Minimal Partial Languages and Automata

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

September 19, 2017

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 total 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 total 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.

Keywords: Automata and formal languages; Regular languages; Partial languages; Partial words; Deterministic finite automata; Non-deterministic finite automata.

1 Introduction

Words, sequences of characters from some alphabet, are natural objects in computer science, mathematics, biology, physics, bioinformatics, music, linguistics, etc. They can model data that can be stored in linear files, they can

[1]Department of Computer Science, University of North Carolina, P.O. Box 26170, Greensboro, NC 27402–6170, USA, `blanchet@uncg.edu`

[2]Computer Science & Engineering, University of Washington, Box 352350, Seattle, WA 98195–2350, USA, `ksgoldner@gmail.com`

[3]Department of Computer Science, Swarthmore College, 500 College Ave, Swarthmore, PA 19081, USA, `shackletonaidan@gmail.com`

approximate biological sequences, to name a few. Words have been extensively studied and several generalizations have been proposed in the literature which include *strings with don't-cares*, introduced in 1974 [12]. Strings with don't-cares are also referred to as *partial words*, whose combinatorics have been studied since 1999 [4, 8]. Although *languages*, or sets of words, have also been extensively studied (see, e.g., [25]), little is known on languages of partial words (see, e.g., [3, 7, 11, 16, 20, 21]).

Words over some finite alphabet Σ are (finite or infinite) sequences of characters from Σ and the set of all finite sequences is denoted by Σ^* (we also refer to elements of Σ^* as *total 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 $\delta : Q \times \Sigma \to Q$, an initial state, and a set of final states. The function δ can be extended to $\hat{\delta} : Q \times \Sigma^* \to Q$, where $\hat{\delta}(q, w)$ is the state reached from q after reading w. The machine M accepts w if and only if $\hat{\delta}(s, w) \in F$. In a DFA, δ is defined for all state-symbol pairs, so there is exactly one computation for any word. In contrast, a *nondeterministic 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 a final state. Two automata are *equivalent* if they recognize the same language, so every NFA has an equivalent DFA. In general, NFAs allow 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 total language, subset of Σ^*, through a $\diamond$*-substitution* $\sigma : \Sigma_\diamond^* \to 2^{\Sigma^*}$, defined such that $\sigma(a) = \{a\}$ for all $a \in \Sigma$ and $\sigma(\diamond) \subseteq \Sigma$. A $\diamond$-substitution, then, maps a partial language to a total language and is completely defined by $\sigma(\diamond)$; e.g., if $\sigma(\diamond) = \{a, b\}$ and $L = \{\diamond a, b\diamond c\}$ then $\sigma(L) = \{aa, ba, bac, bbc\}$. So a partial word is just a particular kind of star-free regular expression where we use the letters, the union, the concatenation, and some subset of the alphabet. In the example above where the subset is $\{a, b\}$, which is defined by the regular expression $a + b$, the partial word $\diamond a$ is equivalent to the regular expression $aa + ba \equiv (a + b)a$ while $(a + b)a + b(a + b)c$ is a regular expression that defines the partial language L using standard notations. By reversing this process, we can compress total 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*. 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 total language $\sigma(L)$. Deterministic finite automata (including $\diamond$-DFAs) often include an "error state", i.e., a sink non-final state. While we account for error states when determining the state complexity of an automaton, we frequently omit them in visual representations of automata for the sake of simplicity.

Motivated by the compression of DFAs into smaller machines, Dassow et al. [11] introduced the $\diamond$-DFAs. Then Balkanski et al. [3] analyzed their state complexity and proved, in particular, 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 $\mathbb{L}(A) = \mathbb{L}(B)$, 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 [17], where n is the number of states in the input DFA, and that $\mathcal{DFA} \to \mathcal{NFA}$-MINIMIZATION is $\mathcal{PSPACE}$-complete [18]. 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 total 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 $\mathbb{L}(M)$. Using the methods from Björklund and Martens [6], it is a simple exercise to show that MINIMAL-$\diamond$-$\mathcal{DFA}$ is PSPACE-complete, so we discuss an approach to approximating minimal $\diamond$-DFAs. We give an approximation via minimal partial languages associated with $\mathbb{L}(M)$ and $\diamond$-substitutions σ. The smallest among associated $\diamond$-DFAs thus provides an approximation for a $\diamond$-DFA with minimal state complexity for $\mathbb{L}(M)$. Note also that Holzer et al. [16] 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 review background material on partial words and languages. In Section 3, we set our notation and introduce the σ-minimal partial languages given a $\diamond$-substitution σ. In Section 4, we approximate minimal finite partial languages, associated with a $\diamond$-substitution σ, by describing our *Minlang* algorithm. We then prove that *Minlang* outputs a $\frac{3}{2}$-approximation for the unique minimal partial language corresponding to the input total language given σ. In Section 5, we show that running *Minlang*, a polynomial number of passes in the size of the input, with our *Redundancy Removal* algorithm produces the minimal partial language. In Section 6, we describe our *Partial Automaton Check* algorithm that given a $\diamond$-DFA M_σ and a finite language L, verifies that $\sigma(\mathbb{L}(M_\sigma)) = L$ and that M_σ is a "contender" for a minimal $\diamond$-DFA for L given σ. We also discuss all our algorithms' runtime. In Section 7, we adapt *Minlang* for infinite languages. Finally in Section 8, we conclude with some open problems.

2 Partial Words and Languages

We recall some background material related to partial words (see [8] for more information).

Let Σ be a finite alphabet and set $\Sigma_\diamond = \Sigma \cup \{\diamond\}$, where $\diamond \notin \Sigma$ is a "hole" symbol representing an "undefined" position. A *partial word* w over Σ is a sequence of characters from $\Sigma_\diamond$. The set of defined positions of w, denoted by $D(w)$, is the set of non-hole positions of w, while $H(w)$ is the set of hole positions of w. A *total word* is a partial word without holes. For example, $ab\diamond bbc\diamond ba$ is a partial word with two holes over $\{a, b, c\}$; here $D(ab\diamond bbc\diamond ba) = \{0, 1, 3, 4, 5, 7, 8\}$ and $H(ab\diamond bbc\diamond ba) = \{2, 6\}$ (here the labelling of positions starts at 0).

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 "$u \subsetneq v$"). For example, $aa\diamond b\diamond \subsetneq aaab\diamond$. Clearly, if $u \subsetneq v$ then $D(u) \subsetneq D(v)$. The set of partial words over Σ together with the binary relation $\subset$ is a poset. 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 $u \cup v$, is the partial word such that $u \subset (u \cup v)$, $v \subset (u \cup v)$, and if $u \subset w$ and $v \subset w$ then $(u \cup v) \subset w$. Note that $D(u \cup v) = D(u) \cup D(v)$. For example, $aa\diamond b\diamond \uparrow a\diamond b\diamond\diamond$ and $(aa\diamond b\diamond \cup a\diamond b\diamond\diamond) = aabb\diamond$. Similarly, the *greatest lower bound* of u and v, denoted by $u \cap v$, is the partial word such that $(u \cap v) \subset u$, $(u \cap v) \subset v$, and if $w \subset u$ and $w \subset v$ then $w \subset (u \cap v)$. Note that $D(u \cap v) = D(u) \cap D(v)$. For example, $(aa\diamond b\diamond \cap a\diamond b\diamond\diamond) = a\diamond\diamond\diamond\diamond$.

A *total language* over Σ is a subset of Σ^*, the set of all total words over

Σ, while a *partial language* over Σ is a subset of $\Sigma_\diamond^*$, the set of all partial words over Σ. A partial language is associated with a total language through a $\diamond$-*substitution* σ over Σ. A $\diamond$-substitution $\sigma : \Sigma_\diamond^* \to 2^{\Sigma^*}$ satisfies $\sigma(a) = \{a\}$ for all $a \in \Sigma$, $\sigma(\diamond) \subseteq \Sigma$, $\sigma(uv) = \sigma(u)\sigma(v)$ for all $u, v \in \Sigma_\diamond^*$, and

$$\sigma(L) = \bigcup_{w \in L} \sigma(w) \text{ for all } L \subseteq \Sigma_\diamond^*.$$

A $\diamond$-substitution, then, maps a partial language to a total language and is completely defined by $\sigma(\diamond)$. For example, if $\sigma(\diamond) = \{a, b\}$ and $L = \{\diamond ac, b\diamond b\diamond\}$ then $\sigma(L) = \{aac, bac, baba, babb, bbba, bbbb\}$.

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". For example, if $\sigma(\diamond) = \{a, b\}$, then $a\diamond$ covers both aa and ab.

We can recognize partial languages by using partial word DFAs. A $\diamond$-*DFA* $M_\sigma = (Q, \Sigma_\diamond, \delta, s, F)$, associated with some $\diamond$-substitution σ, is defined as a DFA that recognizes a partial language L, but that is also associated with the total language $\sigma(L)$. A total word u in $\sigma(L)$ has a *representation* x if x is a partial word in L such that $u \in \sigma(x)$. The $\diamond$-DFAs employ a limited amount of "non-determinism" in the sense that any letter $a \in \Sigma$ can be represented by a transition labelled by a or by $\diamond$. Figure 1 (Top) shows a $\diamond$-DFA, where $b\diamond\diamond ba\diamond \in L$ and $\sigma(b\diamond\diamond ba\diamond) = \{baabaa, baabab, \ldots, bbbbab\}$ contains eight elements. If we were to use this $\diamond$-DFA to find a representation of $baabab$, we could use the path 0012012 labelled by $b\diamond\diamond ba\diamond$, or the path 0012012 labelled by $baaba\diamond$, or the path 0120012 labelled by $\diamond\diamond\diamond ba\diamond$, etc.

