A New Constructive Method for the One-Letter Context-Free Grammars

Ştefan ANDREI and Wei-Ngan CHIN
Singapore-MIT Alliance E4-04-10, 4 Engineering Drive 3, Singapore 117576

***Abstract*— Constructive methods for obtaining the regular grammar counterparts for some sub-classes of the context free grammars (`cfg`) have been investigated by many researchers. An important class of grammars for which this is always possible is the one-letter `cfg`. We show in this paper a new constructive method for transforming arbitrary one-letter `cfg` to an equivalent regular expression of star-height 0 or 1. Our new result is considerably simpler than a previous construction by Leiss, and we also propose a new normal form for a regular expression with single-star occurrence. Through an alphabet factorization theorem, we show how to go beyond the one-letter `cfg` in a straight-forward way.**

***Index Terms*— reduction of a context-free grammar, one-letter context-free language, regular expression**

I. Introduction

The subclass of one-letter alphabet languages has been studied for many years. The result "*Each context-free one-letter language is regular*" was first proven in [13] and re-published in [14] using Parikh mappings. A second method based on the "pumping" lemma for the context-free languages (`cfl`'s) was presented in [10]. Salomaa ([15]) used the *systems of equations* (based on $\cup$, $\cdot$ and $*$ operators) to prove that the star-height of every one-letter alphabet language is equal to 0 or 1. Later, Leiss ([12]) gave the first constructive method by developing a theory of language equations over a one-letter alphabet. Several key theorems were proven and tied together to provide an algorithm which solves any equation of that type. In this paper, we shall present a new simpler method using only a single result, called the *Regularization Theorem*, with the help of a new normal form for one-letter equations.

Like [1], [7], [10], we will use systems of equations to denote `cfg`'s. It is known that for an arbitrary `cfg`, it is undecidable whether its least fixed point can be expressed as a regular expression, in general ([5]). We define a new normal form for the one-letter equations and a new theorem for solving them. Algorithm **A** (Section III) will use this normal form to determine precisely the least fixed point, as an equivalent regular expression. By considering the classes of *one-letter/one-variable factorizable*, we enlarge slightly the class of `cfg`'s for which the construction of a regular expression remains decidable.

Ştefan ANDREI is with Singapore-MIT Alliance, National University of Singapore, CS Programme, Singapore, 117543; e-mail: andrei@comp.nus.edu.sg

Wei-Ngan CHIN is with National University of Singapore, School of Computing, Department of Computer Science, Singapore, 117543; e-mail: chinwn@comp.nus.edu.sg.

II. Preliminaries

We suppose the reader is familiar with the basic notions of the formal language theory, but some important notions are briefly covered here.

A **context-free grammar** is denoted as $G = (V_N, V_T, S, P)$, where V_N/V_T are the alphabets of variables/terminals, ($V = V_N \cup V_T$ is the alphabet of all symbols of G), S is the start symbol and $P \subseteq V_N \times V^*$ is the set of productions. The productions $X \to \alpha_1$, $X \to \alpha_2$, ..., $X \to \alpha_k$ will be denoted by $X \to \alpha_1 \,|\, \alpha_2 \,|\, ... \,|\, \alpha_k$ and the right-hand side of X is denoted by $\mathtt{rhs}(X)$, that is $\{\alpha_1, \alpha_2, ..., \alpha_k\}$. A variable X is a **self-embedded variable** in G if there exists a derivation $X \underset{G}{\overset{*}{\Longrightarrow}} \alpha X \beta$, where α, $\beta \in V^+$ ([6]). G is a **self-embedded grammar** if there exists a self-embedded variable. G is a **reduced grammar** if $\forall\ X \in V$, $S \underset{G}{\overset{*}{\Longrightarrow}} \alpha\, X\, \beta$ and $\forall\ X \in V_N$, $X \underset{G}{\overset{*}{\Longrightarrow}} u$, with $u \in V_T^*$. The empty word is denoted by ϵ. A `cfg` is **proper** if it has no ϵ-productions (i.e. $X \to \epsilon$, $X \in V_N$) and no chain-productions (i.e. $X \to Y$, $X, Y \in V_N$). It is known that for every `cfg` (which doesn't generates ϵ) there exists an equivalent proper `cfg`.

The set of the terminal words attached to the variable X of the grammar G is $L_G(X) = \{w \in V_T^* \mid \exists\ X \underset{G}{\overset{+}{\Longrightarrow}} w\}$ ($\underset{G}{\overset{m}{\Longrightarrow}}$ ($\underset{G}{\overset{+}{\Longrightarrow}}$) means that m (at least one) productions have been applied in G). The set of **sentential forms** of X in G is $S_G(X) = \{\alpha \in V^* \mid \exists\ X \underset{G}{\overset{*}{\Longrightarrow}} \alpha\}$, the set of **sentential forms** of G is $S(G) = S_G(S)$. The **language** of G is $L(G) = S(G) \cap V_T^* = L_G(S)$. All the above sets can be easily extended to words, i.e. $L_G(\alpha) = \{\alpha \in V_T^* \mid \exists\ \alpha \underset{G}{\overset{+}{\Longrightarrow}} w\}$, a.s.o.

