INSTYTUT INFORMATYKI
UNIWERSYTETU WROCŁAWSKIEGO

INSTITUTE OF COMPUTER SCIENCE
UNIVERSITY OF WROCŁAW

ul. Joliot-Curie 15
50–383 Wrocław
Poland

Report 03/07

Artur Jeż, Tomasz Jurdziński, Krzysztof Loryś

Results for extensions of prefix grammars

May 2007

RESULTS FOR EXTENSIONS OF PREFIX GRAMMARS

ARTUR JEŻ, TOMASZ JURDZIŃSKI, KRZYSZTOF LORYŚ

ABSTRACT. Frazier and Page [FPJ94] introduced prefix grammars as a slight modification of Context-Sensitive Grammars (CSG) — with a fixed finite starting set and a restriction, that any production can be applied only to the prefix of a word. They proved that the set of languages generated by those grammars produce coincide with the regular languages. They also gave an inefficient algorithms converting a DFA to a prefix grammar and converting prefix grammars to a $\mathcal{NFA}$. In [RQ02] Ravikumar and Quan gave an efficient polynomial time algorithms for those two problems. They also posed a question what happens, when we enrich the prefix grammar — swap a finite starting set for a regular set or allow applying the productions to the suffixes of the word as well. They conjectured that those enrichments still lead to a regular set. In this paper we prove those conjectures and also give a polynomial time algorithm to transform such a grammar to an equivalent $\mathcal{NFA}$.

Key words: Prefix grammars, regular languages, rewriting systems

1. BACKGROUND AND HISTORY

In [FPJ94] prefix grammars were defined as a triple $G = \langle \Sigma, S, P \rangle$ where Σ is a finite alphabet, S is a finite set of words over this alphabet, called base strings or base words, and P is a finite set of production of the form $\alpha \to \beta$ with α and β — finite words over Σ. A production $\alpha \to \beta$ can be applied to a word w if and only if $w = \alpha w'$ and this application results in a word $\beta w'$. The way of applying productions justifies the name 'prefix grammar'. Language $L(G)$ consists of words generated from base set S by productions P.

It was proved in [FPJ94] that every language defined by a prefix grammar is regular and also every regular language can be described by means of a prefix grammar. This result was strengthened in [RQ02] — given a prefix grammar G we can construct an equivalent $\mathcal{NFA}$ in polynomial time and any DFA can be transformed to a prefix grammar in polynomial time. Also in this paper an example of an infinite family of languages for which minimal prefix grammar is exponential in size of minimal $\mathcal{NFA}$ and an infinite family of languages for which minimal $\mathcal{DFA}$ is exponential in the size of minimal prefix grammar were presented, hence a polynomial time algorith transforming a $\mathcal{NFA}$ to a prefix grammar is not possible.

In [RQ02] some question were posed about the properties of prefix grammars and their possible extensions. In this paper we deal with two of them and some natural natural generalizations. Namely — what happens when we allow the base set to be a regular set? It was conjectured that such grammar still generates regular

language. We prove this conjecture. Another extension of prefix grammars is to allow applying productions from both the beginning and the end of the word and so receive prefix-suffix grammar (please note, that no specific name has been used before). We prove the conjecture that languages generated by such grammars are regular. The results holds even if we allow a base set to be a regular set. For all three proofs we supply polynomial time algorithms to transform the considered variance of prefix grammar to an equivalent $\mathcal{NFA}$.

2. Prefix grammars with regular base set

In this section we deal with the case when starting set is a regular set. We assume that this set is described by an $\mathcal{NFA}$. We give an informal description of the algorithm of transforming it to an equivalent $\mathcal{NFA}$. This algoritm runs in polynomial time in G and given $\mathcal{NFA}$.

We work with a prefix grammar $G = \langle \Sigma, L(M), P\rangle$ which we abbreviate to $\langle \Sigma, M, P\rangle$. Since we usually refer to grammars of the form $\langle \Sigma, S, P\rangle$ for some set S, so it is conveniant to think, that our grammar G has fixed alphabet and productions, but its base set may vary. Therefore for fixed G with alphabet Σ and productions P we define $G(w) := \langle \Sigma, \{w\}, P\rangle$ and $L_G(w) := L(\langle \Sigma, \{w\}, P\rangle)$. Also instead of a single word w we will use sets and automata with an obvious meaning. Also for given (non-deterministic) automata M by M_q we denote automata M with starting state q and by M^q — with only one accepting state q. In both cases we assume, that q is a state of M.

