$\mathscr{L}_k(L\#RS) = \mathscr{L}_k(LSMG)$

Radim Krčmář `<ikrcmar@fit.vutbr.cz>`
Brno University of Technology, Faculty of Information Technology

The talk will describe linear #-rewriting systems (L#RSs), linear simple matrix grammars (LSMGs), and a proof that these two formalisms generate the same family of languages.

L#RSs of degree k start with a k hash string ($\#^k$) and a state; each rewriting rule can change the state and rewrite the i-th # into a string that contains terminals and up to one #. LSMGs of degree k start from a non-terminal S and either directly derive a string of terminals or a string of k non-terminals; k-tuples (matrices) of linear production rules are then used for derivation, where the i-th non-terminal is expanded using the i-th rule; the sentential form after each derivation must have k or 0 non-terminals.

The proof itself has two main parts and the basic idea of each part will be presented. The first part is $\mathscr{L}_k(LSMG) \subseteq \mathscr{L}_k(L\#RS)$, where we construct a L#RS that simulates any given LSMG of the same degree. The basic idea is to store k non-terminals in a state in which the i-th non-terminal corresponds to the i-th # and generate a unique sequence of k rewriting rules in L#RS for each production rule of LSMG. The unique sequencing is achieved by storing the simulated production rule and current progress in the state. The second part is $\mathscr{L}_k(L\#RS) \subseteq \mathscr{L}_k(LSMG)$; its basic idea of simulation is to store the state of L#RS in the first non-terminal and store indices of corresponding #s in non-terminal symbols. The main complication is that #s are erased during rewriting, which decreases the index of following #s by one; for each rewriting rule, we construct production rules for all possible erasure sequences, which exponentially increases the amount of rules, but no more than one constructed rule for each rewriting rule is applicable at any given point.