Extended Free Grammars

Prof. Licia Sbattella

aa 2007-08

Translated and adapted by L. Breveglieri

FREE GRAMMARS EXTENDED BY MEANS OF REGULAR EXPRESSIONS

REGULAR EXPRESSIONS ARE MORE READABLE AND PERSPICUOUS than free grammars, thanks to the iteration operators (star and cross) and to the choice operator (union).

To make free grammars more readable as well, it is often permitted to use regexps also in the right sides of the production rules of the grammar. One thus obtains the so-called EXTENDED FREE GRAMMARS (Extended BNF or EBNF). The grammars of technical languages (programming languages, description languages, etc) are usually presented in extended form.

EBNF production rules can be represented as syntax diagrams. A syntax diagram can be conceived as the flow graph of a syntax analysis algorithm..

> The family LIB of free languages is closed with respect to regular operators (union, star, concatenation); therefore the extended free grammars have the same generating power of the non-extended ones.

EXAMPLE: list of declarations of variable names

$$\textit{character}\ \ \text{testo1, testo2;}\ \textit{real}\ \ \text{temp, result;}$$

$$\textit{integer}\ \ \text{alfa, beta2, gamma;}$$

$\Sigma = \{c, i, r, v, \ \text{','}\ ,\ \text{';'}\}$ where *v* is a variable name

$$\left(\left(c \mid i \mid r\right) v(,v)^{*};\right)^{+}$$

$$S \to SE \mid E \quad E \to AF; \quad A \to c \mid i \mid r \quad F \to v, F \mid v$$

The grammar is longer than the regexp, and makes less evident the existence of a hierarcy of two lists. Moreover, the names A, E and F are arbitrary.

A FREE EXTENDED GRAMMAR (EBNF) G = { V, Σ, P, S } contains exactly $|V|$ rules, each of the form $A \rightarrow \eta$, where η is a regexp over the alphabet $V \cup \Sigma$. For better clarity it is allowed to use derived regular operators, like for instance cross, optionality, repetition, etc.

EXAMPLE: a simplified Algol-like language

$$B \rightarrow b\left[D\right]Ie$$

$$D \rightarrow ((c \mid i \mid r)v(,v)^{*};)^{+}$$

$$I \rightarrow F(;F)^{*}$$

$$F \rightarrow a \mid B$$

A more compact but less readable version. The non-terminal B cannot be eliminated, as it is indispensable to generate nested structures.

$$B \rightarrow b\left[((c \mid i \mid r)v(,v)^{*};)^{+}\right]F(;F)^{*}e$$

$$B \rightarrow b((c \mid i \mid r)v(,v)^{*};)^{*}F(;F)^{*}e$$

$$B \rightarrow b((c \mid i \mid r)v(,v)^{*};)^{*}(a \mid B)(;(a \mid B))^{*}e$$

DERIVATION AND SYNTAX TREE IN THE EXTENDED GRAMMARS

The right side of an extended production rule is a regexp the defines an infinite set of strings. Such strings could be imagined ad the right sides of a grammar G', equivalent to G, but containing infinitely many production rules.

$$A \to (aB)^+$$

$$A \to aB \mid aBaB \mid ....$$

DERIVATION RELATION IN AN EXTENDED GRAMMAR

Given the strings $\eta_1 \quad \eta_2 \in (\Sigma \bigcup V)^*$

one says that η_2 *derives* (immediately) from η_1

$\eta_1 \Rightarrow \eta_2$ if the two strings are factored as follows:

$\eta_1 = \alpha A \gamma, \quad \eta_2 = \alpha \vartheta \gamma$ and there is a rule

$A \to e: \quad e \overset{*}{\Rightarrow} \vartheta$

note that $\eta_1 \quad \eta_2$ ($\alpha A \gamma \quad \alpha \vartheta \gamma$) do not contain neither regular operators nor parentheses. *e* is a regexp.

Similarly, one can define multiple step derivations and the generated language.

EXAMPLE: extended derivation for arithmetic expressions.

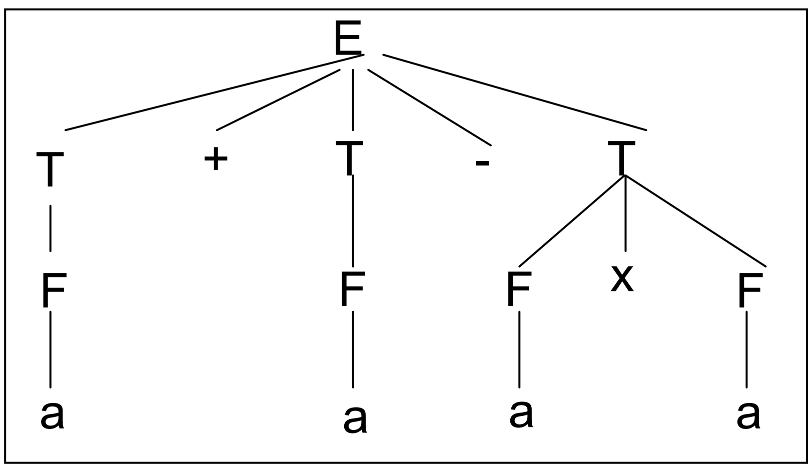
The extended grammar G shown below generates the arithmetic expressions with the four standard operators (in infix notation), round brackets and the symbol *a* to represent generic variables of constants.

