Context-Free Valence Grammars – Revisited*

H.J. Hoogeboom, Universiteit Leiden
http://www.liacs.nl/~hoogeboo/

Abstract

Context-free valence languages (over $\mathbb{Z}^k$) are shown to be codings of the intersection of a context-free language and a blind k-counter language. This AFL-style characterization allows one to infer some of the properties of the family of valence languages, in particular the λ-free normal form proved by Fernau and Stiebe.

1 Introduction

Valence grammars were introduced by Păun as a new way of regulated rewriting ([Pau80], or [DP89, p. 104]): associate with each production of a Chomsky grammar an integer value, the valence of the production, and consider only those derivations for which the valences of the productions used add to zero. The formalism is surprisingly simple, and adding valences to context-free grammars one obtains a family of languages strictly in between context-free and context-sensitive. More importantly, that family has convenient closure properties.

The formalism has been extended to valence monoids other than $\mathbb{Z}$, like positive rational numbers ($\mathbb{Q}^+$ under multiplication), or vectors of integers ($\mathbb{Z}^k$ under componentwise addition).

Fernau and Stiebe study the context-free valence grammars over arbitrary monoids in a uniform presentation, and give additional results for the context-free valence grammars over $\mathbb{Z}^k$ ([FS97], full paper [FS00]). One of their important contributions is that the Chomsky and Greibach normal forms hold in the valence case. This is somewhat unexpected, as chain productions and λ-productions do not contribute to the derived string, and yet their valences have to be taken into account – which makes their removal a nontrivial task.

This paper reconsiders those basic properties of context-free valence grammars over $\mathbb{Z}^k$. It has been observed in the cited papers that the valence mechanism is very similar to the storage type used by the blind

*LIACS Technical Report 01-06. Presentation at *Developments in Language Theory*, July 2001, Vienna, Austria.

counter automata of Greibach [Gre78], in fact, regular valence grammars are easily seen to be equivalent to blind counter automata. Here we make this connection explicit in another way. In the style of AFL theory [Gin75] we characterize context-free valence languages (over $\mathbb{Z}^k$) as homomorphic images of the intersection of a context-free language and a blind k-counter language, see Lemma 3. In this fashion we have explicitly separated the context-free derivation process from the phase where the valences are checked. As the context-free languages and the blind counter language share specific closure properties, generic results from AFL theory allow us to conclude those properties for context-free valence languages (Theorem 6).

Then the characterization is improved by replacing the (arbitrary) homomorphisms by codings (Theorem 5). Technically this is much more complicated, involving the prediction of the valences of derivations that yield the empty string (ending in λ-productions). Apart from its elegance, this characterization has some immediate use. A classic result of Latteux shows that the blind k-counter languages are accepted by *real-time* automata, i.e., automata reading a symbol in each step. The construction of a valence grammar starting with a context-free grammar and such a real-time blind counter automaton does not change the shape of the grammar. From this we immediately conclude the existence of Chomsky and Greibach normal forms for context-free valence grammars, see Theorems 8 and 9, i.e., directly from the existence of those normal forms for context-free grammars. This means that we have avoided the lengthy and involved technical analysis of the derivations in valence grammars [FS00, Sections 5.3–5.5], at the "cost" of applying an abstract result from AFL theory (which thus hides the actual complexity of the construction).

Our motivation to study valence language comes from the formal language theory of splicing. The valence families are powerful enough to model certain restricted types of splicing, while (unlike the larger family of context-sensitive languages) having suitable closure properties [HvV01].

2 Preliminaries

The reader is assumed to be familiar with standard formal language notions, see [Sal73] for a broad introduction. After some generalities, this section presents basic facts on language families, valence grammars, and blind counter automata.

We use λ to denote the empty string, and $||$ to denote the shuffle operation on languages.

For a vector $\vec{v} \in \mathbb{Z}^k$, $\|\vec{v}\|$ denotes the 1-norm $\sum_{i=1}^{k} |v_i|$, where $\vec{v} = (v_1, \ldots, v_k)$. Vectors in $\mathbb{Z}^k$ are added componentwise; the vector all of which elements are 0 is denoted by $\vec{0}$.

