

Constructing Factor Oracles

Loek Cleophas¹, Gerard Zwaan¹, and Bruce W. Watson^{1,2}

¹ Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

² Department of Computer Science, University of Pretoria, Pretoria 0002, South Africa

loek@loekcleophas.com, g.zwaan@tue.nl, bruce@bruce-watson.com

January 5, 2004

Abstract. A *factor oracle* is a data structure for weak factor recognition. It is an automaton built on a string p of length m that is acyclic, recognizes at least all factors of p , has $m + 1$ states which are all final, and has m to $2m - 1$ transitions. In this paper, we give two alternative algorithms for its construction and prove the constructed automata to be equivalent to the automata constructed by the algorithms in [1]. Although these new $\mathcal{O}(m^2)$ algorithms are practically inefficient compared to the $\mathcal{O}(m)$ algorithm given in [1], they give more insight into factor oracles. Our first algorithm constructs a factor oracle based on the suffixes of p in a way that is more intuitive. Some of the crucial properties of factor oracles, which in [1] need several lemmas to be proven, are immediately obvious. Another important property however becomes less obvious. A second algorithm gives a clear insight in the relationship between the trie or dawg recognizing the factors of p and the factor oracle recognizing a superset thereof. We conjecture that an $\mathcal{O}(m)$ version of this trie-based algorithm exists.

1 Introduction

A *factor oracle* is a data structure for weak factor recognition³. It can be described as an automaton built on a string p of length m that (a) is acyclic, (b) recognizes at least all factors of p , (c) has $m + 1$ states (which are all final), and (d) has m to $2m - 1$ transitions (cf. [1]). Some example factor oracles are given in Figure 1.

Factor oracles are introduced in [1] as an alternative to the use of exact factor recognition in many on-line keyword pattern matching algorithms. In such algorithms, a window on a text is read backward while attempting to match a keyword factor. When this fails, the window is shifted using the information on the longest factor matched and the mismatching character.

³ See Subsection 1.2 for the definition of a *factor*.

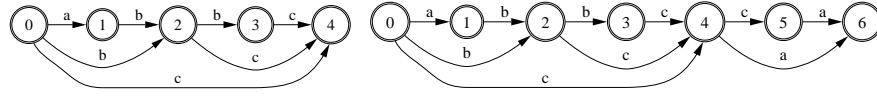


Fig. 1. Factor oracles for abc (recognizing abc , which is not in $\mathbf{fact}(p)$), $abccca$ (recognizing $abc, abcc, abcca, abca, abbca, bbca, bca$, all of which are not in $\mathbf{fact}(p)$)

Instead of an automaton recognizing exactly the set of factors of the keyword, it is possible to use a factor oracle: although it recognizes more strings than just the factors and thus might read backwards longer than necessary, it cannot miss any matches. The advantage of using factor oracles is that they are easier to construct and take less space to represent compared to the automata that were previously used in these factor-based algorithms, such as suffix, factor and subsequence automata. The latter automata lack one of the properties (c) ($m + 1$ states) or (d) (m to $2m - 1$ transitions) and in fact have *more* states or transitions. The factor oracle on the other hand satisfies both and therefore takes less memory space.

The factor oracle is introduced in [1] by means of an $\mathcal{O}(m^2)$ construction algorithm that is used as its definition. Furthermore, an $\mathcal{O}(m)$ sequential construction algorithm is described. It is not obvious by just considering the algorithms that it recognizes at least all factors of p and has m to $2m - 1$ transitions (i.e. that (b) and (d) hold). For both algorithms, a number of lemmas are needed to prove this. In this paper, we give two alternative algorithms for the construction of a factor oracle.

Our first algorithm, in Section 2, constructs a factor oracle based on the suffixes of p . This algorithm is $\mathcal{O}(m^2)$ and thus not of practical interest, but it is more intuitive to understand and properties (b) and (d)—two important properties of factor oracles—are immediately obvious from the algorithm. The acyclicity of the factor oracle however—corresponding to property (a)—is not immediately obvious. Our proof of this property (part of Property 6) is rather involved, whereas the property is immediately obvious from the algorithms in [1]. We prove that the alternative construction algorithm and those given in [1] construct equivalent automata in Section 3.

Section 4 shows that the language of a factor oracle is prefix-, suffix- and therefore factor-closed. A precise characterization of the language (independent of the automaton itself) is still open. We show that the occurrence of repetitions and the occurrence of states (other than the start state) with at least 2 outgoing transitions are necessary for the language to contain strings other than factors.

In Section 5 we present our second algorithm, which constructs a factor oracle from the trie recognizing the factors of p . Although this algorithm is $\mathcal{O}(m^2)$ as well, it gives a clear insight in the relationship between the trie and dawg recognizing the factors of p and the factor oracle recognizing a superset thereof. In addition, we conjecture that an $\mathcal{O}(m)$ trie-based algorithm exists, although we have not investigated this due to lack of time.

Finally, Section 6 gives a summary and overview of future work.

1.1 Related Work

An earlier version of this report appears as [4, Chapter 4]. That thesis also discusses pattern matching algorithms—among them those using factor oracles—and the implementation of the factor oracle as part of the SPARE TIME pattern matching toolkit, a revised and extended version of SPARE PARTS ([11]).

A shorter version of this report—excluding Section 4 on the language of factor oracles—was presented at the 2003 Prague Stringology Conference ([3]).

As mentioned before, factor oracles were introduced in [1] as an alternative to the use of exact factor recognition in many on-line keyword pattern matching algorithms. A pattern matching algorithm using the factor oracle is described in that paper as well.

Apart from their use in pattern matching algorithms, factor oracles have been used in a heuristic to compute repeated factors of a string [8] as well as to compress text [9]. An improvement for those uses of factor oracles is introduced in [10] in the form of the *repeat oracle*.

Related to the factor oracle, the *suffix oracle*—in which only those states corresponding to a suffix of p are marked final—is introduced in [1]. In [2] the factor oracle is extended to apply to a set of strings.

1.2 Preliminaries

A *string* $p = p_1 \dots p_m$ of length m is a sequence of characters from an alphabet V . A string u is a *factor* (resp. *prefix*, *suffix*) of a string v if $v = sut$ (resp. $v = ut$, $v = su$), for $s, t \in V^*$. We will use $\mathbf{fact}(p)$, $\mathbf{pref}(p)$ and $\mathbf{suff}(p)$ for the set of factors, prefixes and suffixes of p respectively. A factor (resp. prefix or suffix) is a *proper* factor (resp. prefix or suffix) of a string p if it does not equal p . We write $u \leq_s v$ to denote that u is a suffix of v , and $u <_s v$ to denote that u is a proper suffix of v .

