Introduction to Language Theory and Compilation: Exercises

Session 6: Grammars revisited

Grammars: quick reminder

A grammar is described by four components $\langle V, T, P, S\rangle$ where:

- V is the set of variables
- T is the set of terminals
- P is the set of production rules

$$P \subseteq (V \cup T)^* V (V \cup T)^* \times (V \cup T)^*$$

- $S \in V$ is the start symbol

Useless symbols

- A symbol $X \in (V \cup T)$ is said to be useless if there doesn't exist a derivation of the form:

$$\underbrace{S \stackrel{*}{\Rightarrow}}_{\text{accessible}} wXy \underbrace{\stackrel{*}{\Rightarrow} wxy}_{\text{productive}}$$

- To remove useless symbols:
  - We remove unproductive symbols (i.e. from which strings of terminals cannot be derived)
  - We remove inaccessible symbols
- In that order!

Useless symbols – examples

- Grammar whose language is empty

$$\begin{array}{rcl} S & \to & aAb \\ A & \to & aA \\ & & bC \\ C & \to & Ab \end{array}$$

- Grammar with an *inaccessible* variable (C)

$$\begin{array}{rcl} S & \to & aAb \\ A & \to & aA \\ & & b \\ C & \to & bA \end{array}$$

Useless symbols – algorithms

Grammar `RemoveUnproductive`(**Grammar** $G = \langle V, T, P, S \rangle$) **begin**

$V_0 \leftarrow \emptyset$;

$i \leftarrow 0$;

repeat

$i \leftarrow i + 1$;

$V_i \leftarrow \{A \mid A \rightarrow \alpha \in P \wedge \alpha \in (V_{i-1} \cup T)^*\} \cup V_{i-1}$;

until $V_i = V_{i-1}$;

$V' \leftarrow V_i$;

$P' \leftarrow$ set of rules of P that do not contain variables in $V \setminus V'$;

`return`($G' = \langle V', T, P', S \rangle$) ;

Useless symbols – algorithms (ctd.)

Grammar `RemoveInaccessible`(**Grammar** $G = \langle V, T, P, S \rangle$) **begin**

$V_0 \leftarrow \{S\}$; $i \leftarrow 0$;

repeat

$i \leftarrow i + 1$;

$V_i \leftarrow \{X \mid \exists\, A \rightarrow \alpha X \beta \text{ in } P \wedge A \in V_{i-1}\} \cup V_{i-1}$;

until $V_i = V_{i-1}$;

$V' \leftarrow V_i \cap V$; $T' \leftarrow V_i \cap T$;

$P' \leftarrow$ set of rules of P that only contain variables from V_i ;

`return`$(G' = \langle V', T', P', S \rangle)$;

Useless symbols – algorithms (ctd.)

Grammar `RemoveUseless`(**Grammar** $G = \langle V, T, P, S\rangle$) **begin**
  Grammar $G_1 \leftarrow$ `RemoveUnproductive`(G) ;
  Grammar $G_2 \leftarrow$ `RemoveInaccessible`(G_1) ;
  `return`(G_2) ;

Exercise 1

Remove the useless symbols in the following two grammars:

a

$$
\begin{aligned}
S &\rightarrow a \mid A \\
A &\rightarrow AB \\
B &\rightarrow b
\end{aligned}
$$

b

$$
\begin{aligned}
S &\rightarrow A \\
  & B \\
A &\rightarrow aB \\
  & bS \\
  & b \\
B &\rightarrow AB \\
  & Ba \\
C &\rightarrow AS \\
  & b
\end{aligned}
$$

Solution for exercise 1.a

- The unproductive symbol removal algorithm stabilises with $V_i = \{S, B\}$ and we thus have:

$$G_1 = \langle\{S, B\}, \{a, b\}, \{S \to a, B \to b\}, S\rangle$$

- We notice B can't be accessed from S in this new grammar and can thus be removed. We then end up with:

$$G' = \langle\{S\}, \{a\}, \{S \to a\}, s\rangle$$

ULB

Solution for exercise 1.b

- Computational steps for V_i:

i	V_i
0	$\emptyset$
1	$\{C, A\}$
2	$\{C, A, S\}$
3	$\{C, A, S\}$

- We thus have the following P':

$$
\begin{array}{lcl}
S & \rightarrow & A \\
A & \rightarrow & bS \\
 & & b \\
C & \rightarrow & AS \\
 & & b
\end{array}
$$

Solution for exercise 1.b (ctd.)

