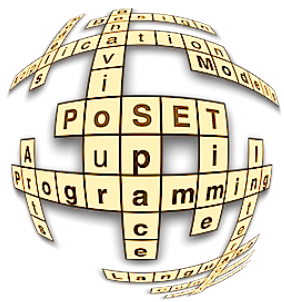
Factor Oracle for Machine Improvisation

Jaime Arias

Université de Bordeaux, LaBRI, UMR 5800
Inria - Bordeaux Sud-Ouest

August 2016



Preliminaries

Preliminaries

Word

A word s is a finite sequence $s = s_1 s_2 \dots s_m$ of length $|s| = m$ on a finite alphabet Σ.

$s =$

a	b	b	c	a	b	c	d	a	b	c

Factor

A word $x \in \Sigma^*$ is a factor of s if and only if s can be written $s = uxv$ with $u, v \in \Sigma^*$. Given integers i, j where $1 \leq i \leq j \leq m$, we denote a *factor* of s as $s[i \dots j] = s_i s_{i+1} \dots s_j$.

$s =$

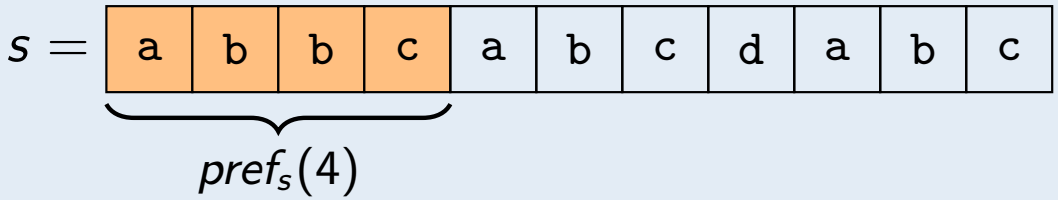
a	b	b	c	a	b	c	d	a	b	c

$s[3, 5]$

Preliminaries

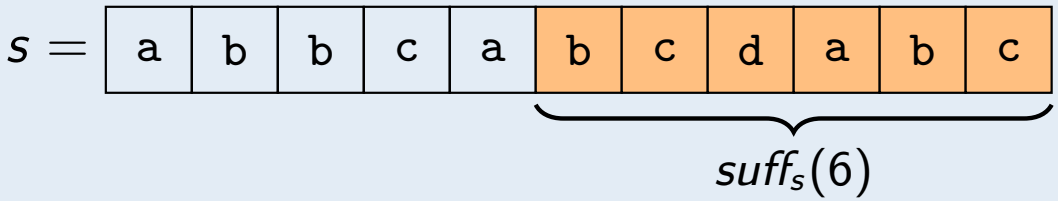
Prefix

A factor x of s is a prefix of s if $s = xu$ with $u \in \Sigma^*$. The ith *prefix* of s, denoted $pref_s(i)$, is the prefix $s[1 \dots i]$.



Suffix

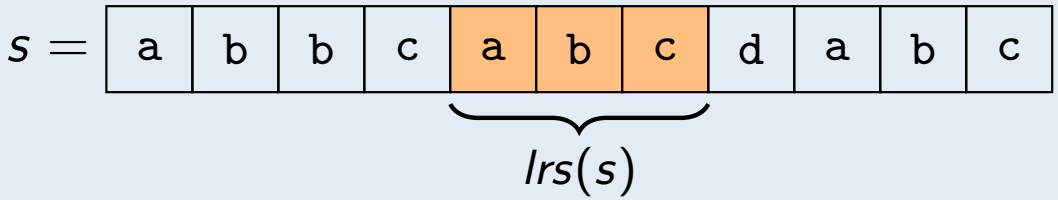
A factor x of s is a suffix of s if $s = ux$ with $u \in \Sigma^*$. The ith *suffix* of s, denoted $suff_s(i)$, is the suffix $s[i \dots m]$.



