5th tutorial (May 10, 2017)

Formale Baumsprachen

Task 10 (*yield(Rec) *and* CF*)

(a) Let $\Delta = \{(_a,)_a, (_b,)_b\}$. Give a context-free grammar that generates D_Δ (i.e. the Dyck language of well-bracketed strings).

(b) Let $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}, \gamma^{(0)}\}$ and G be an rtg over Σ with initial symbol Z and productions

$$Z \to \alpha + \beta + \gamma + \sigma(A, Z) + \sigma(Z, B) \quad A \to \alpha + \gamma + \sigma(A, A) \quad B \to \beta + \gamma + \sigma(B, B).$$

Describe the string languages $\mathsf{yield}_\alpha(L(G))$, $\mathsf{yield}_\beta(L(G))$, and $\mathsf{yield}_\gamma(L(G))$.

(c) Give an rtg H in normal form with $D_\Delta = \mathsf{yield}_e(L(H))$ (see Task 10 (a)) for some e.

(d) Using the construction from the lecture, transform the rtg G (see Task 10 (b)) into a context-free grammar G' such that $L(G') = \mathsf{yield}_\gamma(L(G))$. Is there a simpler context-free grammar generating the same language?

Task 11 (unranked tree automata and Haskell)

In the lecture we considered automata for ranked trees, i.e. trees where each terminal symbol has a fixed number of successors (the rank). In practice it is sometimes desirable for trees to allow an arbitrary number of successors for the terminal symbols, such trees are called unranked trees. In HTML, for example, it is possible for the <**ul**>-tag (unordered list) to have arbitrarily many successors of the form <**li**>…</**li**> (list item). Automata over unranked trees are called unranked tree automata.

(a) Give a definition of unranked tree automata and deterministic unranked tree automata.

(b) Devise a polymorphic data type DUTA q t in Haskell for deterministic unranked tree automata. The type variables q and t represent the states and terminal symbols, respectively.

(c) Implement a function recognize :: DUTA q t -> Tree t -> **Bool** that returns **True** if and only if the first parameter recognizes the second parameter.

(d) Test recognize.

(e) Devise a polymorphic data type NUTA q t in Haskell for unranked tree automata.

(f) Implement a function recognize' :: NUTA q t -> Tree t -> **Bool** that returns **True** if and only if the first parameter recognizes the second parameter.

(g) Test recognize'.

Note The tutorial's time might not suffice for presenting all solutions. Please prepare to ask for the solutions you are most interested in.