2 Construction based on suffixes

Our first alternative algorithm for the construction of a factor oracle constructs a ‘skeleton’ automaton for p —recognizing $\mathbf{pref}(p)$ —and then constructs a path for each of the suffixes of p in order of decreasing length, such that eventually at least $\mathbf{pref}(\mathbf{suff}(p)) = \mathbf{fact}(p)$ is recognized. If such a suffix of p is already recognized, no transition needs to be constructed. If on the other hand the complete suffix is not yet recognized there is a longest prefix of such a suffix that is recognized. A transition on the next, non-recognized symbol is then created, from the state in which this longest prefix of the suffix is recognized, to a state from which there is a path leading to state m that spells out the rest of the suffix.

Build_Oracle_2($p = p_1p_2\dots p_m$)

- 1: **for** i from 0 to m **do**
- 2: Create a new final state i
- 3: **for** i from 0 to $m - 1$ **do**
- 4: Create a new transition from i to $i + 1$ by p_{i+1}
- 5: **for** i from 2 to m **do**
- 6: Let the longest path from state 0 that spells a prefix of $p_i\dots p_m$ end in state j and spell out $p_i\dots p_k$ ($i - 1 \leq k \leq m$)
- 7: **if** $k \neq m$ **then**
- 8: Build a new transition from j to $k + 1$ by p_{k+1}

Note that this algorithm is $\mathcal{O}(m^2)$ (since the operation on line 6 can be implemented using a **while** loop). The factor oracle on p built using this algorithm is referred to as $\text{Oracle}(p)$ and the language recognized by it as $\mathbf{factoracle}(p)$.

The first two properties we give are obvious given our algorithm. They correspond to (b) and (c)-(d) respectively as mentioned in Section 1.

Property 1. $\mathbf{fact}(p) \subseteq \mathbf{factoracle}(p)$.

Proof: The algorithm constructs a path for all suffixes of p and all states are final. \square

Property 2. For p of length m , $\text{Oracle}(p)$ has exactly $m + 1$ states and between m and $2m - 1$ transitions.

Proof: States can be constructed in steps 1-2 only, and exactly $m + 1$ states are constructed there. In step 4 of the algorithm, m transitions are created. In steps 5-8, at most $m - 1$ transitions are created. \square

Property 3 (Glushkov’s property). All transitions reaching a state i of $\text{Oracle}(p)$ are labeled by p_i .

Proof: The only steps of the algorithm that create transitions are steps 4 and 8. In both, transitions to a state i are created labeled by p_i . \square

Property 4 (Weak determinism). For each state of $\text{Oracle}(p)$, no two outgoing transitions of the state are labeled by the same symbol.

Proof: The algorithm never creates an outgoing transition by some symbol if such a transition already exists. \square

We now define function $\text{poccur}(u, p)$ to give the end position of the left-most occurrence of u in p (equivalent to the same function in [1]):

Definition 1. Function $\text{poccur} \in V^* \times V^* \rightarrow \mathbb{N}$ is defined as

$$\text{poccur}(u, p) = \min\{|tu|, p = tuv\} \quad (p, t, u, v \in V^*)$$

\square

Note that if $u \notin \mathbf{fact}(p)$, $\text{poccur}(u, p) = \infty$.

Property 5. For suffixes and prefixes of factors we have:

$$\begin{aligned} uv \in \mathbf{fact}(p) &\Rightarrow \text{poccur}(v, p) \leq \text{poccur}(uv, p) \quad (p, u, v \in V^*) \\ uv \in \mathbf{fact}(p) &\Rightarrow \text{poccur}(u, p) \leq \text{poccur}(uv, p) - |v| \quad (p, u, v \in V^*) \end{aligned}$$

\square

We introduce $\text{min}(i)$ for the minimum length string recognized in state i —either in a partially constructed or in the complete automaton.

In the following property, we use j_i and k_i to identify the values j and k attain when considering suffix $p_i \dots p_m$ of p in steps 5-8 of the algorithm.

Property 6. For the partial automaton constructed according to algorithm **Build-Oracle_2** with all suffixes of p of length greater than $m - i + 1$ already considered in steps 5-8 ($2 \leq i \leq m + 1$), we have that

- i. it is acyclic
- ii. for each h with $1 \leq h < i$, all prefixes of $p_h \dots p_m$ are recognized
- iii. for each state n and outgoing transition to a state $q \neq n + 1$, $q \leq k_{max} + 1$ holds where $k_{max} = \max\{k_h, 1 < h < i \wedge k_h < m\}$
- iv. for each state n , $\text{min}(n)$ is an element of $\mathbf{fact}(p)$, $\text{min}(n)$ is a suffix of each string recognized in n , and $n = \text{poccur}(\text{min}(n), p)$
- v. if $u \in \mathbf{fact}(p)$ is recognized, it is recognized in a state $n \leq \text{poccur}(u, p)$
- vi. for each state n and each symbol a such that there is a transition from n to a state q by a , $\text{min}(n) \cdot a \in \mathbf{fact}(p)$ and $q = \text{poccur}(\text{min}(n) \cdot a, p)$

- vii. for each pair of states n and q , if $\min(n) \leq_s \min(q)$, then $n \leq q$, and as a result, if $\min(n) <_s \min(q)$, then $n < q$
- viii. if w is recognized in state n , then for any suffix u of w , if u is recognized, it is recognized in state $q \leq n$

Proof: See Appendix A. □

Note that Property 6, i. corresponds to property (a) in Section 1.

3 Equivalence to Original Algorithms

A factor oracle as introduced in [1] is built by the following algorithm:

Build_Oracle($p = p_1p_2\dots p_m$)

- 1: **for** i from 0 to m **do**
- 2: Create a new final state i
- 3: **for** i from 0 to $m - 1$ **do**
- 4: Create a new transition from i to $i + 1$ by p_{i+1}
- 5: **for** i from 0 to $m - 1$ **do**
- 6: Let u be a minimal length word in state i
- 7: **for all** $\sigma \in \Sigma, \sigma \neq p_{i+1}$ **do**
- 8: **if** $u\sigma \in \mathbf{Fact}(p_{i-|u|+1}\dots p_m)$ **then**
- 9: Build a new transition from i to^{*}
 $i - |u| + \mathit{poccur}(u\sigma, p_{i-|u|+1}\dots p_m)$ by σ

To prove the equivalence of the automata constructed by the two algorithms, we need the following properties.