Preliminaries

Longest Repeated Suffix (LRS)

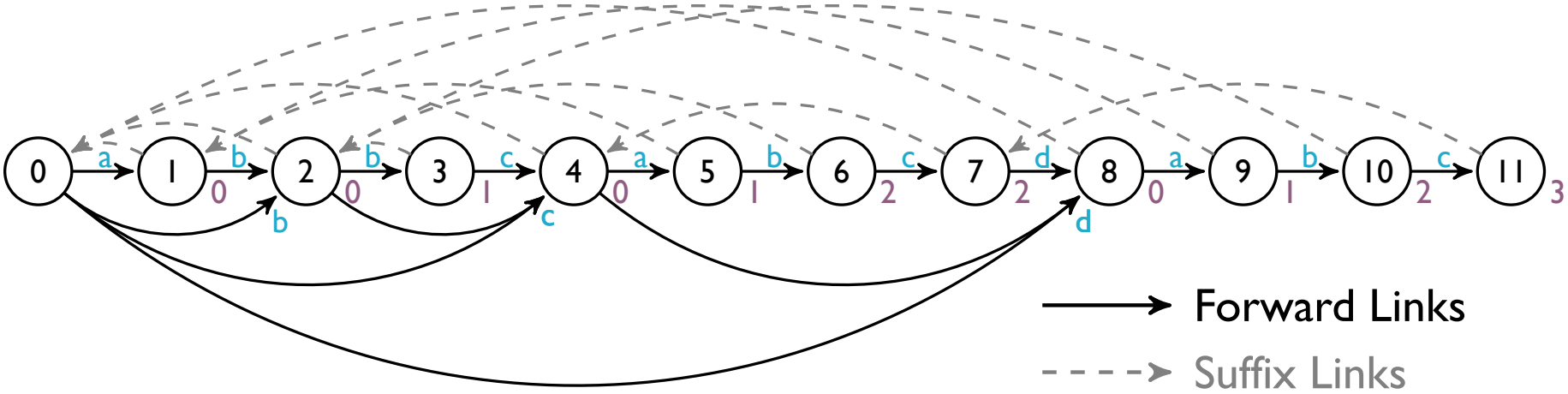
A factor x of s is the longest repeated suffix of s if x is a suffix of s and $|x|$ is maximal.



Factor Oracle

Factor Oracle

Overview

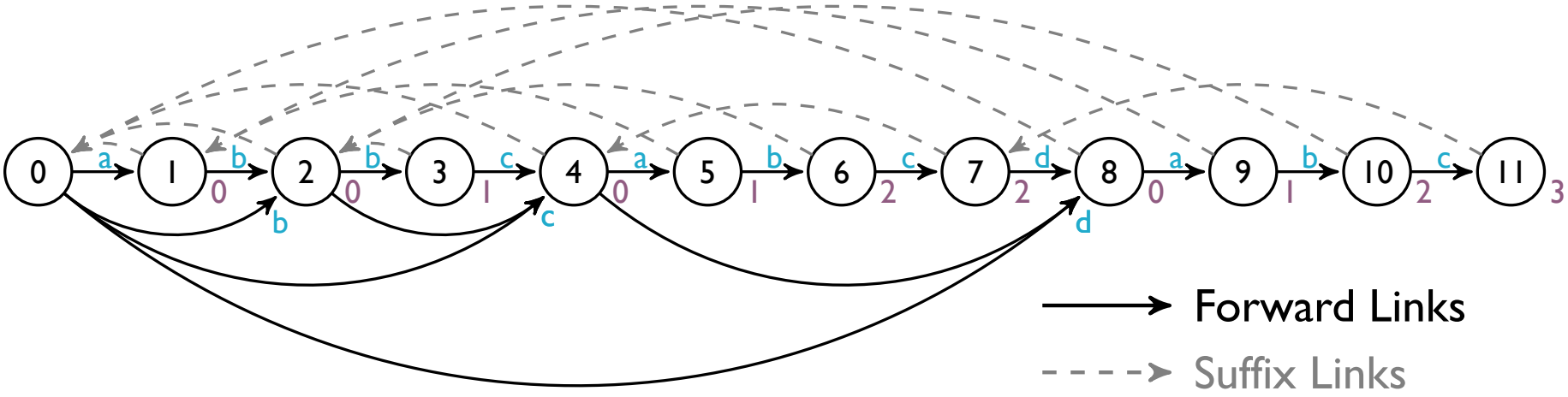


Factor Oracle

The factor oracle of a word s of length m is a *deterministic finite automaton* (Q, q_0, F, δ) where $Q = \{0, 1, \ldots, m\}$ is the set of states, $q_0 = 0$ is the starting state, $F = Q$ is the set of terminal states and δ is the transition function.