- We can now remove the inaccessible symbols:

i	V_i
0	$\{S\}$
1	$\{S, A\}$
2	$\{S, A\}$

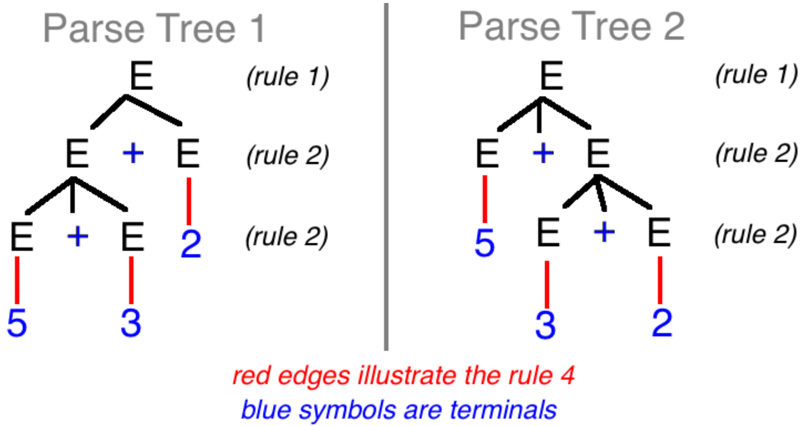
- We thus obtain $G' = \langle V', P', T', S' \rangle$ where:
  - $V' = \{S, A\}$
  - $P' = \{S \rightarrow A, A \rightarrow bS \mid b\}$
  - $T' = \{b\}$
  - $S' = S$

- A grammar G is said to be ambiguous if there exists a word $w \in L(G)$ such that there exists at least two different parse trees for w.
- Example of an ambiguous grammar:

$$\begin{array}{lll} E & \rightarrow & E + E \\ & & E * E \\ & & (E) \\ & & integer \end{array}$$

- In the above example, the word $5 + 3 + 2$ has more than one possible parse tree.

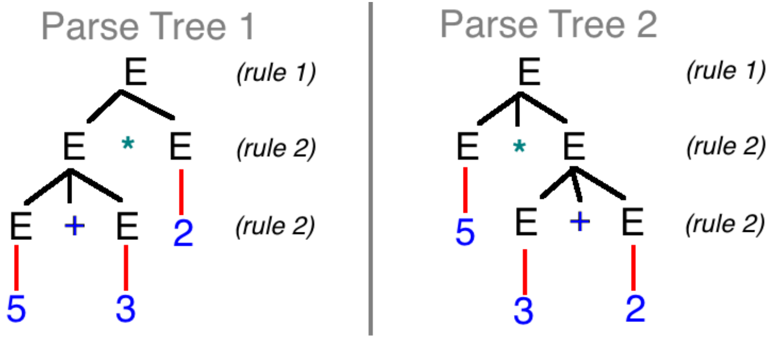
CFG transformations (ctd.)



CFG transformations (ctd.)

- Operator associativity: in the preceding example, the ambiguity arises from the fact that $+$ can be interpreted as left or right associative.
- Operator precedence: it can also be observed on the previous grammar that $*$ did not have precedence over $+$.

CFG transformations (ctd.)



red edges illustrate the rule 4
blue symbols are terminals

What happens if the first + change to a *?

Parse Tree 1	Parse Tree 2
5 + 3 = 8	3 + 2 = 5
8 * 2 = **16**	5 * 5 = **25**

Ambiguity removal

We can solve the associativity problem by transforming G into the following G' which forces left associativity:

$$
\begin{array}{rcl}
E & \rightarrow & E + T \\
 & & E * T \\
 & & T \\
T & \rightarrow & (E) \\
 & & nb
\end{array}
$$

Ambiguity removal (ctd.)

We can then force precedence for $*$ over $+$ by transforming G' into the following G'':

$$
\begin{array}{lcl}
E & \rightarrow & E + T \\
 & & T \\
T & \rightarrow & T * F \\
 & & F \\
F & \rightarrow & (E) \\
 & & nb
\end{array}
$$

Exercise 2

Consider the following grammar:

$$
\begin{array}{rcl}
E & \rightarrow & E \ op \ E \\
 & & ID[E] \\
 & & ID \\
op & \rightarrow & * \\
 & & / \\
 & & + \\
 & & - \\
 & & -> \\
\end{array}
$$

- Show that the grammar is ambiguous.
- The desired operator precedence is the following: $\{[], ->\} > \{*, /\} > \{+, -\}$. Transform the grammar so that it accounts for operator precedence and left associativity.

Solution for exercise 2

$$
\begin{array}{lcl}
E & \rightarrow & E+T \\
 & & E-T \\
 & & T \\
T & \rightarrow & T*F \\
 & & T/F \\
 & & F \\
F & \rightarrow & F->G \\
 & & ID[E] \\
 & & G \\
G & \rightarrow & ID
\end{array}
$$

Left factoring

- Left factoring aims to remove rules that share a common prefix since they make life difficult for predictive parsing.
- Sometimes, left factoring is enough to turn a given CFG into an LL(1) grammar!
- Example :

$$S \rightarrow ab \mid aa$$

is not LL(1). After left factoring, we get:

$$\begin{array}{rcl} S & \rightarrow & aN \\ N & \rightarrow & a \mid b \end{array}$$

which is LL(1)!

