Compiler - Exam

Task 1 - Grammar transformation for LL(1)

We have the following grammar $G_{T_{UP}}$ with the 4 terminals: id ( ) ,

$$\begin{aligned} S &\rightarrow id \\ S &\rightarrow (\ T\ ) \\ T &\rightarrow S\ ,\ T \\ T &\rightarrow S \end{aligned}$$

for tuple expressions for LL(1) parsing.

a

Give a short reason why this grammar is not LL(1), and transform the grammar (using a well-known transformation) to obtain a grammar $G'_{T_{UP}}$ suitable for LL(1) parsing.

To make the grammer LL(1), we want to

1. Eliminate ambiguity
2. Eliminate left-recursion
3. Perform left factorisation where required

Since the two productions for T begins with the same symbol, this means they have overlapping *FIRST* sets and is not LL(1). Therefore we need to left-factor T by making the a simple production with the same common prefix, S and creating another nonterminal, so we get the following grammar $G'_{T_{UP}}$

$$\begin{aligned} S &\rightarrow id \\ S &\rightarrow (\ T\ ) \\ T &\rightarrow S\ R \\ R &\rightarrow\ ,\ T \\ R &\rightarrow \varepsilon \end{aligned}$$

b

For $G'_{T_{UP}}$, determine $FIRST$ sets for all right-hand sides.

We calculate these through fixed-point iteration using the rules from **Algorithm 2.5**[1].

Right-hand side	Initialisation	Iteration 1	Iteration 2	Iteration 3
id	∅	{id}	{id}	{id}
(T)	∅	{ (}	{ (}	{ (}
S R	∅	∅	{id, (}	{id, (}
, T	∅	{ , }	{ , }	{ , }
ε	∅	∅	∅	∅

And then we reach a fixed point.
So we have the following $FIRST$ sets

$$\begin{aligned} FIRST(id) &= \{id\} \\ FIRST(\,(\,T\,)\,) &= \{\,(\,\} \\ FIRST(S\ R) &= \{id,\ (\,\} \\ FIRST(,\ T) &= \{\ ,\ \} \\ FIRST(\varepsilon) &= \emptyset \end{aligned}$$

c

Add a start production $S' \rightarrow S\$$ and determine $FOLLOW$ sets for all non-terminals.

This yields the grammar

$$\begin{aligned} S' &\rightarrow S\$ \\ S &\rightarrow id \\ S &\rightarrow (\ T\ ) \\ T &\rightarrow S\ R \\ R &\rightarrow ,\ T \\ R &\rightarrow \varepsilon \end{aligned}$$

[1]Torben Ægidius Mogensen, Introduction to Compiler Design, page 55

To calculate the *FOLLOW* sets, we follow an algorithm[2] and get the following constraints

For the production $S' \rightarrow S\$$ we add the following constraint since '\$' is seen as a terminal and is therefore the *FIRST* set of what follows S but '\$' is not nullable.

$$\{\$\} \subseteq \{S\}$$

For the production $S \rightarrow id$ there are no nonterminals, so no constraints are added.

For the production $S \rightarrow (\ T\ )$ we add the following constraint since ')' is the *FIRST* set of what follows T but is not nullable.

$$\{\ )\ \} \subseteq \{T\}$$

For the production $T \rightarrow S\ R$ we make a split for each occurence. Looking at the nonterminal S we get the constraint

$$\{\ ,\ \} \subseteq \{S\}$$

since it is the *FIRST* set of R. Furthermore, since R is nullable, we add the following constraint

$$\{T\} \subseteq \{S\}$$

Now looking at R there is nothing in the *FIRST* set, so no constraint is added. However, that nothing follows means that it is nullable, so we add the constraint

$$\{T\} \subseteq \{R\}$$

For the production $R \rightarrow , T$ we get nothing in the *FIRST* set for what follows T meaning it is the same case as above so we add the following constraint

$$\{R\} \subseteq \{T\}$$

For the last production $R \rightarrow \varepsilon$ there are no nonterminals, so no constraints are added.

[2] Torben Ægidius Mogensen, Introduction to Compiler Design, page 59

Now we need to solve these constraints. For each constraint on the form *terminal* $\subseteq$ *nonterminal* we put those into $FOLLOW(nonterminal)$ so we get

$$FOLLOW(S) = \{\$, ',' \}$$
$$FOLLOW(T) = \{\ )\ \}$$
$$FOLLOW(R) = \emptyset$$

Now for each constraint on the form *nonterminal* $\subseteq$ *nonterminal* we add the content from the $FOLLOW$ set of the first nonterminal to the second until a fixed point is reached and we get

$$FOLLOW(S) = \{\$, ',' \ , \ )\}$$
$$FOLLOW(T) = \{\ )\ \}$$
$$FOLLOW(R) = \{\ )\ \}$$

Whereas a fixed point is reached and we have our $FOLLOW$ sets.

d

Determine the look-ahead sets for all productions and point out that $G'_{T_{UP}}$ is LL(1).

Having calculated $FIRST$ and $FOLLOW$ we can use the following rules to determine look-ahead (LA) sets.

$$LA(X \rightarrow \alpha) = \begin{cases} FIRST(\alpha) \cup FOLLOW(X) & \text{if NULLABLE(X)} \\ FIRST(\alpha) & \text{otherwise} \end{cases}$$

So we get the following look ahead sets (disregarding the added start production)

$$LA(S \rightarrow id) = \{id\}$$

Since the right hand side is not nullable.

$$LA(S \rightarrow (\ T\ )) = \{\ (\ \}$$

Since the right hand side is not nullable.

$$LA(T \rightarrow S\ R) = \{id,\ (\}$$

Since the right hand side is not nullable.

$$LA(R \rightarrow \text{,}\ T) = \{\ \text{,}\ \}$$

Since the right hand side is not nullable.

$$LA(R \rightarrow \varepsilon) = \{\ )\ \}$$

Since the right hand side is nullable.

Since for each nonterminal X in the grammar, the look ahead sets for each production for this nonterminal are disjoint, it means the grammar is LL(1).

Task 2 - Extend the equality operation in Paladim to work on arrays