Property 7. For any state i of both **Oracle**(p) (i.e. the factor oracle constructed according to algorithm **Build_Oracle_2** and the factor oracle constructed according to algorithm **Build_Oracle**), if $u = \min(i)$ then

$$u\sigma \in \mathbf{fact}(p_{i-|u|+1}\dots p_m) \equiv u\sigma \in \mathbf{fact}(p)$$

Proof: \Rightarrow : Trivial. \Leftarrow : By Property 6, iv. (for **Build_Oracle_2**) and [1, Lemma 1] (for **Build_Oracle**), $i = \mathit{poccur}(u, p)$. By Property 5, $\mathit{poccur}(u\sigma, p) \geq i$, hence $u\sigma \in \mathbf{fact}(p_{i-|u|+1}\dots p_m)$. □

Property 8. For any state i of an automaton constructed by either algorithm, if $u = \min(i)$ and $u\sigma \in \mathbf{fact}(p)$ then

$$i - |u| + \mathit{poccur}(u\sigma, p_{i-|u|+1}\dots p_m) = \mathit{poccur}(u\sigma, p)$$

* Note that in [1] the term $-|u|$ is missing in the algorithm, although from the rest of the paper it is clear that it is used in the construction of the automata

Proof:

$$\begin{aligned}
& i - |u| + \text{poccur}(u\sigma, p_{i-|u|+1} \dots p_m) \\
= & \quad \{ \text{definition } \text{poccur} \} \\
& i - |u| + \min\{|tu\sigma|, p_{i-|u|+1} \dots p_m = tu\sigma v\} \\
= & \quad \{ u = \min(i), \text{ hence recognized in } i = \text{poccur}(u, p) \} \\
& i - |u| + \min\{|tu\sigma| - (i - |u|), p = tu\sigma v\} \\
= & \quad \{ u\sigma \in \mathbf{fact}(p), \text{ property of min} \} \\
& i - |u| + \min\{|tu\sigma|, p = tu\sigma v\} - (i - |u|) \\
= & \quad \{ \text{calculus, definition } \text{poccur} \} \\
& \text{poccur}(u\sigma, p) \quad \square
\end{aligned}$$

Property 9. The algorithms **Build_Oracle_2** and **Build_Oracle** construct equivalent automata.

Proof: We prove this by induction on the states. Our induction hypothesis is that for each state j ($0 \leq j < i$), $\min(j)$ is the same in both automata, and the outgoing transitions from state j are equivalent for both automata.

If $i = 0$, $u = \min(i) = \varepsilon$ in both automata. Consider a transition created by **Build_Oracle_2**, say to state k by $\sigma \neq p_{i+1}$. Since this transition exists, $u\sigma \in \mathbf{fact}(p)$ and $k = \text{poccur}(u\sigma, p)$ (due to Property 6, vi.). Using Properties 7 and 8, such a transition was created by **Build_Oracle** as well. Similarly, consider a transition created by **Build_Oracle**, say to state k by σ . This transition, say on symbol σ , leads to state $k = i - |u| + \text{poccur}(u\sigma, p_{i-|u|+1} \dots p_m)$ and was created since $u\sigma \in \mathbf{fact}(p_{i-|u|+1} \dots p_m)$ (see the algorithm). Using Properties 7 and 8, such a transition was created by **Build_Oracle_2** as well.

If $i > 0$, using the induction hypothesis and acyclicity of the automata, i has the same incoming transitions and as a result $\min(i)$ is the same for both automata. Using the same arguments as in case $i = 0$, the outgoing transitions from state i are equivalent for both automata.

As a result, the two automata are equivalent. \square

4 Language of Factor Oracles

A definition of the language $\mathbf{factoracle}(p)$ not employing an automaton and its construction algorithm was left as an open question in [1]. It is

straightforward to see that it is bounded from above by $\mathbf{seq}(p)$, the set of all subsequences of p : factor oracles are acyclic, all transitions go from a state i to $j > i$, and all transitions going to some state j are labeled by p_j .

In this section, we show that the language is prefix-, suffix- and hence factor-closed. We also give two properties showing sufficient conditions for $\mathbf{factoracle}(p) \supset \mathbf{fact}(p)$ to hold.

Property 10. The language $\mathbf{factoracle}(p)$ is prefix-closed:

$$w \in \mathbf{factoracle}(p) \Rightarrow \mathbf{pref}(w) \subseteq \mathbf{factoracle}(p) \quad (p, w \in V^*)$$

Proof: Follows directly from the fact that all states of $\text{Oracle}(p)$ —and thus all states on the path from state 0 spelling w —are final. \square

In [1, Lemma 5], the authors prove that the language recognized by a factor oracle is suffix-closed. Their proof is rather hard to follow and we therefore give a slightly different, longer version here.

Property 11. The language $\mathbf{factoracle}(p)$ is suffix-closed:

$$w \in \mathbf{factoracle}(p) \text{ is recognized in state } n \Rightarrow \\ \mathbf{suff}(w) \subseteq \mathbf{factoracle}(p) \text{ and } u \in \mathbf{suff}(w) \text{ is recognized in state } o \leq n \\ (p, u, w \in V^*)$$

Proof:

By induction on $|w|$. It is true if $|w| = 0$ or $|w| = 1$. Assume $|w| \geq 2$ and that it is true for all strings x such that $|x| < |w|$. We show that it is also true for w , recognized in n .

Let $w = xa$ ($x \neq \varepsilon$), x is recognized in h ($0 < h < n$). Consider a proper suffix of w . It either equals ε and is recognized in state $0 \leq n$ or it can be written as va where $v <_s x$.