ULB

Left factoring – algorithm

`LeftFactor`(**Grammar** $G = \langle V, T, P, S \rangle$) **begin**

while *G has at least two rules with the same left-hand side and a common prefix* **do**

Let $E = \{A \rightarrow \alpha\beta, \ldots, A \rightarrow \alpha\zeta\}$ be such a set of rules ;

Let $\mathcal{V}$ be a new variable;

$V = V \cup \mathcal{V}$;

$P = P \setminus E$;

$P = P \cup \{A \rightarrow \alpha\mathcal{V}, \mathcal{V} \rightarrow \beta, \ldots, \mathcal{V} \rightarrow \zeta\}$;

Exercise 3

Left-factor the following production rules:

<stmt> $\rightarrow$ **if** <expr> **then** <stmt-list> **end if**

<stmt> $\rightarrow$ **if** <expr> **then** <stmt-list> **else** <stmt-list> **end if**

Solution for exercise 3

<stmt>	$\rightarrow$	**if** <expr> **then** <stmt-list> <if-tail>
<if-tail>	$\rightarrow$	**end if**
<if-tail>	$\rightarrow$	**else** <stmt-list> **end if**

Left recursion

- For the same reasons, it is often useful to suppress left (or right) recursion in a given grammar.
- The following grammar is left recursive:

$$S \rightarrow S\alpha \mid \beta$$

- It can be transformed into the equivalent (but not left recursive) grammar:

$$\begin{array}{lcl} S & \rightarrow & \mathcal{V}\mathcal{T} \\ \mathcal{V} & \rightarrow & \beta \\ \mathcal{T} & \rightarrow & \alpha\mathcal{T} \mid \varepsilon \end{array}$$

Left recursion removal algorithm

```
RemoveLeftRecursion(Grammar G = ⟨V, T, P, S⟩) begin
    while G contains a left recursive variable A do
        Let E = {A → Aα, A → β, ..., A → ζ} be the set of rules that
        have A as left-hand side ;
        Let 𝒰 and 𝒱 be two new variables ;
        V = V ∪ {𝒰, 𝒱} ;
        P = P \ E ;
        P = P ∪ {A → 𝒰𝒱, 𝒰 → β, ..., 𝒰 → ζ, 𝒱 → α𝒱, 𝒱 → ε} ;
```

Exercise 4

Apply the left recursion removal algorithm to the following grammar:

$$
\begin{array}{rcl}
E & \rightarrow & E + T \\
 & & T \\
T & \rightarrow & T * P \\
 & & P \\
P & \rightarrow & ID
\end{array}
$$

Solution for exercise 4

$$
\begin{array}{lcl}
E & \rightarrow & AB \\
A & \rightarrow & T \\
B & \rightarrow & +TB \\
 & & \varepsilon \\
T & \rightarrow & CD \\
C & \rightarrow & P \\
D & \rightarrow & *PD \\
 & & \varepsilon \\
P & \rightarrow & ID
\end{array}
$$

Exercise 5 – former exam question

Transform the following grammar into an LL(1) grammar:

$$
\begin{array}{lcl}
S & \rightarrow & aE \mid bF \\
E & \rightarrow & bE \mid \epsilon \\
F & \rightarrow & aF \mid aG \mid aHD \\
G & \rightarrow & Gc \mid d \\
H & \rightarrow & Ca \\
C & \rightarrow & Hb \\
D & \rightarrow & ab
\end{array}
$$

ULB

Solution for exercise 5 – useless symbols

1. Remove unproductive symbols: H and C are both unproductive as they are mutually recursive and that they cannot produce anything useful. We can thus remove rules $H \to Ca$, $C \to Hb$ and $F \to aHD$.
2. Remove inaccessible symbols: by removing rule $F \to aHD$, D became inaccessible. We can thus remove $D \to ab$.

Solution for exercise 5 – left recursion and factoring

So far, we have:

$$\begin{array}{lcl} S & \rightarrow & aE \mid bF \\ E & \rightarrow & bE \mid \varepsilon \\ F & \rightarrow & aF \mid aG \\ G & \rightarrow & Gc \mid d \end{array}$$

3. Left recursion removal: $G \rightarrow Gc$ is left recursive. We replace $G \rightarrow Gc \mid d$ with $G \rightarrow dG'$ and $G' \rightarrow cG' \mid \varepsilon$.
4. Left factoring: we replace $F \rightarrow aF \mid aG$ with $F \rightarrow aF'$ and $F' \rightarrow F \mid G$.

Solution for exercise 5 – LL(1) ?

We can now check whether our final grammar is LL(1) by building the corresponding action table and verifying that no conflicts arise.

$$
\begin{array}{rrcl}
(0) & S' & \rightarrow & S\$ \\
(1,2) & S & \rightarrow & aE \mid bF \\
(3,4) & E & \rightarrow & bE \mid \varepsilon \\
(5) & F & \rightarrow & aF' \\
(6,7) & F' & \rightarrow & F \mid G \\
(8) & G & \rightarrow & dG' \\
(9,10) & G' & \rightarrow & cG' \mid \varepsilon
\end{array}
$$

	a	b	c	d	$\$$
S'	P0	P0	×	×	×
S	P1	P2	×	×	×
E	×	P3	×	×	P4
F	P5	×	×	×	×
F'	P6	×	×	P7	×
G	×	×	×	P8	×
G'	×	×	P9	×	P10

... and we're done!