# W2 Resubmission

Zeerak Waseem - csp265

December 9, 2013

# 1 Writing Context Free Grammars

Write grammars over the alphabet $\sum = \{a, b, c\}$

## 1.1 Words which contain more a's than b's

We identify that for each production that contains a 'b' the production must also contain an 'a'.

$$
\begin{aligned}
S &= aS \\
S &= aT \\
T &= aTb \\
T &= \epsilon
\end{aligned}
$$

See the grammarfile `a.grm`

## 1.2 Words that are palindromes

We identify that asides from a single character, all letters must be produced twice, one for each side of the singular character.
It is clear that this language will cause conflicts, as it is not $LR(1)$ or $LL(1)$.

$$
\begin{aligned}
S &= A|B|C|a|b|c|\epsilon \\
A &= aSa \\
B &= bSb \\
C &= cSc
\end{aligned}
$$

See the grammarfile `b.grm`

# 2 $LL(1)$-parser construction

The grammar:

$$
\begin{aligned}
Z &\rightarrow b|X\ Y\ Z \\
Y &\rightarrow \epsilon|c \\
X &\rightarrow Y|a
\end{aligned}
$$

## 2.1 Determine which Nonterminals are nullable and calculate First sets of all right-hand sides of the productions

We start by assuming that no nonterminals are nullable, then we use `algorithm 2.4` to calculate the right hand sides, updating the values on the left hand side, when needed.

$$nullable(Z) = nullable(b) \lor nullable(XYZ)$$
$$nullable(Y) = nullable(\epsilon) \lor nullable(c)$$
$$nullable(X) = nullable(Y) \lor nullable(a)$$

Now we determine whether the equations above are true.

$$nullable(b) = False$$
$$nullable(XYZ) = nullable(X) \land nullable(Y) \land nullable(Z)$$
$$nullable(\epsilon) = True$$
$$nullable(c) = False$$
$$nullable(a) = False$$

We then utilize fixed-point iteration (shown in a table) to determine whether the states change. As we can see, the fourth and fifth iteration introduce no

| RHS | Init | It. 1 | it. 2 | It. 3 | It. 4 | It. 5 |
|---|---|---|---|---|---|---|
| b | false | false | false | false | false | false |
| XYZ | false | false | false | false | false | false |
| $\epsilon$ | false | true | true | true | true | true |
| c | false | false | false | false | false | false |
| Y | false | false | false | true | true | true |
| Nonterminals | | | | | | |
| X | false | false | false | false | true | true |
| Y | false | false | true | true | true | true |
| Z | false | false | false | false | false | false |

changes, so we can stop our calculation at the firth iteration. Thus, our result is:

$$nullable(X) = True$$
$$nullable(Y) = True$$
$$nullable(Z) = False$$

We now wish to calculate the first sets of the right hand sides, for this we use `algorithm 2.5`.

$$first(b) = \{b\}$$
$$first(XYZ) = first(X) \cup first(Y) \cup first(Z)$$
$$first(\epsilon) = \{\emptyset\}$$
$$first(c) = \{c\}$$
$$first(a) = \{a\}$$

| RHS | Init | It. 1 | It. 2 | It. 3 | It. 4 | It. 5 |
|---|---|---|---|---|---|---|
| b | ∅ | b | b | a | a | a |
| XYZ | ∅ | ∅ | ∅ | a, b, c | a, b, c | a, b, c |
| $\epsilon$ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| c | ∅ | c | c | c | c | c |
| Y | ∅ | ∅ | ∅ | c | c | c |
| a | ∅ | a | a | a | a | a |
| Nonterminals | | | | | | |
| X | ∅ | ∅ | a | a | a, c | a, c |
| Y | ∅ | ∅ | c | c | c | c |
| Z | ∅ | ∅ | b | b | a, b, c | a, b, c |

We use fixed-point iteration to determine the sets

Thus, the first sets are:

$$\begin{aligned}
first(b) =& \{b\} \\
first(XYZ) =& \{a, b, c\} \\
first(\epsilon) =& \emptyset \\
first(c) =& \{c\} \\
first(Y) =& \{c\} \\
first(a) =& \{a\}
\end{aligned}$$

## 2.2 Calculate Follow sets for all nonterminals (adding an extra start production to recognise the end of the input, denoted by "$")

Given our grammar, the follow sets are quite easily calculated. An extra nontermianl $Z' \rightarrow Z\$$ has been added.

| Nonterminal | Follow set | Constraint |
|---|---|---|
| Z | {$} | Follow(Z) = Follow(Z) |
| Y | {a, b, c} | Follow(X) ⊆ Follow(Y) |
| X | {a, b, c} | |

### 2.3 Determine the look-aheads sets of all productions and put together a parse table for a predictive parser (as shown in the lecture slides)

$$
\begin{aligned}
la(Z \to b) =& First(b) = \{b\} \\
la(Z \to XYZ) =& First(XYZ) \cup Follow(Z) = \{a, b, c, \$\} \\
la(Y \to c) =& First(c) = \{c\} \\
la(y \to \epsilon) =& First(\epsilon) \cup Follow(Y) = \{a, b, c\} \\
la(X \to Y) =& First(Y) \cup Follow(X) = \{a, b, c\} \\
la(X \to a) =& First(a) = \{a\}
\end{aligned}
$$

I'm unsure of how to create a parser table, but this is my attempt.

| Stack | a | b | c | $ |
|---|---|---|---|---|
| Z | XYZ, 2 | XYZ, 2; b, 1 | XYZ, 2 | error |
| Y | $\epsilon$, 3 | $\epsilon$, 3 | $\epsilon$, 3; c, 4 | error |
| X | Y, 5; a, 6 | Y, 5 | Y, 5 | error |
| a | pop | | | |
| b | | pop | | |
| c | | | pop | |
| $ | | | | accept |

## 3 SLR Parser Construction

My grammer:

$$
\begin{aligned}
S &\to Sa \\
S &\to \epsilon
\end{aligned}
$$

### 3.1 Show that your grammar does not generate conflicts (by providing a parse table)

| | 001 | a | $\epsilon$ | $ | S | entry |
|---|---|---|---|---|---|---|
| S0 | S1 | | | | | S2 |
| S1 | | S3 | | reduce 2 | S4 | |
| S2 | | | | accept | | |
| S3 | | s3 | | reduce 2 | S5 | |
| S4 | | | reduce 3 | | | |
| S5 | | | reduce 1 | | | |

As seen there are no conflicts. This parser table is gathered from the output file generated by mosmlyac with the flag -v set.

### 3.2 Compare your grammar to an equivalent one that uses right-recursion. How does the parse stack grow when parsing input?

$$S \to aS$$
$$S \to \epsilon$$

| | 001 | a | $\epsilon$ | \$ | S | entry |
|---|---|---|---|---|---|---|
| S0 | S1 | | | | | S2 |
| S1 | | S3 | | reduce 2 | S4 | |
| S2 | | | | accept | | |
| S3 | | S3 | | reduce 2 | S5 | |
| S4 | | | reduce 3 | | | |
| S5 | | | reduce 1 | | | |

It is seen that the right recursive grammar builds up a stack before reducing it, while the left recursive reduces from the word go.