Language Families

We use REG and CF to denote the families of regular and of context-free languages. The family of homorphisms is denoted by HOM; the family of codings (letter-to-letter homomorphisms) by COD. Thus, e.g., $\mathsf{HOM}(\mathcal{F})$ denotes the family of homomorphic images of languages from $\mathcal{F}$.

For language families $\mathcal{F}$ and $\mathcal{G}$, $\mathcal{F} \wedge \mathcal{G}$ denotes $\{F \cap G \mid F \in \mathcal{F}, G \in \mathcal{G}\}$.

The family $\mathcal{F}$ is a *trio* (or *faithful cone*) if it is closed under λ-free homomorphisms, inverse homomorphisms, and intersection with regular languages. A *full trio* (or *cone*) is additionally closed under (arbitrary) homomorphisms. The smallest trio and full trio containing the language L are denoted by $\mathcal{M}(L)$, and $\widehat{\mathcal{M}}(L)$, respectively. It can be shown that

$$\widehat{\mathcal{M}}(L) = \{\ g(h^{-1}(L) \cap R) \mid h, g \in \mathsf{HOM}, R \in \mathsf{REG}\ \}.$$

For $\mathcal{M}(L)$ there is an analogous characterization where g is λ-free (Corollary 1 to Theorem 3.2.1 in [Gin75]). A (full) trio of this form is called (full) *principal*, i.e., it is generated by a single language L. It is automatically a (full) *semi*-AFL, i.e., additionally closed under union [Gin75, Proposition 5.1.1].

As explained in [Gin75, Example 5.1.1], it follows from the Chomsky-Schutzenberger Theorem that CF equals $\widehat{\mathcal{M}}(D_2)$; the full trio generated by D_2, the Dyck set on two symbols, a language generated by the context-free grammar with productions $S \to SS \mid a_1 S b_1 \mid a_2 S b_2 \mid \lambda$.

Valence Grammars

A *valence grammar* over $\mathbb{Z}^k$ is a context-free grammar in which every production has an associated value (the valence) from $\mathbb{Z}^k$. A string is in the language of the valence grammar if it can be derived in the usual way, under the additional constraint that the valences of the productions used add up to $\vec{0}$.

Formally the grammar is specified as four-tuple $G = (N, T, R, S)$, where N, T are alphabets of *nonterminals* and *terminals*, $S \in N$ is the *axiom*, and R is a finite subset of $N \times (N \cup T)^* \times \mathbb{Z}^k$, the *rules*. When $(A, w, \vec{r})$ is an element of R, then we write as usual $(A \to w, \vec{r})$, and $A \to w$ is the (underlying) *production* and $\vec{r}$ the *valence* of the rule.

Omitting the valences from the rules of G, we obtain the underlying context-free grammar of G. The derivation relation of G is an obvious extension of the one for context-free grammars; for $x, y \in (N \cup T)^*$ and $\vec{v} \in \mathbb{Z}^k$ we write $(x, \vec{v}\,) \Rightarrow (y, \vec{v} + \vec{r}\,)$ if there is a rule $(A \to w, \vec{r}\,)$, such that $x = x_1 A x_2$, and $y = x_1 w x_2$. Then, the *language* of G equals $L(G) = \{\, w \mid w \in T^*, (S, \vec{0}\,) \Rightarrow^* (w, \vec{0}\,) \,\}$. The family of all *valence languages* over $\mathbb{Z}^k$ is denoted here by $\mathsf{VAL}(\mathbb{Z}^k)$.

Blind Counter Automata

Let $k \geq 0$. A k *blind counter automaton*, or kBCA, is a finite state automaton with k integers $\vec{v} = (v_1, v_2, \ldots, v_k)$ as additional external storage. In each step the automaton may increment and decrement each of its counters. The storage is called *blind* as the automaton is not allowed to test the contents of the counters during the computation. The only, implicit, test is at the end of the computation, as we accept only computations from initial state to final state, that start and end with empty counters.

Formally, an instruction of a kBCA is of the form $(p, a, q, \vec{r}\,)$, where p, q are states, a is an input symbol or λ, and $\vec{r} \in \mathbb{Z}^k$. Executing this instruction, the automaton changes from state p into state q, reads a from its input tape, and changes the counter values from $\vec{v}$ into $\vec{v} + \vec{r}$.