3 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. [3] gave a general 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. Figure 1 illustrates their construction for $n = 3$.

Since the problem of finding a minimal $\diamond$-DFA (over all $\diamond$-substitutions) for the total language L accepted by a given DFA is PSPACE-complete, 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.

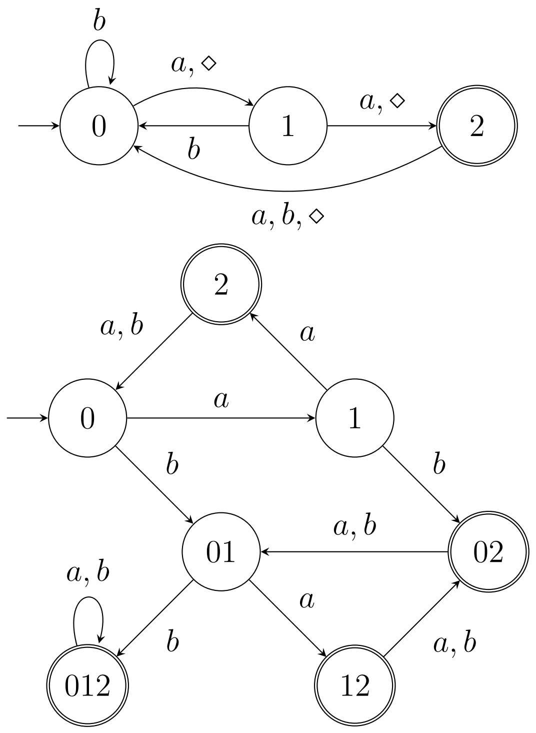

Figure 1: Top: A 3-state $\diamond$-DFA M_σ, with $\sigma(\diamond) = \{a, b\}$. Bottom: The smallest DFA M satisfying $\mathbb{L}(M) = \sigma(\mathbb{L}(M_\sigma))$, which has $2^3 - 1 = 7$ states.

Definition 1. *Let L be a total 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 $x \subsetneq w$.*

Note that 2 does not imply 3. To see this, consider the total language $L = \{aa, ab, ba\}$. Let $\sigma(\diamond) = \{a, b\}$, and let K be the partial language $\{a\diamond, ba\}$. Clearly, $\sigma(K) = L$. The inequality $|K| \leq |L'|$ is true for all partial languages L' satisfying $\sigma(L') = L$, so 2 holds. However, the partial words in K are not as weak as possible; for $w = ba \in K$, there exists $x = \diamond a$ such that $\sigma(x) \subseteq L$ and $x \subsetneq w$. So 3 does not hold. In this example, $L_{\min,\sigma} = \{a\diamond, \diamond a\}$.

The following proposition shows the uniqueness of a partial language satisfying 1, 2 and 3.

Proposition 1. *Let L be a total regular language over alphabet Σ, and let σ be a $\diamond$-substitution over Σ. The partial language $L_{\min,\sigma}$ from Definition 1 is the unique partial language that satisfies 1, 2 and 3 for L.*

Proof. Let L_1 and L_2 be two partial languages such that

1. $\sigma(L_1) = \sigma(L_2) = L$;

2. for all partial languages L' satisfying $\sigma(L') = L$, both $|L_1| \leq |L'|$ and $|L_2| \leq |L'|$ hold;

3. the partial words in L_1 (respectively, L_2) are as weak as possible, i.e., for no partial word $w \in L_1$ (respectively, L_2) does there exist x satisfying $\sigma(x) \subseteq L$ and $x \subsetneq w$.

By 1 and 2, we can deduce that $|L_1| = |L_2|$. So to show that $L_1 = L_2$, it is enough to show that $L_1 \subseteq L_2$. Let $w \in L_1$. Since $\sigma(L_1) = L$ by 1, we have $\sigma(w) \subseteq L$, and let $u \in \sigma(w)$. Thus $u \in L$, and since $\sigma(L_2) = L$ by 1, we have $u \in \sigma(x)$ for some $x \in L_2$. Then $w \subset u$ and $x \subset u$, so by definition of compatibility, $w \uparrow x$. Note that both $\sigma(w) \subseteq L$ and $\sigma(x) \subseteq L$.

Suppose towards a contradiction that $w \neq x$. The containment $w \subsetneq x$ would contradict 3 that the partial words in L_2 are as weak as possible, while the containment $x \subsetneq w$ would contradict 3 that the partial words in L_1 are as weak as possible. Then $w = x$, so $w \in L_2$. □

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.

In the next three sections, we describe our algorithms for approximating minimal $\diamond$-DFAs. The input and output finite languages are represented by listing their words.

4 Our *Minlang* Algorithm

We describe a first algorithm, referred to as *Minlang*, to efficiently approximate the unique minimal partial language for a language with respect to a $\diamond$-substitution. Given as input a finite (total or partial) language L over Σ and a $\diamond$-substitution σ, *Minlang* computes a partial language L_σ such that $\sigma(L_\sigma) = L$ and L_σ approximates $L_{\min,\sigma}$. *Minlang* puts $\diamond$'s where possible, removing explicit "redundancies". It first puts L into a language tree. Then it traverses the tree starting at reverse depth 0, at the height of the tree. While examining nodes at some reverse depth, it considers the question "Do the parents have children on all branches of $\sigma(\diamond)$?". If the answer is yes, *Minlang* consolidates these children into one partial word.

Pseudocode is given below, as well as an example of its execution on the total language $L = \{aaa, aab, aac, aba, abb, aca, acb, bac, cac\}$ with $\sigma(\diamond) = \{a, b, c\}$ (see Figures 2–5). Note that the output L_σ of *Minlang* is not necessarily σ-minimal. A partial word w in L_σ is called *redundant* if there exists a subset X of $L_{\min,\sigma}$ that covers w and is such that for all $x \in X$, neither $x \subset w$ nor $w \subset x$. 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 $aa\diamond$ is 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 5).

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 initial state represented by the root node, the final states by the terminal nodes, and the transitions by the edges. Running standard DFA minimization al-

