A Note on the Complexity of Restricted Attribute-Value Grammars

Leen Torenvliet[1]& Marten Trautwein[2]
University of Amsterdam
Department of Mathematics and Computer Science
Plantage Muidergracht 24
1018 TV Amsterdam

Abstract

The recognition problem for attribute-value grammars (AVGs) was shown to be undecidable by Johnson in 1988. Therefore, the general form of AVGs is of no practical use. In this paper we study a very restricted form of AVG, for which the recognition problem is decidable (though still NP-complete), the R-AVG. We show that the R-AVG formalism captures all of the context free languages and more, and introduce a variation on the so-called *off-line parsability constraint*, the *honest parsability constraint*, which lets different types of R-AVG coincide precisely with well-known time complexity classes.

1 Introduction

Although a universal feature theory does not exist, there is a general understanding of its objects. The objects of feature theories are abstract linguistic objects, e.g., an object "sentence," an object "masculine third person singular," an object "verb," an object "noun phrase." These abstract objects have properties like "tense," "number," "predicate," "subject." The values of these properties are either atomic, like "present" and "singular," or abstract objects, like "verb" and "noun-phrase." The abstract objects are fully described by their properties and their values. Multiple descriptions for the properties and values of the abstract linguistic objects are presented in the literature. Examples are:

1. Feature graphs, which are labeled rooted directed acyclic graphs $G = (V, A)$, where F is a collection of labels, a sink in the graph represents

[1]The author was supported in part by HC&M grant ERB4050PL93-0516.
[2]The author was supported by the Foundation for language, speech and logic (TSL), which is funded by the Netherlands organization for scientific research (NWO)

an atomic value and the labeling function is an injective function $f : V \times A \mapsto F$.

2. Attribute-value matrices, which are matrices in which the entries consist of an attribute and a value or a reentrance symbol. The values are either atomic or attribute-value matrices.

From a computational point of view, all descriptions that are used in practical problems are equivalent. Though there exist some theories with a considerably higher expressive power (Blackburn and Spaan 1993). For this paper we adopt the feature graph description, which we will define somewhat more formally in the next section. Attribute Value Languages (AVL) (Smolka 1992) consist of sets of logical formulas that describe classes of feature graphs, by expressing constraints on the type of paths that can exist within the graphs. To wit: In a sentence like "a man walks" the edges labeled with "person" that leave the nodes labeled "a man" and "walks" should both end in a node labeled "singular." Such a constraint is called a "path equation" in the attribute-value language.

A rewrite grammar (Chomsky 1956) can be enriched with an AVL to construct an Attribute Value Grammar (AVG), which consists of pairs of rewrite-rules and logical formulas. The rewrite rule is applicable to a production (nonterminal) only if the logical formula that expresses the relation between left- and right-hand side of the rule evaluates to true. The recognition problem for attribute-value grammars can be stated as: Given a grammar G and a string w does there exist a derivation in G, that respects the constraints given by its AVL, and that ends in w. As the intermediate productions correspond to feature graphs this question can also be formulated as a question about the existence of a consistent sequence of feature graphs that results in a feature graph describing w. For the rewrite grammar, any formalism in the Chomsky hierarchy (from regular to type 0) can be chosen. From a computational point of view it is of course most desirable to restrict oneself to a formalism that on the one hand gives enough expressibility to describe a large fragment of the (natural) language, and on the other hand is restrictive enough to preserve feasibility. For a discussion on the linguistic significance of such restrictions, see Perrault (1984).

Johnson (1988) proved that attribute-value grammars that are as restrictive as being equipped with a rewrite grammar that is regular can already give rise to an undecidable recognition problem. Obviously, to be of any practical use, the rewrite grammar or the attribute-value language must be more restrictive. Johnson proposed to add the *off-line parsability constraint*, which is respected if the rewrite grammar has no chain- or ϵ-rules. Then, the number of applications in a production is linear and the size of the structure corresponding to the partial productions is polynomial. Hence as by a modification of Smolka's algorithm (Smolka 1992) consistency of intermediate steps can be checked in quadratic time, the complexity of the recognition problem can at most be (nondeterministic) polynomial time. This observation was made in (Trautwein 1995), which also has an *NP*-hardness proof of the recognition problem.

We further investigate the properties of these restricted AVGs (R-AVGs). In the next section, we give some more formal definitions and notations. In Section 3 we show that the class of languages generated by an R-AVG (R-AVGL) includes the class of context free languages (CFL). It follows that any easily parsable class of languages (like CFL) is a proper subset of R-AVGL, unless $P = NP$. Likewise, R-AVGL is a proper subset of the class of context sensitive languages, unless $NP = PSPACE$. In Section 4 we propose a further refinement on the off-line parsability constraint, which allows R-AVGs that respect this constraint to capture *precisely* complexity classes like *NP* or *NEXP*. That is, for any language L that has an *NP*-parser, there exists an R-AVG, say G, such that $L = L(G)$. Though our refinement, the *honest parsability constraint* is probably not a property that can be decided for arbitrary R-AVGs, we show that R-AVGs can be equipped with restricting mechanisms that enforce this property. The techniques that prove Theorem 3.1 and Theorem 4.2 result from Johnson's work. Therefore, the proofs of these theorems are deferred to the appendices.

2 Definitions and Notation

2.1 Attribute-Value Grammars

The definitions in this section are in the spirit of (Johnson 1988, Section 3.2) and (Smolka 1992, Sections 3–4). Consider three sets of pairwise disjoint symbols.

A, the finite set of constants, denoted $(a, b, c, \ldots)$

V, the countable set of variables, denoted $(x, y, z, \ldots)$

L, the finite set of attributes, also called features, denoted $(f, g, h, \ldots)$

Definition 1: An *f-edge* from x to s is a triple (x, f, s) such that x is a variable, f is an attribute, and s is a constant or a variable. A *path*, p, is a, possibly empty, sequence of f-edges $(x_1, f_1, x_2), (x_2, f_2, x_3), \ldots, (x_n, f_n, s)$ in which the x_i are variables and s is either a variable or a constant. Often a path is denoted by the sequence of its edges' attributes, in reversed order, e.g., $p = f_n \ldots f_1$. Let p be a path, ps denotes the path that starts from s, where s is a constant only if p is the empty path. If the path is nonempty, $p = f_n \ldots f_1$ $(n \geq 1)$, then s is a variable. For paths ps and qt we write $ps \doteq qt$ iff p and q start in s and t respectively and end in the same variable or constant. The expression $ps \doteq qt$ is called a *path equation*. A *feature graph* is either a pair $(a, \emptyset)$, or a pair (x, E) where x is the root and E a finite set of f-edges such that:

1. if (y, f, s) and (y, f, t) are in E, then $s = t$;
2. if (y, f, s) is in E, then there is a path from x to y in E.