Factor Oracle

Overview

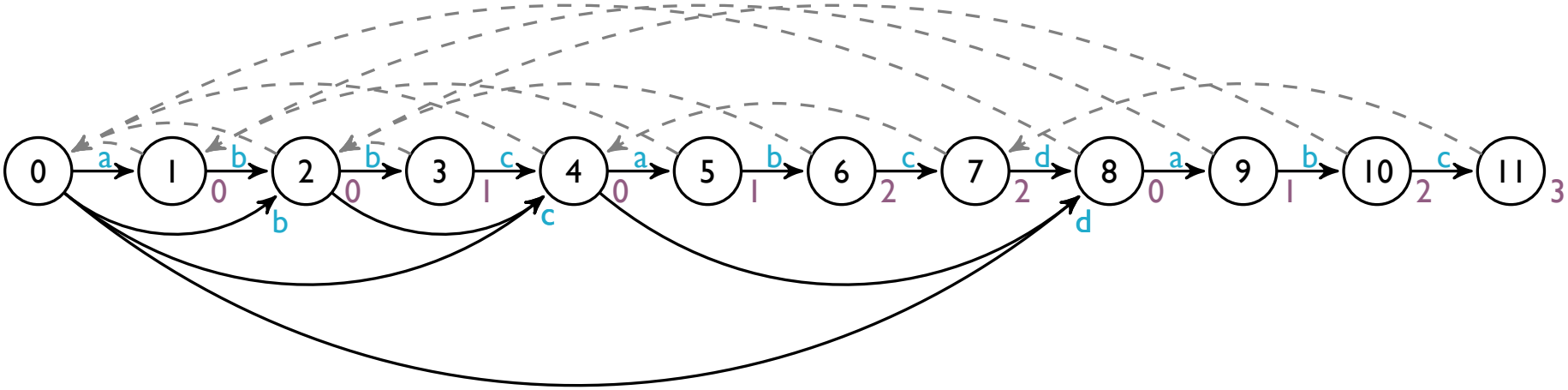


Suffix Link

The suffix link of a state i of the factor oracle of a word s, is equal to the state in which the *longest repeated suffix (lrs)* of $s[1 \dots i]$ is recognized.