The automaton is called *real-time* if none of the instructions read λ.

We use $k\mathsf{BC}$ to denote the family of languages accepted by kBCA.

According to standard AFA interpretation (cf. [Gin75, Lemma 5.2.3]) an automaton can be seen as a finite state transducer translating input symbols into sequences of storage instructions; the input string is accepted only if the resulting sequence of storage instructions is "legal", i.e., it can be executed by the storage, leading from initial to final storage. This implies that the languages accepted are precisely the inverses under finite state transductions of the language of legal storage instructions.

Let Σ_k be the alphabet $\{a_1, b_1, \ldots, a_k, b_k\}$. Define $B_k = \{x \in \Sigma_k^* \mid \#_{a_i}(x) = \#_{b_i}(x) \text{ for each } 1 \leq i \leq k\}$. Observe that B_k models the possible legal instruction sequences to the blind counter storage, interpreting a_i and b_i as increments and decrements of the i-th counter.

As every transducer (and its inverse) can be decomposed into homomorphisms and intersection with regular languages (known as Nivat's Theorem, cf. equation 2.4 in [Sal73, Chapter IV]), the language accepted by a kBCA can be written as $g(h^{-1}(B_k) \cap R))$, where h, g are homomorphisms, and R is a regular language.

This being the characterization of $\widehat{\mathcal{M}}(B_k)$ that was recalled on the

previous page, it then follows that $k\mathsf{BC}$ equals $\widehat{\mathcal{M}}(B_k)$; the full trio generated by B_k. For real-time kBCA h is a λ-free homomorphism, and we obtain along the same lines the trio $\mathcal{M}(B_k)$ generated by B_k.

By a result of Greibach, [Gre78, Theorem 2], blind counter automata are equivalent to their real-time restriction. Unfortunately, the proof of this result does not keep the number of counters constant, see the proof of Lemma 1 in [Gre78], which states "[..] this can be done [..] with delay 0, *by adding another counter*".

The equivalence of the real-time restriction for k blind counter automata, for every fixed k, is a special case of a general language theoretic result of Latteux [Lat79].

Proposition 1 *Real-time kBCA are as powerful as (unrestricted) kBCA.*

Proof. By [Lat79, Theorem II.11] $\widehat{\mathcal{M}}(L) = \mathcal{M}(L)$ for every permutation closure L of a regular language. Hence the proposition follows, as B_k is the permutation closure of the regular language $(a_1b_1)^* \cdots (a_kb_k)^*$. □

For kBCA we have a normal form in which the automaton never changes more than one of its counter values in each step.

Proposition 2 *For each (real-time) kBCA there exists an equivalent (real-time) kBCA such that $\|\vec{r}\| \leq 1$ for each instruction $(p, a, q, \vec{r})$.*

Proof. We use a variant of the normalisation procedure for counter automata, see [FMR68] – a rather flexible method, cf. [HvV00, Lemma 11].

Let m be the maximal amount by which any of the counters can change in one step; hence in k consecutive steps each of the counters is changed by a value at most $\pm km$. The normal form is obtained by accessing the counters in turn, such that in k consecutive steps each of the counters is changed (at most) once. We store the original counter value divided by $2km$ on the counter, while leaving the remainder (the original counter value modulo $2km$) in the finite state.

To be more precise, when a particular counter has just been accessed, the finite state keeps a remainder for that counter in the interval $[-km, \ldots, km-1]$. In the next $k-1$ steps we do not update that counter, but store the changes in the finite memory. Thus, the remainder grows to $v \in [-2km+m, \ldots, 2km-m-1]$. After the k steps we may again access that particular counter. If the original automaton at that moment wants to add $r \in [-m, \ldots, m]$ to the counter, the new automaton now computes $v' = v+r$ in its finite state. If $v' \geq km$ it increments the counter (by one)

and stores $v' - 2km$ in the state; if $v' < -km$ it decrements the counter and stores $v' + 2km$ in the state.