Definition 2: An *attribute-value language* $\mathcal{A}(A, V, L)$ consists of sets of logical formulas that describe feature graphs, by expressing constraints on the type of paths that can exist within the graphs.

- The terms of an attribute-value language $\mathcal{A}(A, V, L)$ are the constants and the variables $s, t \in A \cup V$.
- The formulas of an attribute-value language $\mathcal{A}(A, V, L)$ are path equations and Boolean combinations of path equations. Thus all formulas are either $ps \doteq qt$, where ps and qt are paths, or $\phi \wedge \psi$, $\phi \vee \psi$, or $\neg\phi$, where ϕ and ψ are formulas.

Assume a finite set Lex (of lexical forms) and a finite set Cat (of categories). Lex will play the role of the set of terminals and Cat will play the role of the set of nonterminals in the productions.

Definition 3: A *constituent structure tree* (CST) is a labeled tree in which the internal nodes are labeled with elements of Cat and the leaves are labeled with elements of Lex.

Definition 4: Let T be a constituent structure tree and F be a set of formulas in an attribute-value language $\mathcal{A}(A, V, L)$. An *annotated constituent structure tree* is a triple $<T, F, h>$, where h is a function that maps internal nodes in T onto variables in F.

Definition 5: A *lexicon* is a finite subset of $\text{Lex} \times \text{Cat} \times \mathcal{A}(A, \{x_0\}, L)$. A set of *syntactic rules* is a finite subset of $\bigcup_{i \geq 1} \text{Cat} \times \text{Cat}^i \times \mathcal{A}(A, \{x_0, \ldots, x_i\}, L)$. An *attribute-value grammar* is a triple $<\text{lexicon}, \text{rules}, \text{start}>$, where lexicon is a lexicon, rules is a set of syntactic rules and start is an element of Cat.

Definition 6:

1. **(Balcázar et al. 1988, p .150)** A class $\mathcal{C}$ of sets is *recursively presentable* iff there is an effective enumeration $M_1, M_2, \ldots$ of deterministic Turing machines which halt on all their inputs, and such that $\mathcal{C} = \{L(M_i) \mid i = 1, 2, \ldots\}$.
2. We say that a class of grammars $\mathcal{G}$ is *recursively presentable* iff the class of sets $\{L(G) \mid G \in \mathcal{G}\}$ is recursively presentable.

2.2 Restricted Attribute-Value Grammars

The only formulas that are allowed in the attribute-value language of restricted attribute-value grammars (R-AVGs) are path-equations and conjunctions of path-equations (i.e. disjunctions and negations are out). We will denote the attribute-value language of an R-AVG by $\mathcal{A}'(A, V, L)$ to make the distinction clear. The CST of an R-AVG is produced by a chain- and ϵ-rule free regular grammar. The CST of an R-AVG can be either a left-branching or a right-branching tree, since the grammar contains at most one nonterminal in each rule.

Definition 7: The set of syntactic rules of a restricted attribute-value grammar is a subset of $\bigcup_{i \geq 1, k \leq 1} \text{Cat} \times \text{Lex}^i \times \text{Cat}^k \times \mathcal{A}'(A, \{x_0, x_k\}, L)$. A *restricted attribute-value grammar* is a pair $<$rules, start$>$, where rules is a set of syntactic rules and start is an element of Cat.

Definition 8: An R-AVG $<$rules, start$>$ *generates* an annotated constituent structure tree $<T, F, h>$ iff

1. the root node of T is start, and
2. every internal node of T is licensed by a syntactic rule, and
3. the set F is consistent, i.e., describes a feature graph.

Let $\phi[x/y]$ stand for the formula ϕ in which variable y is substituted for variable x. An internal node v of an annotated constituent structure tree is *licensed* by a syntactic rule $(c_0, l_1, \ldots, l_i, \phi)$ iff

1. the node v is labeled with category c_0, $h(v) = n_0$, and
2. all daughters of v are leaves, which are labeled with $l_1 \ldots l_i$, and
3. $\phi[x_0/n_0]$ is in the set F.

An internal node v of an annotated constituent structure tree is *licensed* by a syntactic rule $(c_0, l_1, \ldots, l_i, c_1, \phi)$ iff

1. the node v is labeled with category c_0, $h(v) = n_0$, and
2. one of v's daughters is an internal node, v_1, which is labeled with category c_1, and $h(v_1) = n_1$, and
3. the daughters of v that are leaves are labeled with $l_1 \ldots l_i$, and
4. $\phi[x_0/n_0, x_1/n_1]$ is in the set F.

3 Weak Generative Capacity

In (Trautwein 1995), it is shown that the recognition problem for R-AVGs is *NP*-complete. This seems to indicate that although the mechanism for generating CSTs in R-AVGs is extremely simple, the generative capacity of R-AVGs is different from the generative capacity of e.g., context free languages (CFLs), which have a polynomial time parsing algorithm (Earley 1970). Yet, a priori, there may exist CFLs that do not have an R-AVG.

Theorem 3.1 *Let L be a context free language. There exists an R-AVG G such that $L = L(G)$.*

Proof. If L is a context free language, then there exists a context free grammar G' in Greibach normal form such that $L = L(G')$. From this grammar G', we can construct a pushdown store M that accepts exactly the words in $L(G') = L$. Such a pushdown store M is actually a finite state automaton M' with a stack S. The finite state automaton M' may be simulated by a chain- and ϵ-rule free regular grammar. Furthermore, we can construct an attribute-value language $\mathcal{A}'(A, V, L)$ that simulates the stack S. Thus it should be clear that there exists an R-AVG G that produces word w iff $w \in L(G')$. Details of this construction are deferred to Appendix A. □

From this we can draw the conclusion that the class of context free languages is indeed a proper subset of the class of R-AVG languages, unless $P = NP$.

Theorem 3.2 *Let $\mathcal{C}$ be a recursively presentable class of grammars such that:*

1. *$G \in \mathcal{C}$ can be decided in time polynomial in $|G|$*
2. *$G \overset{*}{\Rightarrow} w$ can be decided in time polynomial in $|G| + |w|$.*

If every R-AVG G has a grammar in $\mathcal{C}$ then $P = NP$. In fact, for every language L in NP there is an explicit deterministic polynomial time algorithm.

Proof. Let L be a language in NP and $w \in \{0,1\}^*$. Trautwein (Trautwein 1995) provided an R-AVG G and a reduction that maps any formula F onto a string w_F s.t. $G \overset{*}{\Rightarrow} w_F$ iff $F \in SAT$. It was also shown that any R-AVG has a nondeterministic polynomial time, hence deterministic exponential time, recognition algorithm. Suppose every R-AVG G has a grammar in $\mathcal{C}$. Then there exists a $G' \in \mathcal{C}$ with $L(G') = L(G)$. We can decide in polynomial time whether $w_F \in L(G)$ for any w_F. So, $P = NP$.

