Individual Exercises - Lecture 8

1. (optional - but recommended) Follow the studio associated with Crafting a Compiler Chapter 6 http://www.cs.wustl.edu/~cytron/cacweb/Chapters/6/studio.shtml
   For this a virtual machine will be provided with the necessary tools pre installed.

2. Use **CUP** to do Fischer et al exercises 40, 41, 42, 43 on pages 224-233 (exercise 40, 42, 44, 43 on pages 263-264 in GE)
   The virtual machine available on Moodle has CUP and SableCC installed, along with a pdf guide that details how to use them!

   If you want to run CUP directly on your own machine:
   Download Cup -> Extract Jar file to desktop -> create grammar.cup file on desktop
   (optional - download intellij highlighting plugin for cup files)
   Recreate grammar from exercises in grammar.cup file
   Run in cmd :
   cd Desktop
   java -jar java-cup-11b.jar grammar.cup
   See output for answer to exercises

   Example grammar: http://www2.cs.tum.edu/projects/cup/docs.php#intro
   40. Recall the dangling else problem introduced in Chapter 5. A grammar for a simplified language that allows conditional statements follows:

```
1  Start → Stmt $
2  Stmt → if e then Stmt else Stmt
3        | if e then Stmt
4        | other
```

   (a) Explain why the grammar is or is not LALR(1).

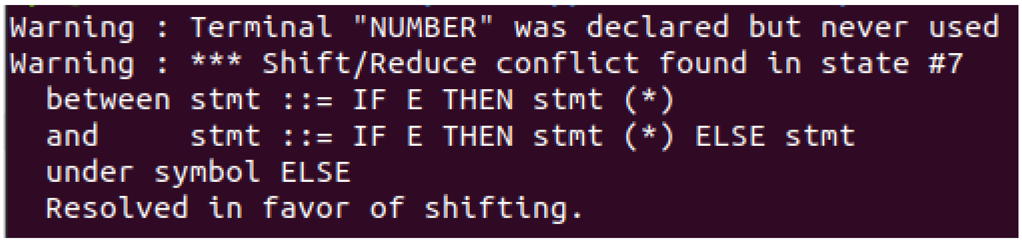
   First we create a grammar.cup file with the following contents:

```
terminal IF, THEN, ELSE, E, OTHER;
terminal Integer NUMBER;

non terminal program, stmt;

program ::= stmt;
stmt ::= IF E THEN stmt ELSE stmt
       | IF E THEN stmt
       | OTHER;
```

Then we run the JavaCup parser generator with this input. The output warns us that there was a shift reduce conflict, but that it was automatically resolved by favoring shift:



```
Warning : Terminal "NUMBER" was declared but never used
Warning : *** Shift/Reduce conflict found in state #7
  between stmt ::= IF E THEN stmt (*)
  and     stmt ::= IF E THEN stmt (*) ELSE stmt
  under symbol ELSE
  Resolved in favor of shifting.
```

From this we can conclude that the grammar is not LALR(1).

41. Consider the following grammar

1	Start	→	Stmt $
2	Stmt	→	Matched
3		|	Unmatched
4	Matched	→	if e then Matched else Matched
5		|	other
6	Unmatched	→	if e then Matched else Unmatched
7		|	if e then Unmatched

(a) Explain why the grammar is or is not LALR(1)

First we create a grammar.cup file to represent the grammar:

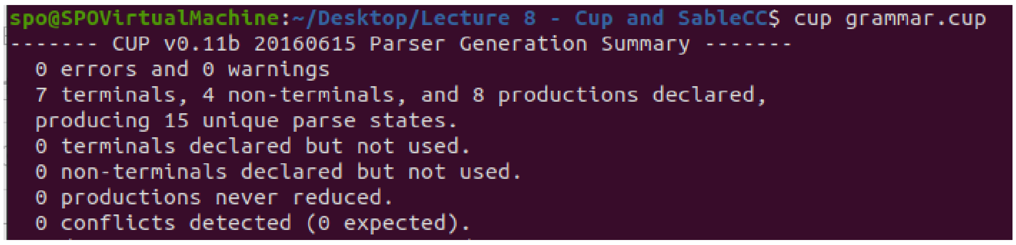
```
terminal IF, THEN, ELSE, E, OTHER;

non terminal program, stmt, matched, unmatched;

program ::= stmt;
stmt    ::= matched | unmatched;

matched   ::= IF E THEN matched ELSE matched
            | OTHER;
unmatched ::= IF E THEN matched ELSE unmatched
            | IF E THEN unmatched;
```

Then we run the CUP parser generator with this grammar:



```
spo@SPOVirtualMachine:~/Desktop/Lecture 8 - Cup and SableCC$ cup grammar.cup
------- CUP v0.11b 20160615 Parser Generation Summary -------
  0 errors and 0 warnings
  7 terminals, 4 non-terminals, and 8 productions declared,
  producing 15 unique parse states.
  0 terminals declared but not used.
  0 non-terminals declared but not used.
  0 productions never reduced.
  0 conflicts detected (0 expected).
```

This does not yield any errors, warnings or conflicts, which means that we have successfully generated a LALR(1) parser for this grammar, thereby proving that the language is LALR(1).

(b) Is the language of this grammar the same as the language of the grammar in Exercise 40? Why or why not?

Recall that the problem of determining whether two given grammars generate the same language is undecidable! This means that the tool cannot have an algorithm that answers this question for all pairs of grammars. It should, however, be noted that in special cases it is possible to determine if two grammars describe the same language.

For this specific example, we can reason that these two grammars do not generate the same language:

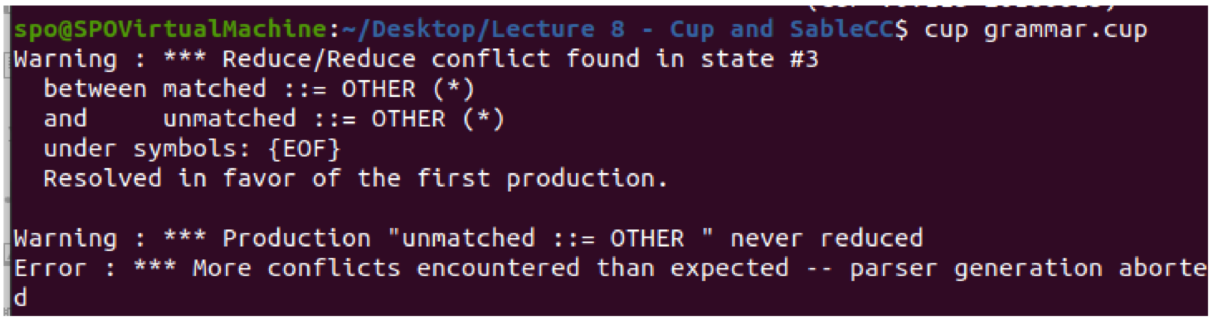
Looking at the grammar from exercise 41 we see that using any of the 'Unmatched' production rules causes infinite recursion. This means that the grammar in exercise 41 cannot produce the string "IF e THEN other" while the grammar from exercise 40 can.

42. Repeat Exercise 41, adding the production Unmatched→other to the grammar.

1	Start	→	Stmt $
2	Stmt	→	Matched
3		|	Unmatched
4	Matched	→	if e then Matched else Matched
5		|	other
6	Unmatched	→	if e then Matched else Unmatched
7		|	if e then Unmatched
8		|	other

(a) Explain why the grammar is or is not LALR(1)

Adding the new production to the grammar we now get an error. CUP is telling us that we get a shift reduce conflict, because of the terminal 'OTHER', meaning the grammar is no longer LALR(1)



```
spo@SPOVirtualMachine:~/Desktop/Lecture 8 - Cup and SableCC$ cup grammar.cup
Warning : *** Reduce/Reduce conflict found in state #3
  between matched ::= OTHER (*)
  and     unmatched ::= OTHER (*)
  under symbols: {EOF}
  Resolved in favor of the first production.

Warning : *** Production "unmatched ::= OTHER " never reduced
Error : *** More conflicts encountered than expected -- parser generation aborte
d
```

(b) Is the language of this grammar the same as the language of the grammar in Exercise 40? Why or why not?