Acceptance as before, by final state and empty counter; a state is final if it is final in the original automaton and if it stores $\vec{0}$. □

3 Characterization of $\mathsf{VAL}(\mathbb{Z}^k)$

Let G be a valence grammar. We say that G is in *2-normal form* if G has only productions of the form $A \to BC$, $A \to B$, $A \to a$, and $A \to \lambda$, where $A, B, C \in N$, and $a \in T$. Indeed this is a normal form, and can be obtained easily similar to Chomsky normal form for context-free grammars, by splitting longer productions, and by introducing "shadow"-nonterminals that introduce terminals ($N_a \to a$) whenever necessary [FS00, Theorem 4.2].

We start by showing that valence languages are the homomorphic images of the intersection of a context-free language and a blind k-counter language. This reminds us of the characterization in the proof of Theorem 6 in [Pau80], in our notation $L(G) = h(L(G') \cap (B_1||T^*))$, where G is a *regular* valence grammar over $\mathbb{Z}^1$ and G' is a regular grammar constructed from G.

Lemma 3 $\mathsf{VAL}(\mathbb{Z}^k) = \mathsf{HOM}(\mathsf{CF} \wedge k\mathsf{BC})$.

Proof. Let G be a valence grammar over $\mathbb{Z}^k$ with terminal alphabet T. We obtain a context-free grammar G' from G by replacing each rule $(A \to w, \vec{r})$ by the production $A \to \vec{r}w$, and by adding the valence vectors that occur in the rules of G to the terminal alphabet T.

Let $\mathcal{A}$ be the blind counter automaton that operates on the terminal alphabet of G', and which interpretes the vectors $\vec{r}$ by adding their values to its counters. The automaton ignores the terminal symbols from T. It is clear that $L(G)$ is equal to $h(L(G') \cap L(\mathcal{A}))$, where h is the homomorphism that acts as identity on T, while it erases the valence vectors.

The converse implication can be obtained using a variant of the classical triple construction for the intersection of a context-free language with a regular language, as used in [FS00, Theorem 4.1] to show closure properties of $\mathsf{VAL}(\mathbb{Z}^k)$. Starting with a context-free grammar G in Chomsky normalform, a real-time blind counter automaton $\mathcal{A}$, and a homomorphism h, we introduce rules $([p, A, r] \to [p, B, q][q, C, r], \vec{0})$ for each production $A \to BC$ of G and states p, q, r of $\mathcal{A}$. Moreover we add terminal rules $([p, A, q] \to h(a), \vec{r})$ whenever there is an instruction $(p, a, q, \vec{r})$ of $\mathcal{A}$, and a production $A \to a$ of G.

The axiom S of the valence grammar gets rules $(S \to [p, S, q], \vec{0}\,)$ such that p is the initial state of $\mathcal{A}$, and q a final state of $\mathcal{A}$. If we want, we can get rid of those initial chain-rules by combining them with rules for the nonterminals $[p, S, q]$. □

As an immediate consequence of this basic characterization we can infer the main closure properties of $\mathsf{VAL}(\mathbb{Z}^k)$, see Theorem 6 below.

The main difficulty in improving the characterization to codings instead of arbitrary homomorphisms, is the removal of λ-rules from the grammar, while taking into account their valences. The main tool is the observation that the *Szilard language* of a context-free grammar, the production sequences used in derivations, has a semilinear Parikh image (see [DP89, Lemma 2.1.9], [FS00, Proposition 5.6]). This implies that we can count the corresponding valences using a sequential device, in our case a blind counter automaton.

For a sequence p of productions, we write $x \Rightarrow^p y$ if y is derived from x using the sequence p.

Lemma 4 *Let $G = (N, T, P, S)$ be a context-free grammar, and let A and B be nonterminals of G. The set $\{p \in P^* \mid A \Rightarrow^p B\}$ has a semilinear Parikh image.*

Proof. Replace each production $\pi : C \to \alpha$ by the production $C \to \pi\alpha$, where π is used as a label to represent the production. Consider the sentential forms of the new grammar, using A as axiom. This is a context-free language, and it remains context-free after intersection with P^*BP^* to select the derivations matching $A \Rightarrow^* B$. By Parikh's theorem, the resulting language has a semilinear image (from which we may drop the occurrence of B). □