If every R-AVG G has a grammar in $\mathcal{C}$, then the algorithm for deciding "$w \in L$?" consists of: use Cook's reduction to produce a formula F that is satisfiable iff $w \in L$; use Trautwein's reduction to produce w_F and R-AVG G; enumerate grammars in $\mathcal{C}$ for the first grammar G' that has a description of length less than $\log\log|w|$ for which $L(G) \cap \{0,1\}^{\leq \log\log|w|} = L(G') \cap \{0,1\}^{\leq \log\log|w|}$ accept iff $w \in L(G')$. This gives a polynomial time algorithm that erroneously accepts or rejects w for only a finite number of strings w. The theorem now follows from the fact that both P and NP are closed under finite variation. □

Corollary 3.3 *If R-AVGs generate only context free languages then $P = NP$.*

In fact it can be shown directly that R-AVGs also produce non-context free languages.

Theorem 3.4 *The context sensitive language $\{a^n b^n c^n\}$ is generated by an R-AVG.*

Proof.(Sketch) Typically, the R-AVG that generates the language $\{a^n b^n c^n\}$ first generates an amount of a's then an amount of b's and finally an amount of c's. Let us assume that the grammar generates i a's. During the derivation, the feature graph can be used to store the amount of a's that is produced. Once the grammar starts to produce b's , the feature graph will force the grammar to generate exactly i b's and next to generate exactly i c's as well. □

4 The Honest Parsability Constraint and Consequences

According to Theorem 3.2, it is unlikely that the languages generated by R-AVGs can be limited to those languages with a polynomial time recognition algorithm. Trautwein (Trautwein 1995) showed that all R-AVGs have nondeterministic polynomial time algorithms. Is it perhaps the case that any language that has a nondeterministic polynomial time recognition algorithm can be generated by an R-AVG. Does there exist a tight relation between time bounded machines and R-AVGs as e.g., between LBAs and CSLs? The answer is that the off-line parsability constraint that forces the R-AVG to have no chain- or ϵ-rules is just too restrictive to allow such a connection. The following trick to alleviate this problem has been observed earlier in complexity theory. The off-line parsability constraint(OLP) (Johnson 1988) relates the amount of "work" done by the grammar to produce a string linearly to the number of terminal symbols produced. It is therefore a sort of honesty constraint that is also demanded of functions that are used in e.g., cryptography. There the deal is, for each polynomial amount of work done to compute the function at least one bit of output must be produced. In such a way, for polynomial time computable functions one can guarantee that the inverse of the function is computable in nondeterministic polynomial time.

As a more liberal constraint on R-AVGs we propose an analogous variation on the OLP

Definition 9: A grammar G satisfies the Honest Parsability Constraint(HPC) iff there exists a polynomial p s.t. for each w in $L(G)$ there exists a derivation with at most $p(|w|)$ steps.

From Smolka's algorithm and Trautwein's observation it trivially follows that any attribute-value grammar that satisfies the HPC (HP-AVG) has an *NP* recognition algorithm. The problem with the HPC is of course that it is not a syntactic property of grammars. The question whether a given AVG satisfies the HPC (or the OLP for that matter) may well be undecidable. Nonetheless, we can produce a set of rules that, when added to an attribute-value grammar *enforces* the HPC. The newly produced language is then a subset of the old produced language with an *NP* recognition algorithm. Because of the fact that our addition may simulate any polynomial restriction, we regain the full class of AVG's that satisfy the HPC. In fact

Theorem 4.1 *The class, P-AVGL, of languages produced by the HP-AVGs is recursively presentable.*

We will give a detailed construction of such a set of rules in Appendix B. The existence of such a set of rules and the work of Johnson now gives the following theorem.

Theorem 4.2 *For any language L that has an NP recognition algorithm, there exists a restricted attribute-value grammar G that respects the HPC and such that $L = L(G)$.*

Proof.(Sketch) Let M be the Turing machine that decides $w \in L$. Use a variation of Johnson's construction of a Turing machine to create an R-AVG that can produce any string w that is recognized by M. Add the set of rules that guarantee that only strings that can be produced with a polynomial number of rules can be produced by the grammar. □

5 Veer out the HPC

Instead of creating a counter of logarithmic size as we do in Appendix B, it is quite straightforward to construct a counter of linear size (or exponential size if there is enough time). In fact, for well-behaved functions, the construction of a counter gives a method to enforce any desired time bound constraint on the recognition problem for attribute-value grammars. For instance, for nondeterministic exponential time we could define the Linear Dishonest Parsability Constraint (LDP) (allowing a linear exponential number of steps) which would give.

Theorem 5.1 *The class of languages generated by R-AVGs obeying the LDP condition is exactly NEXP.*

Acknowledgements

We are indebted to E. Aarts and W.C. Rounds for their valuable suggestions on an early presentation of this work.

References

Balcázar, J., Díaz, J., and Gabarró, J. (1988). *Structural Complexity I.* New York: Springer-Verlag.

Blackburn, P., and Spaan, E. (1993). A modal perspective on the computational complexity of attribute value grammar. *Journal of Logic, Language and Information*, 2(2):129–169.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, 13(2):94–102.

Hopcroft, J., and Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison Wesley.

Johnson, M. (1988). *Attribute-Value Logic and the Theory of Grammar*, vol. 16 of *CSLI Lecture Notes*. Stanford: CSLI.

Perrault, C. (1984). On the mathematical properties of linguistic theories. *Computational Linguistics*, 10(3–4):165–176.

Smolka, G. (1992). Feature-constraint logics for unification grammars. *Journal of Logic Programming*, 12(1):51–87.

Sudkamp, T. (1988). *Languages and Machines: An introduction to the Theory of Computer Science*. Reading, MA: Addison Wesley.

Trautwein, M. (1995). Assessing complexity results in feature theories. ILLC Research Report and Technical Notes Series LP-95-01, University of Amsterdam.

A Simulating a Context Free Grammar in GNF

A context free grammar (CFG) is a quadruple $\langle N, \Sigma, P, S\rangle$, where N is a set of nonterminals, Σ is a set of terminals, P is a set of productions, and $S \in N$ is the start nonterminal. A CFG is in Greibach normal form (GNF) if, and only if, the productions are of one of the following forms, where $a \in \Sigma, A \in N, A_1 \ldots A_n \in N \setminus \{S\}$ and ϵ the empty string (c.f., (Hopcroft and Ullman 1979), (Sudkamp 1988)):

$$\begin{array}{lcl} A & \to & a\, A_1 \ldots A_n \\ A & \to & a \\ S & \to & \epsilon \end{array}$$

Given a GNF $G = \langle N, \Sigma, P, S\rangle$, we can construct a restricted attribute-value grammar (R-AVG) G' that simulates grammar G. R-AVG G' consists of the same set of nonterminals and terminals as GNF G. The productions of R-AVG G' are described by Table 1. The only two attributes of R-AVG G' are TOP and REST. R-AVG G' contains $|N| + 1$ atomic values, one atomic value for each nonterminal and the special atomic value \$. The R-AVG G' uses the feature graph to encode a push-down stack, similar to the encoding of a list. The stack will be used to store the nonterminals that still have to be rewritten.

