Meeting 6: Concrete + Abstract Syntax

\[ \Sigma = \{a, 3\} \]

- \[ A ::= \text{aa} \]
- \[ A ::= \text{aa } A \mid \text{\varepsilon} \]
- \[ A ::= \text{aa } A \mid a \]  \quad A \rightarrow a \]

The language is defined by the grammar:

- \[ A ::= (A) A \mid \text{\varepsilon} \]

The symbols 0, 1, 2, and 3 are generated by the grammar:

- 0, 1, 2
- (())
- (())()
Concrete vs. Abstract Syntax

3-line summary

Concrete syntax refers to the surface syntax of a language = strings

Abstract syntax is the structure of a language = terms / tree

Precedence, Associativity, Ambiguous grammars
Ambiguity

\[ e ::= n \mid e + e \mid e \times e \]

```
1 + (2 \times 3) \quad \rightarrow \times 7
```

\[ (1 + 2) \times 3 \quad \rightarrow \times 9 \]

---

\[ 1 + 2 \times 3 \]

This string can be read in two ways:

This string can be read in two ways.
"ambiguous grammars need parentheses"

\[ e ::= n | e \# e \]

\[ \Sigma = \#, 0, 1, 2, \ldots, 3 \]

\[ (7 \# 6) \# 5 \]

Resolving Ambiguity

\[ e ::= n | n \# e \]

right-associativity

\[ (7 \# 6) \# 5 \]

right recursion
\[ e ::= n | e + e | e \times e \]

\[ 7 + 2 \times 3 \]

precedence = binding tightly

\[ e ::= n | n + e | n \times e \]

\[ e ::= t | t + e \]

\[ t ::= n | n \times t \]

\[ \frac{(7+2)}{3} \]

\[ 7 + (2 \times 3) \]

resolving

"layering of grammar" = "precedence ambiguity"
Better Idea

Use trees directly!

Goal: Separate readability concerns from structural concerns

Abstract Syntax

\[ e ::= n \mid +(e, e) \mid *(e, e) \]

\[ + (\ast (3, 4), 5) \]
\[ 1 + (2 \times 3) \]

1 + 2 \times 3 - bad for a compiler

plus \((1, \text{times} (2, 3))\) - better for a compiler

\[
\text{strings} \xrightarrow{\text{Parse}} \text{trees/terms}
\]
Abstract Syntax

"Give me trees, or give me death."

\[ e ::= n \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid x \mid \text{let } x = e_1 \text{ in } e_2 \]

- \( x \) is a name
- \( n \) is a number

\[ \text{let } x = e_1 \text{ in } e_2 \]

means bind a new name \( x \) to \( e_1 \)
for use in \( e_2 \)
(let \( x \equiv 1 \Rightarrow x + x + z \) )

\[ \alpha \text{-equivalent} \]

(let \( y \equiv 1 \Rightarrow y + y + z \) )

Higher-Order abstract syntax = tree + bound variables can be renamed