Factor Oracle

Overview

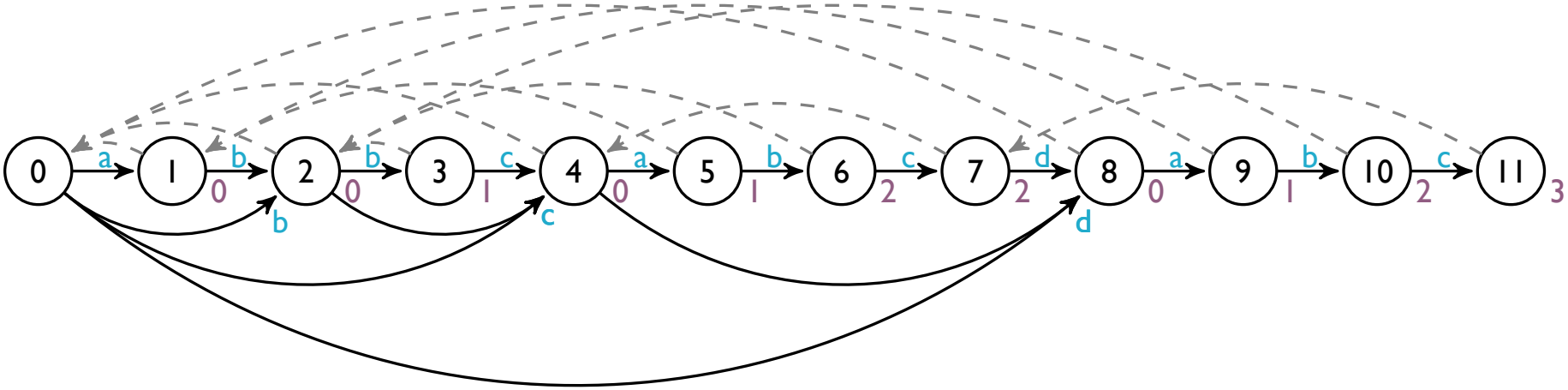


Suffix Links

- $s =$ abbcabcdabc

Factor Oracle

Overview

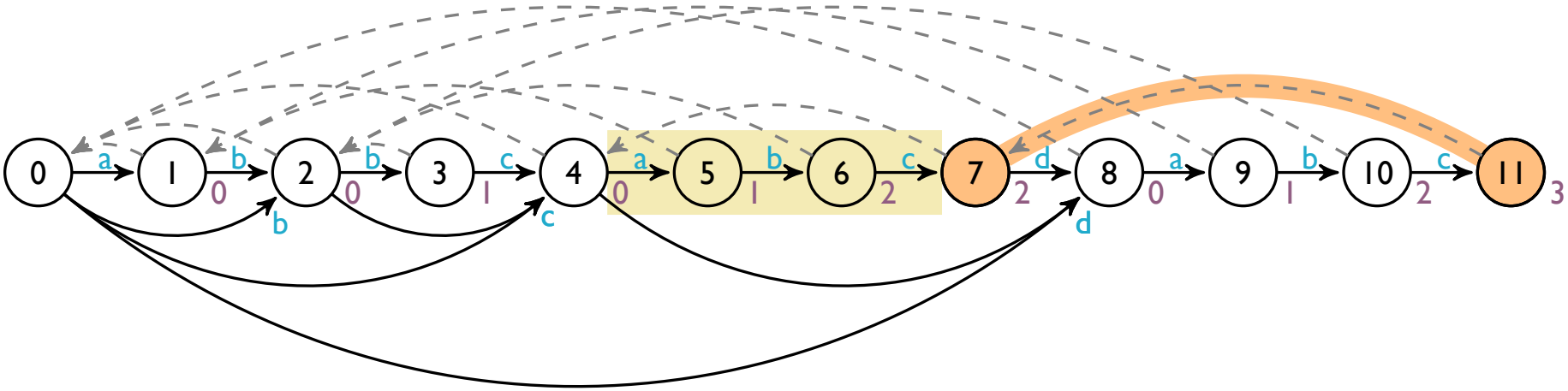


Suffix Links

- $s = \texttt{abbcabcdabc}$
- $lrs(s) = \texttt{abc}$

Factor Oracle

Overview



Suffix Links

- $s = \texttt{abbcabcdabc}$
- $lrs(s) = \texttt{abc}$
- $S(11) = 7$

Factor Oracle

Algorithm

Algorithm 1 Construction of a Factor Oracle

1: **function** FactorOracle($p = p_1 p_2 \dots p_m$)
2: Create a new oracle P with an initial state 0
3: $S_P(0) \leftarrow -1$
4: **for** $i \leftarrow 1, m$ **do**
5: $Oracle(p = p_1 p_2 \dots p_i) \leftarrow$ AddLetter($Oracle(p = p_1 p_2 \dots p_{i-1}), p_i$)
6: **end for**
7: **return** $Oracle(p = p_1 p_2 \dots p_m)$
8: **end function**

Factor Oracle

Algorithm

Algorithm 1 Construction of a Factor Oracle

1: **function** FactorOracle($p = p_1 p_2 \dots p_m$)
2: Create a new oracle P with an initial state 0
3: $S_P(0) \leftarrow -1$
4: **for** $i \leftarrow 1, m$ **do**
5: $Oracle(p = p_1 p_2 \dots p_i) \leftarrow$ AddLetter($Oracle(p = p_1 p_2 \dots p_{i-1}), p_i$)
6: **end for**
7: **return** $Oracle(p = p_1 p_2 \dots p_m)$
8: **end function**

$p =$

a	b	b	c	a	b	c	d	a	b	c

Factor Oracle

Algorithm

Algorithm 1 Construction of a Factor Oracle

1: **function** FactorOracle($p = p_1 p_2 \dots p_m$)
2: Create a new oracle P with an initial state 0
3: $S_P(0) \leftarrow -1$
4: **for** $i \leftarrow 1, m$ **do**
5: $Oracle(p = p_1 p_2 \dots p_i) \leftarrow$ AddLetter($Oracle(p = p_1 p_2 \dots p_{i-1}), p_i$)
6: **end for**
7: **return** $Oracle(p = p_1 p_2 \dots p_m)$
8: **end function**