The three syntactic abbreviations below are used to clarify the simulation. We use represent a stack by a Greek letter, or a string of symbols; the top of the stack is the leftmost symbol of the string. Let x_0 encode a stack γ, then the formulas in the abbreviation PUSH$(A_0 \ldots A_n)$ express that x_1 encodes a stack $A_0 \ldots A_n\gamma$. Likewise, the formulas in the abbreviation POP(A) express that x_0 encodes a stack $A\gamma$, and x_1 encodes the stack γ. The abbreviation EMPTY-STACK expresses that x_0 encodes an empty stack.

$$\begin{array}{rll} \text{PUSH}(A_0 \ldots A_n) & \text{stands for} & \text{TOP}(x_1) \doteq A_0 \wedge \\ & & \text{TOP REST}(x_1) \doteq A_1 \wedge \\ & & \vdots \\ & & \text{TOP REST}^n(x_1) \doteq A_n \wedge \\ & & \text{REST}^{n+1}(x_1) \doteq x_0 \\ \text{POP}(A) & \text{stands for} & \text{TOP}(x_0) \doteq A \wedge \\ & & \text{REST}(x_0) \doteq x_1 \\ \text{EMPTY-STACK} & \text{stands for} & x_0 \doteq \$ \end{array}$$

We have to prove that GNF G and its simulation by R-AVG G' generate (almost) the same language. Obviously, R-AVG G' cannot generate the empty string. However, for all non-empty strings the following theorem holds.

Theorem A.1 *Start nonterminal S of GNF G derives string α ($\alpha \in \Sigma^+$) if, and only if, start nonterminal S of R-AVG G' derives string α with the empty stack.*

Productions of GNF G		Productions of R-AVG G'	
$S \to aA_1 \dots A_n$	$\rightsquigarrow$	$S \to aA_1$	
		PUSH$(A_2 \dots A_n) \wedge$ EMPTY-STACK	
$A \to aA_1 \dots A_n$	$\rightsquigarrow$	$A \to aA_1$	
		PUSH$(A_2 \dots A_n)$	$(A \neq S)$
$S \to a$	$\rightsquigarrow$	$S \to aB$	$\forall B \in N \setminus \{S\}$
		POP$(B) \wedge$ EMPTY-STACK	
$S \to a$	$\rightsquigarrow$	$S \to a$	
		EMPTY-STACK	
$A \to a$	$\rightsquigarrow$	$A \to aB$	$\forall B \in N \setminus \{S\}$
		POP(B)	$(A \neq S)$
$A \to a$	$\rightsquigarrow$	$A \to a$	
		EMPTY-STACK	
$S \to \epsilon$	neglected		

Table 1: Simulating productions of GNF G by R-AVG G'

Proof. There are two cases to consider. First, S derives string α in one step. Second, S derives string α in more than one step. The lemma below is needed in the proof of the second case.

Case I Let start nonterminal S derive string α in one step. GNF G contains a production $S \to \alpha$ iff R-AVG G' contains a production $S \to \alpha$ with the equation EMPTY-STACK. So, S derives α in a derivation of GNF G iff S derives α with an empty stack in the derivation of R-AVG G'.

Case II Initial nonterminal S of GNF G derives string $\alpha = \beta\beta'$ in more than one step iff there is a left-most derivation $S \stackrel{*}{\Rightarrow} \beta A \Rightarrow \beta\beta'$. GNF G contains production $A \to \beta'$ iff R-AVG G' contains production $A \to \beta'$ with the equation EMPTY-STACK. By the next lemma: $S \stackrel{*}{\Rightarrow} \beta A$ iff $S \stackrel{*}{\Rightarrow} \beta A$ with the empty stack. Hence S derives α for GNF G iff S derives α with empty stack for R-AVG G'. □

Lemma A.2 *Start nonterminal S derives $\alpha A\gamma$ ($\alpha \in \Sigma^+, A\gamma \in (N \setminus \{S\})^+$) in a left-most derivation of GNF G if, and only if, nonterminal S derives αA with stack $\gamma\$$ ($\$$ is the bottom-of-stack symbol) in the derivation of R-AVG G'.*

Proof. The lemma is proven by induction on the length of the derivation.

Basis If S derives $\alpha A\gamma$ in one step, then GNF G contains production $S \to \alpha A\gamma$ and R-AVG G' contains production $S \to \alpha A$ with stack $\gamma\$$. If S derives αA with stack $\gamma\$$ in one step, then R-AVG G' contains production $S \to \alpha A$ with stack $\gamma\$$ and GNF G contains production $S \to \alpha A\gamma$.

Induction The induction hypotheses states that $S \stackrel{n}{\Rightarrow} \alpha A\gamma$ in GNF G iff $S \stackrel{n}{\Rightarrow} \alpha A$ with stack $\gamma\$$ in R-AVG G'. Next, we distinguish three cases.

1. GNF G contains a production $A \to aA_1A_2 \dots A_n$. Hence there is a left-most derivation $S \overset{n+1}{\Rightarrow} \alpha a A_1 A_2 \dots A_n \gamma$. GNF G contains the production $A \to aA_1A_2 \dots A_n$ iff R-AVG G' contains a production $A \to aA_1$ with equation $\textsc{push}(A_2 \dots A_n)$. Since the induction hypotheses states that there is a derivation $S \overset{n}{\Rightarrow} \alpha A$ with stack $\gamma\$$, there is a derivation $S \overset{n+1}{\Rightarrow} \alpha a A_1$ with stack $A_2 \dots A_n \gamma\$$.
2. GNF G contains a production $A \to a$ and $\gamma = B'\gamma'$. Hence there is a left-most derivation $S \overset{n+1}{\Rightarrow} \alpha a B' \gamma'$. GNF G contains the production $A \to a$ iff R-AVG G' contains productions $A \to aB$ with equation $\textsc{pop}(\mathrm{B})$, for all $B \in N \setminus \{S\}$. Hence by the induction hypotheses, there is a derivation $S \overset{n+1}{\Rightarrow} \alpha a B'$ with stack $\gamma'\$$.
3. GNF G contains a production $A \to a$ and $\gamma = \epsilon$. Then there is a left-most derivation $S \overset{n+1}{\Rightarrow} \alpha a$. GNF G contains the production $A \to a$ iff R-AVG G' contains production $A \to a$ with equation EMPTY-STACK. Hence by the induction hypotheses, there is a derivation $S \overset{n+1}{\Rightarrow} \alpha a$ with stack \$. □

Because every context free language is generated by some GNF G, every context free language is generated by some R-AVG G'.