We now show that it is possible to replace the homomorphisms in Lemma 3 by codings. For this we need the notion of the *binary structure* of a derivation tree of a valence grammar in 2-normal form (or of its underlying context-free grammar). It suffices to introduce that notion in an informal way. This binary structure is created from the derivation tree by deleting each node that has no terminal symbols among its descendants. The remaining tree can be decomposed into "linear" derivations $(A, \vec{0}\,) \Rightarrow_G^* (A', \vec{v}\,)$ leading into a rule which is either of the type $(A' \to BC, \vec{r}\,)$ or of the type $(A' \to a, \vec{r}\,)$. In fact, this decomposition by the binary structure induces a decomposition of the original derivation itself (but we need to find a way to include the valences of the rules that were cut from the tree).

Theorem 5 $\mathsf{VAL}(\mathbb{Z}^k) = \mathsf{COD}(\mathsf{CF} \wedge k\mathsf{BC})$.

Proof. By Lemma 3 it suffices to prove the inclusion from left to right. Let $G = (N, T, R, S)$ be a valence grammar over $\mathbb{Z}^k$. We may assume that G is in 2-normal form. In Lemma 3 the new grammar G' was constructed such that the (valence of) every production used in the derivation according to G was present in the language. As a coding cannot remove symbols, we cannot insert information in the string other than by associating it to terminals of $L(G)$.

Thus, we will code in our new grammar G' only the binary structure of the derivation tree. Actually, this structure is a presentation of the full derivation, up to derivations of the form $A \Rightarrow^* B$. These "linear" derivations may not only involve chain productions, but also large subtrees with empty yield. The blind counter language used in the intersection will not only take care of the computation of the valences, but it will also verify that those "linear" derivations exist in G.

The context-free grammar G' contains the productions

- $A \to B[A.BC]C$, for $A, B, C \in N$, where $[A.BC]$ is a new terminal symbol making the applied production visible in the language;
- $A \to [A.a]$, for $A \in N$ and $a \in T$, where again $[A.a]$ is a new terminal symbol;
- $S' \to [.S]S$, $S' \to \lambda$.

The terminal alphabet of G' consists the new symbols $[A.BC]$, $[A.a]$, and $[.S]$. The axiom of G' is the new nonterminal S'; the other nonterminals of G' are copied from G. Note G' does not depend on R.

Observe that $L(G')$ consists of strings of the form $\pi_1\rho_1\pi_2\rho_2 \ldots \pi_n\rho_n$ recording the binary structure of a possible derivation tree in the in-order of the nodes in the tree. Each π_i is of type $[A.BC]$, each ρ_i is of type $[A.a]$ (with the exception of $\pi_1 = [.S]$, which was introduced to obtain an even number of symbols). If $\rho_i = [A_i.a_i]$, then $a_1a_2 \ldots a_n$ is a string over the terminal alphabet of G, which *may* have a derivation with the given binary structure. The intersection with the blind counter language will get rid of unwanted binary structures.

We now consider the blind counter automaton $\mathcal{A}$ that should accept input of the form $\pi_1\rho_1\pi_2\rho_2 \ldots \pi_n\rho_n$ only if the productions coded in that string correspond to "linear" derivations in G, with valences adding to $\vec{0}$.

Reading a symbol π, with $\pi = [A.BC]$, $\mathcal{A}$ tries to accomplish the following. It guesses a rule $(A' \to BC, \vec{r}\,)$ of G, and a derivation $(A, \vec{0}\,) \Rightarrow^*_G$

$(A', \vec{v})$ to match a part of a derivation in G. The resulting vectors $\vec{r}$ and $\vec{v}$ are added using the counters of $\mathcal{A}$. Once the production $A' \to BC$ is guessed, the nondeterministic simulation of the "linear" derivation $A \Rightarrow_G^* A'$ can be done by $\mathcal{A}$ using Lemma 4. As the sequence of productions for such a derivation is an element of a semilinear set, $\mathcal{A}$ may simulate a finite state automaton that accepts a regular language with the same semilinear image.

For a symbol $\rho = [A.a]$, $\mathcal{A}$ guesses a rule $(A' \to a, \vec{r})$ and a matching derivation $(A, \vec{0}) \Rightarrow_G^* (A', \vec{v})$ following the same principle. The symbol $\pi_1 = [.S]$ can be ignored.