Algorithm 1 *Minlang* Given as input a finite (total or partial) 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
          a 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
```

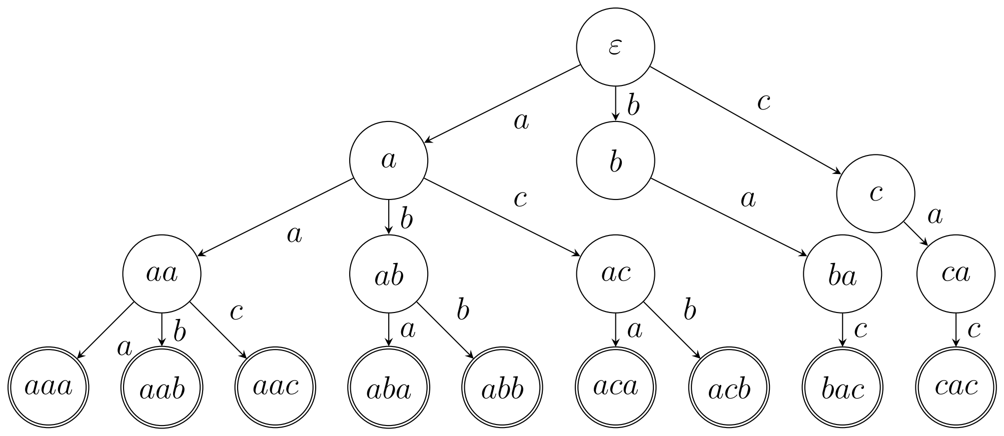

Figure 2: Step 1: The initial prefix tree for $L = \{aaa, aab, aac, aba, abb, aca, acb, bac, cac\}$. *Minlang* begins at the leaf nodes, compiling $\mathcal{C}$ when a node's parent has children for every letter in $\sigma(\diamond) = \{a, b, c\}$. This consolidates the children of aa into a single child, $aa\diamond$.

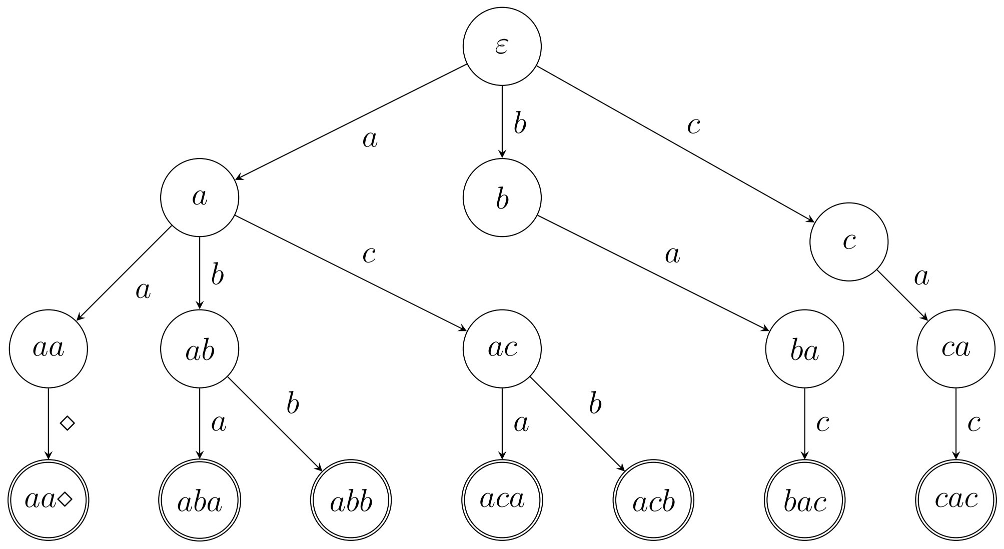

Figure 3: Step 2: *Minlang* examines nodes at reverse depth 1 and their parents, finding children for every letter in $\sigma(\diamond)$ at the node a. Then $\mathcal{C} = \{a, b\}$, adding a transition from a to $a\diamond$ and from $a\diamond$ to children $a\diamond a$ and $a\diamond b$, removing the b and c branches from a, but leaving the a branch as it does not contain any words in $\mathcal{C}$.

gorithms on this $\diamond$-DFA results in an approximation of a minimal $\diamond$-DFA for L.

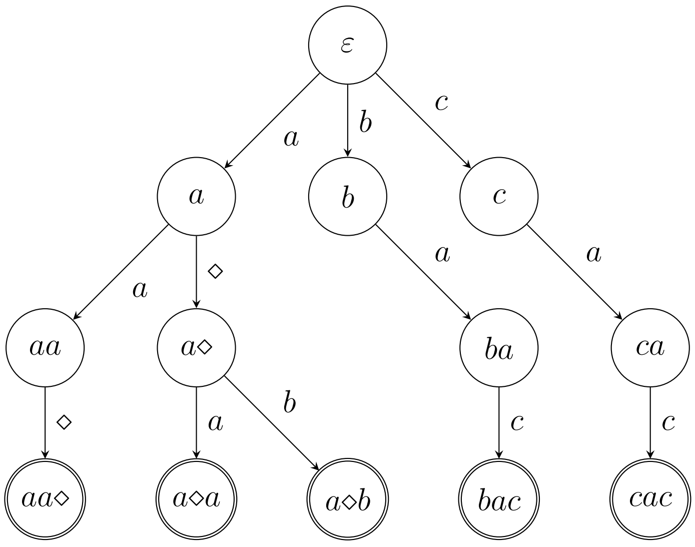

Figure 4: Step 3: *Minlang* examines nodes at reverse depth 2 and their parents, finding children for every letter in $\sigma(\diamond)$ at the node ε. Then $\mathcal{C} = \{ac\}$, adding a transition from ε to $\diamond$ and from $\diamond$ a terminal path to $\diamond ac$, removing the b and c branches from ε, but leaving the a branch as it does not contain any words in $\mathcal{C}$.

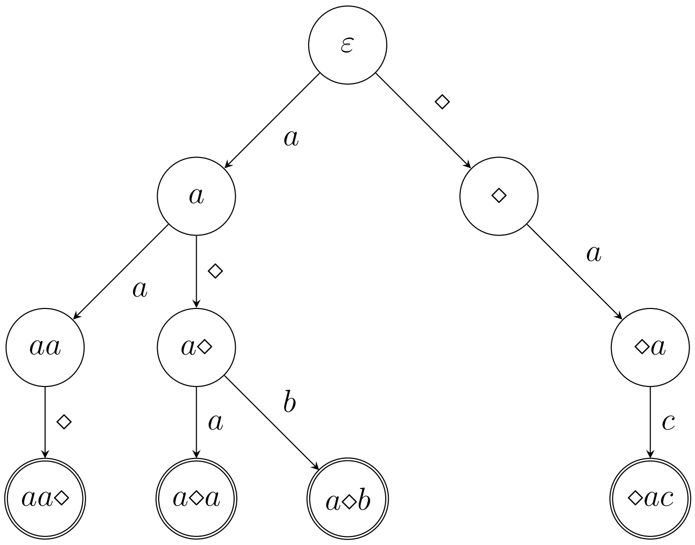

Figure 5: Step 4: The final tree output by *Minlang*. The words in the partial language L_σ are the labels of the terminal nodes in the tree.

Proposition 2. *The language L_σ output by* Minlang *satisfies $\sigma(L_\sigma) = L$.*

Proof. First, we say that 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 base case, consider a node u' at reverse depth 0, with parent u, in the tree for L_σ. Here u' is at a depth equal to the height of the tree. If $u' = ua$ for some $a \in \Sigma$, then $\{xa \mid x \in \sigma(u)\}$ is in the tree for L, while if $u' = u\diamond$, then $\{xa \mid x \in \sigma(u), a \in \sigma(\diamond)\}$ is in the tree for L. Now, consider a node u' at reverse depth 0, with parent u, in the tree for L. If *Minlang* finds that from u, there is a transition labeled by a in the tree for L, for each $a \in \sigma(\diamond)$, then each such ua has a representation that ends with $\diamond$ in the tree for L_σ. And if *Minlang* does not find such transitions from u, then $u' = ua$ for some $a \in \Sigma$, and u' has a representation that ends in a in the tree for L_σ.

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., $\{xa \mid x \in \sigma(u)\}$, 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., $\{xa \mid x \in \sigma(u), a \in \sigma(\diamond)\}$, is in the tree for L.

Consider a node u' at reverse depth n, with parent u, in the tree for L. Suppose that *Minlang* finds that from u, there is a transition labeled by a, for each $a \in \sigma(\diamond)$. By the inductive hypothesis, each node uav, where $v \neq \varepsilon$, has a representation $x\diamond y$ in the tree for L_σ such that $|x| = |u|$ and $|y| = |v|$, and so ua has a representation $x\diamond$. Otherwise, set $u' = ua$ for some $a \in \Sigma$. By the inductive hypothesis, each node uav, where $v \neq \varepsilon$, has a representation xay in the tree for L_σ such that $|x| = |u|$ and $|y| = |v|$, and so ua has a representation xa. In either case, u' has a representation in the tree for L_σ. □

The next four lemmas give properties of the language L_σ output by *Minlang*.

Lemma 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 $x \uparrow w$.*

Proof. By Definition 1 and Proposition 2, $\sigma(L_{\min,\sigma}) = \sigma(L_\sigma) = L$, 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, $x \uparrow w$. □

Lemma 2. *For $w \in L_{\min,\sigma}$, there is no $x \in L_\sigma \setminus L_{\min,\sigma}$ such that $x \subset w$.*

Proof. 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 $x \subsetneq w$, and since $\sigma(L_\sigma) = L$ by Proposition 2, we know that $\sigma(x) \subseteq L$ contradicting Definition 1(3). □

Lemma 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 $w \subsetneq x$.*

Proof. 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 base case, suppose towards a contradiction that $w = w'\diamond \in L_{\min,\sigma}$ and $x = x'a \in L_\sigma$, where $a \in \Sigma$ and $x' \in \Sigma^*$, are such that $w \subset x$. Then since $\sigma(w) \subseteq L$ and $w' \subset x'$, we have that $\{x'b \mid b \in \sigma(\diamond)\} \subseteq \sigma(w) \subseteq L$. Hence the node x', that represents a path from the root node to x', has children on every branch of $\sigma(\diamond)$, with each child marked as a terminal node. Then *Minlang* adds ε to $\mathcal{C}$ and iterates over each child, never removing ε, and hence adding a $\diamond$-transition from x' to $x'\diamond$ and unmarking the $\sigma(\diamond)$ children of x' as terminal nodes, possibly deleting them if they do not branch, hence removing all redundant words from $\{x'b \mid b \in \sigma(\diamond)\}$. Thus *Minlang* switches the a to a $\diamond$ in the last character's index, a contradiction. So the suffix of length one of x is identical to the suffix of length one of w.

For the inductive step, suppose towards a contradiction that $w = w'\diamond v \in L_{\min,\sigma}$ and $x = x'av' \in L_\sigma$, where $w \subset x$, $a \in \Sigma$, $x' \in \Sigma^*$, and $|v| = |v'| = k$. By the inductive hypothesis, $v = v'$. Since $w' \subset x'$, we have that $\{x'by \mid b \in \sigma(\diamond), y \in \sigma(v)\} \subseteq \sigma(w) \subseteq L$. Hence the node x' has children on every branch of $\sigma(\diamond)$, and each node $x'bv$ is marked as terminal. Thus *Minlang* adds v to $\mathcal{C}$ and iterates over each child $x'b$, never removing v. It adds a $\diamond$-transition from x' to $x'\diamond$ and unmarks the $x'bv$ nodes as terminal nodes, possibly deleting them and the path to them if they contain no other branches to differing paths or are not terminal nodes. *Minlang* then adds a terminal path from $x'\diamond$ to $x'\diamond v$, so it switches the a to a $\diamond$ in the $(k+1)$th last character's index, a contradiction. We conclude that the suffix of length $k+1$ of x is identical to the suffix of length $k+1$ of w. □

Lemma 4. *For $w \in L_{\min,\sigma}$ and $x \in L_\sigma$, if $x \uparrow w$, then $w \in L_\sigma$.*

Proof. 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 base case, consider $|w| = 1$. Suppose towards a contradiction that $x \subsetneq w$. By Proposition 2, $\sigma(L_\sigma) = L$ and so since $x \in L_\sigma$, we have $\sigma(x) \subseteq L$.

By Definition 1(3), we reach a contradiction. Thus, $w \subset x$ and by Lemma 3, $w = x$. So $w \in L_\sigma$.

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 $d \in \sigma(\diamond)$. If we consider each language L_{d_σ} constructed from taking the sub-tree of the tree for 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 Lemma 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 these lemmas, we can easily derive the following proposition.

Proposition 3. *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, there exists some $x \in L_\sigma$ such that $x \uparrow w$. By Lemma 4, $w \in L_\sigma$. $\square$

Figure 6 illustrates a minimal $\diamond$-DFA for L_σ, the language output by *Minlang* that has more states than a minimal $\diamond$-DFA for the total language L as a result of redundancies in the *Minlang* approximation.

Given that *Minlang* can output an imperfect approximation even for simple $\diamond$-DFAs as a result of redundancies, it is important to establish a worst-case bound for the error in the approximation of $L_{\min,\sigma}$.