B Constructing an Honestly Parsable Attribute-Value Grammar

In this section we show how to add a binary counter to an attribute-value grammar (AVG). This counter enforces the Honest-Parsability Constraint (HPC) upon the AVG. To keep this section legible we sometimes use the attribute-value matrices (AVMs) as descriptions. In Section B.2, we show how to create a counter for the AVG. In Section B.3 we show how to extend the syntactic rules and the lexicon of the AVG.

B.1 Arithmetic by AVGs

We start with a little bit of arithmetic.

Natural numbers. The AVMs below encode natural numbers in binary notation. The sequences of attributes 0 and 1 in these AVMs encode natural numbers, from least- to most-significant bit. The attribute V has value 1 (or 0) if, and only if, it has a sister attribute 1 (or 0).

1. The AVMs $\begin{bmatrix} \text{V} & 0 \\ 0 & + \end{bmatrix}$ and $\begin{bmatrix} \text{V} & 1 \\ 1 & + \end{bmatrix}$ encode the natural numbers zero and one.

2. The AVMs $\begin{bmatrix} \text{V} & 0 \\ 0 & [F] \end{bmatrix}$ and $\begin{bmatrix} \text{V} & 1 \\ 1 & [F] \end{bmatrix}$ encode natural numbers iff the AVM $[F]$ encodes a natural number.

Syntactic rules that tests two numbers for equality. Assume a nonterminal A with some AVM $\begin{bmatrix} \text{N} & [F] \\ \text{M} & [H] \end{bmatrix}$, where $[F]$ and $[H]$ encode natural number x and y, respectively. We present one syntactic rule that derives from this nonterminal A a nonterminal B with AVM $\begin{bmatrix} \text{N} & [F] \\ \text{M} & [H] \end{bmatrix}$ if $x = y$.

$$\boxed{\begin{array}{l} A \rightarrow\ B \\ \quad \text{N}(x_0) \doteq \text{M}(x_0) \\ \quad \wedge\, x_0 \doteq x_1 \end{array}}$$

Table 2: The rule to test two numbers for equality.

Clearly, this simple test takes one step. A more sophisticated test, which also tests for inequality, would compare $[F]$ and $[H]$ bit-by-bit. Such a test would take $O(\min(\log(x), \log(y))) = O(\min(|[F]|, |[H]|))$ derivation steps.

Syntactic rules that multiply by two. Assume a nonterminal A with some AVM $\begin{bmatrix} \text{N} & [F] \end{bmatrix}$, where $[F]$ encodes natural number x. We present one syntactic rule that derives from this nonterminal A a nonterminal B with the AVM $\begin{bmatrix} \text{N} & [H] \end{bmatrix}$, where $[H]$ encodes natural number $2x$.

The number N in $[H]$ equals two times N in $[F]$ if, and only if, the least-significant bit of N in $[H]$ is 0, and the remaining bits form the same sequence as the number N in $[F]$. Multiplication by two takes one derivation step.

$$\boxed{\begin{array}{l} A \rightarrow\ B \\ \quad \text{V N}(x_1) \doteq 0 \\ \quad \wedge\, \text{N}(x_0) \doteq 0\ \text{N}(x_1) \end{array}}$$

Table 3: The rule to multiply by two.

Syntactic rules that increments by one. Assume a nonterminal A with some AVM $\begin{bmatrix} \text{N} & [F] \end{bmatrix}$, where $[F]$ encodes natural number x. We present five syntactic rules that derive from this nonterminal A a nonterminal C with AVM $\begin{bmatrix} \text{N} & [H] \end{bmatrix}$, where $[H]$ encodes natural number $x + 1$.

The increment of N requires two additional pointers in the AVM of A: attribute P points to the next bit that has to be incremented; attribute Q points to the most-significant bit of the (intermediate) result. These additional pointers are hidden from the AVMs of the nonterminals A and C.

The five rules from Table 4 increment N by one. Nonterminal A rewrites, in one or more steps, to nonterminal C, potentially through a number of nonterminals B.

$A' \rightarrow C'$ $\textsc{v}\ \textsc{n}(x_0) \doteq 0$ $\wedge 0\ \textsc{n}(x_0) \doteq 1\ \textsc{n}(x_1)$ $\wedge \textsc{v}\ \textsc{n}(x_1) \doteq 1$	$A' \rightarrow B$ $\textsc{v}\ \textsc{n}(x_0) \doteq 1$ $\wedge 1\ \textsc{n}(x_0) \doteq \textsc{p}(x_1)$ $\wedge 0\ \textsc{n}(x_1) \doteq \textsc{q}(x_1)$ $\wedge \textsc{v}\ \textsc{n}(x_1) \doteq 0$	$B \rightarrow B$ $\textsc{v}\ \textsc{p}(x_0) \doteq 1$ $\wedge 1\ \textsc{p}(x_0) \doteq \textsc{p}(x_1)$ $\wedge \textsc{n}(x_0) \doteq \textsc{n}(x_1)$ $\wedge \textsc{v}\ \textsc{q}(x_0) \doteq 0$ $\wedge 0\ \textsc{q}(x_0) \doteq \textsc{q}(x_1)$
$B \rightarrow C'$ $\textsc{v}\ \textsc{p}(x_0) \doteq 0$ $\wedge \textsc{v}\ \textsc{q}(x_0) \doteq 1$ $\wedge 0\ \textsc{p}(x_0) \doteq 1\ \textsc{q}(x_0)$ $\wedge \textsc{n}(x_0) \doteq \textsc{n}(x_1)$	$B \rightarrow C'$ $\textsc{v}\ \textsc{p}(x_0) \doteq 1$ $\wedge 1\ \textsc{p}(x_0) \doteq +$ $\wedge \textsc{n}(x_0) \doteq \textsc{n}(x_1)$ $\wedge \textsc{v}\ \textsc{q}(x_0) \doteq 0$ $\wedge \textsc{v}\ 0\ \textsc{q}(x_0) \doteq 1$ $\wedge 1\ 0\ \textsc{q}(x_0) \doteq +$	

Table 4: Five rules to increment N by one.

The first and fourth rule of Table 4 state that adding one to a zero bit sets this bit to one and ends the increment. The second and third rule state that adding one to a one bit sets this bit to zero and the increment continues. The fifth rule states that adding one to the most-significant bit sets this bit to zero and yields a new most-significant one bit. We claim that $A \stackrel{*}{\Rightarrow} C$ takes $O(\log(x)) = O(|[F]|)$ derivation steps.