A **permutation** with n elements is an one-to-one correspondence from $\{1, ..., n\}$ to $\{1, ..., n\}$, the set of all permutations with n elements is denoted by Π_n. $\mathbf{N}$ denotes the set of natural numbers; $\overline{1,n}$ denotes the set $\{1, ..., n\}$, $i,\ j \in \overline{1,n}$ denotes $i \in \overline{1,n}$, $j \in \overline{1,n}$.

We continue by providing some results related to the *system of equations* ([1]). The systems of equations are extremely concise for modeling `cfl`'s ([7], [10]). The notions of substitution, solution, and equivalence can be found in [1], [11].

Definition 2.1: Let $G = (\{X_1, ..., X_n\}, V_T, X_1, P)$ be a `cfg`. A **system of (X_i-)equations** over G is a vector $\mathcal{P} = (\mathcal{P}_1, ..., \mathcal{P}_n)$ of subsets of V^*, usually written as: $X_i = \mathcal{P}_i,\ \forall\ i \in \overline{1,n}$, with $\mathcal{P}_i = \{\alpha \in V^* \mid X_i \to \alpha \in P\}$.

The next classical result gives one method for computing the minimal solution of a system of equations by derivations ([1]).

Theorem 2.1: Let $G = (\{X_1, ..., X_n\}, V_T, X_1, P)$ be a `cfg`. Then the vector $L_G = (L_G(X_1), ..., L_G(X_n))$ is the least solution of the associated system.

The next theorem refers to a well known transformation which "eliminates" X from a linear $X-$equation ([3], [15], [11]). From now on, unless specified otherwise, we will use notations $\alpha = \alpha_1 + ... + \alpha_m$, $\beta = \beta_1 + ... + \beta_n$, where m, and $n \in \mathbf{N}$. We shall use $X \notin \beta$ to mean $X \notin \beta_j$, $\forall\ j \in \overline{1, n}$.

Theorem 2.2: Let $X = \alpha\, X + \beta$ be an $X-$equation, where $X \notin \alpha$, and $X \notin \beta$. The least solution is $X = \alpha^*\beta$, and if $\epsilon \notin \alpha$, then this is unique.

III. One-letter CFG and its Regular Construction

In this section, we shall give a new constructive method for regularizing one-letter `cfg`'s that is more concise and general than the method proposed by Leiss ([12]). The commutativity plays an important role for transforming the one-letter `cfg`'s and this is covered in the following lemma.

Lemma 3.1: Let $G = (V_N, \{a\}, S, P)$ be a one-letter `cfg`. The set of all **commutative grammars** of G is $\mathcal{G}_{com}(G) = \{(V_N, \{a\}, S, P_{com}),$ where $P_{com} = \{X \to \alpha_{\pi(1)} ... \alpha_{\pi(k)} \mid X \to \alpha_1 ... \alpha_k \in P,\ \pi \in \Pi_k\}$. Then for every $G_{com} \in \mathcal{G}_{com}(G)$, it follows $L(G) = L(G_{com})$.

Proof It can be easily proved by induction on l, $l \geq 1$, that for any $X \in V_N$, we have: (1) $X \overset{l}{\underset{G}{\Longrightarrow}} a^n$ iff $X \overset{l}{\underset{G_{com}}{\Longrightarrow}} a^n$. Complete proof can be found in [2]. ∎

Lemma 3.2 allows the symbols of any sentential form to be re-ordered in an one-letter `cfg`. Its proof is similar to Lemma 3.1.

Lemma 3.2: Let $G = (V_N, \{a\}, S, P)$ be an one-letter `cfg` and let us consider the derivation $\alpha_1 ... \alpha_k \overset{*}{\underset{G}{\Longrightarrow}} a^n$. For any $\pi \in \Pi_k$, we have $\alpha_{\pi(1)} ... \alpha_{\pi(k)} \overset{*}{\underset{G}{\Longrightarrow}} a^n$.

The next lemma shows how star-operations are flattened in the one-letter `cfg`'s.

Lemma 3.3: Let $G = (V_N, \{a\}, S, P)$ be an one-letter `cfg` and α_1, ..., α_n some words over $V_N \cup \{a\}$. Then the following properties hold: $L_G((\alpha_1 + ... + \alpha_n)^*) = L_G(\alpha_1^* ... \alpha_n^*) = L_G((\alpha_1^* ... \alpha_n^*)^*)$, $L_G((\alpha_1 \alpha_2^* ... \alpha_n^*)^*) = \epsilon + L_G(\alpha_1 \alpha_1^* \alpha_2^* ... \alpha_n^*)$.