According to the induction hypothesis, v is recognized in state $l \leq h$. Let $\bar{x} = \min(h)$ and $\bar{v} = \min(l)$. Due to Property 6, iv., $\bar{x} \leq_s x$ and $\bar{v} \leq_s v$. We now prove that $\bar{v} \leq_s \bar{x}$. If $l = h$, then $\bar{v} = \bar{x}$. Now consider the case $l < h$. Since $v \leq_s x$ and $\bar{v} \leq_s v$, $\bar{v} \leq_s x$. Due to Property 6, vii., $\bar{x} \not\leq_s \bar{v}$. Thus, since \bar{v} and \bar{x} both are suffixes of x , $\bar{v} \leq_s \bar{x}$. Since \bar{x} is recognized in h and there is a transition by a from h , by Property 6, vi. we have that $\bar{x}a \in \mathbf{fact}(p)$ and $n = \text{poccur}(\bar{x}a, p)$. Since $\bar{x}a \in \mathbf{fact}(p)$, $\bar{x}az \leq_s p$ for some $z \in V^*$, hence $\bar{v}az \leq_s p$ as well, hence $\bar{v}a$ is recognized. Since \bar{v} is recognized in l , there is a transition by a from l to a state o . We know that $o = \text{poccur}(\bar{v}a, p)$ due to Property 6, vi. Since $\bar{v}a \leq_s \bar{x}a$, $\text{poccur}(\bar{v}a, p) \leq \text{poccur}(\bar{x}a, p)$ due to Property 5 and hence $o \leq n$. \square

Property 12. The language **factoracle** is factor-closed:

$$w \in \mathbf{factoracle}(p) \Rightarrow \mathbf{fact}(w) \in \mathbf{factoracle}(p) \quad (p, w \in V^*)$$

Proof: Follows from Properties 10 and 11 and $\mathbf{fact}(w) = \mathbf{pref}(\mathbf{suff}(w))$. \square

Remark 1. For $\mathbf{fact}(p)$, the equality $\mathbf{fact}(p)^r = \mathbf{fact}(p^r)$ holds for all $p \in V^*$. The equality $\mathbf{factoracle}(p)^r = \mathbf{factoracle}(p^r)$ does not hold for all $p \in V^*$. An example of a p for which the equality does not hold is $p = baabba$. The factor oracles for p and p^r are given in Figure 2. It is clear that $bab \in \mathbf{factoracle}(baabba)^r$ but $bab \notin \mathbf{factoracle}((baabba)^r)$. \square

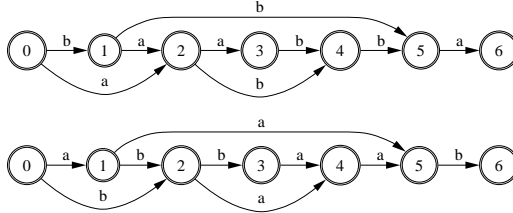


Fig. 2. Factor oracles for $baabba$ and $abbaab$

Due to Property 12, it must be possible to characterize a function $skip(p)$ returning a set of strings such that $\mathbf{factoracle}(p) = \mathbf{fact}(skip(p))$. The exact definition of this function $skip$ is still unclear, but the following two properties give some insight in the language $\mathbf{factoracle}(p)$:

Property 13 (Relationship between non-factor strings and repetitions). If there are no repetitions of any symbol a in p , $\mathbf{factoracle}(p) \subseteq \mathbf{fact}(p)$.

Proof: We prove this based on an induction hypothesis on the partial factor oracle constructed according to the algorithm, with the suffixes of p upto $p_i \dots p_m$ (i.e. of length $\geq m - i + 1$) already added to the automaton. Our induction hypothesis is that the transitions in this partial factor oracle are exactly those from j to $j + 1$ on p_{j+1} for $0 \leq j < m$ and those from 0 to j on p_j for $1 \leq j \leq i$ and that the language recognized by this partial factor oracle is a subset of or equal to $\mathbf{fact}(p)$.

In case $i = 1$, the automaton clearly recognizes $\mathbf{pref}(p)$, and $\mathbf{pref}(p) \subseteq \mathbf{fact}(p)$.

Assume that the induction hypothesis holds for $0 \leq j \leq i$. The algorithm will construct a transition from 0 to $i+1$ by p_{i+1} in step 8 due to the absence of repetitions in p . New strings recognized will be $\mathbf{pref}(p_{i+1} \dots p_m)$, and $\mathbf{pref}(p_{i+1} \dots p_m) \subseteq \mathbf{fact}(p)$. Thus, the language recognized is a subset of or equal to $\mathbf{fact}(p)$ and the transitions are exactly those from j to $j+1$ for $0 \leq j < m$ and those from 0 to j for $1 \leq j \leq i+1$. \square

Property 14. If there is no state $i > 0$ in the factor oracle on p with at least 2 outgoing transitions, $\mathbf{factoracle}(p) = \mathbf{fact}(p)$.

Proof: In this case every path from state 0 to state m is labeled by a suffix of p . \square

5 Construction based on Trie

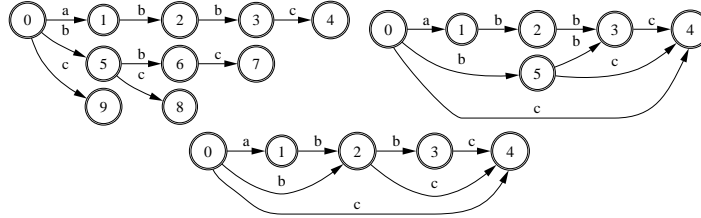


Fig. 3. (Left to right, top to bottom) Trie, DAWG and factor oracle recognizing $\mathbf{fact}(abc)$, $\mathbf{fact}(abc)$ and $\mathbf{fact}(abc) \cup \{abc\}$ respectively

There is a close relationship between the data structures $\mathbf{Trie}(\mathbf{fact}(p))$ —the *trie* ([7]) on $\mathbf{fact}(p)$ —recognizing exactly $\mathbf{fact}(p)$, $\mathbf{DAWG}(\mathbf{fact}(p))$ —the *directed acyclic word graph* ([6, 5]) on $\mathbf{fact}(p)$ —recognizing exactly $\mathbf{fact}(p)$, and $\mathbf{Oracle}(p)$ —the factor oracle on p —which recognizes at least $\mathbf{fact}(p)$.

It is a well known fact that $\mathbf{DAWG}(\mathbf{fact}(p))$ can be constructed from $\mathbf{Trie}(\mathbf{fact}(p))$ by merging states whose right languages are identical (see for example [6, 5]). The factor oracle as defined by $\mathbf{Oracle}(p)$ can also be constructed from $\mathbf{Trie}(\mathbf{fact}(p))$, by merging states whose right languages have identical longest strings (which are suffixes of p). An example of a trie, DAWG and factor oracle for the factors of abc can be seen in Figure 3.