Again, the tool cannot help us determine the equality of the languages described by the grammars. Looking at the grammars, it intuitively seems like they are generating the same strings.

43. Consider the following grammar:

```
1  Start      → Stmt $
2  Stmt       → Matched
3             | Unmatched
4  Matched    → if e then Matched else Matched
5             | other
6  Unmatched  → if e then Matched else Unmatched
7             | if e then Stmt
```

(a) Explain why the grammar is or is not LALR(1)

Creating the grammar.cup file:

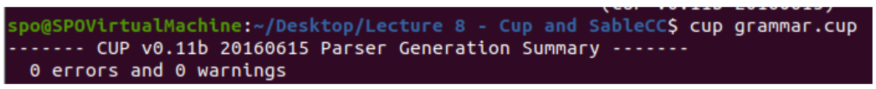
```
terminal IF, THEN, ELSE, E, OTHER;

non terminal program, stmt, matched, unmatched;

program ::= stmt;
stmt    ::= matched | unmatched;

matched   ::= IF E THEN matched ELSE matched
              | OTHER;
unmatched ::= IF E THEN matched ELSE unmatched
              | IF E THEN stmt;
```

Running cup on it we get a successfully generated parser:



```
spo@SPOVirtualMachine:~/Desktop/Lecture 8 - Cup and SableCC$ cup grammar.cup
------- CUP v0.11b 20160615 Parser Generation Summary -------
  0 errors and 0 warnings
```

...meaning the grammar is LALR(1).

(b) Is the language of this grammar the same as the language of the grammar in Exercise 40? Why or why not?

Again, the tool cannot help us determine the equality of the languages described by the grammars. Looking at the grammars, it intuitively seems like they are generating the same strings.

3. Familiarize yourself with SableCC by browsing their web pages
   Visit here: https://sablecc.org
   A virtual machine with SableCC installed can be found on moodle along with a guide for running it.

4. Use **SableCC** to do Fischer et al exercises 40, 41, 42, 43 on pages 224-233 (exercise 40, 42, 44, 43 on pages 263-264)
   The virtual machine available on Moodle has CUP and SableCC installed, along with a pdf guide that details how to use them!

40. Recall the dangling else problem introduced in Chapter 5. A grammar for a simplified language that allows conditional statements follows:

```
1  Start → Stmt $
2  Stmt → if e then Stmt else Stmt
3        | if e then Stmt
4        | other
```

Explain why the grammar is or is not LALR(1).

We create the following SableCC grammar:

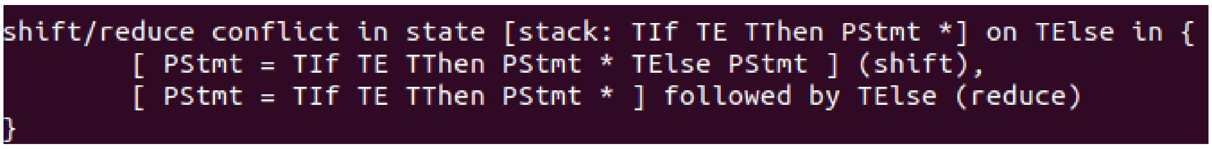
```
Helpers
  tab   = 9;

Tokens
  e    = 'exp';
  other = 'other';
  if = 'if';
  then = 'then';
  else = 'else';
  blank = ' ' | tab;

Ignored Tokens
  blank, eol;

Productions
start = stmt;
stmt = {one}   if e then [fst]:stmt else [snd]:stmt
     | {two}   if e then stmt
     | {three} other;
```

When trying to generate a parser with SableCC we get:



```
shift/reduce conflict in state [stack: TIf TE TThen PStmt *] on TElse in {
        [ PStmt = TIf TE TThen PStmt * TElse PStmt ] (shift),
        [ PStmt = TIf TE TThen PStmt * ] followed by TElse (reduce)
}
```

Meaning we have shift/reduce conflict and the grammar is not LALR(1).