Proof Focusing to the first equality, we have to prove that: $(\alpha_1 + ... + \alpha_n)^* \overset{*}{\underset{G}{\Longrightarrow}} a^m$ iff $\alpha_1^* ... \alpha_n^* \overset{*}{\underset{G}{\Longrightarrow}} a^m$. Based on Lemma 3.2, the words α_1, ..., α_n can be commuted in any order. We proceed by induction on n. First, let us suppose that $n = 2$. The inclusion $L_G((\alpha_1 + \alpha_2)^*) \supseteq L_G(\alpha_1^* \alpha_2^*)$ is obvious. For the other inclusion, let us take $\beta = (\alpha_1 + \alpha_2)^n$, $n \geq 0$. It can be rewritten $\beta = \alpha_1^{n_1}\, \alpha_2^{n_2} ... \alpha_1^{n_k - 1}\, \alpha_2^{n_k}$, where $n_i \in \overline{0, n}$, $\forall\ i \in \overline{1, k}$, and $\sum_{i=1}^{k} n_i = n$. Using $\alpha_1\alpha_2 = \alpha_2\alpha_1$ applied several times, we get $\beta = \alpha_1^{n_1 + ... + n_{k-1}}\, \alpha_2^{n_2 + ... + n_k}$. So $L(\beta) \subseteq L_G(\alpha_1^*\, \alpha_2^*)$, therefore $L_G((\alpha_1 + \alpha_2)^*) = L_G(\alpha_1^* \alpha_2^*)$.

Now, we suppose true for $n = m > 2$ and prove it for $n = m + 1$. We have $L_G((\alpha_1 + ... + \alpha_m + \alpha_{m+1})^*) = L_G(((\alpha_1 + ... + \alpha_m) + \alpha_{m+1})^*) = L_G((\alpha_1 + ... + \alpha_m)^* \alpha_{m+1}^*) = L_G((\alpha_1 + ... + \alpha_m)^*) \cdot L_G(\alpha_{m+1}^*) = L_G(\alpha_1^* ... \alpha_m^*) \cdot L_G(\alpha_{m+1}^*) = L_G(\alpha_1^* ... \alpha_m^* \alpha_{m+1}^*)$.

For the other identities, we shall use some equations for regular expressions from [15]: $(\alpha^*)^* = \alpha^*$ and $(\alpha\,\beta^*)^* = \epsilon + \alpha(\alpha + \beta)^*$. Therefore $L_G((\alpha_1^* ... \alpha_n^*)^*) = L_G(((\alpha_1 + ... + \alpha_n)^*)^*) = L_G((\alpha_1 + ... + \alpha_n)^*) = L_G(\alpha_1^* ... \alpha_n^*)$ and $L_G((\alpha_1 \alpha_2^* ... \alpha_n^*)^*) = L_G(((\alpha_1(\alpha_2 + ... + \alpha_n)^*)^*) = L_G(\epsilon + \alpha_1(\alpha_1 + ... + \alpha_n)^*) = \epsilon + L_G(\alpha_1 \alpha_1^* \alpha_2^* ... \alpha_n^*)$. ∎

Definition 3.1: We say that the equation $X = \mathcal{P}$ is in the **one-letter normal form** (abbreviated by **OLNF**) if $\mathcal{P} = \alpha\, X + \beta$, where $X \notin \beta$.

Theorem 3.1: Let $G = (\{X_1, ..., X_n\}, \{a\}, X_1, P)$ be an one-letter reduced `cfg`. Then every attached X_i-equation can be transformed into OLNF.

Proof Let $X_i = \alpha\, X_i + \beta$ be an arbitrary X_i-equation. Because G is reduced, it follows that $\beta \neq \emptyset$, otherwise there will be no terminal word in $L_G(X_i)$. Based on Lemma 3.2, it follows that the symbols of α can be commuted in $\mathcal{P}_i$ in such a way that X_i will be on the last position. Next, by distributivity $(\gamma_1 \cdot X_i + \gamma_2 \cdot X_i = (\gamma_1 + \gamma_2) \cdot X_i)$, it is obvious that every X_i-equation can be transformed in this form. The only possible term of $\mathcal{P}_i$ for which X_i cannot be commuted until the last position is $\alpha'(\beta'\, X_i)^*$. In that case, $\alpha'(\beta'\, X_i)^*$ will be rewritten into $\alpha'(\epsilon + (\beta'\, X_i)^*(\beta'\, X_i)) = \alpha' + \alpha'\, \beta'(\beta'\, X_i)^* X_i$. Now, if $X_i \notin \alpha'$ then the X_i-equation is in OLNF, otherwise the transformation will continue and stop after a finite number of steps. ∎

By doing this transformation together with the (flattening) Lemma 3.3, Theorem 3.2 can be viewed as a generalization of Leiss's results (consisting of Theorems 3.1, 4.1, and 4.2 from [12]).

The next theorem is a tool for eliminating the occurrences of the variable X in a rhs of an $X-$equation. This is a generalization of Theorem 2.2, and a key ingredient of Algorithm **A**. Let us denote by $\alpha[\beta/X]$ the word obtained by replacing every $X-$occurrence in α with β. Of course, the substitution is valid iff X does not occur in β.

