CS4158

Tutorial week 6

Here is a Grammar

S -> xAT

A -> AY

A -> ag

T -> mn

T -> ma

T -> p

Y -> q

Y -> f.

**1: Convert this grammar into a grammar that can be parsed using LL(1) parsing.**

S -> xAT {x}

The next production is left recursive in that the first thing an A breaks down into is an A (itself). That means we have the situation:

A->AY {a}

A->ag {a} //ie: the predict set doesn’t let us know which rule to follow to break down an A

// so this is not an LL(1) grammar

Standard refactoring involves 2 new non-terminals N1, and N2:

A -> N1 N2 {a}

N1 -> ag {a}

N2 -> Y N2 {q, f}

N2 -> lambda {m, p} //only one where it is not ‘first’ and ‘predict’, but ‘follow’ and ‘predict’

// is – we go past the end of the RHS and have to see what follows N2

//to determine predict set

The grammar also has a common prefix in that the RHS of 2 T-breakdown rules start with the same thing:

T-> mn {m}

T->ma {m} //again we can see how the predict set does not allow us decide which rule to

//follow to break down a T => not LL(1)

Again, a standard refactoring changes that, with a new N3 non-terminal:

T->m N3 {m}

B->n {n}

B->a {a} //again, all rules that have the same non-terminal on the LHS have different predict

//sets, so it is now LL(1)

T -> p {p}

Y-> q {q}

Y-> f {f}

**2: Calculate the first, follow and predict sets for the grammar, showing that it is LL(1) parse-able.**

The predict-sets are detailed above in brackets beside each rule. These are the same is the first-sets in all cases except one; the N2-> lambda rule. In all other rules the terminals in the predict set are encountered before the RHS of the rule is exhausted. For example, in the rule T->m N3, there is no way we can get past this first m terminal, so it must be the next thing in the language instance if we break down T according to this rule, and that M is on the RHS of that rule.

But in the N2-> lambda rule there is nothing to the RHS of the rule, so we fall off the end (ie we have fully matched off the N2 on the LHS). So, to calculate the predict set we have to see what might FOLLOW an N2 and we look for other rules with an N2 on the RHS. That is what we mean when we say that we have to use the follow-set to identify the predict set for N2->lambda.

We find one such rule: A-> N1 N2. But, if we have fully matched off this N2, then we have fully matched off to the end of this rule also, meaning that we have fully matched off the LHS of the rule – an A. So then we have to look to see what could possibly follow an A on the RHS of any production rule.

We find an A on the RHS of one rule: S->xAT. We see that T can break down to an m or a p, so {m,p} becomes the FOLLOW set (and predict-set) for N2-> lambda.

**3: Use the grammar and predict set to assess if xma, xagp, xagqfqma are valid language instances.**

xagp (for example – the only example I did in class!)

S (predict x) -> xAT, x consumed, so we are now targeting A for breakdown

A (predict a) -> N1 N2, so now we are targeting N1

N1 (predict a) -> ag, so ag is consumed, meaning that N1 is fully matched off and we can target N2

N2 (predict p) -> lambda, meaning that N2 is fully matched, and so A-> N1 N2 is now fully matched

So we can fall back to the S->xAT and target the T

T (predict p) -> p, so p consumed, so we have fully matched off the S (our start symbol) at the end of the language instance => a successful parse: xagp is a valid language instance of that grammar.

**4: What is the job of the lexer?**

To identify the individual words/tokens/elements who’s structure is to be subsequently assessed by the parser. For programming language this includes keywords, identifiers, punctuation and mathematical operators.

5: Describe the 4 types of Grammar Chomsky identified. Characterize each of them in terms of the constraints they have and the memory requirements of an automaton that could analyze them.

Types:

Unrestricted Grammars: Any combination of terminals/non-terminals on the LHS, and any combination of terminals/non-terminals on the RHS. Otherwise, there are no restrictions;

Language instances cannot be parsed using a Turing machine

Context Sensitive Grammars: Any combination of terminals/non-terminals on the LHS, and any combination of terminals/non-terminals on the RHS. Otherwise, the only restriction is that the LHS must be shorter than the RHS;

Language instances can be parsed using a Turing machine with infinite memory;

Context Free Grammars: Only one non-terminal on the LHS, and any combination of terminals/non-terminals on the RHS.

Language instances can be parsed using a Turing machine with memory related to the LANGUAGE INSTANCE size

Regular Grammars: : Only one non-terminal on the LHS and, at most, one non-terminal and one terminal on the RHS.

Language instances can be parsed using a Turing machine with memory related to the GRAMMAR size

A->aB

A->B

A->a

A-> lambda