Theorem 1. *The partial language L_σ output by* Minlang *is a $\frac{3}{2}$-approximation of $L_{\min,\sigma}$.*

Proof. For every redundant element $w \in L_\sigma$, there must be at least $|\sigma(\diamond)|$ words in $L_{\min,\sigma}$ that cover it. Recall that for every word x in the covering set $X_w \subseteq L_{\min,\sigma}$ of w, to make w redundant, we must have $x \uparrow w$ but neither $x \subset w$ nor $w \subset x$. Hence every x contains a hole that w does not. However, throughout the set X_w, the holes must occur in at least two different indices, otherwise a word strictly weaker than w would be in a partial language for L, and hence w would not be output by *Minlang*, i.e., there can be at most $|\sigma(\diamond)| - 1$ words in X_w with a hole at a certain index when w does not have a hole there.

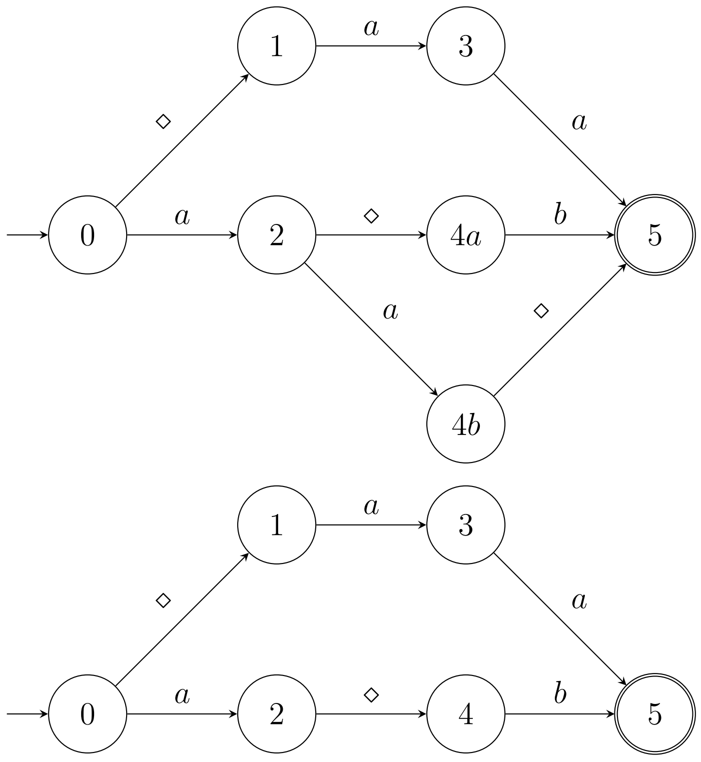

Figure 6: Top: A minimal $\diamond$-DFA recognizing $L_\sigma = \{\diamond aa, a\diamond b, aa\diamond\}$, where $\sigma(\diamond) = \{a, b\}$, output by *Minlang* has 8 states. Bottom: A minimal $\diamond$-DFA recognizing $L_{\min,\sigma} = \{\diamond aa, a\diamond b\}$, which is also a minimal $\diamond$-DFA for the total language $L = \sigma(L_\sigma) = \sigma(L_{\min,\sigma})$, has 7 states. Error states are omitted. The partial word $aa\diamond$ is redundant.

We aim to maximize the redundancy ratio $\frac{|L_\sigma|}{|L_{\min,\sigma}|}$ to show the worst case of redundancy in L_σ and to find a bound for L_σ as an approximation of $L_{\min,\sigma}$. The smallest example of redundancy is illustrated in Figure 6. The redundancy ratio is precisely $\frac{3}{2}$ there.

To exaggerate the redundancy ratio, we attempt to expand the number of redundant words in L_σ while adding as few words as possible to $L_{\min,\sigma}$. We consider a construction of these two sets that maximizes the ratio. For the redundant words, we choose w and w' that have differing non-hole characters at one index, say i, but are otherwise identical. The covering set X_w of w must have words w_1, with hole at position j_1, and w_2, with hole at position j_2, $j_1 \neq j_2$, where j_1 and j_2 are not holes in w. A similar statement is true for the covering set $X_{w'}$ of w'. Letting $|\sigma(\diamond)| - 1$ of the words have a hole at position i is clearly optimal in order to maximize the intersection between X_w and $X_{w'}$. Then X_w (respectively, $X_{w'}$) also requires a word w_0 (respectively, w'_0) with a hole at an index different from i, and w_0 and w'_0 must be distinct, as they must match w and w', respectively, at position i.

We can create up to $|\sigma(\diamond)| - 1$ redundant words in this way, adding $|\sigma(\diamond)|$ words to $L_{\min,\sigma}$ for the first and an additional word for each of the others. If we were to create $|\sigma(\diamond)|$ such words, though, *Minlang* would add a $\diamond$ at the position where the redundant words differ. Suppose we have k redundant words. We have that $L_{\min,\sigma}$ contains the $|\sigma(\diamond)| - 1$ words with a hole at the differing index and k variations of the word without a hole at that index to cover the k redundant words. Hence $|L_{\min,\sigma}| = |\sigma(\diamond)| - 1 + k$ and $|L_\sigma| = |L_{\min,\sigma}| + k$. Clearly the ratio is the largest when k is largest. Hence, in the worst case, $|L_{\min,\sigma}| = |\sigma(\diamond)| - 1 + |\sigma(\diamond)| - 1$ and $|L_\sigma| = |\sigma(\diamond)| - 1 + |\sigma(\diamond)| - 1 + |\sigma(\diamond)| - 1$ which is precisely a redundancy ratio of $\frac{3}{2}$. □

To illustrate Theorem 1, let $\sigma(\diamond) = \{a, b, c\}$. Consider $X_1 = \{a\diamond a, a\diamond b, a\diamond c\}$ as the set that covers $w = aa\diamond$. Clearly each $x \in X_1$ satisfies $x \uparrow w$ but neither $x \subset w$ nor $w \subset x$. However, our choice of X_1 implies that $\sigma(a\diamond\diamond) \subseteq L$, and hence $w \notin L_\sigma$ by Lemma 3. Thus, we can have at most $|\sigma(\diamond)| - 1$ words in the covering set, with a hole at a certain index when w does not have a hole there, that have a hole at the same index. Take for example, $X_2 = \{a\diamond a, a\diamond b, \diamond ac\}$ to be the set that covers $w = aa\diamond$. Consider expanding the set of redundant words to $\{aa\diamond, \diamond ab\}$. We would then need $|\sigma(\diamond)|$ elements to cover the hole in $aa\diamond$ and $|\sigma(\diamond)|$ more elements to cover the hole in $\diamond ab$. A better candidate for worst case is $\{aa\diamond, ab\diamond\}$. Then $a\diamond a$ and $a\diamond b$ cover both, so we only need variations $\diamond ac$ and $\diamond bc$. Thus, $L_\sigma = \{a\diamond a, a\diamond b, \diamond ac, \diamond bc, aa\diamond, ab\diamond\}$ and $L_{\min,\sigma} = \{a\diamond a, a\diamond b, \diamond ac, \diamond bc\}$, yielding the worst-case redundancy ratio of $\frac{3}{2}$. This may seem a large error bound, and indeed it is. Even in this example, though, we have $|L| = |\sigma(L_\sigma)| = 12$, so *Minlang* reduces the size of the language by half, and the ratio of input size to output size for redundancy-maximizing input increases with $|\sigma(\diamond)|$.

We next state *Minlang*'s runtime.

Theorem 2. *The runtime of* Minlang *is polynomial in the size of the input.*

Proof. Let L be a finite language over constant-sized alphabet Σ with $|L| = n$, let σ be a $\diamond$-substitution, and let ℓ be the length of the longest word in L. If $|\Sigma| = 2$, then we prove that $O(\ell n + n^4)$ is an upper bound on the runtime of *Minlang* on input L and σ; if $|\Sigma| > 2$, then we prove that the bound can be lowered to $O(\ell n + n^3)$.

We first note the worst-case runtime occurs when the tree is as dense as possible; i.e., when n is close to $|\Sigma|^\ell$. This may seem counter-intuitive, a larger input leading to a smaller tree is often a good thing, but since the bulk of the computation involves comparing paths to leaves and other terminal nodes, a denser tree results in increased computation for a node at a given depth.

In the worst case, each node k distance from its lowest leaf can have no more than $(|\Sigma|+1)^k$ descendant leaves, and a total of no more than $2(|\Sigma|+1)^k$ terminal paths. After compiling $\mathcal{C}$ from these paths, *Minlang* must then look up weaker and stronger terminal paths for each $w \in \mathcal{C}$ in each of $|\sigma(\diamond)| - 1$ siblings of the current node. For a given sibling, for a given w, finding these paths takes $O((|\sigma(\diamond)|+1)^k)$ time. The time used for actual tree modifications is dwarfed by this lookup phase. Finally, *Minlang* must check no more than $|\Sigma|^{\ell-k-1}$ nodes for a given k (one per node at the level above the current one). Since, in this situation, the worst-case runtime occurs when $\sigma(\diamond) = \Sigma$, the total runtime for *Minlang* is given by

$$O\left(\ell n + \textstyle\sum_{k=0}^{\ell-1}(|\Sigma|+1)^k|\Sigma|(|\Sigma|+1)^{k+1}|\Sigma|^{\ell-k-1}\right) = O\left(\ell n + (|\Sigma|+1)^{2\ell}\right).$$

Because n is approximately $|\Sigma|^\ell$, we have $(|\Sigma|+1)^{2\ell} < n^3$ when $|\Sigma| \geq 3$, and $(|\Sigma|+1)^{2\ell} < n^4$ when $|\Sigma| = 2$. This gives the desired bounds. $\square$

5 Our *Redundancy Removal* Algorithm

In general, *Minlang* produces a good approximation for $L_{\min,\sigma}$, but we can do better. We describe a second algorithm, referred to as *Redundancy Removal*, 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., $w \notin X$ and $\sigma(w) \subseteq \sigma(X)$. 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 Figures 2–5, *Redundancy Removal* is illustrated by Figure 7.

Algorithm 2 *Redundancy Removal* Given as input L_σ, the output 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 $\overline{r} = \sigma(r) \setminus \{r \cup v \mid r \uparrow v$ 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 $L_\sigma$
```