Theorem 3.2: (Regularization) Let $G = (V_N, \{a\}, S, P)$ be an one-letter reduced `cfg`, $X \in V_N$ and $X = \alpha\, X + \beta$ be an OLNF $X-$equation. Then, the least solution of the $X-$equation is $X = (\alpha[\beta/X])^*\beta$, and if G is proper, then this solution is unique.

Proof Before starting the proof, let us refer to the uniqueness of the solution. Because G is proper, it follows that G has no ϵ-productions and chain-productions, so $\epsilon \notin \alpha$, and $\epsilon \notin \beta$. Similarly to Theorem 2.2, it easily follows that the solution of the $X-$equation is unique. Without loss of generality, by applying finitely many times Lemmas 3.2 and 3.3, we suppose that α can be viewed as a regular expression over $V_N \cup \{a\}$ of star-height 0 or 1. So, its general form is $\alpha = \sum_{i=1}^{t} \alpha_{0,i}(\alpha_{1,i} X^{k_{1,i}})^* ... (\alpha_{m,i} X^{k_{m,i}})^*$. For simplicity, let us focus on $(\alpha_{1,i} X^{k_{1,i}})^*$. Based on commutativity, $(\alpha_{1,i} X^{k_{1,i}})^* = \{(\alpha_{1,i} X^{k_{1,i}})^{n_{1,i}} \mid n_{1,i} \geq 0\} = \{\alpha_{1,i}^{n_{1,i}} X^{k_{1,i} \cdot n_{1,i}} \mid n_{1,i} \geq 0\}$. Hence, $\alpha = \sum_{i=1}^{t} \alpha_{0,i}(\alpha_{1,i}^{n_{1,i}} X^{k_{1,i} \cdot n_{1,i}}) ... (\alpha_{m,i}^{n_{m,i}} X^{k_{m,i} \cdot n_{m,i}})$ $= \sum_{i=1}^{t} \alpha_{0,i} \alpha_{1,i}^{n_{1,i}} ... \alpha_{m,i}^{n_{m,i}} X^{k_{1,i} \cdot n_{1,i} + ... + k_{m,i} \cdot n_{m,i}}$. This will be denoted by $\alpha = \sum_{i=1}^{t} \alpha'_i X^{Q_i}$, where α'_i are words over $(V_N - \{X\}) \cup \{a\}$ and Q_i are (linear) polynomials in variables $n_{j,i} \in \mathbf{N}$, ($k_{j,i} \in \mathbf{N}$ are constants).

Therefore, the initial $X-$equation becomes $X = (\sum_{i=1}^{t} \alpha'_i X^{Q_i})X + \beta$, which corresponds to the following $X-$productions in G: $X \to \alpha'_1 X^{Q_1} X \mid ... \mid \alpha'_t X^{Q_t} X \mid \beta_1 \mid ... \mid \beta_n$. Because $X \notin \alpha'_i$, $\forall\ i \in \overline{1,t}$, and $X \notin \beta_j$, $\forall\ j \in \overline{1,n}$, it follows that $S_G(X)$ can be generated by applying several times (e.g. $s-$times) productions of the form $X \to \alpha'_i X^{Q_i} X$, $i \in \overline{1,t}$, followed by productions of the form $X \to \beta_j$, $j \in \overline{1,n}$ in order to remove all the occurrences of X. According to Lemma 3.2, we can re-order the symbols in any sentential form, and thus apply the current X-production to the last occurrence of the variable X, so we get the general $X-$derivations: $X \overset{s}{\underset{G}{\Longrightarrow}} \alpha'_{i_1}...\alpha'_{i_s} X^{Q_{i_1}}...X^{Q_{i_s}} X$, where i_1, ..., $i_s \in \overline{1,t}$. After applying $Q_{i_1} + ... + Q_{i_s} + 1$ productions of type $X \to \beta_j$, $j \in \overline{1,n}$, we obtain the words $\alpha'_{i_1}...\alpha'_{i_s}\beta_{j_{1,1}}...\beta_{j_{1,Q_{i,1}}}...\beta_{j_{s,1}}...\beta_{j_{s,Q_{i,s}}}\beta_j$. According to Lemma 3.2, $L_G(\alpha'_{i_1}..\alpha'_{i_s}\beta_{j_{1,1}}..\beta_{j_{1,Q_{i,1}}}..\beta_{j_{s,1}}..\beta_{j_{s,Q_{i,s}}}\beta_j) = L_G(\alpha_{i_1}\beta_{j_{1,1}}..\beta_{j_{1,Q_{i,1}}}..\alpha_{i_s}\beta_{j_{s,1}}..\beta_{j_{s,Q_{i,s}}}\beta_j)$. Because the words $\alpha_{i_1}\beta_{j_{1,1}}...\beta_{j_{1,Q_{i,1}}}...\alpha_{i_s}\beta_{j_{s,1}}...\beta_{j_{s,Q_{i,s}}}\beta_j$ correspond to $(\alpha[\beta/X])^*\beta$, then it follows that the solution of the $X-$equation is $X = (\alpha[\beta/X])^*\beta$. ∎

Algorithm **A** is based on the representation of the one-letter `cfg` as a system of equations. Then this system of equations is solved in order to obtain an equivalent regular expression. As we assume reduced `cfg`, each recursive $X-$equation must have at least one term without any occurrence of X.