Theorem 1. *For every prefix grammar G and every $\mathcal{NFA}$ M language $L_G(M)$ is regular.*

Proof. Consider any word u belonging to $L_G(M)$. This word can be obtained from (at least one) word w accepted by M. Let us fix this word, its accepting computation of M and its transformation to u. Word w can be written as $w = w_1w_2$ with w_1, w_2 — words (perhaps trivial) and w_2 is the longest suffix not modified by productions of prefix grammar while deriving u. Hence $u = u_1w_2$. The derivation of u from w induces in a natural way a derivation of u_1 from w_1. We look closer to this derivation, we denote the words obtained in consecutive steps by $x_1, \ldots, x_n$, where $x_1 = w_1$ and $x_n = u_1$, and so $w = w_1w_2 = x_1w_2 \xrightarrow{P} x_2w_2 \xrightarrow{P} \ldots \xrightarrow{P} x_nw_2 = u'w_2 = u$. Since whole w_1 was modified there is some transformation, that changes the whole x_i for some $i \in [1, n-1]$— because the last letter of some x_i must be modified and we can apply productions only to prefixes. Hence there is a production $x_i \to x_{i+1}$ in our prefix grammar, denote $x := x_{i+1}$.

Note, that u belongs to language $L_G(x)w_2$. In fact, $L_G(x)w_2 \subset L_G(M)$ — all those words can be derived from w. More, if we take a word w_1w' that it is accepted by M and in its accepting computation after reading w_1 automaton M is in the same state as after reading w_1 in fixed accepting computation for w_1w_2 then $L_G(x)w' \subset L_G(M)$: those words can be derived from w_1w', w_1w' is accepted by M and so it is a base word of $G(M)$. Condition on w' means that M_q accepts w'. So $u \in L_G(x)L(M_q) \subset L_G(M)$ and $L_G(x)L(M_q)$ is a regular set (because

$L_G(x)$ and $L(M_q)$ are regular sets). Now there are only finitely many pairs (x, q) — x is a right side of some production in P and q is a state of M. Since every word u in $L_G(M)$ belongs to some set $L_G(x)L(M_q)$ (perhaps many) then

$$L_G(M) = \bigcup_{(x,q)\in I} L_G(x)L(M_q)$$

for some finite set I, hence it is a regular language. □

This theorem answers the question stated in [RQ02]. A natural question arises — is it possible to transform given grammar $\langle \Sigma, M, P\rangle$ to an equivalent $\mathcal{NFA}$? And if so, what is the complexity of such a task? The answer is not obvious — the proof of the previous theorem does not give direct answer to this question, however, it actually gives a good hint — we prove that this task can be done in polynomial time.

Theorem 2. *We can construct $\mathcal{NFA}$ N recognizing language $L(\langle \Sigma, M, P\rangle)$ in time $Poly(|M|) \cdot Poly(|G|)$.*

Proof. For every pair (x, q) s.t. $q \in Q_M, x \in \{\beta : \alpha \to \beta \in P\}$ we want to decide, whether $L_G(x)L(M_q)$ is a subset of $L_G(M)$. It is equivalent to deciding, whether there is a word w, such that $w \to x$ and M after reading w is in state q. Also if M is non-deterministic, then any computation ending in q is good for us. Let us define $G^R = \langle \Sigma, S, P\rangle^R := \langle \Sigma, S, \{\beta \to \alpha : \alpha \to \beta \in P\}\rangle$ — a prefix grammar obtained from G by changing the directions of its productions. So we need to answer, whether $G^R(x) \cap M_q \neq \emptyset$. Since we can transform any prefix grammar to equivalent $\mathcal{NFA}$ in polynomial time, we can also answer the question of non-emptiness of $G^R(x) \cap M_q \neq \emptyset$ in polynomial time. We do it for every pair of (x, q), which ends the proof.