41. Consider the following grammar

1	Start	→	Stmt $
2	Stmt	→	Matched
3		|	Unmatched
4	Matched	→	if e then Matched else Matched
5		|	other
6	Unmatched	→	if e then Matched else Unmatched
7		|	if e then Unmatched

(a) Explain why the grammar is or is not LALR(1)

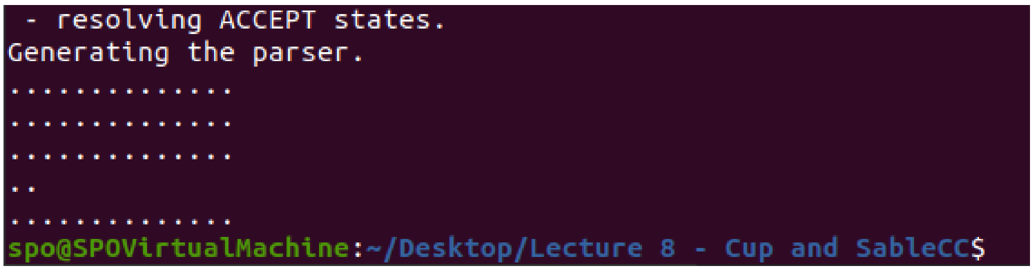
We use a slightly modified version of the previous grammar:

```
...
Productions
start = stmt;
stmt = {one} matched
     | {two} unmatched;

matched = {one} if e then [first]:matched else [second]:matched
        | {two} other;

unmatched = {one} if e then matched else unmatched
          | {two} if e then unmatched;
```

and run SableCC on it:



```
 - resolving ACCEPT states.
Generating the parser.
..............
..............
..............
..
..............
spo@SPOVirtualMachine:~/Desktop/Lecture 8 - Cup and SableCC$
```

No errors, we now have a LALR(1) parser based on the grammar.

(b) Is the language of this grammar the same as the language of the grammar in Exercise 40? Why or why not?

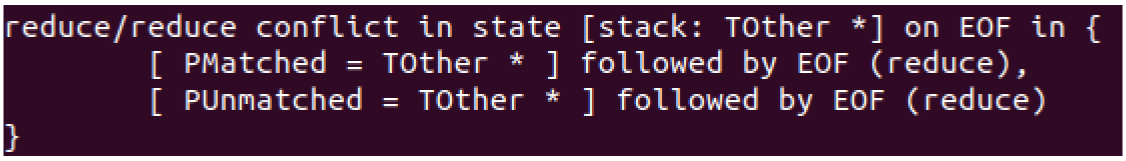
See the answer to the Cup version of this exercise (found earlier in this document).

42. Repeat Exercise 41, adding the production Unmatched→other to the grammar.

1	Start	→	Stmt $
2	Stmt	→	Matched
3		|	Unmatched
4	Matched	→	if e then Matched else Matched
5		|	other
6	Unmatched	→	if e then Matched else Unmatched
7		|	if e then Unmatched
8		|	other

(a) Explain why the grammar is or is not LALR(1)

Again using the same grammar file but adding the ‘other’ rule and then running SableCC on it we get:



```
reduce/reduce conflict in state [stack: TOther *] on EOF in {
        [ PMatched = TOther * ] followed by EOF (reduce),
        [ PUnmatched = TOther * ] followed by EOF (reduce)
}
```

A reduce reduce conflict, meaning not LALR(1)

(b) Is the language of this grammar the same as the language of the grammar in Exercise 40? Why or why not?