Algorithm A

Input: $G = (\{X_1, ..., X_n\}, \{a\}, X_1, P)$ a reduced and proper one-letter `cfg`
Output: $L_G = (L_G(X_1), ..., L_G(X_n))$, and $L_G(X_i)$ is regular, $\forall\ i \in \overline{1,n}$
Method:
1. Construct $X_i = \mathcal{P}_i$, $\forall\ i \in \overline{1,n}$ as in Definition 2.1;
2. **for** $i := 1$ **to** n **do begin**
3. Transform X_i-equation into OLNF
4. $\mathcal{P}_i = (\alpha[\beta/X_i])^*\beta$;
5. Apply Lemma 3.3 to obtain the star-height 0 or 1 for $\mathcal{P}_i$
6. **for** $j := i+1$ **to** n **do** $\mathcal{P}_j = \mathcal{P}_j[\mathcal{P}_i/X_i]$;
endfor
7. **for** $i := n-1$ **downto** 1 **do**
8. **for** $j := n$ **downto** $i+1$ **do begin**
9. $\mathcal{P}_i = \mathcal{P}_i[\mathcal{P}_j/X_j]$;
10. Apply Lemma 3.3 to obtain the star-height 0 or 1 for $\mathcal{P}_i$
endfor
11. $L_G = (X_1, ..., X_n)$

Theorem 3.3: Algorithm **A** is correct and performs a finite number of steps.

Proof The lines 1, 11 are due to Definition 2.1 and Theorem 2.1, respectively. The instructions between lines 3-5 are based on Theorem 3.2 and Lemma 3.3 and imply that $\forall\ i \in \overline{1,n}$, $\mathcal{P}_i$ doesn't contain X_i. Line 6 ensures that $\forall\ i \in \overline{1,n}$, $\mathcal{P}_i$ doesn't contain any X_j with $j < i$. The occurrences of X_j from $\mathcal{P}_i$, where $j > i$ are replaced with terminal words at the lines 7-10. After the execution of Algorithm **A**, $\mathcal{P}_i$ is a regular expression over $\{a\}$ of star-height 0 or 1, thus $L_G(X_i)$ is regular, $\forall\ i \in \overline{1,n}$. By induction on i, it can be easily proved that according to Lemma 3.3, $\mathcal{P}_i$ has the star-height 0 or 1. ∎

As a remark, due to the nested **for** instructions (2-6 and 7-10), if we suppose that the steps 3-6 and 9-10 require constant time in n, then the time-complexity of Algorithm **A** is $\mathcal{O}(n^2)$.

Example 3.1: Let us consider $G = (\{X_1, X_2\}, \{a\}, X_1, P)$ with P given by the following productions: $X_1 \to a\, X_1\, X_2 \mid a$, $X_2 \to X_1\, X_2 \mid a\, a$. Line 1 of Algorithm **A** will construct the system: $X_1 = a\, X_1\, X_2 + a$, $X_2 = X_1\, X_2 + a^2$. After executing line 4, we get $X_1 = (a\, X_2)^*\, a$, and after line 6, we obtain $X_2 = a\,(a\, X_2)^*\, X_2 + a^2$. At the next iteration, Algorithm **A** will provide $X_2 = (a\,(a^3)^*)^* a^2$, and after line 5, $X_2 = a^2 + a^3 \cdot a^*(a^3)^*$. At line 9, it follows $X_1 = a(a^3 + a^4 \cdot a^*(a^3)^*)^*$, and after line 10, $X_1 = (a^3)^* \cdot (a + a^5 \cdot a^* \cdot (a^3)^* \cdot (a^4)^*)$. ∎

As a remark, in Algorithm **A** the order of eliminating X_i can be arbitrary. For instance, by eliminating X_2, followed by X_1, we get the equivalent simpler expressions: $X_1 = a + a^4 \cdot a^*$ and $X_2 = a^2 \cdot a^*$. We shall next show that every factor of the one-letter regular expression can be reduced to only one occurrence of $*$.

Definition 3.2: We say that $e = e_1 + ... + e_n$ (where each e_i contains only $\cdot$ and $*$ operators) is in **single-star normal form** iff $\forall\ i \in \overline{1,n}$, e_i has at most one occurrence of $*$.

This normalization is captured in the following theorem.

Theorem 3.4: Every regular expression over an one-letter alphabet can be transformed into an equivalent single-star normal form.