When $\mathcal{A}$ receives the empty string as input, it guesses a derivation $(S, \vec{0}) \Rightarrow_G^* (A, \vec{v})$ and a rule $(A \to \lambda, \vec{r})$.

It is now straightforward to verify that the intersection $L(G') \cap L(\mathcal{A})$ represents $L(G)$ in the above manner, and that $L(G)$ can be obtained from this intersection in $\mathsf{CF} \wedge k\mathsf{BC}$ by projecting the $[A.a]$ symbols to their a-components.

Of course, we cannot hide the π_i using a coding. The languages we need are obtained from $L(G')$ and $L(\mathcal{A})$ by an inverse homomorphism g^{-1} which combines $\pi_i \rho_i$ into a single symbol $[\pi_i \rho_i]$. Recall that both CF and $k\mathsf{BC}$ are closed under inverse homomorphisms.

Again, $g^{-1}(L(G')) \cap g^{-1}(L(\mathcal{A})) = g^{-1}(L(G') \cap L(\mathcal{A}))$ represents $L(G)$, which can be obtained from this new language using a coding that maps $[\pi_i \rho_i]$ into a_i. □

4 Implications

In this section we apply our basic characterizations of $\mathsf{VAL}(\mathbb{Z}^k)$ to obtain some of the (known) properties of this family.

Theorem 6 $\mathsf{VAL}(\mathbb{Z}^k)$ *is a (full) principal semi-*AFL.

Proof. According to Theorem 5.5.1(e) in [Gin75], if $\mathcal{F}$ and $\mathcal{G}$ are full principal semi-AFLs, then so is $\mathsf{HOM}(\mathcal{F} \wedge \mathcal{G})$. Both CF and $k\mathsf{BC}$ are full principal semi-AFLs, as discussed, hence so is $\mathsf{VAL}(\mathbb{Z}^k) = \mathsf{HOM}(\mathsf{CF} \wedge k\mathsf{BC})$. Similarly we find that $\mathsf{VAL}(\mathbb{Z}^k)$ is a principal semi-AFL, i.e., it is of the form $\mathcal{M}(L)$ rather than $\widehat{\mathcal{M}}(L)$, by [Gin75, Theorem 5.5.1(d)] and the characterization with codings, Theorem 5. Here we additionally use the fact that both CF and $k\mathsf{BC}$ are principal semi-AFLs, by the real-time results (of Greibach and Latteux). □

In fact, as is shown in [Gin75], the generator of the full principal semi-AFL $\mathsf{VAL}(\mathbb{Z}^k)$ is the shuffle $D_2||B_k$ of the respective generators for CF and $k\mathsf{BC}$, after renaming one of the alphabets to make them disjoint.

This also suggests a machine interpretation as (obviously) $D_2||B_k$ describes the "legal" instruction sequences of the storage type push-down combined with k blind counters cf. [FS00, Remark. 5.3]. The connection between AFL and automata discussed before Proposition 1 for kBCA now yields the following characterization (cf. [Gin75, Section 4.6]).

Theorem 7 $\mathsf{VAL}(\mathbb{Z}^k)$ *is the family of languages accepted by (real-time) automata equipped with a push-down and k blind counters.*

As a corollary to our characterizations we now obtain the "Chomsky II" normal form, Proposition 5.16 from [FS00].

Theorem 8 *For each valence grammar over $\mathbb{Z}^k$ there is an equivalent valence grammar $G = (N, T, R, S)$ over $\mathbb{Z}^k$, such that each rule is either of the form $(A \to BC, \vec{0})$, or of the form $(A \to a, \vec{r})$, with $A, B, C \in N$, $a \in T$, and $\|\vec{r}\| \leq 1$.*

Proof. Let G be an arbitrary valence grammar over $\mathbb{Z}^k$. By Theorem 5, we may write $L(G) = h(L(G') \cap L(\mathcal{A}))$, where h is a coding, G' a context-free grammar, and $\mathcal{A}$ is a kBCA. By classic formal language theory we may assume that G' is in Chomsky normal form, and according to Latteux we may assume that $\mathcal{A}$ is real-time (Proposition 1). Additionally we may assume that $\|\vec{r}\| \leq 1$ for each of the counter instructions of $\mathcal{A}$ (Proposition 2).

