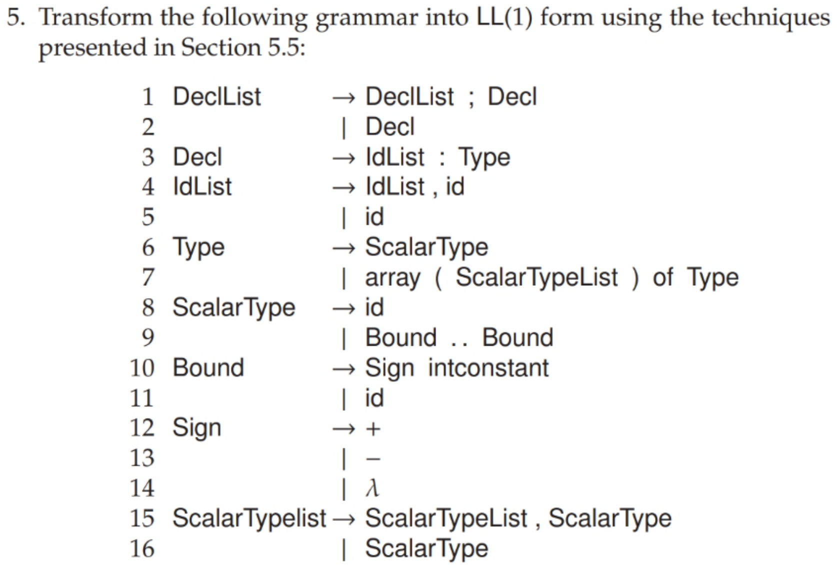
Gabriel Arnell

Lab 9

Crafting a Compiler 5.5



5. Transform the following grammar into LL(1) form using the techniques presented in Section 5.5:

```
 1 DeclList        → DeclList ; Decl
 2                 | Decl
 3 Decl            → IdList : Type
 4 IdList          → IdList , id
 5                 | id
 6 Type            → ScalarType
 7                 | array ( ScalarTypeList ) of Type
 8 ScalarType      → id
 9                 | Bound .. Bound
10 Bound           → Sign intconstant
11                 | id
12 Sign            → +
13                 | −
14                 | λ
15 ScalarTypelist  → ScalarTypeList , ScalarType
16                 | ScalarType
```

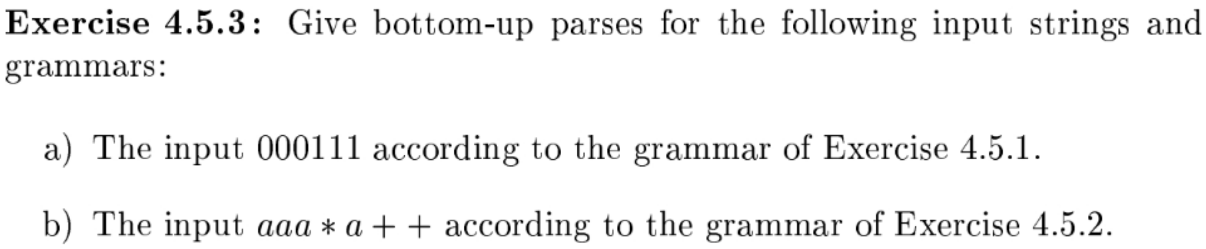
Issues:

- If parsed LL(1) style, the DeclList will always break after it parses the final Decl and expects 1 more after it as per the DeclList productions. This can be fixed by parsing the first Decl as a Decl instead of a first set to a DeclList.
- Two productions of IdList involve have a first set of {id}
    - We can fix this by making the **IdList, id** flip into an **id, idList** production and have the other production be empty.
- Scalar type has a production for bound, and a production for id that both can start with id
    - We should remove the Id production in Scalar type and let it just use Bound as the id. We then need to create a second production set to consider that ScalarType could be a single **id** or a **id .. Bound**.
- ScalarTypeList has two productions that have first set of {id} due to it only being able to reference itself. We can fix this by having its productions be **ScalarType, ScalarTypeList** and ***empty***

DeclList	Decl; DeclList *empty*
Decl	IdList : Type
IdList	id, idList *empty*
Type	ScalarType array (ScalarTypeList) of Type

ScalarType	Id EndBound Sign intconstant .. Bound
Bound	Id Sign intconstant
EndBound	.. Bound *empty*
Sign	+ - λ
ScalarTypelist	ScalarType, ScalarTypeList *empty*

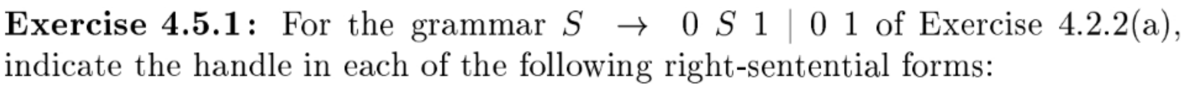
Dragon 4.5.3



Exercise 4.5.3: Give bottom-up parses for the following input strings and grammars:

a) The input 000111 according to the grammar of Exercise 4.5.1.

b) The input $aaa * a + +$ according to the grammar of Exercise 4.5.2.

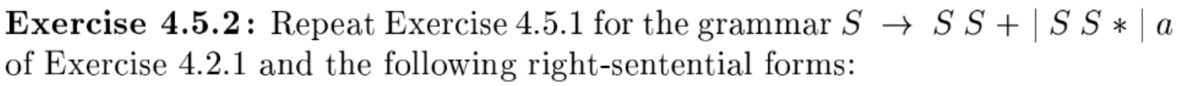


Exercise 4.5.1: For the grammar $S \rightarrow 0\ S\ 1 \mid 0\ 1$ of Exercise 4.2.2(a), indicate the handle in each of the following right-sentential forms:

Grammar: S -> 0 S 1
-> 0 1
Input: 000111

Steps:
Stack: 0
Nothing can be replaced.
Stack: 00
No replace
Stack: 000
No replace:
Stack 0001
We replace 01 with S
Stack: 00S(01)
No replace
Stack: 00S(01)1

We can replace 0S1 with S
Stack: 0S(0S(01)1)1
We can replace with S
Stack: S(0S(0S(01)1)1)
We only have S on the stack, which is the goal production so we have completed it.



Exercise 4.5.2: Repeat Exercise 4.5.1 for the grammar $S \to S\,S + \mid S\,S * \mid a$ of Exercise 4.2.1 and the following right-sentential forms:

Grammar:
S -> S S +
  -> S S *
  -> a
Input:
aaa*a++
Stack: a
We replace a with S
Stack: S
Can't replace, let's shift
Stack: Sa
Replace a with S
Stack: SS
No replace
Stack: SSa
Replace a with S
Stack: S S S
No replace, shift the *
SSS*
Replace SS* with S
Stack: SS
Shift the a
Stack: SSa
Replace the A with an S
Stack: SSS
Shift the +
Stack: SSS+
Replace SS+ with S
Stack: SS
Shift the +
Stack: SS+
Replace SS+ with S
Stack: S
S is the goal production, there are no terminals left and nothing on stack. Compilation is complete.