$p =$

a	b	b	c	a	b	c	d	a	b	c

Factor Oracle

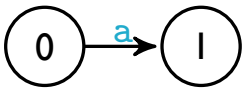
Algorithm

Algorithm 2 Incremental update of Factor Oracle

1: **function** AddLetter($Oracle(p = p_1, p_2 \dots p_m), \sigma$)
2: Create state $m + 1$
3: Create a new transition from m to $m + 1$ labeled by σ $\triangleright\ \delta(m, \sigma) = m + 1$
4: $k \leftarrow S_p(m)$
5: $\pi_1 \leftarrow m$

$p =$

a	b	b	c	a	b	c	d	a	b	c



$m = 0$

$k = -1$

$\pi_1 = 0$

Factor Oracle

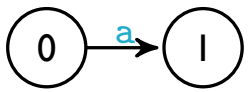
Algorithm

Algorithm 2 Incremental update of Factor Oracle

6: **while** $k > -1$ and there is no transition from k by σ **do**
7: Create a new transition from k to $m+1$ by σ $\triangleright\ \delta(k, \sigma) = m + 1$
8: $\pi_1 \leftarrow k$
9: $k \leftarrow S_p(k)$
10: **end while**

$p =$

a	b	b	c	a	b	c	d	a	b	c



$m = 0$

$k = -1$

$\pi_1 = 0$

Factor Oracle

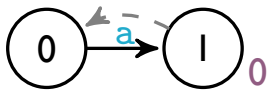
Algorithm

Algorithm 2 Incremental update of Factor Oracle

```
11:  if k = −1 then
12:      S_pσ ← 0
13:      lrs_pσ ← 0
14:  else
15:      S_pσ ← state that leads the transition from k by σ
16:      lrs_pσ ← LengthCommonSuffix(π_1, S(m + 1) − 1) + 1
17:  end if
```

$p =$

a	b	b	c	a	b	c	d	a	b	c



$m = 0$

$k = -1$

$\pi_1 = 0$

Factor Oracle

Algorithm

Algorithm 2 Incremental update of Factor Oracle

18: $k \leftarrow \text{FindBetter}(m+1, p[m+1-lrs(m+1)])$
19: **if** $k \neq 0$ **then**
20: $lrs_{p\sigma} \leftarrow lrs(m+1)+1$
21: $S_{p\sigma} \leftarrow k$
22: **end if**
23: $T(S_{p\sigma}) \leftarrow T(S(m+1)) \cup \{m+1\}$
24: **return** $Oracle(p = p_1 p_2 \ldots p_m \sigma)$
25: **end function**

Factor Oracle

Algorithm

Algorithm 3 Find Better Algorithm

```
1: function FindBetter(i, a)
2:     for all the elements j of T(i) in increasing order do
3:         if lrs(j) = lrs(i) and p[j − lrs(i)] = a then
4:             return j
5:         end if
6:     end for
7:     return 0
8: end function
```

Factor Oracle

Algorithm

Algorithm 3 Find Better Algorithm

```
1: function FindBetter(i, a)
2:     for all the elements j of T(S(i)) in increasing order do
3:         if lrs(j) = lrs(i) and p[j − lrs(i)] = a then
4:             return j
5:         end if
6:     end for
7:     return 0
8: end function
```

Factor Oracle

Algorithm

Algorithm 4 Length Common Suffix Algorithm

```
1:  function LengthCommonSuffix(π1, π2)
2:      if S(π1) = π2 then
3:          return lrs(π1)
4:      else
5:          while S(π1) ≠ S(π2) do
6:              π2 ← S(π2)
7:          end while
8:      end if
9:      return min(lrs(π1), lrs(π2))
10: end function
```

Thank you for your attention! ☺

Factor Oracle for Machine Improvisation

Jaime Arias

Université de Bordeaux, LaBRI, UMR 5800
Inria - Bordeaux Sud-Ouest

August 2016