Definition 2. We define $\mathbf{Trie}(S)$ as a 5-tuple $\langle Q, V, \delta, \varepsilon, F \rangle$ where S is a finite set of strings, $Q = \mathbf{pref}(S)$ is the set of states, V is the

alphabet, δ is the transition function, defined by

$$\delta(u, a) = \begin{cases} ua & \text{if } ua \in \mathbf{pref}(S) \\ \perp & \text{if } ua \notin \mathbf{pref}(S) \end{cases} \quad (u \in \mathbf{pref}(S), a \in V),$$

ε is the single start state and $F = S$ is the set of final states. \square

Property 15. For $u, v \in \mathbf{fact}(p)$ we have :

$$uv \in \mathbf{fact}(p) \wedge (\forall w : uw \in \mathbf{fact}(p) : |w| \leq |v|) \quad \Rightarrow uv \in \mathbf{suff}(p)$$

$$\begin{aligned} uv_1 \in \mathbf{fact}(p) \wedge (\forall w : uw \in \mathbf{fact}(p) : |w| \leq |v_1|) \\ \wedge uv_2 \in \mathbf{fact}(p) \wedge (\forall w : uw \in \mathbf{fact}(p) : |w| \leq |v_2|) \Rightarrow v_1 = v_2 \end{aligned}$$

\square

Property 16. For $u \in \mathbf{fact}(p)$ and $C \in \mathbb{N}$,

$$(\forall w : uw \in \mathbf{fact}(p) : |w| \leq C) \equiv (\forall w : uw \in \mathbf{suff}(p) : |w| \leq C)$$

Proof: \Rightarrow : trivial. \Leftarrow : Let $ux \in \mathbf{fact}(p)$, then $(\exists y : : uxy \in \mathbf{suff}(p))$, hence $(\exists y : : |xy| \leq C)$, and since $|y| \geq 0$, $|x| \leq C$. \square

Using Properties 15 and 16, $\mathit{max}_p(u)$ can be defined as the unique longest string v such that $uv \in \mathbf{suff}(p)$:

Definition 3. Define $\mathit{max}_p(u) = v$ where v is such that

$$uv \in \mathbf{suff}(p) \wedge (\forall w : uw \in \mathbf{suff}(p) : |w| \leq |v|)$$

\square

We now present our trie-based construction algorithm for factor oracles:

Trie_To_Oracle($p = p_1p_2\dots p_m$)

- 1: Construct **Trie**($\mathbf{fact}(p)$)
- 2: **for** i from 2 to m **do**
- 3: Merge all states u for which $\mathit{max}_p(u) = p_{i+1}\dots p_m$ into the single state $p_1\dots p_i$

The order in which the values of i are considered is not important. In addition, note that it is not necessary to consider the states u for which $\mathit{max}_p(u) = p_2\dots p_m$ since there is precisely one such state u in $\mathbf{Trie}(\mathbf{fact}(p))$, $u = p_1$. Due to Property 15, it is sufficient to only consider suffixes of p as longest strings.

Also note that the intermediate automata may be nondeterministic, but the final automaton will be weakly deterministic (as per Property 4).

The above algorithm has complexity $\mathcal{O}(m^2)$ (assuming that $max_p(u)$ was computed during construction of the trie). The construction of a Trie can be done in $\mathcal{O}(m)$ time however, and the merging of the states is similar to minimization of an acyclic automaton, which can also be done in $\mathcal{O}(m)$. We therefore conjecture that an $\mathcal{O}(m)$ trie-based factor oracle construction algorithm exists, although we have not investigated this due to lack of time.

To prove that algorithm **Trie_To_Oracle** constructs Oracle(p), we define a partition on the states of the trie, induced by an equivalence relation on the states.

Definition 4. Relation \sim_p on states of Trie(**fact**(p)) is defined by

$$t \sim_p u \equiv max_p(t) = max_p(u) \quad (t, u \in \mathbf{fact}(p))$$

Note that relation \sim_p is an equivalence relation. □

We now show that the partitioning into sets of states of Trie(**fact**(p)) induced by \sim_p , is the same as the partitioning of Trie(**fact**(pa)) induced by \sim_{pa} , restricted to the states of Trie(**fact**(p)), i.e.

Property 17.

$$t \sim_p u \equiv t \sim_{pa} u \quad (t, u \in \mathbf{fact}(p), a \in V)$$

Proof:

$$\begin{aligned} & t \sim_p u \\ \equiv & \quad \{ \text{definition } \sim_p \} \\ & max_p(t) = max_p(u) \\ \equiv & \quad \{ \} \\ & max_p(t)a = max_p(u)a \\ \equiv & \quad \{ (\star) \} \\ & max_{pa}(t) = max_{pa}(u) \\ \equiv & \quad \{ \text{definition } \sim_{pa} \} \\ & t \sim_{pa} u \end{aligned}$$

where we prove (\star) by

$$\begin{aligned}
& v = \mathit{max}_{pa}(u) \\
\equiv & \quad \{ \text{definition } \mathit{max}_{pa} \} \\
& uv \in \mathbf{succ}(pa) \wedge (\forall w : uw \in \mathbf{succ}(pa) : |w| \leq |v|) \\
\equiv & \quad \{ u \in \mathbf{fact}(p), \text{ hence } (\exists x : : uxa \in \mathbf{succ}(pa)), \\
& \quad \text{hence } |xa| > 0 \text{ and } |v| > 0; \mathbf{succ}(pa) = \mathbf{succ}(p)a \cup \{\varepsilon\} \} \\
& uv \in \mathbf{succ}(p)a \wedge (\forall w : uw \in \mathbf{succ}(pa) : |w| \leq |v|) \\
\equiv & \quad \{ |v| > 0, \text{ introduction } v' \} \\
& uv \in \mathbf{succ}(p)a \wedge (\forall w : w \neq \varepsilon \wedge uw \in \mathbf{succ}(pa) : |w| \leq |v|) \wedge v = v'a \\
\equiv & \quad \{ \mathbf{succ}(pa) = \mathbf{succ}(p)a \cup \{\varepsilon\} \} \\
& uv \in \mathbf{succ}(p)a \wedge (\forall w : w \neq \varepsilon \wedge uw \in \mathbf{succ}(p)a : |w| \leq |v|) \wedge v = v'a \\
\equiv & \quad \{ w = w'a \} \\
& uv \in \mathbf{succ}(p)a \wedge (\forall w' : uw'a \in \mathbf{succ}(p)a : |w'a| \leq |v'a|) \wedge v = v'a \\
\equiv & \quad \{ \} \\
& uv \in \mathbf{succ}(p)a \wedge (\forall w' : uw' \in \mathbf{succ}(p) : |w'| \leq |v'|) \wedge v = v'a \\
\equiv & \quad \{ v = v'a \} \\
& uv' \in \mathbf{succ}(p) \wedge (\forall w' : uw' \in \mathbf{succ}(p) : |w'| \leq |v'|) \wedge v = v'a \\
\equiv & \quad \{ \text{definition } \mathit{max}_p \} \\
& v' = \mathit{max}_p(u) \wedge v = v'a \\
\equiv & \quad \{ \text{elimination } v' \} \\
& v = \mathit{max}_p(u)a
\end{aligned}$$

□

Property 18. Algorithm **Trie_To_Oracle** constructs $\mathit{Oracle}(p)$.

