## Programming Language Translation

## Practical 4 Handin – Group J

### Task 2 Palindromes

**Grammar 1:**

**Does it describe palindromes? If not, why not?**

Grammar 1 does not describe palindromes since it is non-terminating. Every production it creates has at least 1 non-terminal and so creates a never ending cycle continuously calling itself again and again.

**Is it an LL(1) grammar? If not, why not?**

Yes. At any point in the grammar the parser must be able to decide on the basis of a single look-ahead symbol which of several alternatives have to be selected. This grammar is therefore LL(1) since either of the options it can follow begins with distinct tokens (“a” or “b”).

**Grammar 2:**

**Does grammar 2 describe palindromes? If not, why not?**

No, since the single “a” or “b” options in the production, while allowing termination of the grammar, inhibits the ability to produce ALL palindromes. This grammar accepts palindromes of an odd length (such as ababa), but would fail a palindrome of an even length. This is due to the only terminating replacement for the ‘Palindrome’ production being a single ‘a’ or ‘b’, so while it accounts for checking a palindrome from the outside inwards, the centre HAS to end with a single ‘a’ or ‘b’, which only accounts for palindromes of even length.

**Is it an LL(1) grammar? If not, why not?**

No since the parser will not be able to decide which alternative to take. This is easily seen by the fact that the token ‘a’ is the start of at least 2 different productions, the same with the token ‘b’.

**Grammar 3:**

**Does grammar 3 describe palindromes? If not, why not?**

No, though the use of the square brackets now allows for all even length palindromes, it doesn’t account for odd length palindromes that need to terminate with a single character in the centre.

**Is it an LL(1) grammar? If not, why not?**

No since there are LL(1) conflicts caused by “a” and “b” being at the start and successor of a deletable structure.

**Grammar 4:**

**Does grammar 4 describe palindromes? If not, why not?**

No since it will accept non-palindromes such as “ababb”

**Is it an LL(1) grammar? If not, why not?**

No since there are several alternatives that all start with the same symbol and Palin is a deletable structure since it describes all languages of “a” and “b”.

**Can you find a better grammar to describe palindromes? If so, give it, if not, explain why not.**

Yes. With a look-ahead symbol one can determine which of several alternatives to take and therefore reduce ambiguity.

### Task 3 Thinking about ambiguity

**Which of the following statements are true? Justify your answers.**

**(a) An LL(1) grammar cannot be ambiguous.**

True. At any point in the grammar the parser must be able to decide on the basis of a single look-ahead symbol which of several alternatives have to be selected. An ambiguous grammar can produce different parse trees for the same sentence. An ambiguous grammar would therefore have multiple alternatives for the parser to select. Therefore an LL(1) grammar cannot be ambiguous.

**(b) A non-LL(1) grammar must be ambiguous.**

A non-LL(1) is an LL(k) where k > 1. This allows a grammar, over and above the properties of a LL(1), to look ahead k symbols in the production. This means that an LL(k) is a subset of LL(k+1). Hence an LL(k) where k > 1 could describe the same grammar. LL(1) grammars cannot be ambiguous therefore there is an LL(K) grammar that is not ambiguous.

**(c) An ambiguous language cannot be described by an LL(1) grammar.**

Not without the help of resolvers. This allows the parser to determine which alternative to take in instances of LL(1) conflicts.

(d) It is possible to find an LL(1) grammar to describe any non-ambiguous language.

An LL(1) grammar cannot be ambiguous, this does not imply that the inverse is true.

### Task 4 Meet the family

**Is it LL(1) compliant? If not, which productions break the LL(1) rules, and why?**

No. Family, pets and parents are productions that break the LL(1) rules. Since Dad and mom are at the start of several alternatives. “cat” and “dog” are at the start of several alternatives. Since At any point in the grammar the parser must be able to decide on the basis of a single look-ahead symbol which of several alternatives have to be selected. This results in LL(1) conflicts as the parser is unable to determine which alternative to take with only a single look-ahead symbol.

**Can you find an equivalent grammar that does obey the LL(1) constraints? If so, give it. If not, explain why you think it canot be done.**

### Task 5 RPN

**Are the given grammars equivalent?**

The languages are equivalent because they generate the same set of sentences.

**Is either (or both) ambiguous?**

RPN2 is ambiguous since two different parse trees can be derived for the same sentence. RPN1 is not ambiguous since only one parse tree can be derived for the same sentences (CHECK THIS).

**Do either or both conform to the LL(1) conditions? If not, explain clearly where the rules are broken, and come up with an LL(1) grammar that describes RPN notation, or else explain why it might be necessary to modify the language itself to overcome any problems you have uncovered.**

Both fail to conform to LL(1) conditions. The first production in RPN1 has three alternatives that can all start with number. In RPN2 the “-” symbol is the start and successor of a deletable structure. This is because RPN2 can end in a “-” and unaryOP can begin with “-”. The parser therefore doesn’t know when the RPN2 derivation ends and where the unaryOP begins.