Proof If e is a regular expression of the star-height 1 (the case 0 is trivial) then it can be written as $e = e_1 + ... + e_n$, where $\forall\ i \in \overline{1,n}$, $e_i = a^{m_{0,i}}(a^{m_{1,i}})^*...(a^{m_{k_i,i}})^*$, where $m_{1,i} < ... < m_{k_i,i}$. We suppose, without loss of generality, that the cases $m_{s,i} = m_{s+1,i}$ are excluded based on the property $\alpha^*\alpha^* = \alpha^*$. Let $G(a_1, ..., a_k)$ be the greatest number b such that the Diophantine equation $a_1\, x_1 + ... + a_k\, x_k = b$ has no solution in $\mathbf{N}$, where the greatest common divisor of $a_1, ..., a_k$ is 1 (notation $gcd(a_1, ..., a_k) = 1$). This means that for any $b > G(a_1, ..., a_k)$ the equation $a_1\, x_1 + ... + a_k\, x_k = b$ has always solution in $\mathbf{N}$. Let us denote by $F(a_1, ..., a_k)$ the set of all natural numbers less than $G(a_1, ..., a_k)$ such that the above equation has solution in $\mathbf{N}$. According to [8], if $a_1 < ... < a_k$ and $gcd(a_1, ..., a_k) = 1$, then $G(a_1, ..., a_k) \leq (a_k - 1)(a_1 - 1)$.

Denoting $d = gcd(m_{1,i}, ..., m_{k_i,i})$, due to the above Diophantine equation, it follows that e_i can be equivalently transformed into $a^{m_{0,i}} \cdot \left(\epsilon + a^{d \cdot n_1} + ... + a^{d \cdot n_s} + (a^d)^{(\frac{m_{k,i}}{d} - 1)(\frac{m_{1,i}}{d} - 1) + 1} \cdot (a^d)^*\right)$, where n_1, ..., $n_s \in F(\frac{m_{1,i}}{d}, ..., \frac{m_{k,i}}{d})$. In this way, each factor e_i of e has at most one star, so e is in single-star normal form. ∎

A particular case of the above theorem is to reduce the expression $(a^m)^* \cdot (a^n)^*$ for which $m \equiv 0 (mod\, n)$. So, $gcd(m, n) = m$, hence by Theorem 3.4, it follows that $(a^m)^* \cdot (a^n)^* = \epsilon + (a^m) \cdot (a^m)^* = (a^m)^*$. Considering the `cfg` from Example 3.1, we can reduce $X_1 = a \cdot (a^3)^* + a^5 \cdot a^*$ and $X_2 = a^2 + a^3 \cdot a^*$.

Example 3.2: For instance, the following regular expressions of star-height 1 are reduced to the single-star normal form: $(a^2)^*(a^3)^* = \epsilon + a^2a^*$, $(a^4)^*(a^6)^* = \epsilon + a^4(a^2)^*$ and $(a^4)^*(a^6)^*(a^9)^* = \epsilon + a^4 + a^6 + a^8 + a^9 + a^{10} + a^{12} \cdot a^*$. ∎

Our main result, based on Theorem 3.2, is considerably simpler and more general than the constructive method given by Leiss [12]. Firstly, we needed only a single (more general) theorem to facilitate the construction of an equivalent regular expression for an arbitrary one-letter `cfg`. Secondly, the substitution of all the $X-$occurrences by β is done in one step, as opposed to multiple steps used by Leiss's procedure. We now explore a straight-forward way to go beyond one-letter `cfg`'s through the use of alphabet factorisation.

IV. Beyond one-letter CFG's

As is well-known, the non self-embedded variables/`cfg`'s are easily converted to the regular sublanguages. Theorem 4.1 (proven in [2]) shows that any `cfg`, G, generates a regular language if all its self-embedded variables can be shown to generate regular languages.

Theorem 4.1: Let G be an arbitrary reduced and proper `cfg`. If for all self-embedded variables X the language $L_G(X)$ is regular, then $L(G)$ is regular.

In the following, we shall combine the property of an one-letter alphabet, together with self-embeddedness, in order to obtain a more powerful class of `cfg`'s which generates regular languages.

Definition 4.1: A `cfg` $G = (V_N, V_T, S, P)$ is called **one-letter factorizable** iff for every self-embedded variable X, $L_G(X) \subseteq \{a\}^*$, where $a \in V_T$.

In other words, if G is one-letter factorizable, then every self-embedded variable has the corresponding language defined over (only) one-letter alphabet.

Now, the notion of one-variable factorizable will be introduced. This notion is somehow *dual* to one-letter factorizable, by considering at most one occurrence of a variable A_i in $\texttt{rhs}(X_i)$.

Definition 4.2: We say that $G = (V_N^1 \cup V_N^2, V_T, X_1, P)$ where $V_N^1 = \{X_1, ..., X_n\}$ and $V_N^2 = \{A_1, ..., A_n\}$ ($V_N^1 \cap V_N^2 = \emptyset$) is **one-variable factorizable** iff for every self-embedded variable X_i the $\texttt{rhs}(X_i) \subseteq \{X_i, A_i\}^*$ and $\texttt{rhs}(A_i) \subseteq V_T^*$.

