Parsing Expression Grammars

PEG = (Σ (alphabet), N (nonterminals), R (rules, one per nonterminal), S (starting symbol)).

Parsing expression tries to recognise a substring straight ahead, left-to-right (i.e., a prefix of the remaining string) and remove it from the string (unless stated otherwise). Language consists of strings which are entirely recognised by the entire expression.

Parsing expressions are comprised of Σ, N, ε, FAIL, and their closure under the following operations:

- fg – tries to recognise f, then tries g on the remainder
- f/g – tries to recognise f, then tries g on the same string if the first attempt FAILs
- $\&f$ – tries to recognise f, then forgets if anything was removed (but remembers FAIL)
- $!f$ – tries to recognise f, then forgets if anything was removed, FAILing iff the internal expression doesn't.

A symbol from the alphabet is removed if it matches and FAILs if it does not; a nonterminal symbol is replaced with the corresponding rule, then recognition continues.

Can be recognised in linear time by going right-to-left instead of left-to-right.

Scaffolding automata

SA = (Σ (alphabet), Γ (internal alphabet), d (degree), k (depth), Q (states), q_0 (initial state), F (accepting states), δ (transition function)).

Builds a d-regular (with d outcoming edges being distinctly labelled with numbers from 1 to d, some possibly absent) directed graph as it recognises a string. The initial graph consists of a single vertex with no label. Transition depends on the internal state, the labels in the k-neighbourhood of the latest vertex, and the freshly read symbol; it produces a new state, a label for the new vertex, and a list of d vertices from the neighbourhood (some possibly absent) to point the new edges into. Accepts if the state is accepting.

Central theorem: a language is recognised by a PEG iff its reverse is recognised by a SA.

Proof: simulation in both directions.

Theorem 2: for every computable function f there is such a computable function g that the language $\{f(x)\#^{g(x)}x\}_x$ is recognised by a PEG.

Proof (was not finished in time): use the scaffolding automaton to simulate a Turing machine, using the extra # symbols to write the computation states on the internal vertex labels in a back-and-forth fashion (copy the last in reverse order, then flip again while processing the step).

Theorem 3: There is no pumping lemma for PEGs (that is, a totally computable function A that for every PEG G and every large enough ($|x| > n_G$) string x from $L(G)$ the output $y = A(G, x)$ is in $L(G)$ and $|y| > |x|$).

Proof (was not finished in time): use Theorem 2 and a bit of recursive trickery to construct an automaton that always recognises a bigger next string than a given function predicts.