Proof: By induction on $|p| = m$. If $m = 0$, $p = \varepsilon$, and $\mathit{Trie}(\mathbf{fact}(\varepsilon)) = \mathit{Oracle}(\varepsilon)$. If $m = 1$, $p = a$ ($a \in V$), and $\mathit{Trie}(\mathbf{fact}(a)) = \mathit{Oracle}(a)$. If $m > 1$, $p = xa$ ($x \in V^*$, $a \in V$), and we may assume the algorithm to construct part $\mathit{Oracle}(x)$ of $\mathit{Oracle}(xa)$ correctly (using $\mathbf{fact}(ua) = \mathbf{fact}(u) \cup \mathbf{succ}(u)a$, $\mathit{Trie}(\mathbf{fact}(xa))$ being an extension of $\mathit{Trie}(\mathbf{fact}(x))$, and $\mathit{Oracle}(xa)$ being an extension of $\mathit{Oracle}(x)$ (which is straightforward to see from algorithm **Build_Oracle_2** as well as [1, page 57, after Corollary 4]), and Property 17). Now consider the states of this partially converted automaton in which suffixes of x are recognized. By construc-

tion of the trie, there are transitions from these states by a . The factor oracle construction according to algorithm **Oracle_Sequential** in [1] creates $\text{Oracle}(xa)$ from $\text{Oracle}(x)+a$ (i.e. the factor oracle for x extended with a single new state m reachable from state $m-1$ by symbol $p_m = a$) by creating new transitions to state m from those states in which suffixes of x are recognized and that do not yet have a transition on a . Since **Trie_To_Oracle** merges all states t for which $\text{max}_{xa}(t) = a$ into the single state m , $\text{Oracle}(xa)$ is constructed correctly from $\text{Trie}(\mathbf{fact}(xa))$. \square

6 Conclusions and Future Work

We have presented two alternative construction algorithms for factor oracles and shown the automata constructed by them to be equivalent to those constructed by the algorithms in [1]. Although both our algorithms are $\mathcal{O}(m^2)$ and thus practically inefficient compared to the $\mathcal{O}(m)$ sequential algorithm given in [1], they give more insight into factor oracles.

Our first algorithm is more intuitive to understand and makes it immediately obvious, without the need for several lemmas, that the factor oracle recognizes at least $\mathbf{fact}(p)$ and has m to $2m-1$ transitions.

Our second algorithm gives a clear insight into the relationship between the trie or dawg recognizing $\mathbf{fact}(p)$ and the factor oracle recognizing a superset thereof. We conjecture that an $\mathcal{O}(m)$ trie-based algorithm for the construction of factor oracles exists, although we have not investigated this due to lack of time.

Although an automaton-independent characterization of the language $\mathbf{factoracle}(p)$ remains to be defined, we have given clear proofs that $\mathbf{fact}(p) \subseteq \mathbf{factoracle}(p)$. In addition, we have shown some sufficient conditions for $\mathbf{fact}(p) \subset \mathbf{factoracle}(p)$ to hold.

We are still working on an automaton-independent characterization of the language. Such a characterization would enable us to calculate how many strings are recognized that are not factors of the original string. This could be useful in determining whether to use a factor oracle-based algorithm in pattern matching or not.

As stated in [1], the factor oracle is not minimal in terms of number of transitions among the automata with $m+1$ states recognizing at least $\mathbf{fact}(p)$. We note that it is not even minimal among the subset of such automata having Glushkov's property (see Figure 4).

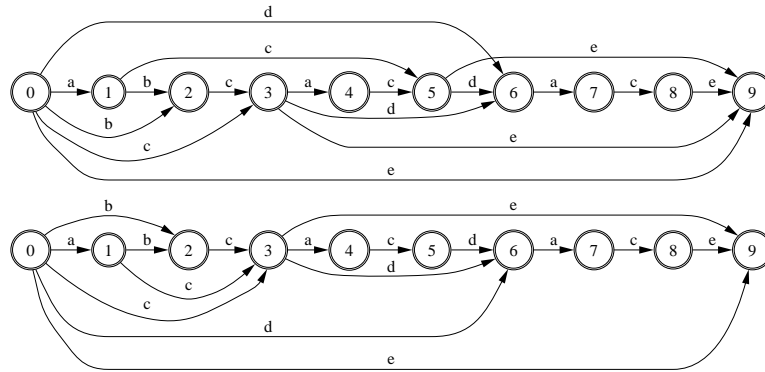


Fig. 4. Factor oracle recognizing a superset of $\mathbf{fact}(p)$ (including for example $cace \notin \mathbf{fact}(p)$) and alternative automaton with $m + 1$ states satisfying Glushkov's property yet recognizing a *different* superset of $\mathbf{fact}(p)$ (including for example $acacdade \notin \mathbf{factoracle}(p)$, but not $cace$) and having less transitions, for $p = abcacdade$.