Now reassemble G' and $\mathcal{A}$ into a valence grammar, using the triple construction, cf. the proof of Lemma 3. We obtain a valence grammar in Chomsky II normal form. □

Of course, an important consequence of this normal form is the equivalence of context-free valence grammars with their λ-free restriction, one of the important contributions of [FS00]; in their notation $\mathcal{L}(\mathrm{Val}, \mathrm{CF}, \mathbb{Z}^k) = \mathcal{L}(\mathrm{Val}, \mathrm{CF}\text{-}\lambda, \mathbb{Z}^k)$.

In the same way as Chomsky II above, we obtain Greibach normal form, [FS00, Proposition 5.17]. Note that we do not have to extend the classical construction (to obtain Greibach starting from Chomsky) from CF to $\mathsf{VAL}(\mathbb{Z}^k)$ as we directly use the classical result for the context-free languages.

Theorem 9 *For each valence grammar over $\mathbb{Z}^k$ there is an equivalent valence grammar $G = (N, T, R, S)$ over $\mathbb{Z}^k$, such that each rule is of the form $(A \to a\alpha, \vec{r})$, with $A \in N$, $\alpha \in N^*$, $|\alpha| \leq 2$, $a \in T$, and $\|\vec{r}\| \leq 1$.*

Proof. Similar to the previous proof. Reassemble a context-free grammar G' in Greibach normal form with a real-time kBCA with the $\|\vec{r}\| \leq 1$ restriction: productions of the form $A \to a$, $A \to aB_1$, and $A \to aB_1B_2$ can be combined with the instruction $(p, a, q, \vec{r})$ to form the rules $([p, A, q] \to h(a), \vec{r})$, $([p, A, s] \to h(a)[q, B_1, s], \vec{r})$, and $([p, A, s] \to h(a)[q, B_1, q_1][q_1, B_2, s], \vec{r})$ respectively, for all states q_1, s. □

5 Final Words

The generality of the result of Latteux, $\mathcal{M}(L) = \widehat{\mathcal{M}}(L)$ for the permutation closure L of any regular language, suggests other types of valence grammars to apply the AFL techniques. Consider the permutation closure A_k of $(a_1 \dots a_k)^*$, i.e., the strings in which each letter from $\{a_1, \dots, a_k\}$ occurs an equal number of times. Then A_k describes a set of legal storage operations, here interpreted as increments on a vector of k natural numbers, with acceptance if all these "counters" are equal.

Then we have, in the style of Theorem 5, $\mathsf{VAL}(\mathbb{N}^k) = \mathsf{COD}(\mathsf{CF} \wedge \mathcal{M}(A_k))$, where $\mathsf{VAL}(\mathbb{N}^k)$ denotes the family accepted by the corresponding context-free valence grammars, i.e., valences in $\mathbb{N}^k$ and the condition that all components in the total valence of the derivation are equal. At first glance this type of valence grammar does not seem to fit the framework of [FS00] as the additional valence condition is not on the identity $\vec{0}$ but on equality of the components.

Unfortunately, this does not lead to an extension of the model, as it can be shown that $\mathsf{VAL}(\mathbb{N}^{k+1}) = \mathsf{VAL}(\mathbb{Z}^k)$, either directly, or by observing that $\mathcal{M}(A_{k+1}) = \mathcal{M}(B_k)$. For example, $\mathsf{VAL}(\mathbb{N}^{k+1}) \supseteq \mathsf{VAL}(\mathbb{Z}^k)$, as $B_k = h^{-1}(A_{k+1})$ for the homomorphism that maps b_i to $a_1 \dots a_{i-1}a_{i+1} \dots a_{k+1}$ (the string of all a's, except a_i) and leaves each a_i unchanged. This relation can be better understood if we interpret the symbols $a_1, \dots, a_k, a_{k+1}$ in another way as storage instructions: $a_1, \dots, a_k$ increment a counter, as before, but a_{k+1} acts as a synchronous decrement on all k counters.

Acknowledgements. The author is most obliged to Joost Engelfriet: for lending his copy of [Gin75] and teaching how to read it. The referees of DLT'01 kindly commented on a previous version of this paper.

