CS3133 Foundations of Computer Science HW4

Keith DeSantis

9/28/21

Symbols for Convenience: ∅ ≠ Σ λ ⊆ ⊄ ϵ δ ≡ ≢ ∉ ∃
Collaboration with Kush Shah and Samuel Parks

186 MakeMoreCFGs)

b) $a^n b^m c^m d^{2n}$ | n,m are natural numbers

S → aSdd | K | λ
K → bKc | λ

Explanation:

K clearly generates any string that looks like $b^m c^m$.
S generates strings of the form $a^n K d^{2n}$.
By combining them in this way we get the $a^n b^m c^m d^{2n}$, as we wanted.

c) $a^n b^m$ | 0 <= n <= m <= 2n

S → aSb | aSbb | λ

Explanation:

S_1 → aSb | λ clearly represents $a^n b^n$
S_2 → aSbb | λ clearly represents $a^m b^{2m}$
By having S → aSb | aSbb | λ we get $a^n b^x$ where n <= x <= 2n, since at the least number of b's possible is the same number of a's (if we choose aSb repeatedly), while the most number of b's possible is twice the numbers of a's (if we choose aSbb repeatedly).

d) $a^m b^n c^k$ where m=n or m=k

S → S_1 | S_2
S_1 → aS_1bC | λ
C → cC | λ
S_2 → aS_2c | B | λ
B → bB | λ

Explanation:

This is because, by theorem, the union of L(M_1) and L(M_2) = L(M), where M is out desired CFG, M_1 is a CFG for $a^m b^m c^k$ and M_2 is a CFG for $a^m b^n c^m$.

M_1:
$S_1 \rightarrow aS_1bC \mid \lambda$
$C \rightarrow cC \mid \lambda$

C denotes c^k, meaning S_1 denotes $a^m b^m c^k$

M_2:
$S_2 \rightarrow aS_2c \mid B \mid \lambda$
$B \rightarrow bB \mid \lambda$

B denotes b^n, meaning S_2 denotes $a^m b^k c^m$

Therefore, the union of these two gives our desired CFG.

e) $\{ a^i b^i c^k d^k \mid i,k >= 0 \}$

$S \rightarrow MK$
$M \rightarrow aMb \mid \lambda$
$K \rightarrow cKd \mid \lambda$

Explanation:

Clearly M denotes $a^i b^i$.
Clearly K denotes $c^k d^k$.
Therefore MK denotes $a^i b^i c^k d^k$.

f) $\{ a^i b^j c^k d^m \mid i,j,k,m >= 0$, and $(i = j$ or $k = m) \}$

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow KCD$
$K \rightarrow aKb \mid \lambda$
$C \rightarrow cC \mid \lambda$
$D \rightarrow dD \mid \lambda$

$S_2 \rightarrow ABN$
$A \rightarrow aA \mid \lambda$
$B \rightarrow bB \mid \lambda$
$N \rightarrow cNd \mid \lambda$

Explanation:

Similarly to part d, this is because when we have the union of two CFL's that results in our desired CFL ($L(S_1)$ and $L(S_2) = L(S)$), our desired CFG can be constructed as $S \rightarrow S_1 \mid S_2$.

$L(S_1)$ is all strings $a^i b^i c^k d^m \mid i,k,m >= 0$
$L(S_2)$ is all strings $a^i b^j c^k d^k \mid i,j,k >= 0$
Therefore, $L(S_1) \cup L(S_2) = L(S)$.

Explanations for S_1 and S_2 –

S_1:
C denotes c^k
D denotes d^m
K denotes $a^i b^i$
Therefore, S_1 denotes $a^i b^i c^k d^m$

S_2:
A denotes a^i
B denotes b^j
N denotes $c^k d^k$
Therefore, S_2 denotes $a^i b^j c^k d^k$

Therefore $L(S_1) \cup L(S_2) = L(S)$.