References

1. Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Efficient Experimental String Matching by Weak Factor Recognition. In *Proceedings of the 12th conference on Combinatorial Pattern Matching*, volume 2089 of *LNCS*, pages 51–72, 2001.
2. Cyril Allauzen and Mathieu Raffinot. Oracle des facteurs d'un ensemble de mots. Technical Report 99-11, Institut Gaspard-Monge, Université de Marne-la-Vallée, 1999.
3. Loek Cleophas, Gerard Zwaan, and Bruce W. Watson. Constructing Factor Oracles. In *Proceedings of the Prague Stringology Conference 2003*, 2003.
4. Loek G.W.A. Cleophas. Towards SPARE Time: A New Taxonomy and Toolkit of Keyword Pattern Matching Algorithms. Master's thesis, August 2003.
5. Maxime Crochemore and Wojciech Rytter. *Text Algorithms*. Oxford University Press, 1994.
6. Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology - Text Algorithms*. World Scientific Publishing, 2003.
7. E. Fredkin. Trie memory. *Communications of the ACM*, 3(10):490–499, 1960.
8. Arnaud Lefebvre and Thierry Lecroq. Computing repeated factors with a factor oracle. In L. Brankovic and J. Ryan, editors, *Proceedings of the 11th Australasian Workshop on Combinatorial Algorithms*, pages 145–158, 2000.
9. Arnaud Lefebvre and Thierry Lecroq. Compror: on-line lossless data compression with a factor oracle. *Inf. Process. Lett.*, 83(1):1–6, 2002.
10. Arnaud Lefebvre, Thierry Lecroq, and J. Alexandre. Drastic improvements over repeats found with a factor oracle. In E. Billington, D. Donovan, and A. Khodkar, editors, *Proceedings of the 13th Australasian Workshop on Combinatorial Algorithms*, pages 253–265, 2002.
11. Bruce W. Watson and Loek Cleophas. SPARE Parts: A C++ toolkit for String PAttern REcognition. *Software: Practice and Experience*, 2003. To be published.

A Proof of Property 6

We first consider the automaton constructed in steps 1-4 of the algorithm. It is straightforward to verify that the properties hold for $i = 2$.

Now assume that the properties hold for the automaton with all suffixes of p of length greater than $m - i + 1$ already considered. We prove that they also hold for the automaton after the suffix of length $m - i + 1$, $p_i \dots p_m$, has been considered.

If $k = m$ in step 6, suffix $p_i \dots p_m$ is already recognized, no new transition will be created, the automaton does not change and the properties still hold.

If $k < m$, then we need to prove that each of the properties holds for the new automaton.

Ad i: By v., string $p_i \dots p_k$ is recognized in state $j \leq \text{poccur}(p_i \dots p_k, p)$. Since $p_i \dots p_k \leq_s p_1 \dots p_k$ and $\text{poccur}(p_1 \dots p_k, p) = k$, $\text{poccur}(p_i \dots p_k, p) \leq k$ due to Property 5. Since $j \leq k$, the transition created from j to $k + 1$ is a forward one.

Ad ii: Trivial.

Ad iii: We prove that the property holds for the new automaton by showing that $k = k_i \geq k_{max}$, i.e. k will become the new k_{max} .

If $k_{max} = -\infty$, $k \geq k_{max}$ clearly holds.

If $k_{max} > -\infty$, assume that $k_{max} > k$, then there is an h such that $1 < h < i \wedge k_h < m \wedge k_h = k_{max}$. Factor $p_h \dots p_k$ is recognized in $g \leq k$ due to ii. and v.

If $g = k$, then $p_h \dots p_k$ is recognized in k and $p_h \dots p_m$ is recognized in m ; so $k_h = m$ which contradicts $k_h < m$.

If $g < k$, then $p_h \dots p_k$ is recognized in $g < k$. Since $p_i \dots p_k$ is recognized in $j = j_i$ and $p_i \dots p_k \leq_s p_h \dots p_k$, due to viii., $j \leq g$.

If $j = g$, then $p_h \dots p_k$ is the longest prefix of $p_h \dots p_m$ recognized by the old automaton, which contradicts ii.

If $j < g$, then $j < g < k$. We know that $\min(g) \leq_s p_h \dots p_k$ (using iv.), $\min(j) \leq_s p_h \dots p_k$ (using iv. and $p_i \dots p_k \leq_s p_h \dots p_k$) and therefore that $\min(j) <_s \min(g)$ (due to vii.). Let l be the state to which the transition by p_{k+1} from g leads, i.e. l is the state in which $p_h \dots p_{k+1}$ is recognized. Using vi., we have that $l = \text{poccur}(\min(g) \cdot p_{k+1}, p)$. Using Property 5 we have that $l \leq \text{poccur}(p_h \dots p_{k+1}, p)$ and the latter is $\leq k + 1$ due to the definition of poccur (since $k + 1$ marks the end of an occurrence of $p_h \dots p_{k+1}$). We have $\text{poccur}(\min(j) \cdot p_{k+1}, p) \leq \text{poccur}(\min(g) \cdot p_{k+1}, p) = l$ since $\min(j) \leq_s \min(g)$. We want to prove that $k + 1 \leq \text{poccur}(\min(j) \cdot p_{k+1}, p)$. Assume that $\text{poccur}(\min(j) \cdot p_{k+1}, p) < k + 1$. If the first occurrence of

$\min(j) \cdot p_{k+1}$ starts before position i of p , then it is a prefix of a suffix of p longer than $p_i \dots p_m$ and thus by ii. $\min(j) \cdot p_{k+1}$ is recognized. Since $\min(j)$ is recognized in j , a transition from j by p_{k+1} must exist and we have a contradiction. If the first occurrence of $\min(j) \cdot p_{k+1}$ starts at or after position i of p , then there exists a shortest string x such that $x \cdot \min(j) \cdot p_{k+1} \in \mathbf{pref}(p_i \dots p_k)$ and $x \cdot \min(j) \cdot p_{k+1}$ is recognized in a state $\leq j$. But then $x \cdot \min(j)$ is recognized in a state $n < j$. By viii., since $\min(j) \leq_s x \cdot \min(j)$, this means that $\min(j)$ is recognized in state $s \leq n < j$ and we have a contradiction. Thus $k + 1 \leq \text{poccur}(\min(j) \cdot p_{k+1}, p) \leq l$ and therefore, since $l \leq k + 1$ holds, $l = k + 1$. In that case, $p_h \dots p_{k+1}$ is recognized in $l = k + 1$ and $p_h \dots p_m$ is recognized in m . But then $k_h = m$, and we have a contradiction.

Thus, $k_{max} = k_h \leq k = k_i$ and iii. holds for the new automaton.

Ad iv: Let $s = \min(j)$, $t = \min(k+1)$ and $u = \min(h)$ ($k+1 \leq h \leq m$) respectively in the old automaton. Due to the proof of iii., $k = k_i \geq k_{max}$ and therefore a unique path between $k + 1$ and h exists, labeled r , and—due to iv— $u \leq_s tr$.

