the language machine – from lambda to language and grammar

The *language machine* is an efficient and usable toolkit for language and grammar that is published as free software under the Gnu GPL. At its core is an engine for applying unrestricted grammatical substitution or rewriting rules.

In effect the *language machine* contains the lambda calculus: functions in the lambda calculus and its near relatives can be represented and evaluated by normal order reduction without recourse to side-effect actions, using only a limited subset of rules in the *language machine* **-** rules which treat functions as grammatical substitutions.

It follows that the system of rules in the *language machine* is Turing-complete, and that the languages it can in theory recognise are the languages that can be generated by Chomksy type-0 grammars with unrestricted generative rules.

The lm-diagram is a diagram which shows what happens as rules are applied in the *language machine,* and so it can in theory display any conceivable process of evaluation or analysis in the lambda calculus or the *language machine*.

The limited set of rules that represent functions in the lambda calculus can be extended with unrestricted rules that recognise and substitute nested sequences of grammatical symbols and variable bindings. Such rules are the analytic equivalent of the rules in Chomsky type-0 generative grammars, extended with variable bindings.

All rule applications in the l*anguage machine* are triggered by mismatch between a goal symbol and an input symbol. Rules can be tied to both of these, to the input symbol alone as bottom-up rules, or to the goal symbol alone as top-down rules. Rules can recurse to left, to right and to centre, with left, right, and bracket priority levels. Variable bindings may be used to record the analysis and construct transformations.

The input symbol in each mismatch event may come from an external stream, or from a sequence of symbols and variable bindings that has been produced during the substitution phase of some other rule application. Rules that do pure computation consume no external symbols.

A distinction is commonly made between generative and analytic approaches to grammar: almost all work in computational linguistics derives ultimately from Noam Chomksy's generative model of grammar.

If rules in the *language machine* tend to consume input symbols so as to reduce them to a small number of nonterminal symbols which represent grammatical concepts, they can be said to operate as an analytical grammar.

If rules in the *language machine* tend to generate more symbols than they consume, they can be said to operate as a generative engine.

In practice, useful applications of the *language machine* combine grammatical analysis with generative or computational interpretation using side-effect actions and external calls. But if generative grammars do exist as more than mere abstractions of thought, they can be directly implemented by rules in the *language machine*, whereas generative grammars cannot themselves directly perform any kind of analysis.

Rules in the *language machine* are simple replacements of one pattern by another, where nesting is permitted in both phases of rule application (recognition and substitution), where each phase may produce any number of symbols and where either phase may be empty. As simple replacements, they may provide a fruitful model for physical and causal processes beyond language itself, a model that can be visualised in the lm-diagram and that can be shown to be Turing-complete.

So the *language machine* is directly usable free software, a complete basis for grammar, and a starting point for new directions in language research.