□

3. PREFIX-SUFFIX GRAMMARS

We now extend the definition of a prefix grammar to a prefix-suffix grammar. We define it as follows: a prefix-suffix grammar is a tuple $\langle \Sigma, S, P_p, P_s\rangle$ where P_s is a set of prefix productions and P_p is a set of suffix production, the rest of definition is equivalent to prefix grammar that is a production $\alpha \to \beta \in P_p$ can be applied to a word αw resulting in a word βw and production $\alpha \to \beta \in P_s$ can be applied to a word $w\alpha$ resulting in a word $w\beta$. Language $L(G)$ consists of words generated from S by productions $P_s \cup P_p$. In [RQ02] it was asked, whether those grammars describe the regular languages? We prove this conjecture.

Theorem 3. *For every prefix-suffix grammar $G(S)$ where S is a finite set language $L_G(S)$ is regular.*

Proof. It is enough to show it for $S = \{w\}$. We prove this theorem in two steps. First let us consider the language $L_G^{\text{no overlaps}}(w) \subset L_G(w)$ of those words u, that have derivation with no overlaps between prefix and suffix productions (although

they may as well have some overlapping derivations), that is: if a letter was produced by a suffix (resp. prefix) production, it is not used by any prefix (resp. suffix) production. We can visualize this idea as painting letters from both productions in red and blue and saying, that are are no red-blue repaintings. We claim, that $L_G^{\text{no overlaps}}(w)$ is regular. Then we reduce the general question to the first one, in the spirit of Theorem 1 and Theorem 2.

To prove, that $L_G^{\text{no overlaps}}(w)$ is a regular language it is sufficient to notice that:

$$L_G^{\text{no overlaps}}(w) = \bigcup_{\substack{w_1,w_2:\\ w_1w_2=w}} L_{G_{prefix}}(w_1)L_{G_{suffix}}(w_2)$$

where G_{prefix} (resp. G_{suffix}) is a prefix (resp. suffix) grammar obtained from G by removing its suffix productions (resp. prefix productions). Since both $L_{G_{prefix}}(w_1)$ and $L_{G_{suffix}}(w_2)$ are regular and the sum is over finite set, then indeed $L_G^{\text{no overlaps}}(w)$ is regular.

To move to the general case let us suppose, that u is obtained from w by productions of G and that during the derivation, there was at least one overlap. Let us look to the last overlap — the last production that creates letters later used by production of another kind. There are two cases — this production can be suffix or prefix, w.l.o.g. we consider the latter. So let the number of this production be p (as prefix). Let us look to the first suffix production, that uses its letters. Denote s as number of this production. There may be some production between p and s. Let us first consider the prefix production between them. We know, that letters produced by them are not used by any suffix productions. Also they do not use any letters produced by suffix productions. So we may move them all after s without changing the result of derivation. In the same way any suffix production between p and s do not use any letters produced by p, so we may move them before p, without changing the derived word. So w.l.o.g. we may assume, that p and s are consecutive productions. But after applying them both we receive word x, on which derivation is non overlapping. Also x is not long — it consists of w_1 —- proper prefix of some right side of prefix derivative and w_2 — some right side of suffix derivative. The same argument applies for the last overlap of the second type. We say that x is a seed (for w) if x can be obtained from w by productions from G and two last productions are of different type and are overlapping. Clearly, if x is a seed, then $L_G^{\text{no overlaps}}(x) \subset L_G(w)$. Also if u belongs to $L_G(w)$, then it belongs to $L_G^{\text{no overlaps}}(w)$ or there exists a seed x, such that u belongs to $L_G^{\text{no overlaps}}(x)$. Hence

$$L_G(w) = L_G^{\text{no overlaps}}(w) \cup \bigcup_{x\in I} L_G^{\text{no overlaps}}(x)$$

with x a non overlapping seed and I — some finite set. And so $L_G(w)$ is a regular set. □

A natural question arises, whether it is possible to construct an efficient algorithm to transform a prefix-suffix grammar $G(x)$ to an equivalent $\mathcal{NFA}$. The answer is positive.

Theorem 4. *There is a polynomial time algorithm, that transforms any prefix-suffix grammar $G(S)$ (with a finite base set S) to an equivalent $\mathcal{NFA}$.*

Before we prove this theorem let us first state a technical lemma. During the derivation there may be many overlaps. We divide the whole derivation into steps between two consecutive overlaps. So it is natural to ask the following question. Let G be a prefix-suffix grammar with a base word w. Is it possible to derive a word u with just one overlap during the derivation and that two overlapping productions are the last two?

