Two-level grammars are more expressive than Type 0 grammars
Or are they?

Dick Grune
Dept. of Mathematics and Computer Science
Vrije Universiteit
De Boelelaan 1081
1081 HV Amsterdam
dick@cs.vu.nl

ABSTRACT

Under some simple conditions, two-level grammars can generate languages with unbounded numbers of terminal symbols; Type 0 grammars cannot.

Introduction

In 1967, Sintzoff[1] showed that for every semi-Thue system there is a two-level or VW (van Wijngaarden) grammar. In 1974, van Wijngaarden[2] showed that the production mechanism of VW grammars can be used to simulate a Turing machine. In fact both proved that $L(VW) \supseteq L(Type\ 0)$.

An observation

Surprisingly, it is almost trivial to show that $L(VW) \supset L(Type\ 0)$, i.e., that there are well-defined sets of sequences of symbols that can be generated by a VW grammar but not by a Type 0 grammar. An example is the set

$$S=\{t_1^n \cdots t_k^n \mid n \geq 0, k>0,\ t_1 \cdots t_k \text{ are different symbols}\} \quad (1)$$

i.e. the set of all strings consisting of an unbounded number (k) of different symbols, each occurring the same number of times (n) in succession. A Type 0 grammar cannot produce this set since for such a grammar the number of terminal symbols in it is limited in its definition. The VW grammar in Figure 1 (taken from Grune and Jacobs[3] with substantial modification) produces the set, with t_k corresponding to i^k symbol.

```
N ::  n N ; ε .
K ::  i K ; ε .
ZERO ::  ε .

start: sequence of Ki symbols each N times.

sequence of Ki symbols each N times:
            sequence of K symbols each N times, Ki symbol N times.
sequence of ZERO symbols each N times: ε .

K symbol Nn times: K symbol, K symbol N times.
K symbol ZERO times: ε .
```

Figure 1.

In the classical application of a VW grammar, the ALGOL 68 report[4], the correspondence between protonotions ending in symbol and their representations is specified by enumeration (in § 9.4). The easiest way to specify such a correspondence for the grammar of Figure 1 is to say that the representation of a symbol of the form i... symbol is that same i... symbol. (There is no reason why representations of symbols should consist of contiguous areas of ink or be of limited length: **i**, Θ and **implementation_module** can very well be representations of symbols.) If one wants to avoid the ... in the above correspondence, it is easy to supply a short (finite)

recipe for mapping protonotions denoting symbols onto representations. The recipe could, e.g., first remove all sequences of i-s of length five from the protonotion and add a corresponding number of s to the representation, and then handle the remaining i-s in a similar way.

We note that even if we allow the mechanism of specifying representations by recipe rather than by enumeration to a Type 0 grammar, the latter still cannot generate the set *S*: for an unbounded number of different symbols to be produced they all have to occur somewhere in the right-hand sides of rules.

There are no infinite actions involved in the generation of elements of the set *S* from the grammar in Figure 1. Once a terminal production of the metanotion K has been substituted in the right-hand side of the start symbol, the number of terminal symbols in the the result is fixed and finite. Likewise there are no unusual problems in the syntactic and semantic analysis of an element of the set *S*. Once the parse tree has been constructed, inspection of the (only) direct descendant of the start symbol reveals how many different symbols the element contained.

An attempt at analysis

The above observation raises three related questions: what is exactly a terminal symbol, what is it exactly that a grammar produces, and is *S* a recursively enumerable (r.e.) set?

For a Type 0 grammar, these questions are almost trivial to answer. A terminal symbol in a Type 0 grammar is a named atomic entity on which only one operation is defined: concatenation; the terminal symbols in a particular Type 0 grammar are given by enumeration in the specification of the grammar (and are therefore a finite set). And a Type 0 grammar produces the concatenation of zero or more of such terminal symbols, the generated string. There is still a small problem left, though. In order to do something with the generated string we have to be able to recognize the terminal symbols in it, to see, e.g., that the first symbol is **program**. For this human recognition we require each terminal symbol to have a *representation*. Traditionally for a Type 0 grammar, the representation and the symbol name are the same, disregarding type font differences and other irrelevancies. In this view, we have only those terminal symbols that we specify ourselves and which we can recognize from their representations. There cannot be an unbounded set of terminal symbols since we can only specify finite sets of them and there are no others. Not only is the set *S* not r.e., it is not even a well-defined set at all.

For a VW grammar, the situation is completely different. It does not produce a sequence of terminal symbols, it produces a comma-separated sequence of protonotions, each ending in symbol ([4], § 1.1.3.1.f and § 1.1.3.2); protonotions are sequences of small syntactic marks; and small syntactic marks are defined in the grammar by enumeration. In

```
i symbol, i symbol, i i symbol, i i symbol                    (2)
```

i, s, y, m, b, o and l are small syntactic marks and i symbol and i i symbol are protonotions. The sentence generated by the grammar is obtained by taking each protonotion, finding a representation for it and replacing it by that representation ([4], § 9.3.b), while at the same time deleting the separating comma. If a protonotion is met for which there is no representation, the generated sequence of protonotions was a blind alley and does not form a sentence in the language; this happens, e.g., for the brief pragmat symbol in the grammar in the ALGOL 68 report. Traditionally in VW grammars, the representation of a terminal symbol differs greatly from its name (= the protonotion ending in symbol). The separation of name and representation allows us an easy way to accommodate unbounded sets of terminal symbols: we agree to accept the name of the terminal symbol as its representation (as we customarily do in Type 0 grammars). Since the names are generated by the grammar, there can be an unbounded number of them.

In this view, the set *S* is a r.e. set and can be produced by a VW grammar (or a computer program) but not by a Type 0 grammar. Although it is certainly possible to write a Type 0 grammar that produces the above sequence (2), the latter would not consist of four but of 34 terminal symbols: i, s, y, m, b, etc. A mechanism that combines a number of terminal symbols into a new terminal symbol is not a feature of a Type 0 grammar.

Conclusion

There seem to be two consistent views. In one, we accept only the terminal symbol representations we have specified ourselves. Then *S* is not a set and $L(VW) = L(Type\ 0)$, but we deny ourselves the pleasure of unbounded sets of terminal symbols. In the other view, we allow the grammar to construct representations of terminal symbols for us (which we then agree to recognize). Then *S* is a r.e. set and $L(VW) \supset L(Type\ 0)$. Whether these unbounded sets of terminal symbols are in any way useful is not yet clear.

Acknowledgements

I thank Evert Wattel and an anonymous referee for asking pertinent questions and good advice.

Literature

1. M. Sintzoff, ‘‘Existence of a van Wijngaarden syntax for every recursively enumerable set,’’ *Annales de la Société Scientifique de Bruxelles* **81**(II), pp. 115-118 (1967).
2. A. van Wijngaarden, ‘‘The generative power of two-level grammars,’’ pp. 9-16 in *Automata, Languages and Programming; Lecture Notes in Computer Science #14*, ed. J. Loeckx, Springer-Verlag, Berlin (1974).
3. Dick Grune and Ceriel Jacobs, *Parsing Techniques, A Practical Guide*, Ellis Horwood, Chichester, England (1990), p. 322.
4. A. van Wijngaarden, B.J. Mailloux, J.E.L. Peck, C.H.A. Koster, M. Sintzoff, C.H. Lindsey, L.G.L.T. Meertens, and R.G. Fisker, ‘‘Revised report on the algorithmic language ALGOL 68,’’ *Acta Inform.* **5**, pp. 1-236 (1975).