Recall that by Proposition 3, $L_{\min,\sigma} \subseteq L_\sigma$. We next prove that *Redundancy Removal* maintains the relationship $L_{\min,\sigma} \subseteq L_\sigma$ while removing from L_σ any partial words not in $L_{\min,\sigma}$.

Theorem 3. *Given as input a finite language L over Σ and a $\diamond$-substitution σ,* Minlang *followed by* Redundancy Removal *returns $L_{\min,\sigma}$.*

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 Removal* removes the elements of the output of *Minlang*, L_σ, that are redundant. It follows directly from Proposition 3 and our claim that $L_\sigma = L_{\min,\sigma}$.

To prove our claim, consider some element x that is removed by our *Redundancy Removal.* 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 $uae' \in L_\sigma$ for all such e', and $uae' \subset uae$. 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

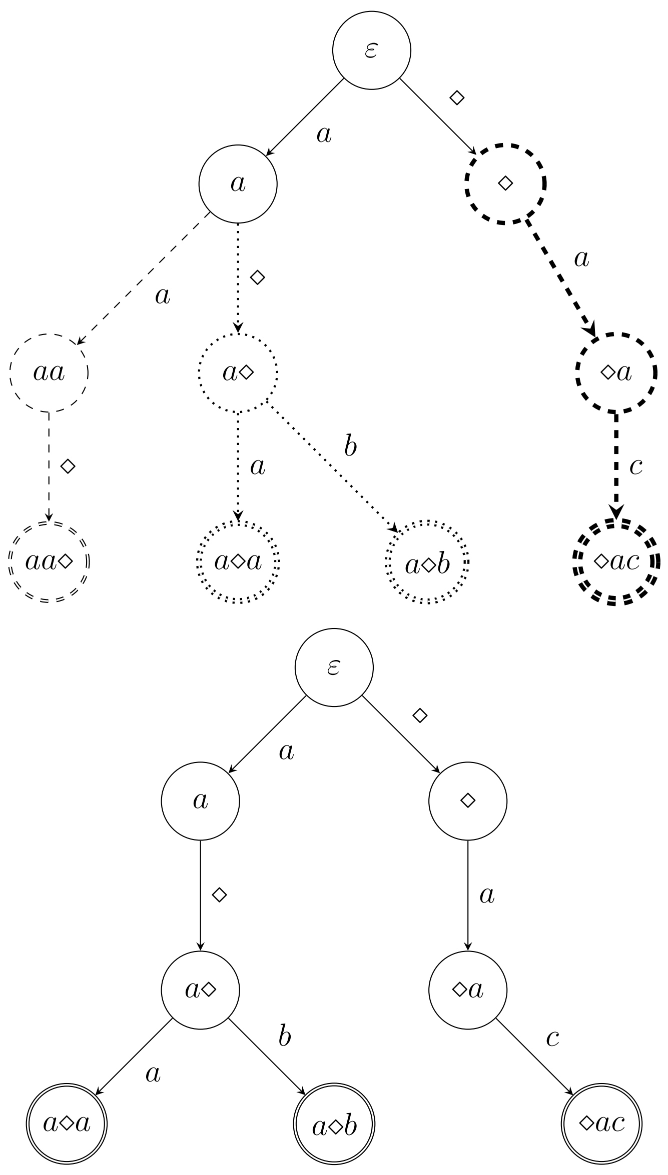

Figure 7: Top: For $u = \varepsilon$, the thick dashed path gives the set $V = \{ac\}$ from the pseudocode of *Redundancy Removal*. The thin dashed path is the only element $r = a\diamond \in R_a$. Then $\overline{r} = \{aa, ab, ac\} \setminus \{ac\} = \{aa, ab\}$. The dotted paths from a labeled $\diamond a, \diamond b$ represent $e' \subset e \in \overline{r}$, hence we delete r. Bottom: The final language tree for the $\{a, b, c\}$-minimal partial language for L, $L_{\min,\sigma} = \{a\diamond a, a\diamond b, \diamond ac\}$.

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 $\{r \cup v\}$ is removed from $\sigma(r)$. Removing from $\sigma(r)$ all elements $r \cup z$ 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.

It follows that after the redundancy removal, there are no elements $x \in L_\sigma$ such that $x \notin L_{\min,\sigma}$. □

We finally state *Minlang* with *Redundancy Removal*'s runtime. For "dense" languages (i.e., when the cardinality of the language is large relative to the length of its longest word), the bound on runtime is significantly worse than that for *Minlang* without *Redundancy Removal*; however, the existence of a polynomial bound for an optimal minimization algorithm is itself a significant result.

Theorem 4. *The runtime of* Minlang *followed by* Redundancy Removal *is polynomial in the size of the input.*

Proof. Let L be a finite language over constant-sized alphabet Σ with $|L| = n$, let σ be a $\diamond$-substitution, and let ℓ be the length of the longest word in L. If $|\Sigma| = 2$, then we prove that $O(\ell n^4 + n^5)$ is an upper bound on the runtime of *Minlang* followed by *Redundancy Removal* on input L and σ; if $|\Sigma| > 2$, then we prove that the bound can be lowered to $O(\ell n^3 + n^4)$.

For the redundancy removal specifically, the worst case occurs when a $\diamond$-transition is present in (nearly) all nodes, but a-transitions remain for nearly all $a \in \sigma(\diamond)$. Then the tree contains $O(|\Sigma|^k)$ paths with a $\diamond$-transition at depth k, and finding all terminals from a depth-k node requires $O(|\Sigma|^{\ell-k})$ time. Thus, we derive the following bounds:

Task/Loop	Bound
Number of possible $u\diamond$ at depth k	$(\lvert\Sigma\rvert + 1)^k$
Compiling V for given $u\diamond$	$O((\lvert\Sigma\rvert + 1)^{\ell-k})$
Number of possible children ua	$\lvert\Sigma\rvert$
Compiling R_a for given a	$O((\ell - k)(\lvert\Sigma\rvert + 1)^{\ell-k}(\lvert\Sigma\rvert + 1)^{\ell-k})$
Size of R_a	$(\lvert\Sigma\rvert + 1)^{\ell-k}$
Compiling $\overline{r}$	$O((\ell - k + \lvert\Sigma\rvert^{\ell-k})(\lvert\Sigma\rvert + 1)^{\ell-k})$
Checking $\overline{r}$ against $uae' \in L_\sigma$	$O((\ell - k)(\lvert\Sigma\rvert + 1)^{\ell-k}(\lvert\Sigma\rvert + 1)^{\ell-k})$
Removing r from the tree	$O(\ell - k)$

While some of these bounds are obvious, a few merit discussion. Compiling R_a requires comparing all terminal suffixes of a with all elements of V. Both sets have $O(|\Sigma|^{\ell-k})$ elements, and a compatibility check requires

$O(\ell - k)$ time, so the stated bound follows. For the compilation of $\overline{r}$, generating $\sigma(r)$ requires $O(|\Sigma|^{\ell-k})$ time (with the upper bound reached when r is entirely composed of $\diamond$ symbols), while computing the set of least upper bounds requires performing $O(\ell - k)$ comparison and replacement operations between r and each element of V. The resulting set has size $O(|\Sigma|^{\ell-k})$, so calculating the associated set of total words (through application of σ) takes $O(|\Sigma|^{\ell-k}|\Sigma|^{\ell-k})$ time. The resulting set is by definition a subset of $\sigma(r)$, so finding $\overline{r}$ using set difference algorithms takes $O((\ell - k)|\Sigma|^{\ell-k})$ time, and the bound follows.