Lemma 5. *For prefix-suffix grammar G and words u, w it is possible to decide in polynomial time, whether u belongs to $L_G(w)$ and its derivation has exactly one overlap and two overlapping productions are the last two.*

Proof. W.l.o.g. we can assume, that the last production s was a suffix one and it overlapped some prefix production p. Let those two productions be of the form $\alpha_p \to \beta_p$ and $\alpha_s \to \beta_s$ respectively. So if we look at the word u' that was obtained after applying p but before s, it begins with β_p and ends with α_s and those two words overlap in u'. Since u' was derived in a non-overlapping way, then w can be partitioned to w_1, w_2 such that $w = w_1 w_2$ and u' can be partitioned to $u' = u'_1 u'_2$ satisfying:

(1) β_p is a prefix of u'_1
(2) u'_1 is derived by prefix part of G from word w_1
(3) u'_2 is derived by suffix part of G from word w_2

All those condition represents the idea that we in parallel derive a word u'_1 from w_1 by prefix part and u'_2 by suffix part from w_2, the last prefix production is p and then it is possible to apply s that will overlap p, hence using whole u'_2 and some part of u'_1. Choosing all possible $\alpha_p \to \beta_p$, $\alpha_s \to \beta_s$, w_1, w_2, u'_1, u'_2 and validating them can be done in polynomial time — we first choose β_p and α_s combine them in an overlapping way to a word u'. Then we divide this word to parts u'_1 and u'_2 with desired properties. We undo the production p on u'_1 and obtain u''_1. Now we divide w to two words w_1 and w_2 and check, whether it is possible to derive u''_1 from w_1 by prefix part of G and u'_2 from w_2 by suffix part, which can be done in polynomial time. □

We now prove the theorem 4

Proof. It is enough to prove this theorem for $S = \{w\}$. Language $L_G(w)$ can be written as $L_G(w) = L_1 \cup L_2$ where L_1 consists of those words, that have at least one non-overlapping derivation and L_2 consists of those words, that have at least one overlapping derivation. Both those sets are regular (consult Theorem 3). We construct M_1 and M_2 — $\mathcal{NFA}$'s that recognize L_1 and L_2 respectively. The case of L_1 is fairly easy, as we already know, that $L_1 = \bigcup_{u,u':w=uu'} L_{G_{\text{prefix}}}(u) L_{G_{\text{suffix}}}(u')$. Since $L_{G_{\text{prefix}}}(u)$ (respectively $G_{\text{prefix}}(u')$) can be transformed to equivalent $\mathcal{NFA}$ in time polynomial in $|G_{\text{prefix}}|$ and $|u|$ (resp. $|G_{\text{suffix}}|$ and $|u'|$) and the sum is over set of size $|w|$ then we construct M_1 in polynomial time.

The analysis for L_2 is again based on searching for the overlaps (as in lemma 5 we can assume, that overlapping productions are consecutive). We look to the derivation of a word u from w and to every overlap during this process. The most natural way of reconstructing the derivation of u is to first derive all words, that were obtained just after the first overlap. This can be done in polynomial time (by Lemma 5), denote the acquired set by S_1. Then we look for all words obtained just after the second overlap (excluding productions taking part in the first overlap), which is equivalent to searching for words derived from S_1 with only one overlap, denote those words by S_2. In the same way we construct S_n. We set $S_0 = \{w\}$ for convenience. And then we derive non overlapping derivation from base set $S = \bigcup_{n\geq 0} S_n$. Note, that if $\bigcup_{n=0}^{n=k} S_n = \bigcup_{n=0}^{n=k+1} S_n$ then $S = \bigcup_{n=0}^{n=k} S_n$. We give a polynomial upper bound on S.