Theorem 4.2: (Factorization) The following facts hold:

(a) An one-letter factorizable `cfg` generates a regular language.

(b) An one-variable factorizable `cfg` generates a regular language.

Proof (a) Let $G = (V_N, V_T, S, P)$ be a one-letter factorizable `cfg`. For every self-embedded variable $X \in V_N$, we know that $L_G(X) \subseteq \{a\}^*$. So due to Theorem 3.3, it follows that $L_G(X)$ is regular. Applying Theorem 4.1, it follows that $L(G)$ is regular.

(b) Let $G = (V_N^1 \cup V_N^2, V_T, X_1, P)$ be a one-variable factorizable `cfg`, where $V_N^1 = \{X_1, ..., X_n\}$, $V_N^2 = \{A_1, ..., A_n\}$ ($V_N^1 \cap V_N^2 = \emptyset$) and for every self-embedded variable X_i the $\texttt{rhs}(X_i) \subseteq \{X_i, A_i\}^*$ and $\texttt{rhs}(A_i) \subseteq V_T^*$.

Let us construct the `cfg` $G' = (V_N^1, V_N^2 \cup V_T, X_1, P')$, where $P' = P - \{A_i \to w \mid A_i \in V_N^2\}$. Because for every self-embedded variable X_i the $\texttt{rhs}(X_i) \subseteq \{X_i, A_i\}^*$, it follows that $L_{G'}(X_i) \subseteq \{A_i\}^*$. Hence $L_{G'}(X_i)$ is an one-letter language, so based on Algorithm **A**, it results that $L_{G'}(X_i)$ is a regular language. By applying Theorem 4.1, it follows that $L(G')$ is regular.

Now, let us consider the substitution $\sigma : V_N^2 \cup V_T \to V_T^*$, such that $\sigma(A_i) = \{\texttt{rhs}(A_i)\}$, $\forall\ i \in \overline{1,n}$ and $\sigma(a) = a$, $\forall\ a \in V_T$. Because $\{\texttt{rhs}(A_i)\}$ is a finite set of words, it follows that σ is a regular substitution. Obviously, $L(G) = \sigma(L(G'))$ and according to closure of the regular languages under the regular substitutions, it results that $L(G)$ is regular. ∎

Example 4.1: Let $G = (\{S, A, B\}, \{a, b, c\}, S, P)$ be a `cfg` with the following set of productions P: $S \to A\,B\,S \mid c$, $A \to a\,A\,a\,a\,A\,a \mid a$, $B \to b\,B\,B \mid b\,b\,b$. The set of the self-embedded variables is $\{A, B\}$, and $L_G(A) \subseteq \{a\}^*$, $L_G(B) \subseteq \{b\}^*$, so G is one-letter factorizable. Based on Algorithm **A**, we get $L_G(A) = \{(a^5)^{n_1}\, a \mid n_1 \geq 0\}$ and $L_G(B) = \{(b^4)^{n_2}\, b^3 \mid n_2 \geq 0\}$. Now, $L_G(S) = (L_G(A) \cdot L_G(B))^* \cdot c = \{((a^5)^{n_1}\, a\, (b^4)^{n_2}\, b^3)^{n_3}\, c \mid n_1,\, n_2,\, n_3 \geq 0\}$, so $L(G) = L_G(S)$ is regular. ∎

Example 4.2: Let $G = (\{S, A\}, \{\texttt{(}, \texttt{)}\}, S, P)$ be a `cfg` with productions P given by $S \to S\,S \mid A\,S\,A \mid \epsilon$, and $A \to \texttt{(} \mid \texttt{)}$. Obviously, by Definition 4.2, G is one-variable factorizable. Similarly to the proof of Theorem 4.2, we get the equation $S = (S + A^2)S + \epsilon$. Now based on Algorithm **A**, it results that $S = (A^2)^*$, so according to the $A-$productions, we get the regular language $L(G) = \{\{\texttt{(}, \texttt{)}\}^2\}^*$. ∎

V. Concluding Remarks

We summarize and compare some previous work on one-letter alphabet language. The class of one-letter alphabet languages were used by considering pushdown automata, whose memory consists of one-letter language. Boasson ([4]) called this kind of pushdown automata *counters* and the accepted language *one-counter* language. He proved that the family of one-counter languages is a proper subfamily of `cfl`'s.

The class of one-letter alphabet languages can be handled by considering finite-state automata. In [8], the problem of converting the (one-way) nondeterministic and two-way deterministic finite-state automata is hard to simulate by (one-way) deterministic finite-state automata, even for only one-letter alphabet languages. He proved that $\mathcal{O}(e^{\sqrt{n \log n}})$ states are sufficient to simulate an $n-$state (one-way) nondeterministic finite automaton recognizing a one-letter language by a (one-way) deterministic finite automaton.

