Meeting 5: Syntax

Today

Review/Questions?
Grammars and Context-Free Languages

Boolean Operators

\[ \text{AND} \quad \&\& : (\text{Bool}, \text{Bool}) \Rightarrow \text{Bool} \]

\[ \text{OR} \quad \lor : (\text{Bool}, \text{Bool}) \Rightarrow \text{Bool} \]

\[ \text{NOT} \quad ! : \text{Bool} \Rightarrow \text{Bool} \]

\[ < : (\text{Int}, \text{Int}) \Rightarrow \text{Bool} \]

Induction - Scala / Math

\[ \text{Not} \]

- Boolean operators
- Qualifications for well-typedness

Well-Typedness

\[ a : \text{Int}, b : \text{Int} \]

\[ \text{if} \ (3 < 4) \ (1, 2) \ \text{else} \ (a, b) \quad : \ (\text{Int}, \text{Int}) \]

3 < 4 : Boolean
3 : Int
4 : Int

(1, 2) : (Int, Int)
1 : Int
2 : Int

(a, b) : (Int, Int)

by environment
def sumFirstN(n: Int): Int {
    require(n ≥ 0)
    def s(n: Int, acc: Int): Int = n match {
        case 0 ⇒ acc
        case _ ⇒ s(n - 1, n + acc)
    }
    s(n, 0)
}

Theorem: For any N s.t. n ≥ 1,
\[ \sum_{i=0}^{n} i \]

Proof by Induction on n
Inductive Case N = N + 1 for some N

IH: sumFirst(N) → \( \sum_{i=0}^{N} i \)
→ sumFirst(N + 1) → s(N, 0) → s(N - 1, N + 0) → s(N + 1, 0)
Grammars (and Syntax)

**Take-home**

1. A *context-free grammar* describes a *context-free language*, which we often use to describe PLS.

2. A context-free grammar is composed of terminal, non-terminal, and productions.

3. A derivation exhibits a sentence in a language recognized by a grammar.

**Why**: Syntax

User: What can you write?

Designer: Users will use it and know your intent.

Implementer: What input is ok.
1. Syntax — What you can write

   Defining a language

   = the form of expressions, statements, programs

2. Semantics — What an expression \( a \# b \) means?

   means = how \( \frac{1}{a} \) evaluates to

   Eg: \( a \land b \) Scala \( \lor \)

   \( a \land b \) ML string concatenation

---

Language Terminology
Language Terminology

A language \( L \) is a set of strings of characters drawn from some alphabet \( \Sigma \).

A sentence is a string in a language.

A (context-free) grammar describes (context-free) languages.

BNF (Backus-Naur Form) is a notation for writing context-free grammars.
BNF

non-terminals | terminals | productions |
chars in our alphabet |

\[ \Sigma = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 \} \]

Number \[ \ ::= \_ \| 2\| 3\| 4\| 5\| 6\| 7\| 8\| 9\| 0 \]

74 \in L(Number) \hspace{1cm} \text{No}

Big Number \[ \ ::= \] BigNumber Number

Show \( 34 \in L(\text{BigNumber}) \) | Number

Derivation

Big Number \[ \rightarrow \] BigNumber Number

\[ \rightarrow \] Number Number

\[ \rightarrow \] 3 Number \[ \rightarrow \] 3 4
stmt ::= AssignStmt | ReturnStmt

AssignStmt ::= Id ::= Expr

ReturnStmt ::= return Expr

Expr ::= Id + Id | Id | Number

Id ::= a | b | c | ... | z

1. \( a+b ::= c+d \) \( N_0 \)
2. \( a ::= b + c + d \) \( N_2 \)
3. \( myvar ::= myvar1 + myvar2 \) \( N_0 \)
4. \( a ::= b+1 \) \( N_0 \)
5. \( a ::= 1+b \) \( N_0 \)