199 ArithAmb1)

G:
$E \rightarrow E+E \mid E*E \mid I$
$I \rightarrow a \mid b \mid c$

G*:
$E \rightarrow T+E \mid T$
$T \rightarrow I*T \mid I$
$I \rightarrow a \mid b \mid c$

a)
G*:
$E \rightarrow E+T \mid T$
$T \rightarrow T*I \mid I$
$I \rightarrow a \mid b \mid c$

b)
G*:
$E \rightarrow E+T \mid E-T \mid T$
$T \rightarrow T*I \mid I$
$I \rightarrow a \mid b \mid c$

c)
G*:
E → E +T | E - T | T
T → T*I | I
I → K↑I | K
K → a | b | c

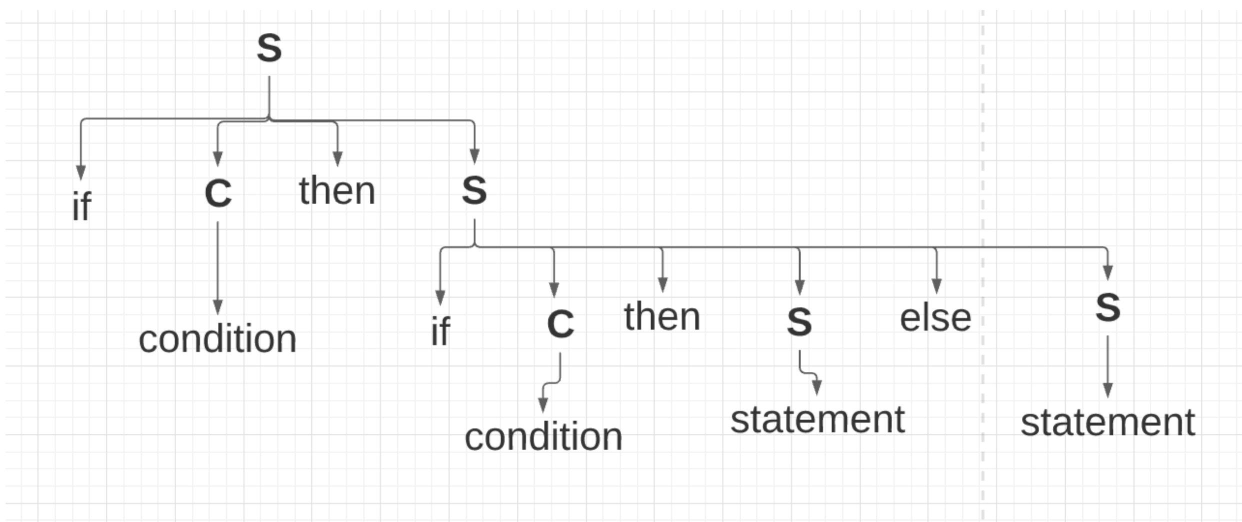
d)
S → S = E | S < E | E
E → E +T | E - T | T
T → T*I | I
I → K↑I | K
K → a | b | c

