COMP 317: Semantics of Programming Languages

Lecture 2 Summary

BNF is used to specify the syntax of numerals.

<Digit>	::=	`0` | `1`
<Numeral>	::=	*<Digit>* | *<Numeral>* *<Digit>*

BNF rules define a **language**: a set of strings. Every string in the language is generated by the rules. The only two strings in the language of *<Digit>*s are `0` and `1`. Every string in the language of *<Numeral>*s is either a *<Digit>* or of the **form** *N D* where *N* is a *<Numeral>* and *D* is a *<Digit>*.

This allows us to define functions on a language **inductively**: by considering the forms that the strings of the language can take, following the BNF options. For example, we define the integer [[*D*]]*digit* as a function of *<Digit> D* by considering the two options defining *<Digit>*: *D* is either `0` or `1`. Thus we define the function:

[[`0`]]*digit*	=	0
[[`1`]]*digit*	=	1

Similarly, we define the integer [[*N*]]*numeral* as a function of *<Numeral> N* by noting that every *<Numeral>* is either a *<Digit>* or of the form *N D*:

[[*D*]]*numeral*	=	[[*D*]]*digit*	for any *<Digit> D*
[[*N D*]]*numeral*	=	2 * [[*N*]]*numeral* + [[*D*]]*digit*	for any *<Numeral> N and <Digit> D*

Key Points

The key points that will recur throughout the module are:

- **Syntax** is strings: things that are written down. Syntactic **languages** are sets of strings, and these can be defined by BNF.
- **Denotation functions** map syntax to semantics, and can be defined **inductively** on the form of their inputs: each BNF option defines one form that the input can take.

You should be able to:

- use BNF to define languages
  see the tutorial on BNF by Lars Marius Garshol, and the Problem Sheet for Week 2;
- define functions inductively
  see the Problem Sheet for Week 2.

Grant Malcolm