References

[DP89] J. Dassow, G. Păun. *Regulated Rewriting in Formal Language Theory.* EATCS Monographs in Theoretical Computer Science, vol. 18. Springer-Verlag, 1989.

[FS97] H. Fernau, R. Stiebe. Regulation by Valences. In: B. Rovan (ed.) Proceedings of MFCS'97, Lecture Notes in Computer Science, vol. 1295, pages 239-248. Springer-Verlag, 1997.

[FS00] H. Fernau, R. Stiebe. Sequential Grammars and Automata with Valences. Technical Report WSI-2000-25, Wilhelm- Schickard-Institut für Informatik, Universität Tübingen, 2000. Submitted. Available via `www.informatik.uni-tuebingen.de/bibliothek/wsi-reports.html`

[FMR68] P.C. Fischer, A.R. Meyer, A.R. Rosenberg. Counter Machines and Counter Languages. *Mathematical Systems Theory* 2 (1968) 265-283.

[Gin75] S. Ginsburg. *Algebraic and Automata-theoretic Properties of Formal Languages*, Fundamental Studies in Computer Science, vol. 2, North-Holland, 1975.

[Gre78] S.A. Greibach. Remarks on Blind and Partially Blind One-Way Multicounter Machines. *Theoretical Computer Science* 7 (1978) 311- 324.

[HvV00] H.J. Hoogeboom, N. van Vugt. Fair Sticker Languages. *Acta Informatica* 37 (2000) 213-225.

[HvV01] H.J. Hoogeboom, N. van Vugt. Upper Bounds for Restricted Splicing. LIACS Technical Report 01-05, 2001. Submitted.

[Lat79] M. Latteux. Cônes Rationnels Commutatifs. *Journal of Computer and Systems Sciences* 18 (1979) 307-333.

[Pau80] G. Păun. A New Generative Device: Valence Grammars. *Revue Roumaine de Mathématiques Pures et Appliquées* 6 (1980) 911-924.

[Sal73] A. Salomaa. *Formal Languages.* ACM Monograph Series, Academic Press, 1973.

A Motivation

This appendix describes our motivation for considering valence grammars.

In the language theory of splicing systems the cutting and recombination of DNA molecules with the help of restriction enzymes is modelled by an operation on languages: two strings $x = x_1x_2$ and $y = y_1y_2$ are recombined to give the new string $z = x_1y_2$. The place where the two original strings are cut (in between x_1 and x_2, and y_1 and y_2, respectively) is specified by a set of rules. We refer to [HPP97] for an overview.

The power of this operation is studied by looking at its effect on languages in the Chomsky hierarchy under different types of rules. If the initial language L is regular and the rules are given as context-free language R (or reversely, L is context-free and R regular) then the operation can be modelled as follows. Both L and R are coded in a context-free language L_R in which the strings are of the form $x_1\#x_2\$y_1\#y_2$, the splicing operation σ can be performed by a finite state transduction, mapping the above string into the result x_1y_2. Consequently, the resulting splicing language $\sigma(L, R)$ is context-free.

Additional constraints involving the lengths of the two strings that participate in the splicing (increasing splicing, same length splicing [KPS96]) require that $|x_1x_2| = |y_1y_2|$ or that $|x_1| \geq |y_1|$. As a consequence, L_R is no longer context-free, but context-sensitive, a family not closed under finite state transductions.

In order to get a reasonable upper bound on the family to which $\sigma(L, R)$ belongs, we discovered the valence grammars (more precisely, additive context-free valence grammars) to be a suitable formalism. The family of valence languages is powerful enough to model initial language L and rules R with additional constraints, it is strictly within the family of context sensitive languages, and finally, is closed under finite state transductions.

References

[HPP97] T. Head, G. Păun, D. Pixton. Language Theory and Molecular Genetics: Generative Mechanisms Suggested by DNA Recombination. In: G. Rozenberg and A. Salomaa (eds.) Handbook of Formal Languages, volume 2. Springer-Verlag, 1997.

[KPS96] L. Kari, G. Păun, A. Salomaa. The Power of Restricted Splicing with Rules from a Regular Language. *Journal of Universal Computer Science*, 2 (224-240) 1996.