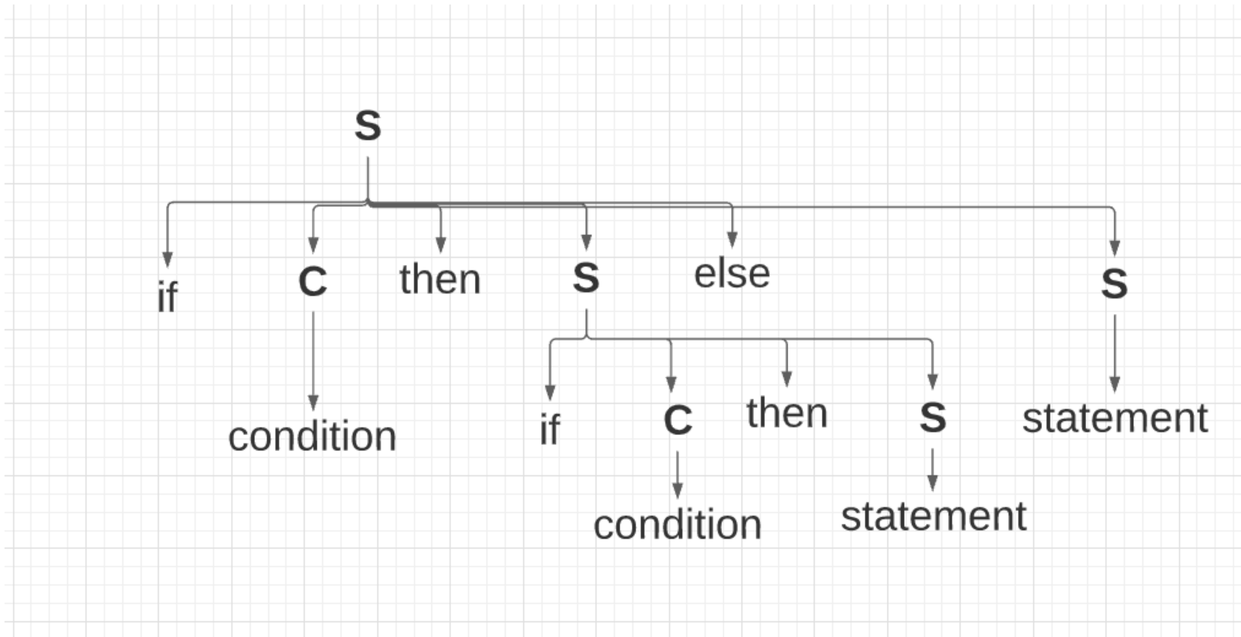
PLAmb)

Σ = { if, then, else, statement, condition }

S → if C then S | if C then S else S | statement
C → condition

1)





Here are two parse trees that are different but give the same string, that string being "if condition then if condition then statement else statement". This ambiguity arose from some kind of precedence ambiguity related to the if, then, and else operators.

2)

The string:

"if condition1 then if condition2 then statement1 else statement2"

Could be executed different depending on which "if" operator the ending "else" is connected to (the key parenthesis in the following examples are highlighted red). For example, if it were to be read as:

if (condition1) then (if (condition2) then (statement1)) else (statement2)

Here the else is connected to the first if, so if condition1 is false, statement2 is executed.

It could also be read as:

if (condition1) then (if (condition2) then (statement1) else (statement2))

Here the else is connected to the second if, so statement2 is only executed if condition1 is true AND condition2 is false, while before statement2 had to dependency on condition2.

In code it would take these two forms:

```
if (condition1) {
	if (condition2) {
		statement1;
	}
} else {
	statement2;
}

	VS

if (condition1) {
	if (condition2) {
		statement1;
	} else {
		statement2;
	}
}
```

3)

Σ = { if, then, else, statement, condition }

S → if C then S | if C then T else S | statement
T → if C then T else T | statement
C → condition

This grammar construction ensures that any statement that is within an "if_else" statement MUST either be "statement" or contain an "else." This removes the possibility of dangling else's and allows the strings to be built from the inside out in a way.

ElimUseless)

Starting Grammar:

S → dS | A | C
A → aA | a
B → bB | b
C → cC

Rewritten Grammar:

$S \rightarrow dS \mid aS \mid a$

GNFPractice)

$E \rightarrow T + E \mid T$
$T \rightarrow I * T \mid I$
$I \rightarrow 0 \mid 1$

Transform to GNF:

Step 1:
$E \rightarrow T + E \mid T$
$T \rightarrow 0 * T \mid 1 * T \mid 0 \mid 1$

Step 2:
$E \rightarrow 0 * T + E \mid 1 * T + E \mid 0 + E \mid 1 + E \mid 0 * T \mid 1 * T \mid 0 \mid 1$
$T \rightarrow 0 * T \mid 1 * T \mid 0 \mid 1$

The CFG is now in GNF.