The class of one-letter alphabet languages was covered in [9], where an efficient conversion from a finite-state automaton over one-letter alphabet to a context-free grammar in Chomsky normal form was proposed. The authors of [9] showed that any $n-$states one-letter deterministic finite automata can be simulated by a Chomsky normal form grammar with $\mathcal{O}(n^{2/3})$ variables, respectively the non-deterministic automata requires $\mathcal{O}(n^{1/3})$ variables. In our paper, Algorithm **A** takes in its input an one-letter reduced and proper `cfg` and provide the equivalent regular expression in single-star normal form.

The one-letter languages have been used recently in [16] for the decomposition of finite languages.

Our work has advanced the frontier of research in one-letter `cfg`'s by providing a much simpler constructive method for transforming into regular expressions using one-letter normal form. We also introduced a factorization result that enabled us to go beyond one-letter languages in a straight-forward way. This helps to enlarge the class of `cfg`'s that could be regularized.

We thank to the unknown referees for their very useful remarks, suggestions and comments which improved the paper.

REFERENCES

[1] Auteberg, J., Berstel, J., Boasson, L.: Context-Free Languages and Pushdown Automata. *Handbook of Formal Languages. Word, Language, Grammar*. Vol. 1, Eds. G.Rozenberg, A. Salomaa. Springer Verlag, Berlin (1997) 111-174

[2] Andrei, Şt., Cavadini, S., Chin, W.-N.: Transforming self-embedded context-free grammars into regular expressions. *Faculty of Computer Science*. TR02-06, `http://www.infoiasi.ro/~tr/tr.pl.cgi`, Iaşi University, România (2002) 1-25

[3] Arden, D.N.: Delayed logic and finite state machines. *Theory of computing machine design*. Univ. of Michigan Press, Ann Arbor (1960) 1-35

[4] Boasson, L.: Two iteration theorems for some families of languages. *J. Comput. System Sci.* 7(6), (1973) 583-596

[5] Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. *Z. Phonetik. Sprachwiss. Kommunikationsforsch.* 14 (1961) 143-172

[6] Chomsky, N.: On certain formal properties of grammars. *Information and Control.* vol. 2 (1959) 137-167

[7] Chomsky, N., Schützenberger, M.P.: The algebraic theory of context-free languages. *Computer Programming and Formal Systems.* (P. Braffort and D. Hirschberg, eds.) Amsterdam, North-Holland (1963) 118-161

[8] Chrobak, M.: Finite automata and unary languages. *Theoretical Computer Science.* 47 (1986) 149-158

[9] Domaratzki, M., Pighizzini, G., Shallit, J.: Simulating finite automata with context-free grammars. *Information Processing Letters.* 84 (2002) 339-344

[10] Ginsburg, S., Rice, H. G.: Two families of languages related to ALGOL. *Journal of the Association for Computing Machinery*. vol. 9 (1962) 350-371

[11] Kuich, W., Urbanek, F.J.: Infinite linear systems and one counter languages. *Theoretical Computer Science.* 22 (1983) 95-126

[12] Leiss, E.L.: Language equations over a one-letter alphabet with union, concatenation and star: a complete solution. *Theoretical Computer Science.* 131 (1994) 311-330

[13] Parikh, R. J.: Language-generating devices. *Quaterly Progress Report*. No. 60, Research Laboratory of Electronics, M.I.T. (1961) 199-212

[14] Parikh, R. J.: On context-free languages. *Journal of the Association for Computing Machinery*. Vol. 13, (1966) 570-581

[15] Salomaa, A.: *Theory of Automata.* Pergamon Press. Oxford (1969)

[16] Salomaa, A., Yu, S.: On the Decomposition of Finite Languages. *Developments in Language Theory: Foundations, Applications, and Perspectives.* Eds. G. Rozenberg, W. Thomas. World Scientific. Scientific (2000)

Ştefan Andrei is a Research Fellow in the National University of Singapore under the Singapore-MIT Alliance (SMA). He has received his B.Sc. and M.Sc. in Computer Science, in 1994 and 1995, respectively, from Iaşi University, România and PhD in Computer Science in 2000 from the Hamburg University, Germany. Between 1997 and 2000, he got the following academic awards: DAAD scholarship, TEMPUS S_JEP 11168-96 scholarship and World Bank Joint Japan Graduate Scholarship at Fachbereich Informatik, Hamburg Universitaet, Germany. He is currently working on formal languages, compilers and real-time systems. More details about him can be found at `http://www.infoiasi.ro/~stefan`

Wei Ngan Chin is an Associate Professor at the Department of Computer Science, School of Computing, National University of Singapore, and a Fellow in the Computer Science Programme of the Singapore-MIT Alliance. He has received his B.Sc. and M.Sc. in Computer Science, in 1982 and 1983, respectively, from University of Manchester, United Kingdom, and PhD in Computing, in 1990 from the Imperial College of Science, Technology and Medicine, United Kingdom. His current research interests are functional programming, program transformation, parallel systems, software models and methods. More details about him can be found at `http://www.comp.nus.edu.sg/~chinwn/`