Rules, similar to the ones above, can be given that decrement the attribute N by one. We only have to take a little extra care that the number 0 cannot be decremented.

Syntactic rules that sum two numbers. In this section we use the previous test and increment rules (indicated by =). Assume a nonterminal A with some AVM $\begin{bmatrix} \textsc{n} & [F] \\ \textsc{m} & [H] \end{bmatrix}$, where $[F]$ and $[H]$ encode natural number x and y, respectively. We present syntactic rules (Table 5–8) that derive from this nonterminal A a nonterminal C with AVM $\begin{bmatrix} \textsc{n} & [F'] \\ \textsc{m} & [H] \end{bmatrix}$, where $[F']$ encodes the natural number $x + y$.

$A \rightarrow A'$ $\textsc{m}(x_0) \doteq \textsc{m}(x_1)$ $\wedge \textsc{n}(x_0) \doteq \textsc{p}(x_1)$ $\wedge \textsc{m}(x_1) \doteq \textsc{q}(x_1)$ $\wedge \textsc{r}(x_1) \doteq \textsc{n}(x_1)$	$C' \rightarrow C$ $\textsc{n}(x_0) \doteq \textsc{n}(x_1)$ $\wedge \textsc{m}(x_0) \doteq \textsc{m}(x_1)$

Table 5: Two rules to hide the auxiliary pointers.

The increment of N by M is similar to the increment by one. Here, three additional pointers are required: the attributes P and Q point to the bits in N and M respectively that have to be summed next; attribute R points to the most-significant bit of the (intermediate) result. In the addition two states are

$A' \to A'$ V $\mathrm{P}(x_0) \doteq 0$ $\wedge$ V $\mathrm{Q}(x_0) \doteq 0$ $\wedge$ V $\mathrm{R}(x_0) \doteq 0$ $\wedge$ 0 $\mathrm{P}(x_0) \doteq \mathrm{P}(x_1)$ $\wedge$ 0 $\mathrm{Q}(x_0) \doteq \mathrm{Q}(x_1)$ $\wedge$ 0 $\mathrm{R}(x_0) \doteq \mathrm{R}(x_1)$ $\wedge\, \mathrm{N}(x_0) \doteq \mathrm{N}(x_1)$ $\wedge\, \mathrm{M}(x_0) \doteq \mathrm{M}(x_1)$	$A' \to A'$ V $\mathrm{P}(x_0) \doteq 1$ $\wedge$ V $\mathrm{Q}(x_0) \doteq 0$ $\wedge$ V $\mathrm{R}(x_0) \doteq 1$ $\wedge$ 1 $\mathrm{P}(x_0) \doteq \mathrm{P}(x_1)$ $\wedge$ 0 $\mathrm{Q}(x_0) \doteq \mathrm{Q}(x_1)$ $\wedge$ 1 $\mathrm{R}(x_0) \doteq \mathrm{R}(x_1)$ $\wedge\, \mathrm{N}(x_0) \doteq \mathrm{N}(x_1)$ $\wedge\, \mathrm{M}(x_0) \doteq \mathrm{M}(x_1)$	$A' \to A'$ V $\mathrm{P}(x_0) \doteq 0$ $\wedge$ V $\mathrm{Q}(x_0) \doteq 1$ $\wedge$ V $\mathrm{R}(x_0) \doteq 1$ $\wedge$ 0 $\mathrm{P}(x_0) \doteq \mathrm{P}(x_1)$ $\wedge$ 1 $\mathrm{Q}(x_0) \doteq \mathrm{Q}(x_1)$ $\wedge$ 1 $\mathrm{R}(x_0) \doteq \mathrm{R}(x_1)$ $\wedge\, \mathrm{N}(x_0) \doteq \mathrm{N}(x_1)$ $\wedge\, \mathrm{M}(x_0) \doteq \mathrm{M}(x_1)$
$B \to B$ V $\mathrm{P}(x_0) \doteq 1$ $\wedge$ V $\mathrm{Q}(x_0) \doteq 0$ $\wedge$ V $\mathrm{R}(x_0) \doteq 0$ $\wedge$ 1 $\mathrm{P}(x_0) \doteq \mathrm{P}(x_1)$ $\wedge$ 0 $\mathrm{Q}(x_0) \doteq \mathrm{Q}(x_1)$ $\wedge$ 0 $\mathrm{R}(x_0) \doteq \mathrm{R}(x_1)$ $\wedge\, \mathrm{N}(x_0) \doteq \mathrm{N}(x_1)$ $\wedge\, \mathrm{M}(x_0) \doteq \mathrm{M}(x_1)$	$B \to B$ V $\mathrm{P}(x_0) \doteq 0$ $\wedge$ V $\mathrm{Q}(x_0) \doteq 1$ $\wedge$ V $\mathrm{R}(x_0) \doteq 0$ $\wedge$ 0 $\mathrm{P}(x_0) \doteq \mathrm{P}(x_1)$ $\wedge$ 1 $\mathrm{Q}(x_0) \doteq \mathrm{Q}(x_1)$ $\wedge$ 0 $\mathrm{R}(x_0) \doteq \mathrm{R}(x_1)$ $\wedge\, \mathrm{N}(x_0) \doteq \mathrm{N}(x_1)$ $\wedge\, \mathrm{M}(x_0) \doteq \mathrm{M}(x_1)$	$B \to B$ V $\mathrm{P}(x_0) \doteq 1$ $\wedge$ V $\mathrm{Q}(x_0) \doteq 1$ $\wedge$ V $\mathrm{R}(x_0) \doteq 1$ $\wedge$ 1 $\mathrm{P}(x_0) \doteq \mathrm{P}(x_1)$ $\wedge$ 1 $\mathrm{Q}(x_0) \doteq \mathrm{Q}(x_1)$ $\wedge$ 1 $\mathrm{R}(x_0) \doteq \mathrm{R}(x_1)$ $\wedge\, \mathrm{N}(x_0) \doteq \mathrm{N}(x_1)$ $\wedge\, \mathrm{M}(x_0) \doteq \mathrm{M}(x_1)$

Table 6: Rules when the carry bit is not changed.