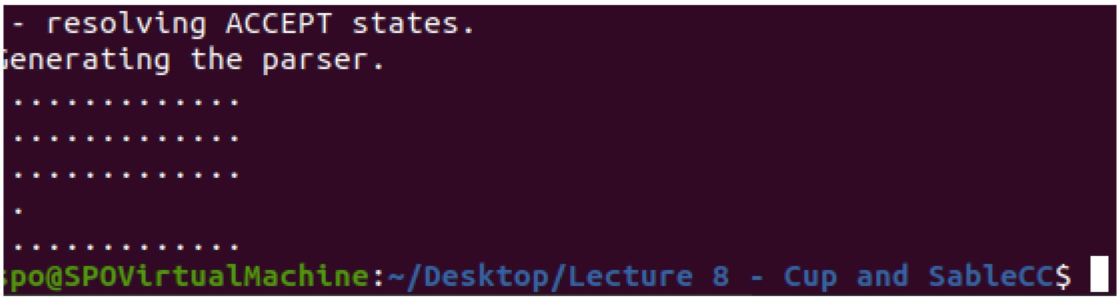
Same answer as the Cup version of this exercise (found earlier in this document).

43. Consider the following grammar:

1	Start	→	Stmt $
2	Stmt	→	Matched
3		|	Unmatched
4	Matched	→	if e then Matched else Matched
5		|	other
6	Unmatched	→	if e then Matched else Unmatched
7		|	if e then Stmt

(a) Explain why the grammar is or is not LALR(1)

Again we modify the previous grammar slightly by replacing ‘unmatched’ with stmt in the last rule. We run SableCC on it and get:



```
 - resolving ACCEPT states.
enerating the parser.
.............
.............
.............
.
.............
po@SPOVirtualMachine:~/Desktop/Lecture 8 - Cup and SableCC$ 
```

The grammar is LALR(1)

(b) Is the language of this grammar the same as the language of the grammar in Exercise 40? Why or why not?

Same answer as the Cup version of this exercise (found earlier in this document).

5. Use The Context Free Grammar Checker on http://smlweb.cpsc.ucalgary.ca/ or its newest version on http://mdaines.github.io/grammophone/ to do Fischer et al exercises 40, 41, 42, 43 on pages 224-233 (exercise 40, 42, 44, 43 on pages 263-264)

   40. Recall the dangling else problem introduced in Chapter 5. A grammar for a simplified language that allows conditional statements follows:

1	Start	→ Stmt $
2	Stmt	→ if e then Stmt else Stmt
3		| if e then Stmt
4		| other

   Explain why the grammar is or is not LALR(1).
   First enter the grammar into the online grammar checker on http://smlweb.cpsc.ucalgary.ca/ :

```
S -> Stmt.
Stmt -> if e then Stmt else Stmt
    | if e then Stmt
    | other.
```

   Next click vital statistics. This brings up a screen saying that the grammar is not LL(1). To see if the grammar is LALR(1) click on ‘generate LALR(1) automaton’. Here we see that a shift/reduce conflict is encountered (dangling else), and for this reason the grammar is not LALR(1).

   41. Consider the following grammar

1	Start	→ Stmt $
2	Stmt	→ Matched
3		| Unmatched
4	Matched	→ if e then Matched else Matched
5		| other
6	Unmatched	→ if e then Matched else Unmatched
7		| if e then Unmatched

   (a) Explain why the grammar is or is not LALR(1)
   We enter the grammar into the grammar checker:

```
S -> Matched
   | Unmatched.
Matched -> if e then Matched else Matched
   | other.
Unmatched -> if e then Matched else Unmatched
   | if e then Unmatched.
```

   This shows a screen stating that the grammar is not LL(1), but clicking on ‘generate LALR(1) automaton’ we see that it is in fact able to generate a LALR(1) parse table for this grammar.

(b) Is the language of this grammar the same as the language of the grammar in Exercise 40? Why or why not?
Same answer as the Cup version of this exercise (found earlier in this document).