If $|sp_{k+1}r| \geq |u|$, u remains the minimal length string recognized in state h . Since $s \leq_s p_i \dots p_k$, $sp_{k+1}r \leq_s p_i \dots p_{k+1}r$. Since $u \leq_s tr$, $tr \leq_s p_1 \dots p_{k+1}r$ and $|sp_{k+1}r| \geq |u|$, $u \leq_s sp_{k+1}r$ and—due to iv.— $u \leq_s s'p_{k+1}r$ as well for any s' recognized in state j .

If $|sp_{k+1}r| < |u|$, $sp_{k+1}r$ is the new minimal length string recognized in state h . Since $s \leq_s p_i \dots p_k$, $sp_{k+1}r \leq_s p_i \dots p_{k+1}r$. Since $u \leq_s tr$, $tr \leq_s p_1 \dots p_{k+1}r$ and $|sp_{k+1}r| < |u|$, $sp_{k+1}r \leq_s u$ and—due to iv.— $sp_{k+1}r \leq_s s'p_{k+1}r$ as well for any s' recognized in state j .

Since $p_i \dots p_{k+1}r$ was not recognized before, it is not a prefix of p , $p_2 \dots p_m$, ..., $p_{i-1} \dots p_m$ (using ii.), hence $\text{poccur}(p_i \dots p_{k+1}r, p) = k + 1 + |r|$. Since $s \leq_s p_i \dots p_k$, $\text{poccur}(sp_{k+1}r, p) \leq k + 1 + |r|$. Assume that $\text{poccur}(sp_{k+1}r, p) < k + 1 + |r|$, then $p_i \dots p_{k+1}r = usp_{k+1}rv$ ($u, v \in V^*$, $v \neq \varepsilon$, $|u|$ minimal), since $sp_{k+1}r$ cannot start before p_i because in that case it would have already been recognized by the old automaton. Factor us is recognized in state $g < j$ (using i.) and—since viii. holds— $s \leq_s us$ is recognized in a state $o \leq g < j$. This contradicts s being recognized in j . As a result $\text{poccur}(sp_{k+1}r, p) = k + 1 + |r|$.

Ad v: Any new factor of p recognized after creation of the transition from j to $k + 1$ has the form $vp_{k+1}r$ and is recognized in $k + 1 + |r|$ with $v \in \mathbf{fact}(p)$ recognized in state j . Since $k + 1 + |r| = \text{poccur}(\min(k+1)r, p)$ (using iii., iv. holding for the new automaton plus the fact that k is the new k_{max}) and $\min(k + 1) \cdot r \leq_s vp_{k+1}r$ due to iv. holding for the new automaton, $k + 1 + |r| \leq \text{poccur}(vp_{k+1}r, p)$ using Property 5.

Ad vi: The states n we have to consider are $n = j$ and $n = h$ for $k + 1 \leq h \leq m$.

For $n = j$, a new transition to $k + 1$ is created and by iv., $\min(j) \leq_s p_i \dots p_k$, hence we have $\min(j) \cdot p_{k+1} \leq_s p_i \dots p_{k+1}$, $p_{k+1-|\min(j)|} \dots p_{k+1} = \min(j) \cdot p_{k+1}$, $\min(j) \cdot p_{k+1} \in \mathbf{fact}(p)$ and $\text{poccur}(\min(j) \cdot p_{k+1}, p) \leq k + 1$. Since $\min(j) \cdot p_{k+1}$ is recognized in state $k + 1$, due to v. for the new automaton, $k + 1 \leq \text{poccur}(\min(j) \cdot p_{k+1}, p)$. Therefore $k + 1 = \text{poccur}(\min(j) \cdot p_{k+1}, p)$.

For $n = h$ with $k + 1 \leq h \leq m$, $\min(h)$ changes to $sp_{k+1}r$ if and only if $|sp_{k+1}r| < |u|$ (with r, s, u as in the proof of iv.). We know that $ua \in \mathbf{fact}(p)$ and $q = \text{poccur}(ua, p)$. Since $sp_{k+1}r \leq_s u$, $sp_{k+1}ra \leq_s ua$, hence $sp_{k+1}ra \in \mathbf{fact}(p)$ as well and $\text{poccur}(sp_{k+1}ra, p) \leq \text{poccur}(ua, p) = q$, but due to v., $q \leq \text{poccur}(sp_{k+1}ra, p)$ hence $q = \text{poccur}(sp_{k+1}ra, p)$.

Ad vii: Assume $\min(n) \leq_s \min(q)$. We have $\text{poccur}(\min(n), p) \leq \text{poccur}(\min(q), p)$ due to Property 5, which according to iv. is equivalent to $n \leq q$.

Ad viii: By induction on $|w|$. It is true if $|w| = 0$ or $|w| = 1$. Assume that it is true for all strings x such that $|x| < |w|$. We will show that it is also true for w , recognized in n .

Let $w = xa$ ($x \neq \varepsilon$), x is recognized in h ($0 < h < n$). Consider a proper suffix of w , recognized in state q . It either equals ε and is recognized in state $0 \leq n$ or it can be written as va where $v <_s x$.

Suffix va of w is recognized, therefore suffix v of x is recognized and according to the induction hypothesis, v is recognized in state $l \leq h$. Let $\bar{x} = \min(h)$ and $\bar{v} = \min(l)$. Due to iv. for the new automaton, $\bar{x} \leq_s x$ and $\bar{v} \leq_s v$. We now prove that $\bar{v} \leq_s \bar{x}$. If $l = h$, then $\bar{v} = \bar{x}$. Now consider the case $l < h$. Since $v \leq_s x$ and $\bar{v} \leq_s v$, $\bar{v} \leq_s x$. Due to vii., $\bar{x} \not\leq_s \bar{v}$. Thus, since \bar{v} and \bar{x} both are suffixes of x , $\bar{v} \leq_s \bar{x}$. Since \bar{x} is recognized in h and there is a transition by a from h to n , by vi. for the new automaton we have that $\bar{x}a \in \mathbf{fact}(p)$ and $n = \text{poccur}(\bar{x}a, p)$. Since \bar{v} is recognized in l and there is a transition by a from l to q , $\bar{v}a \in \mathbf{fact}(p)$ and $q = \text{poccur}(\bar{v}a, p)$ due to vi. for the new automaton. Since $\bar{v}a \leq_s \bar{x}a$, $\text{poccur}(\bar{v}a, p) \leq \text{poccur}(\bar{x}a, p)$ due to Property 5 and hence $q \leq n$.

We have shown that the properties hold for every partial automaton during the construction. Consequently, they hold for the complete automaton Oracle(p). \square