An upper bound on the total runtime of *Redundancy Removal* is then given by

$$\begin{aligned}
&O\Bigg(\sum_{k=1}^{\ell}(|\Sigma|+1)^{k}((|\Sigma|+1)^{\ell-k}+|\Sigma|((\ell-k)(|\Sigma|+1)^{\ell-k}(|\Sigma|+1)^{\ell-k}\\
&\qquad\qquad +(|\Sigma|+1)^{\ell-k}((\ell-k+|\Sigma|^{\ell-k})(|\Sigma|+1)^{\ell-k}\\
&\qquad\qquad +(\ell-k)(|\Sigma|+1)^{\ell-k}(|\Sigma|+1)^{\ell-k}+\ell-k))\Bigg)\\
&\quad = O\left(\ell(|\Sigma|+1)^{2\ell}+|\Sigma|^{\ell}(|\Sigma|+1)^{2\ell}\right).
\end{aligned}$$

Recalling from the runtime analysis of *Minlang* that $(|\Sigma| + 1)^{2\ell}$ is $O(n^3)$ for $|\Sigma| > 2$ and $O(n^4)$ for $|\Sigma| = 2$, we arrive at the stated bounds. Since the bounds dominate those of *Minlang*, the total runtime bound for *Minlang* followed by *Redundancy Removal* is $O(\ell n^3 + n^4)$ for ternary and larger alphabets and $O(\ell n^4 + n^5)$ for binary alphabets. □

6 Our *Partial Automaton Check* Algorithm

We describe a third algorithm, referred to as *Partial Automaton Check*, that verifies if $\sigma(\mathbb{L}(M_\sigma)) = L$ when given as input a $\diamond$-DFA M_σ, associated with a $\diamond$-substitution σ, and a finite language L over Σ.

We say that a $\diamond$-DFA $(Q, \Sigma_\diamond, \delta, s, F)$, associated with a $\diamond$-substitution σ, has *reduced transition complexity* if for all $p, q \in Q$ such that $\delta(p, \diamond) = q$, there is no $a \in \sigma(\diamond)$ such that $\delta(p, a) = q$. Figure 8 illustrates this concept of reduced transition complexity.

A *contender* for a minimal $\diamond$-DFA for a finite language L is a $\diamond$-DFA M_σ such that $|\mathbb{L}(M_\sigma)| \le |L|$, in order to comply with Proposition 4.

Proposition 4. *Let L' be the partial language accepted by a minimal $\diamond$-DFA with reduced transition complexity associated with a $\diamond$-substitution σ for a finite language L. Then $|L'| \le |L|$.*

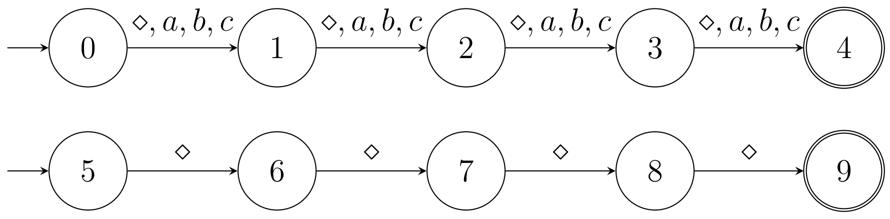

Figure 8: Top: A $\diamond$-DFA without reduced transition complexity recognizing the partial language $\{\diamond, a, b, c\}^4$. Bottom: A $\diamond$-DFA with reduced transition complexity recognizing the partial language $\{\diamond\diamond\diamond\diamond\}$.

Proof. Let $M = (Q_M, \Sigma, \delta_M, s_M, F_M)$ be a DFA that accepts a finite language L and let σ be a $\diamond$-substitution. Let $M_\sigma = (Q, \Sigma_\diamond, \delta, s, F)$ be the $\diamond$-DFA constructed from M. For a state q, we define the suffix set of q, denoted by $S(q)$, to be the set of partial words on which state q transitions to a final state. For a state p, we define a *superstate* p of state q to be such that $S(q) \subseteq S(p)$. Let L' be the partial language recognized by M_σ.

First, we claim that in the construction of M_σ, any $\diamond$-transition added shares or replaces an existing transition for a letter in $\sigma(\diamond)$, i.e., any $\diamond$-transition can only be placed on the same edge as an existing transition and the state out of which the $\diamond$ transitions must contain transitions for every letter in $\sigma(\diamond)$. We prove our claim by induction on $|H(u)|$, where $u\diamond v$ is a partial word in $L' \setminus L$ with $u, v \in \Sigma_\diamond^*$.

For the base case, suppose $|H(u)| = 0$. Let $\hat{\delta}(s, u) = p$ and for the added $\diamond$-transition, let $\delta(p, \diamond) = q$. Then for all $v' \in S(q)$, $u\diamond v' \in L'$, hence $uav' \in L$ for all $a \in \sigma(\diamond)$. It follows that p transitions on every $a \in \sigma(\diamond)$. Furthermore, if $\delta(p, a) \neq q$, then $\delta(p, a) = r$ for some superstate r of q in order for $\sigma(u\diamond v') \in L$ for all $v' \in S(q)$. For the inductive step, suppose $|H(u)| = k$. By the inductive hypothesis, because we added each $\diamond$-transition inside u to some existing transition for $a \in \sigma(\diamond)$, there is some total word $\hat{u}$ such that $\hat{u} \in \sigma(u)$ and $\hat{\delta}_M(s, \hat{u}) = p$. Hence, if $\sigma(u\diamond v) \in L$, then $\hat{u}a\sigma(v) \subseteq L$ for all $a \in \sigma(\diamond)$, and hence p must have transitioned to a state on every $a \in \sigma(\diamond)$ such that v is in the suffix set of each of these states. Our claim follows.

Now, suppose towards a contradiction that the partial language L' recognized by a minimal $\diamond$-DFA $M'_\sigma = (Q, \Sigma_\diamond, \delta, s, F)$ with reduced transition complexity associated with a $\diamond$-substitution σ for the language L is such that $|L'| > |L|$.

We consider the construction of L' from L. In order to have $|L'| > |L|$, at some point, we must insert a weaker word $u\diamond v$ into L', where $ubv \in L$ for all $b \in \sigma(\diamond)$, $u, v \in \Sigma^*$, and not remove any words of the form ubv from L'.

Furthermore, for each redundancy we remove by adding partial words to L', we must have another instance of adding a partial word without removing any redundancies. Hence we maintain that $\sigma(L') = L$ and now $|L'| > |L|$. Consider the partial word $u\diamond v$ that was added to L' such that prior to this addition, $|L'| \leq |L|$, but after it, $|L'| > |L|$. If we are altering L' in order to create a minimal $\diamond$-DFA for L, then it must be that by adding $u\diamond v$, we can merge states in M'_σ.

Let $\hat{\delta}(s, u) = p$ and $\delta(p, \diamond) = q$. As we do not remove redundancies when we add $\delta(p, \diamond) = q$ to M'_σ, we do not delete any transitions. If this allows states to merge, it must do so by enlarging the set of transitions out of p to match exactly some other state r in M'_σ such that we can merge p and r.

If A, the set of all $a \in \sigma(\diamond)$ such that $\delta(p, a) = q$, is non-empty, then by adding $\delta(p, \diamond) = q$, the $\diamond$-DFA is not of reduced transition complexity. Hence it must be that $A \cap \sigma(\diamond) = \emptyset$. Let B be the set of all $b \in \Sigma$ such that $\delta(r, b) = q$. If other state-letter transition pairs differ between p and r that are not concerned with transitioning to q, adding $\delta(p, \diamond) = q$ will not allow p and r to be merged. Then it must be that the set of transitions A and B to q is the only place where p and r differ.

It is necessary to assume that $B = A \cup \{\diamond\}$, as otherwise, adding $\delta(p, \diamond) = q$ would not allow the states to merge. But if r transitions to q on $\diamond$ and no elements of $\sigma(\diamond)$, then it must be that on every letter of $\sigma(\diamond)$, r transitions to a superstate of q. But all of the other transitions out of r are identical to those out of p, hence p and r should already have been merged before $\delta(r, \diamond) = q$ was added, and certainly $\delta(p, \diamond) = q$ will not be added now.

It follows from the induction that for $u, v \in \Sigma_\diamond^*$ that no $\diamond$-transitions would be added without removing redundancies from the induction. $\square$

To illustrate Algorithm 3, consider the $\diamond$-substitution $\sigma(\diamond) = \{a, b\}$, the $\diamond$-DFA M_σ from Figure 9 (Top), and the finite language $L = \{aaa, baa, aba, abb\}$ over $\{a, b\}$. In Line 1, after running standard DFA minimization on M_σ, we get the DFA from Figure 9 (Bottom). In Line 2, the list of all paths from 0 to 6 is $P = \{0 - 12 - 345 - 6, 0 - 1 - 3 - 6, 0 - 12 - 4 - 6\}$; here no path grows longer than $\ell = 3$, the length of the longest word in L. In Line 3, the partial language from P constructed from the labels of paths in P is $L' = \{aaa, aab, baa, abb\}$; here $|L'| = |L| = 4$. In Lines 4 and 5, the length ℓ' of the longest word in L' is such that $\ell' = \ell = 3$. In Line 6, the result of running *Minlang* on L' and σ is $L'_\sigma = \{aa\diamond, a\diamond b, \diamond aa\}$. In Line 7, the result of running *Redundancy Removal* on L'_σ is $L'_\sigma = \{a\diamond b, \diamond aa\}$. In Line 8, running *Minlang* with *Redundancy Removal* on L creates $L_{\min,\sigma} = \{\diamond aa, ab\diamond\}$. In Lines 9 and 10, since $L'_\sigma \neq L_{\min,\sigma}$, the algorithm returns false, i.e., $\sigma(\mathbb{L}(M_\sigma)) \neq L$.

