[class 11]

CHARACTERIZATION OF CFLs (not in book)

- Balanced parentheses language $\Sigma = \{(,)\}$

$x \in \Sigma^*$ is balanced if: (matching parentheses are well-nested)

i) for every prefix y of x: $\#(y) \geq \#)(y)$

ii) $\#(x) = \#)(x)$

It can be proved that the grammar (guess which?)

$G: S \to \varepsilon \mid (S) \mid SS$

generates exactly the language of all balanced parentheses!

- Generalized BPLs

A family of BPLs indexed by n, the number of types of parentheses: $\Sigma_n = \{[_1, [_2, \ldots, [_n,]_1,]_2, \ldots,]_n\}$

$G_n: \quad S \to \varepsilon \mid [_1 S]_1 \mid \ldots \mid [_n S]_n \mid SS$

Informally, $x \in \Sigma_n^*$ is balanced if matching parentheses are well-nested

statements in structured programming languages!

begin ... end

Actually, every CFL is a variation on the theme!!

THM (Chomsky – Schützenberger)

For every CFL $A \subseteq \Sigma^*$ there exist:

- $n \geq 1$
- regular language $R \subseteq \Sigma_n^*$
- homomorphism/renaming $h: \Sigma_n \to \Sigma^*$

such that:

$$A = h(L(G_n) \cap R)$$

Ex $\{a^n b^n \mid n \geq 0\} = h_{\substack{[_1 \mapsto a \\ ]_1 \mapsto b}}\left(L(G_1) \cap L([^*]^*)\right)$

Homework: Apply the Theorem to characterize the set of palindromes over $\{a,b\}$.

The theorem reveals the internal structure of CFLs!

This idea is at the bottom of structured programming, or structured document descriptions in DTD.

BPLs are easy to parse, and it is trivial to build DPDAs for them (if you take away ε) with 1 control state!

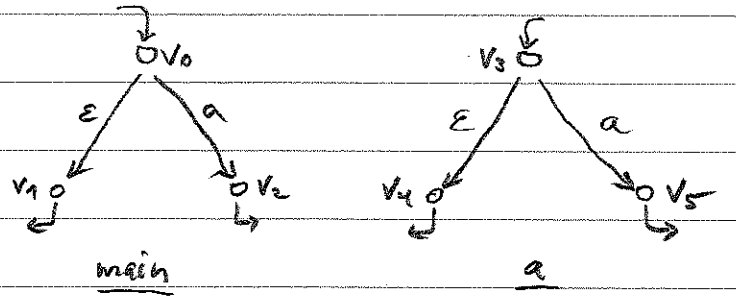
Later, in the light of PDTs and product construction ...

- unified productions for stack re-writing (push left pars, empty on right par)
- R corresponds to "finite control"

• Modelling Control Flow Behaviour of Procedural Programs

The program structure is captured by a flow graph:

F_{sys}:



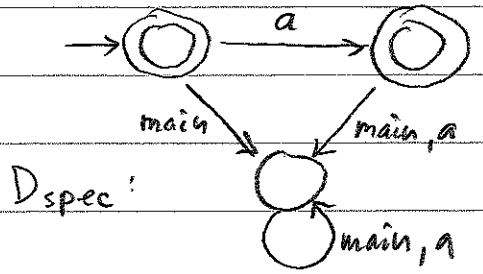
The behaviour of this program, capturing procedure calls and returns, can be modelled by a CFG $G_{sys} = (V, T, P, S)$

- $V = \{V_0, V_1, \ldots, V_5\}$ are the flowgraph nodes
- $T = \{main, a\}$ are the method names
- $S = V_0$ the entry node of method main
- P are productions induced by the flow graph
  - for every transfer edge
    $V_0 \to V_1 \qquad V_3 \to V_4$
  - for every call edge
    $V_0 \to a V_3 V_2 \qquad V_3 \to a V_3 V_5$
  - for every return node
    $V_1 \to \varepsilon \quad V_2 \to \varepsilon \quad V_4 \to \varepsilon \quad V_5 \to \varepsilon$

$L(G_{sys})$ consists of the terminating method invocation sequences, Program executions correspond to left-most derivations!

$$V_0 \underset{lm}{\Rightarrow} a V_3 V_2 \underset{lm}{\Rightarrow} \underbrace{aa}_{\text{method invoc. sequence}} \underbrace{V_3}_{\text{current control point}} \underbrace{V_5 V_2}_{\text{unresolved calls}} \underset{lm}{\Rightarrow} aa V_4 V_5 V_2 \underset{lm}{\Rightarrow} aa V_5 V_2 \underset{lm}{\Rightarrow} aa V_2 \underset{lm}{\Rightarrow} aa$$

Safety properties about safe sequencing of method calls can be specified with DFAs:



D_{spec}:

"the program is only allowed to do one method invocation, namely to a"

Can be checked (but for terminating executions only!) in the standard way:

$$L(G_{sys}) \subseteq L(D_{spec}) \Leftrightarrow L(G_{sys} \times \bar{D}_{spec}) = \emptyset$$

The product can even be defined directly for flow graphs!

CFG $F_{sys} \times \bar{D}_{spec}$

- Testing emptyness for CFLs

For $G = (V, T, P, S)$, $L(G)$ is non-empty iff S is generating.

The generating symbols of a CFG are defined inductively; see Section 7.1.2, p. 264

$S \to ab \mid aSA$
$A \to b$

- all terminals are generating
- X is generating if there is a production $X \to \alpha \in P$ such that all symbols in α are generating

- Reachable symbols: dead code detection!