42. Repeat Exercise 41, adding the production Unmatched→other to the grammar.

1	Start	→	Stmt $
2	Stmt	→	Matched
3		|	Unmatched
4	Matched	→	if e then Matched else Matched
5		|	other
6	Unmatched	→	if e then Matched else Unmatched
7		|	if e then Unmatched
8		|	other

(a) Explain why the grammar is or is not LALR(1)
Add the ‘other’ rule to the previous grammar.
Clicking on generate LALR(1) automaton we now get a warning that we have a reduce/reduce conflict, meaning the grammar is no longer LALR(1).

(b) Is the language of this grammar the same as the language of the grammar in Exercise 40? Why or why not?
See answer for exercise 42.b from the CUP tool.

43. Consider the following grammar:

1	Start	→	Stmt $
2	Stmt	→	Matched
3		|	Unmatched
4	Matched	→	if e then Matched else Matched
5		|	other
6	Unmatched	→	if e then Matched else Unmatched
7		|	if e then Stmt

(a) Explain why the grammar is or is not LALR(1)
We enter the grammar into the online grammar checker:

```
S -> Smt.
Stmt -> Matched
    | Unmatched.
Matched -> if e then Matched else Matched
    | other.
Unmatched -> if e then Matched else Unmatched
    | if e then Stmt.
```

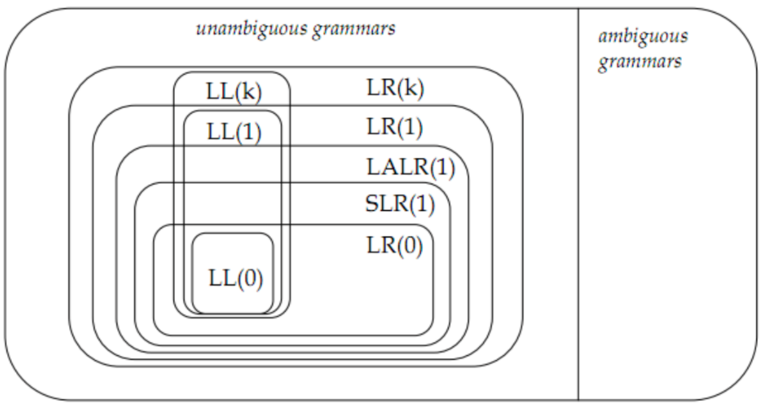
Clicking on ‘generate LALR(1)’ we get a parse table and a message showing that the grammar is LALR(1).
(b) Is the language of this grammar the same as the language of the grammar in Exercise 40? Why or why not?
Same answer as the Cup version of this exercise (found earlier in this document).

Group Exercises - Lecture 08

1. Discuss the outcome of the individual exercises
   Did you all agree on the answers?

2. Discuss pro. And cons. of LR parsing (e.g. as compared to LL parsing)
   Have a look at this map. An LR(k) parser can parse all languages from the LL parser class, while this is not the case the other way around. The LR parsers can, however, be more difficult to implement, so if the language is within for example the LL(1) realm, it may be a good idea to create an LL(1) parser.

   LL(1) versus LR(k)
   A picture is worth a thousand words:



3. Discuss the pros. and cons. of CUP vs SableCC
   Looking back at the exercises: It was a lot easier to formulate the simple grammars in Cup. A lot of small details were required to correctly specify the grammars in SableCC. For example a nonterminal appearing twice on the rhs of a rule requires both occurrences to be named:
   stmt = if e then stmt else stmt // Will not compile due stmt occuring twice
   vs.
   stmt = if e then [one]:stmt else [two]:stmt // Will compile

   Generally the learning curve seems to be steeper for SableCC compared to Cup. The added complexity does however have some benefits. The previous example allows us to access subnodes by their name instead of their position.

4. Follow the Lab associated with Crafting a Compiler Chapter 6
   http://www.cs.wustl.edu/~cytron/cacweb/Chapters/6/lab.shtml
   Find them inside Eclipse in the virtual machine provided on moodle..