$A' \to B$ V $\mathrm{P}(x_0) \doteq 1$ $\wedge$ V $\mathrm{Q}(x_0) \doteq 1$ $\wedge$ V $\mathrm{R}(x_0) \doteq 0$ $\wedge$ 1 $\mathrm{P}(x_0) \doteq \mathrm{P}(x_1)$ $\wedge$ 1 $\mathrm{Q}(x_0) \doteq \mathrm{Q}(x_1)$ $\wedge$ 0 $\mathrm{R}(x_0) \doteq \mathrm{R}(x_1)$ $\wedge\, \mathrm{N}(x_0) \doteq \mathrm{N}(x_1)$ $\wedge\, \mathrm{M}(x_0) \doteq \mathrm{M}(x_1)$	$B \to A'$ V $\mathrm{P}(x_0) \doteq 0$ $\wedge$ V $\mathrm{Q}(x_0) \doteq 0$ $\wedge$ V $\mathrm{R}(x_0) \doteq 1$ $\wedge$ 0 $\mathrm{P}(x_0) \doteq \mathrm{P}(x_1)$ $\wedge$ 0 $\mathrm{Q}(x_0) \doteq \mathrm{Q}(x_1)$ $\wedge$ 1 $\mathrm{R}(x_0) \doteq \mathrm{R}(x_1)$ $\wedge\, \mathrm{N}(x_0) \doteq \mathrm{N}(x_1)$ $\wedge\, \mathrm{M}(x_0) \doteq \mathrm{M}(x_1)$

Table 7: Rules when the carry bit is changed.

distinguished. In the one state, the carry bit is zero, indicated by nonterminal A'. In the other state, the carry bit is one, indicated by nonterminal B. We claim that $A \stackrel{*}{\Rightarrow} C$ takes $O(\max(\log(x), \log(y))) = O(\max(|[F]|, |[H]|))$ derivation steps.

Syntactic rules that sum a sequence of numbers. In this section we use the previous summation rules (indicated by $=$). Assume a nonterminal A with some AVM $\begin{bmatrix} \mathrm{L} & [F'] \end{bmatrix}$, where $[F']$ encodes a list of numbers. To wit

$$[F'] = \begin{bmatrix} \mathrm{F} & [G_1] \\ \mathrm{R} & \begin{bmatrix} \mathrm{F} & [G_2] \\ \mathrm{R} & \ldots \begin{bmatrix} \mathrm{F} & [G_n] \\ \mathrm{R} & + \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

where $[G_i]$ encodes natural number x_i. We present syntactic rules (Table 9) that derive from this nonterminal A a nonterminal B with AVM $\begin{bmatrix} \mathrm{SUML} & [F] \\ \mathrm{L} & [F'] \end{bmatrix}$,

$A' \rightarrow C'$ $\textsc{p}(x_0) \doteq +$ $\wedge\, \textsc{q}(x_0) = i$ $\wedge\, \textsc{r}(x_0) = j$ $\wedge\, i = j$ $\wedge\, x_0 \doteq x_1$	$A' \rightarrow C'$ $\textsc{p}(x_0) = i$ $\wedge\, \textsc{q}(x_0) \doteq +$ $\wedge\, \textsc{r}(x_0) = j$ $\wedge\, i = j$ $\wedge\, x_0 \doteq x_1$	$A' \rightarrow C'$ $\textsc{p}(x_0) \doteq +$ $\wedge\, \textsc{q}(x_0) \doteq +$ $\wedge\, \textsc{r}(x_0) \doteq +$ $\wedge\, x_0 \doteq x_1$
$B \rightarrow C'$ $\textsc{p}(x_0) \doteq +$ $\wedge\, \textsc{q}(x_0) = z$ $\wedge\, \textsc{r}(x_0) = z + 1$ $\wedge\, x_0 \doteq x_1$	$B \rightarrow C'$ $\textsc{p}(x_0) = z$ $\wedge\, \textsc{q}(x_0) \doteq +$ $\wedge\, \textsc{r}(x_0) = z + 1$ $\wedge\, x_0 \doteq x_1$	$B \rightarrow C'$ $\textsc{p}(x_0) \doteq +$ $\wedge\, \textsc{q}(x_0) \doteq +$ $\wedge\, \textsc{v}\ \textsc{r}(x_0) \doteq 1$ $\wedge\, 1\ \textsc{r}(x_0) \doteq +$ $\wedge\, x_0 \doteq x_1$

Table 8: Rules that stop the summation.

where $[F]$ encodes the natural number $\Sigma_i x_i$.

The summation requires an additional pointer in the AVM $[F']$: attribute P points to the next element in the list that has to be summed. We claim that $A \stackrel{*}{\Rightarrow} B$ takes $O(\Sigma_i \log(x_i)) = O(|[F']|)$ derivation steps.

$A \rightarrow A'$ $\textsc{v}\ \textsc{n}(x_1) \doteq 0$ $\wedge\, 0\ \textsc{n}(x_1) \doteq +$ $\wedge\, \textsc{l}(x_0) \doteq \textsc{l}(x_1)$ $\wedge\, \textsc{l}(x_0) \doteq \textsc{p}(x_1)$	$A' \rightarrow A'$ $\textsc{suml}(x_0) = y$ $\wedge\, \textsc{f}\ \textsc{p}(x_0) = z$ $\wedge\, \textsc{suml}(x_1) = y + z$ $\wedge\, \textsc{r}\ \textsc{p}(x_0) \doteq \textsc{p}(x_1)$ $\wedge\, \textsc{l}(x_0) \doteq \textsc{l}(x_1)$	$A' \rightarrow B$ $\textsc{p}(x_0) \doteq +$ $\wedge\, \textsc{suml}(x_0) \doteq \textsc{suml}(x_1)$ $\wedge\, \textsc{l}(x_0) \doteq \textsc{l}(x_1)$

Table 9: Three rules that sum a list of numbers.

B.2 Creating a Counter of Logarithmic Size

Create an AVM of the following form:

$$\begin{bmatrix} \textsc{counter} & \begin{bmatrix} \textsc{size} & \begin{bmatrix} 1 \cup 0 & \ldots[1\ +] \end{bmatrix} \\ \textsc{n} & \begin{bmatrix} \textsc{v} & 1 \cup 0 \\ 1 \cup 0 & \begin{bmatrix} \textsc{v} & 1 \cup 0 \\ \ldots & [1\ +] \end{bmatrix} \end{bmatrix} \\ \textsc{m} & \begin{bmatrix} \textsc{v} & 1 \cup 0 \\ 1 \cup 0 & \begin{bmatrix} \textsc{v} & 1 \cup 0 \\ \ldots & [1\ +] \end{bmatrix} \end{bmatrix} \\ \textsc{poly} & \begin{bmatrix} 1 \cup 0 & \ldots[1\ +] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Attribute COUNTER is used to distinguish the AVMs that encodes the counter from those in the original attribute-value grammar. We will neglect the attribute COUNTER in the remainder of this section, because it is not essential here. The attributes SIZE, N, M and POLY encode natural numbers. The attribute SIZE records the size of the string that will be generated. The attribute POLY records the maximum number of derivation steps that is allowed for a string of size SIZE. The attributes N and M are auxiliary numbers.

The construction of the counter starts with an initiation-step. The further construction of the counter consists of cycles of two phases. Each cycle starts in nonterminal A.

