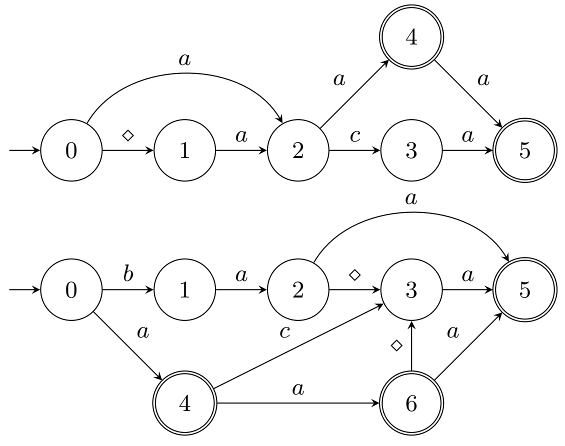

Fig. 7. Top: A minimal ◇-DFA, associated with $\sigma(\diamond) = \{a, b\}$, having 7 states including the error state, a sink non-accept state. Bottom: A minimal ◇-DFA for the same full language, associated with $\sigma(\diamond) = \{a, c\}$, having 8 states including the error state.

References

1. Balkanski, E., Blanchet-Sadri, F., Kilgore, M., Wyatt, B.J.: Partial word DFAs. In: Konstantinidis, S. (ed.) CIAA 2013, 18th International Conference on Implementation and Application of Automata. Lecture Notes in Computer Science, vol. 7982, pp. 36–47. Springer-Verlag, Berlin, Heidelberg (2013)
2. Björklund, H., Martens, W.: The tractability frontier for NFA minimization. Journal of Computer and System Sciences 78, 198–210 (2012)
3. Blanchet-Sadri, F.: Algorithmic Combinatorics on Partial Words. Chapman & Hall/CRC Press, Boca Raton, FL (2008)
4. Dassow, J., Manea, F., Mercaş, R.: Regular languages of partial words. Information Sciences 268, 290–304 (2014)
5. Groz, B., Maneth, S., Staworko, S.: Deterministic regular expressions in linear time. In: PODS 2012, 31th ACM Symposium on Principles of Database Systems. pp. 49–60 (2012)
6. Holzer, M., Jakobi, S., Wendlandt, M.: On the computational complexity of partial word automata problems. IFIG Research Report 1404, Institut für Informatik, Justus-Liebig-Universität Gießen, Arndtstr. 2, D-35392 Gießen, Germany (May 2014)
7. Hopcroft, J.E.: An n log n algorithm for minimizing states in a finite automaton. Tech. rep., DTIC Document (1971)
8. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM Journal on Computing 22, 1117–1141 (1993)