Referring to the notation used in the pseudocode of Algorithm 3, for a

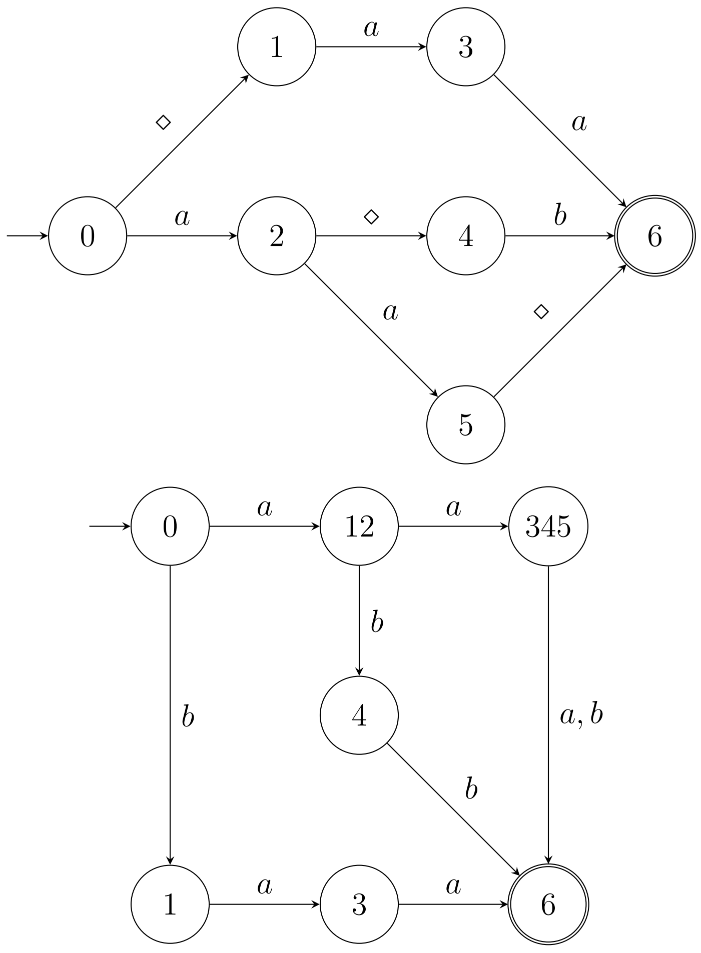

Figure 9: Top: A $\diamond$-DFA M_σ recognizing $\{\diamond aa, a\diamond b, aa\diamond\}$, with $\sigma(\diamond) = \{a, b\}$. Bottom: The minimal DFA M satisfying $\mathbb{L}(M) = \sigma(\mathbb{L}(M_\sigma))$. Error states are omitted.

Algorithm 3 *Partial Automaton 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 $\sigma(\mathbb{L}(M_\sigma)) = L$

```
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 the partial language $L'$ from $P$, i.e., partial words in $L'$ are labels
   of paths in $P$ (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 $\sigma$ (at most $\ell$ passes)
7: run *Redundancy Removal* on $L'_\sigma$
8: run *Minlang* with *Redundancy Removal* on $L$, creating $L_{\min,\sigma}$
9: **if** $L'_\sigma = L_{\min,\sigma}$ **then return true**
10: **else return false**
```

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 $z \subsetneq x$, and $\sigma(w) \subseteq \sigma(X)$.

Theorem 5. *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 passes) on L' and then running* Redundancy Removal *is equal to $L_{\min,\sigma}$, where ℓ denotes the length of the longest word in L.*

Proof. First, we claim that if L is a language over alphabet Σ and σ is a $\diamond$-substitution over Σ, then for any partial language L' such that $\sigma(L') = L$, if L'_σ is the language produced by running *Minlang*, as many passes 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 base case, consider the factorizations $w = uv = ua$ and $x = u'v' = u'a'$ where $|a| = |a'| = 1$. Since X covers w, we have $u'a' = x \uparrow w = ua$, and so $u' \uparrow u$ and $a' \uparrow a$. Suppose towards a contradiction that $a = \diamond$ and $a' \neq \diamond$. Since $X \subseteq L'_\sigma$ covers w, $u'b \in L'_\sigma$ for all $b \in \sigma(\diamond)$, and a pass of *Minlang* clearly results in $u'\diamond \in L'_\sigma$. This implies $u'\diamond \subsetneq u'a'$, contradicting the fact that X is a weakest covering set for w. So $a \neq \diamond$ or $a' = \diamond$, and trivially, $v' = a' \subset a = v$. 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 towards a contradiction that $a = \diamond$ and $a' \neq \diamond$. 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 $u'\diamond y' \subsetneq u'a'y'$, leading to a contradiction as for the base case. So $a \neq \diamond$ or $a' = \diamond$. To have $u'a' \uparrow ua$, we must have $a' \subset a$, hence $v' = a'y' \subset ay = v$.

Thus, for every $w \in L_{\min,\sigma}$, we have $x \subset w$ for all $x \in X$. By Lemmas 2 and 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 prove our claim, 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 3, we have that *Redundancy Removal* on L'_σ removes all redundant elements of L'_σ, resulting in simply $L_{\min,\sigma}$. □

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

Theorem 6. *Given as input a finite language L, a $\diamond$-substitution σ, and a $\diamond$-DFA M_σ,* Partial Automaton Check *runs in polynomial time in the size of the input. It properly verifies that $\sigma(\mathbb{L}(M_\sigma)) = L$ and that M_σ is a contender for a minimal $\diamond$-DFA for L given σ.*

Proof. To see that *Partial Automaton Check* runs in polynomial time, consider an input $\diamond$-DFA $M_\sigma = (Q, \Sigma_\diamond, \delta, s, F)$. Let $V = |Q|$. Let E denote the total number of transitions in M_σ, which is $O(|\Sigma|V)$ as each state can have at most $|\Sigma| + 1$ transitions. Using standard DFA minimization takes $O(V \log V)$ time [17]. Compiling the list P of all paths from s to each $f \in F$ takes $O(V + E)$ time using breadth-first search. Since $|L'| \leq |L|$, by Proposition 4, compiling L' from P takes $O(n)$ time, where $n = |L|$. We define ℓ to be the length of the longest word in L, noting that ℓ is the same for all partial languages with the same σ-image. Additionally, we let $n = |L|$. Running *Minlang* on L', at most ℓ passes, takes $O(\ell^2 n + \ell n^4)$ time by Theorem 2, and running *Redundancy Removal* on the resulting language takes $O(n^5 + \ell n^4)$ time by Theorem 4. Similarly, running *Minlang* with *Redundancy Removal* on L takes $O(n^5 + \ell n^4)$ time. Then the comparison of $L_{\min,\sigma}$ to L'_σ takes $O(\ell n)$ time since $|L_{\min,\sigma}| \leq |L|$ by definition of the σ-minimal partial language for L. Because $V = O(\ell n)$ and $E = O(|\Sigma||V)$, the runtime of *Partial Automaton Check* is given by $O(|\Sigma|\ell n + \ell^2 n + \ell n^5)$, which is polynomial in the input size.

To see that *Partial Automaton Check* properly verifies that $\sigma(\mathbb{L}(M_\sigma)) = L$ 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 initial state s to all final states $f \in F$ and consolidates the paths into a partial language $L' = \mathbb{L}(M_\sigma)$. However, by Proposition 4, 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 Removal* on L and *Minlang*, at most ℓ passes, on L' followed by *Redundancy Removal*. By Theorem 3 and Theorem 5, 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 so, it returns true, and if not, it returns false (Lines 9–10). Hence it returns true if and only if $\sigma(\mathbb{L}(M_\sigma)) = L$ and M_σ is a contender for a minimal $\diamond$-DFA for L given σ. □

7 Adapting *Minlang* for Infinite Languages

It is well known that the problem of minimizing NFAs or regular expressions is PSPACE-complete [24], which is in contrast with the problem of minimizing DFAs [17]. The problem remains PSPACE-complete even when specifying the regular language by a DFA [18]. Gramlich and Schnitzer [13] proved inapproximability results for the minimization problem for NFAs or regular expressions. Other works on the complexity of finite automata problems include [2, 5, 10, 15, 19, 23, 26].

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 joint together at the outermost 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 joint 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 for infinite regular languages runs in polynomial time of the input regular expression using the same analysis as for *Minlang*.