Initiation step and first phase. The initiation-step sets the numbers SIZE and N to 0, and the numbers M and POLY to 1. In the first phase of each cycle, the numbers SIZE and N are incremented by 1.

$S \rightarrow A$ $\mathrm{v}\ \mathrm{SIZE}(x_1) \doteq 0$ $\wedge\ 0\ \mathrm{SIZE}(x_1) \doteq +$ $\wedge\ \mathrm{v}\ \mathrm{N}(x_1) \doteq 0$ $\wedge\ 0\ \mathrm{N}(x_1) \doteq +$ $\wedge\ \mathrm{v}\ \mathrm{M}(x_1) \doteq 1$ $\wedge\ 1\ \mathrm{M}(x_1) \doteq +$ $\wedge\ 1\ \mathrm{POLY}(x_1) \doteq +$	$A \rightarrow B$ $\mathrm{SIZE}(x_0) = x$ $\wedge\ \mathrm{SIZE}(x_1) = x + 1$ $\wedge\ \mathrm{N}(x_0) = y$ $\wedge\ \mathrm{N}(x_1) = y + 1$ $\wedge\ \mathrm{M}(x_0) \doteq \mathrm{M}(x_1)$ $\wedge\ \mathrm{POLY}(x_0) \doteq \mathrm{POLY}(x_1)$

Table 10: Initiation-step and first phase.

The second phase of the cycle. In this phase the numbers N and M are compared. If N is twice M, then (*i*) number POLY is extended by k bits, (*ii*) number M is doubled, and (*iii*) number N is set to 0. If N is less than twice M, nothing happens.

The left rule of the second phase doubles the number M in the second and the third equation. The test "Is N equal to 2M?" therefore reduces to one (the first) equation. The fourth equation extend the number POLY with k bits. The fifth and sixth equations set the number N to 0.

The right rule is always applicable. If the right rule is used where the left rule was applicable, then the number N will never be equal to 2M in the rest of the derivation. Thus POLY will not be extended any more.

$B \rightarrow A$ $\mathrm{N}(x_0) = \mathrm{M}(x_1)$ $\wedge \mathrm{M}(x_0) = x$ $\wedge \mathrm{M}(x_1) = 2x$ $\wedge \mathrm{POLY}(x_0) \doteq 0^k\ \mathrm{POLY}(x_1)$ $\wedge \mathrm{v}\ \mathrm{N}(x_1) \doteq 0$ $\wedge 0\ \mathrm{N}(x_1) \doteq +$	$B \rightarrow A$ $x_0 \doteq x_1$

Table 11: The second phase.

We claim that the left rule appears $\log(n)$ times and the right rule $O(n)$ times in a derivation for input of size n. Obviously, the number POLY is

$O(2^{k \log i}) = O(i^k)$ when the number SIZE is i.

B.3 From AVG to HP-AVG

In this section we show how to transform an AVG into an AVG that satisfies the HPC (HP-AVG). Since all computation steps of the HP-AVG only require a linear amount of derivation steps, total derivations of HP-AVGs have polynomial length.

We can divide the attributes of the HP-AVG into two groups. The attributes that encode the counters, and the attributes of the original AVG. The former will be embedded under the attribute COUNTER, the latter under the attribute GRAMMAR. In the sequel, we mean by $\phi|\text{GRAMMAR}$ the formula ϕ embedded under the attribute GRAMMAR, i.e., the formula obtained from ϕ by substituting the variables x_i by $\text{GRAMMAR}(x_i)$.

The HP-AVG is obtained from the AVG in three steps: change the start nonterminal, the lexicon and the syntactic rules. First, the HP-AVG contains the rules of the previous section, which construct the counter. The nonterminal S from Table 10 is the start nonterminal of the HP-AVG. For the nonterminal A the start nonterminal of the AVG is taken. Nonterminal B from Table 11 is a fresh nonterminal, not occurring in the AVG.

Second, the HP-AVG contains an extension of the lexicon of the AVG. The entries of the lexicon are extended in the following way. The size of the lexical form is set to one, and the amount of derivation steps is zero. Thus, if (w, X, ϕ) is the lexicon of the AVG, then (w, X, ψ) is the lexicon of the HP-AVG, where

$$\begin{array}{rcl} \psi & = & \phi|\text{GRAMMAR} \\ & \wedge & \text{V SIZE COUNTER}(x_0) \doteq 1 \\ & \wedge & \text{1 SIZE COUNTER}(x_0) \doteq + \\ & \wedge & \text{POLY COUNTER}(x_0) \doteq + \end{array}$$

Third, the HP-AVG contains extensions of the syntactic rules of the AVG. The syntactic rules are extended in the following way. The numbers POLY and SIZE of the daughter nonterminals are collected in the lists PLIST and SLIST. Both lists are summed. The number SIZE of the mother nonterminal is equal to the sum of SIZE's, and the number POLY of the mother nonterminal is one more than the sum of POLY's. Thus, if $(X_0, X_1, \ldots, X_n, \phi)$ is a syntactic rule of the AVG, then $(X_0, X_1, \ldots, X_n, \psi)$ is a syntactic rule of the HP-AVG, where

$$\begin{array}{rcl} \psi & = & \phi|\text{GRAMMAR} \\ & \wedge & \text{SUMS COUNTER}(x_0) = \Sigma\,\text{SLIST COUNTER}(x_0) \\ & \wedge & \text{SIZE COUNTER}(x_0) = \text{SUMS COUNTER}(x_0) \\ & \wedge & \text{SUMP COUNTER}(x_0) = \Sigma\,\text{PLIST COUNTER}(x_0) \\ & \wedge & \text{SUMP COUNTER}(x_0) = y \\ & \wedge & \text{POLY COUNTER}(x_0) = y + 1 \end{array}$$

$$\wedge \quad \text{F R}^i \text{ SLIST COUNTER}(x_0) \doteq \text{SIZE COUNTER}(x_i) \quad (0 \leq i < n)$$
$$\wedge \quad \text{R}^n \text{ SLIST COUNTER}(x_0) \doteq +$$
$$\wedge \quad \text{F R}^i \text{ PLIST COUNTER}(x_0) \doteq \text{POLY COUNTER}(x_i) \quad (0 \leq i < n)$$
$$\wedge \quad \text{R}^n \text{ PLIST COUNTER}(x_0) \doteq +$$

Now, a derivation for the HP-AVG starts with a nondeterministic construction of a counter SIZE with value n and a counter POLY with value $O(n^k)$. Then, the derivation of the original AVG is simulated, such that (i) the mother nonterminal produces a string of size n if, and only if the daughter nonterminals together produce a string of size n, and (ii) the mother nonterminal makes n^k+1 derivation steps if, and only if the daughter nonterminals together make n^k derivation steps.