Extracting FSSQs from Regular Stringsets

Dakotah Lambert

1 Introduction

In this module we define and implement a method for extracting strictly piecewise (SP) constraints from arbitrary regular stringsets. These constraints are also known as forbidden subsequences, due to the nature of SP stringsets. Although extracting these constraints is an exponential-time operation, the implementation clarified a linear-time algorithm to generate an SP approximation of a stringset.

Definition 1. Let $v = \sigma_1\sigma_2\ldots\sigma_n \in \Sigma^*$. The *shuffle ideal* of v, SI(v), is the set:

$$\mathrm{SI}(v) = \{u_0\sigma_1u_1\sigma_2u_2\ldots\sigma_nu_n : u_i \in \Sigma^*\}.$$

If $w \in \mathrm{SI}(v)$, we say v is a *subsequence* of w, $v \sqsubseteq w$. Determining whether v is a subsequence of w is quite a simple algorithm:

```
(⊑) :: (Eq a) ⇒ [a] → [a] → Bool
[ ]      ⊑ _ = True
(v : vs) ⊑ ws
    | isEmpty notV = False
    | otherwise    = vs ⊑ tail notV
    where notV = dropWhile (≢ v) ws
```

2 Augmenting a regular stringset

A regular stringset can be augmented by adding transitions to its representative DFA, or by augmenting the set of accepting states. By the nature of the construction described herein, we only need the first method.

Let $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F\rangle$ be an automaton. Define the *subsequence-closure* of $L(\mathcal{M})$ as the smallest superset of $L(\mathcal{M})$ that is closed under deletion. This can be implemented as a function that inserts a transition on ε wherever a transition occurred in δ.

```
subsequenceClosure :: (Ord n, Ord e) ⇒ FSA n e → FSA n e
subsequenceClosure ⟨Q, Σ, δ, q₀, F⟩ = ⟨Q, Σ, (δ ∪ δ′), q₀, F⟩
    where δ′ = map (λ(a $\xrightarrow{x}$ b) → (a $\xrightarrow{\varepsilon}$ b)) δ
```

Let $\mathcal{M}' = \text{subsequenceClosure}\,\mathcal{M}$, and let $u, v, w \in \Sigma^*$ such that $uvw \in L(\mathcal{M}')$. Then $uv \in L(\mathcal{M}')$, as every transition taken in w can be replaced by transitions on ε. Similary vw and uw are in $L(\mathcal{M}')$. Thus $L(\mathcal{M}')$ is SP. Note that since all of the original transitions remain intact, $L(\mathcal{M}) \subseteq L(\mathcal{M}')$.

3 Extracting forbidden subsequences

Lemma 1. *The set of forbidden subsequences of $L(\mathcal{M}')$ is a subset of that of $L(\mathcal{M})$.*

Proof. Suppose the set of forbidden subsequences of $L(\mathcal{M}')$ is not a subset of the set of forbidden subsequences of $L(\mathcal{M})$. Then there exists some sequence u that is forbidden in $L(\mathcal{M}')$ but not forbidden in $L(\mathcal{M})$. It follows that some string w in $L(\mathcal{M})$ contains u as a subsequence. But since $L(\mathcal{M}) \subseteq L(\mathcal{M}')$, this string w also occurs in $L(\mathcal{M}')$. This contradicts the assumption that u is a forbidden subsequence of $L(\mathcal{M}')$. Thus the set of forbidden subsequences of $L(\mathcal{M}')$ is a subset of the set of forbidden subsequences of $L(\mathcal{M})$. ■

Lemma 2. *The set of forbidden subsequences of $L(\mathcal{M})$ is a subset of that of $L(\mathcal{M}')$.*

Proof. Suppose the set of forbidden subsequences of $L(\mathcal{M})$ is not a subset of the set of forbidden subsequences of $L(\mathcal{M}')$. Then there exists some sequence u that is forbidden in $L(\mathcal{M})$ but not forbidden in $L(\mathcal{M}')$. It follows that some string v in $L(\mathcal{M}')$ contains u as a subsequence. But since $L(\mathcal{M}')$ was formed by allowing a computation to skip transitions of $\mathcal{M}$, there exists a string w in $L(\mathcal{M})$ such that $v \sqsubseteq w$. By the transitive property of subsequences, this means $u \sqsubseteq w$. This contradicts the assumption that u is a forbidden subsequence of $L(\mathcal{M})$. Thus the set of forbidden subsequences of $L(\mathcal{M})$ is a subset of the set of forbidden subsequences of $L(\mathcal{M}')$. ■

Since each is a subset of the other, it follows that the set of forbidden subsequences in $L(\mathcal{M}')$ is exactly the same set as those forbidden in $L(\mathcal{M})$. Because of this, collecting the minimal forbidden subsequences of $L(\mathcal{M}')$ will give us the strictly piecewise constraints on $L(\mathcal{M})$.

Since $\mathcal{M}'$ is SP, it suffices to simply traverse $\mathcal{M}'$, finding all minimal (in terms of subsequence) paths from q_0 to the fail-state ($\mathbf{0}$). A cyclic path is necessarily non-minimal, so it suffices to check only the acyclic paths. Since every state in an SP automaton is accepting save for a possible unique non-accepting sink, we can merely check paths in the complement. Since this complement has exactly one accepting state, and exactly one initial state, we can also use the acyclic paths of its reversal, which prevents having to reverse every extracted string.

```
collectFSSQs :: (Ord n, Ord e) ⇒ FSA n e → {[e]}
collectFSSQs f = map (unsymbols ∘ labels) ∘
  filter (maybe False ((∈ finals f′)) ∘ endstate) $
  acyclicPaths f′
  where f′ = normalize ∘ FSA.reverse ∘ complement $ subsequenceClosure f
```

Further, we can derive a minimal set of forbidden subsequences by filtering out elements that are generated by others. Formally, if $v \sqsubseteq w$, for v and w forbidden subsequences, w is not a minimal forbidden subsequence.

```
$collectMinimalFSSQs :: (Ord\ n, Ord\ e) \Rightarrow FSA\ n\ e \rightarrow \{[e]\}$
$collectMinimalFSSQs = Set.fromList \circ$
                       $filterAbsorbed \circ$
                       $sort_{(comparing\ \|\cdot\|)} \circ$
                       $Set.toList \circ$
                       $collectFSSQs$
  where $filterAbsorbed\ (x : xs) = x : filterAbsorbed\ (filter\ (\lambda y \rightarrow x \not\sqsubseteq y)\ xs)$
        $filterAbsorbed\ \_ = [\,]$
```

This concludes the algorithm for extracting minimal forbidden subsequences, and thus for extracting strictly-piecewise factors from an arbirary regular stringset.

4 Testing whether a stringset is strictly piecewise

If $\mathcal{M}$ already represents an SP stringset, then $\mathcal{M}'$ will recognize the same stringset as its source. This makes for quite the simple test to determine whether a stringset is SP:

```
$isSP :: (Ord\ n, Ord\ e) \Rightarrow FSA\ n\ e \rightarrow Bool$
$isSP\ f = f \equiv subsequenceClosure\ f$
```