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 $x \subsetneq w$. 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

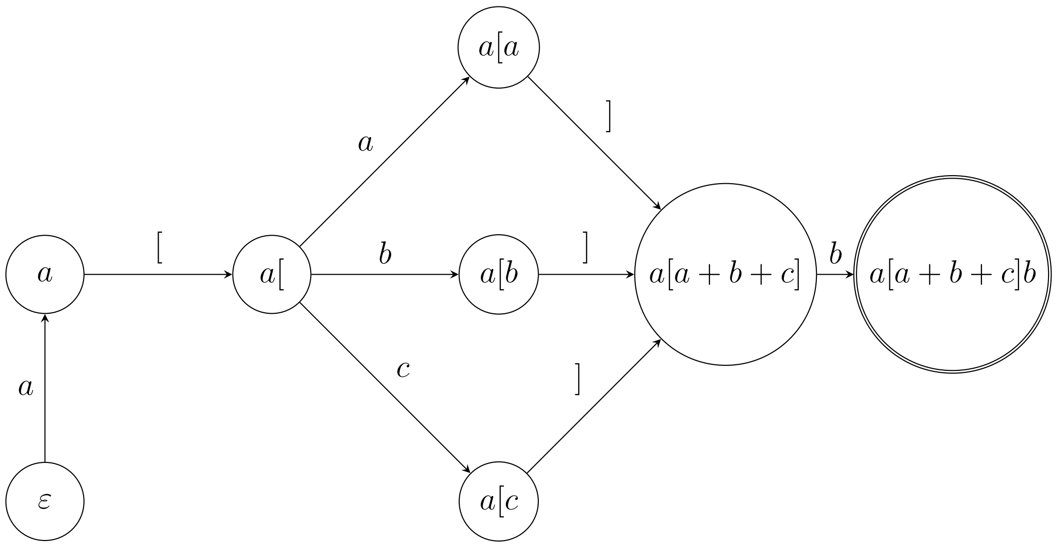

Figure 10: A language tree representing the regular expression $a(a+b+c)^*b$. Running the modified *Minlang* produces the desired regular expression $a\diamond^*b$, with $\sigma(\diamond) = \{a, b, c\}$.

configuration of a loop, such as the unambiguous form from [14]. 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 $u(vyu)^*v$ 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 $(ab)^*\diamond$.

8 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. Figure 11 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 $\mathbb{L}(M)$, associated with σ, has the same state complexity as a minimal $\diamond$-DFA for $\mathbb{L}(M)$ 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.

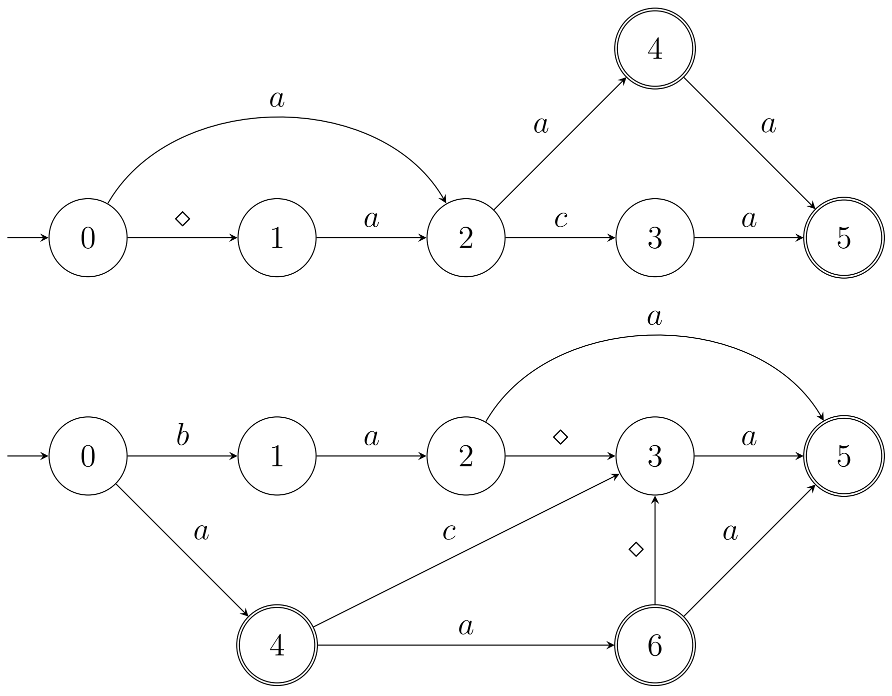

Figure 11: Top: A minimal $\diamond$-DFA, associated with $\sigma(\diamond) = \{a, b\}$, having 7 states including the error state, a sink non-final state. Bottom: A minimal $\diamond$-DFA for the same total language, associated with $\sigma(\diamond) = \{a, c\}$, having 8 states including the error state.

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 total language L, computes the σ-minimal partial language for L. This is our own implementation, we do not use any known automata library.

Acknowledgements

Project sponsored by the National Security Agency under Grant Number H98230-15-1-0232. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notation herein. This manuscript is submitted for publication with the understanding that the United States Government is authorized to reproduce and distribute reprints.

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

Part of this paper was presented at CIAA 2014 [9]. We thank Andrew Lohr, Robert Mercaş, Vishali Vadakattu, and the referees of preliminary versions of this paper for their very valuable comments and suggestions.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] J. Amilhastre, P. Janssen, and M.-C. Vilarem. FA minimisation heuristics for a class of finite languages. In O. Boldt and H. Jürgensen, editors, *4th International Workshop on Implementing Automata, Potsdam, Germany*, volume 2214 of *Lecture Notes in Computer Science*, pages 1-12. Springer, 2001.

[3] E. Balkanski, F. Blanchet-Sadri, M. Kilgore, and B. J. Wyatt. On the state complexity of partial word DFAs. *Theoretical Computer Science*, 578:2–12, 2015.

[4] J. Berstel and L. Boasson. Partial words and a theorem of Fine and Wilf. *Theoretical Computer Science*, 218:135-141, 1999.

[5] J.-C. Birget. Intersection and union of regular languages and state complexity. *Information Processing Letters*, 43:185-190, 1992.

[6] H. Björklund and W. Martens. The tractability frontier for NFA minimization. *Journal of Computer and System Sciences*, 78:198–210, 2012.

[7] F. Blanchet-Sadri. Codes, orderings, and partial words. *Theoretical Computer Science*, 329:177202, 2004.

[8] F. Blanchet-Sadri. *Algorithmic Combinatorics on Partial Words*. Chapman & Hall/CRC Press, Boca Raton, FL, 2008.

[9] F. Blanchet-Sadri, K. Goldner, and A. Shackleton. Minimal partial languages and automata. In M. Holzer and M. Kutrib, editors, *CIAA 2014, 19th International Conference on Implementation and Application of Automata, Giessen, Germany*, volume 8587 of *Lecture Notes in Computer Science*, pages 110–123. Springer International Publishing Switzerland, 2014.

[10] S. Cho and D. T. Huynh. The parallel complexity of finite-state automata problems. *Information Computation*, 97:1-22, 1992.

[11] J. Dassow, F. Manea, and R. Mercaş. Regular languages of partial words. *Information Sciences*, 268:290–304, 2014.

[12] M. J. Fischer and M. S. Paterson. String-matching and other products. In R. M. Karp, editor, *Complexity of Computation*, , SIAM-AMS Proceedings, volume 7, pages 113-126. American Mathematical Society, 1974.

[13] G. Gramlich and G. Schnitzer. Minimizing nfa's and regular expressions. *Journal of Computer and System Sciences*, 73:908–923, 2007.

[14] B. Groz, S. Maneth, and S. Staworko. Deterministic regular expressions in linear time. In *PODS 2012, 31th ACM Symposium on Principles of Database Systems*, pages 49–60, 2012.

[15] H. Gruber and M. Holzer. Computational complexity of NFA minimization for finite and unary languages. In *LATA 2007, 1st International Conference on Language and Automata Theory and Applications, Tarragona, Spain*, Technical Report 35/07, pages 261-272, 2007. Research Group on Mathematical Linguistics, Universitat Rovira i Virgili.

[16] M. Holzer, S. Jakobi, and M. Wendlandt. On the computational complexity of partial word automata problems. In S. Bensch, R. Freund, and F. Otto, editors, *NCMA 2014, 6th Workshop on Non-Classical Models for Automata and Applications, Kassel, Germany*, volume 304 of *books@ocg.at*, pages 131–146. Österreichische Computer Gesellschaft, 2014.

[17] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical report, DTIC Document, 1971.

[18] T. Jiang and B. Ravikumar. Minimal nfa problems are hard. *SIAM Journal on Computing*, 22:1117–1141, 1993.

[19] N. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68-85, 1975.

[20] P. Leupold. Languages of partial words - how to obtain them and what properties they have. *Grammars*, 7:179-192, 2004.

[21] G. Lischke. Restoration of punctured languages and similarity of languages. *Mathematical Logic Quarterly*, 52:20-28, 2006.

[22] F. Manea and C. Tiseanu. The hardness of counting full words compatible with partial words. *Journal of Computer and System Sciences*, 79:7–22, 2013.

[23] F. R. Moore. On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. *IEEE Transactions on Computing*, 20:1211-1214, 1971.

[24] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT 1972, 13th Annual IEEE Symposium on Switching and Automata Theory*, pages 125–129, 1972.

[25] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[26] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. *In 5th Symposium on Theory of Computing*, pages 1-9, 1973.