We call a word crucial, if it can be obtained from any word by applying two overlapping productions. We show that there can be only polynomially many crucial words and hence S is polynomial, because all its words are crucial. There are two cases — the last production was prefix or suffix one. W.l.o.g. we assume, that it was suffix one. Then if we undo the last production we receive a word beginning with β_p for some prefix production $\alpha_p \to \beta_p$ and ending with some α_s for some suffix production $\alpha_s \to \beta_s$ and those two words have nontrivial overlap. Notice, that there are $O(|G|^3)$ such pairs, because such pairs we have to choose s, choose p, and decide, how much α_s and β_p overlap. Now we apply production $\alpha_s \to \beta_s$ and receive a crucial word, hence there are only polynomially many crucial words and so S has polynomially many elements at the most. So the process of calculating S is in fact polynomial — if no element was added in step n, then we can quit, and no more then polynomially many elements can be added. □

To make computations more effective we can create a graph with vertices labeled by triples (α_n, β_m, u) with α_n and β_m come from different kind of productions and u is a crucial word in S (obtained from α_n, β_m in a proper way). Also a special vertex for w is added. Directed edge from x to y represents the fact that y can be derived from x with just one overlap (usual assumption, that overlapping productions are two last ones applies). Computing, whether edge exists or not is polynomial. So searching for reachable vertices is polynomial.

Also it should not surprise the reader, that if we allow a base set for prefix-suffix grammar to be a regular set, then also $L_G(M)$ is a regular set as well and equivalent $\mathcal{NFA}$ can be constructed in polynomial time. The proof combines ideas from the previous proofs, so we give only a sketch of it.

Theorem 6. *The language $L_G(M)$ for any finite (nondeterministic) automata M and prefix-suffix grammar is regular. Also it is possible to transform it to equivalent $\mathcal{NFA}$ in polynomial time.*

Proof. We divide this language into two parts: L_1 — those words that have at least one non-overlapping derivation and L_2 — those words that have at least one overlapping derivation. Those languages may have some words in common. We deal with those cases separately, for each of them we prove that it is regular and construct an equivalent $\mathcal{NFA}$.

We look at the words of L_2 — in the same way as in previous theorem, those words form a regular language $\bigcup_{x \in I} L_G^{\text{no overlaps}}(x)$ for some finite set I of crucial words. Also for every such x it is possible to construct $\mathcal{NFA}$ recognizing $L_G^{\text{no overlaps}}(x)$ in polynomial time. There are only polynomially many possible x, so we only have to decide, which one to take and combine their $\mathcal{NFA}$'s. We take exactly those words that can be derived from $L(M)$ by G. But this is equivalent to ask, for which crucial u it is possible to transform it by G^R to some word recognized by M (being more strict — we should undo the last production and then apply any productions from G^R). Since we already know, how to transform $G^R(u)$ to equivalent $\mathcal{NFA}$, this question can be answered in polynomial time. Note, that automata constructed in this way recognizes all words from L_2 and perhaps some words from L_1 as well, but this does us no harm.

If u belongs to L_1 and hence has non overlapping derivation, then the word w belonging to $L(M)$ that it is derived from, can be partitioned to three words — $w = w_1 w_2 w_3$ satisfying that w_1 is the longest prefix modified by the prefix part of the grammar, w_3 is the longest suffix modified by the suffix productions and w_2 is never modified during the derivation. We can associate x with w_1 and y with w_3 as in theorem 1 and theorem 2 — they are prefix and suffix entirely modified (in one step each) during the derivation. Also p is a state, that is acquired after reading w_1 and q after reading $w_1 w_2$ by M (we fix some accepting computation for w). So we have a tuple (x, y, p, q) and have to sum up languages $L_{G_{\text{prefix}}}(x) L(M_p^q) L_{G_{\text{suffix}}}(y)$ for (x, y, p, q) in some finite (in fact - polynomial) set. In the same way as in theorem 2 for (x, y, p, q) we can construct $\mathcal{NFA}$ equivalent to $L(G_{\text{prefix}}(x)) L(M_p^q) L(G_{\text{suffix}}(y))$ in polynomial time and decide which tuples to take. This claims, that we can construct $\mathcal{NFA}$ for L_2 in polynomial time. So the same applies for $L_G(M)$. And so we are done. □

REFERENCES

[FPJ94] M. Frazier and C.D. Page Jr. Prefix grammars: An alternative characterization of the regular languages. *Inf. Proc. Let.*, 51(4):67–71, 1994.

[RQ02] B. Ravikumar and L. Quan. Efficient Algorithms for Prefix Grammars. 2002.