$$E \Rightarrow [+ \mid -] T ((+ \mid -) T)^* \quad T \to F ((\times \mid /) F)^* \quad F \to (a \mid '(' E ')')$$

leftmost derivation

$$E \Rightarrow T + T - T \Rightarrow F + T - T \Rightarrow a + T - T \Rightarrow a + F - T \Rightarrow$$

$$\Rightarrow a + a - T \Rightarrow a + a - F \times F \Rightarrow a + a - a \times F \Rightarrow a + a - a \times a$$



In EBNF, the arity of a tree node may be unlimited, in general. However, as the tree gets larger the depth gets lower.

A non-extended but ambiguous grammar is ambiguous also when conceived as extended (it is a special case thereof). However, the presence of regexps in the production rules may give rise to specific ambiguity forms.

$a^*b \mid ab^*$ numbered as $a_1^*b_2 \mid a_3b_4^*$

is ambiguous because ab is

derivable as a_1b_2 or as a_3b_4

$S \rightarrow a^*b \mid ab^*$ is ambiguous

MORE GENERAL GRAMMARS – CHOMSKY CLASSIFICATION

Grammar	*Rule Type*	*Lang. Family*	*Recognizer Device*
Type 0 (recursively denumerable)	$\beta \to \alpha$ where $\alpha, \beta \in (\Sigma \cup V)^+$ and $\beta \neq \varepsilon$	Recursively denumerable languages.	Turing machine.
Type 1 (context-dependent)	$\beta \to \alpha$ where $\alpha, \beta \in (\Sigma \cup V)^+$ and $\lvert \beta \rvert \leq \lvert \alpha \rvert$	Context-dependent languages.	Turing machine with a memory tape of length equal to that of the string to be recognized.

CHOMSKY CLASSIFICATION (continued)

Grammar	*Rule Type*	*Lang. Family*	*Recognizer Device*
Tipo 2 (context- free or BNF form)	$A \to \alpha$ where A is a nonterm. and $\alpha \in (\Sigma \bigcup V)^*$	Contex-free languages or BNF languages or algebraic languages.	Pushdown automaton (non-deterministic).
Tipo 3 (right or left uni-linear)	R-lin: $: A \to uB$ L-lin: $: A \to Bu$ where A is a nonterm. $u \in \Sigma^*$ $B \in (V \bigcup \varepsilon)$	Regular or rational languages.	Finite state automaton or sequential machine.

The language families mentioned before are in a hierarchical relation of strict containment.

Differences: form of the rules and properties of the recognizer devices

type 0, 1 and 2:	automaton with unbounded memory
type 3:	automaton with finite memoty

CLOSURE WITH RESPECT TO:
UNION, CONCAT., STAR, MIRROR, INTERS. WITH REG. LANG.: 0, 1, 2, 3

CLOSURE WITH RESPECT TO COMPLEMENT: 1, 3

TYPE 0: there does not exist any general algorithm to decide whether a string belongs to a given language (the membership problem is demi-decidable).
TYPE 1, 2: there exists a general algorithm to decide whether a string belongs to a given language (the membership problem demi-decidable).
TYPE 3: it is decidable whether two grammars are weakly equivalent.
TYPE 0 e 1: the derivation cannot be represented as a syntax tree; more complex graph structures are necessary.

EXAMPLE OF A TYPE 1 GRAMMAR (just for completeness):

$$L = \left\{ a^n b^n c^n \mid n \geq 1 \right\}$$

$$G(\text{ type 1 }):\quad 1.\ S \to aSBC \quad 3.\ CB \to BC \quad 5.\ bC \to bc$$

$$2.\ S \to abC \quad 4.\ bB \to bb \quad 6.\ cC \to cc$$

Differently to what happens in the free grammars, the language generated by means of leftmost derivations is different from that generated by means of rightmost derivations.

REPLICAS OR LIST
OF MATCHINGS

$$L_{replic} = \left\{ uu \mid u \in \Sigma^{+} \right\}$$

$$\Sigma = \left\{ a, b \right\} \quad x = abbbabbb$$

Type 1 grammars are seldom used to design artificial languages for compiler construction (one exception: Algol-68 was defined by means of two-level grammars). Structures like replicas and matching lists are defined by resorting to semantic methods (attribute grammars or syntax-driven translation).

Bibliography

– S. Crespi Reghizzi, *Linguaggi Formali e Compilazione*, Pitagora Editrice Bologna, 2006

– Hopcroft, Ullman, *Formal Languages and their Relation to Automata*, Addison Wesley, 1969

– A. Salomaa – *Formal Languages*, Academic Press, 1973

– D. Mandrioli, C. Ghezzi – *Theoretical Foundations of Computer Science*, John Wiley & Sons, 1987

– L. Breveglieri, S. Crespi Reghizzi, *Linguaggi Formali e Compilatori: Temi d’Esame Risolti,* web site (eng + ita)