CS571
- Notes 01
- Introduction, Motivation, Abstract Syntax

**Motivation**
- For language designers
  - Accurate, unambiguous goals; exploration of alternative paradigms; definition; standardization; PL research
- For language implementors
  - Specification of mechanisms and structures; guide for methods

**Motivation**
- For language learners
  - Provides understanding of model of computation (normative programming styles)
- For language users
  - Reference for “pushing the envelope”
Abstract Syntax

- Concrete syntax is about parsing a character string into a program.
- Abstract syntax is about representing semantically meaningful structures.
- The link between the two are parse trees.

Example:

\[ x := y + 2 \]

The concrete tokens are: `ident`, `assign-op`, `ident`, `plus`, `integer`.

Its parse tree could be:

```
  :=
   +
   |
   x
   |
   y
   +
   2
```

Abstract syntax

- Semantically we can remove all the concrete elements.
- The semantics are clear even though all the terminal symbols from the concrete string have disappeared.
- Thus the goal of abstract syntax is representation, not parsing.

Example of an Imperative Language

- Abstract syntax (i.e., semantically meaningful) categories:
  - `P` = Programs
  - `C` = Commands
  - `E` = Expressions
  - `I` = Identifiers
  - `N` = Numerals
  - `B` = Booleana

The grammar:

\[
P ::= C \\
C ::= \text{nop} \mid C_1 ; C_2 \mid I := E \mid \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ end} \mid \text{while } B \text{ do } C \text{ end} \\
E ::= N \mid I \mid E_1 OA E_2 \\
OA ::= \text{plus} \mid \text{minus} \mid \text{mult} \mid \text{divide} \\
B ::= \text{true} \mid \text{false} \mid I \text{ OR } E \\
\text{OR} ::= \text{equal} \mid \text{greater} \mid \text{less} \\
I ::= \ldots \text{any identifier} \ldots \\
N ::= \text{zero} \mid \text{one} \mid \text{two} \ldots \\
B ::= \text{true} \mid \text{false} \mid \text{any boolean} \ldots \\
I ::= \ldots \text{any identifier} \ldots \\
N ::= \ldots \text{any numeral} \ldots \\
\]

Example program:

\[
x := \text{three}; \\
y := \text{two}; \\
\text{while } x \text{ greater zero do } \\
\text{if } y \text{ equal } x \text{ then } \\
\text{nop} \\
\text{else } \\
y := x \text{ minus one} \\
\text{end} \\
\text{end}
\]
Example of a Functional Language

- Syntactic categories:
  - P ∈ Programs
  - F ∈ Functions
  - E ∈ Expressions
  - I ∈ Identifiers
  - N ∈ Numerals

- The grammar:
  - P ::= F1 F2...Fn E
  - F ::= I(I1,I2,...In) = E
  - E ::= N | I | E1 O E2 | if E1 then E2 else E3 | let I = E1 in E2 | I(E1,E2,...En)
  - O ::= plus | minus | mult | divide | equal | greater | less

Example program:

even(x) = if x then odd(x minus one) else one
odd(x) = if x then even(x minus one) else one

even(four)