Recall syntax directed translation (SDT)

SDT = translating from a sequence of tokens into a sequence of actions/other form, based on underlying syntax

To define a syntax-directed translation

- augment CFG with translation rules
  - define translation of LHS non-terminal as a function of: constants, translations of RHS non-terminals, values of terminals (tokens) on RHS

To translate a sequence of tokens using SDT (conceptually)

- build parse tree
- use translation rules to compute translation of each non-terminal (bottom-up)
- translation of sequence of tokens = translation of parse tree's root non-terminal

Examples

- translate tokenized stream to an integer value
- translate tokenized stream to a string

For parsing, we'll need to translate tokenized stream to abstract-syntax tree (AST)
Abstract syntax trees

\( \text{AST} = \text{condensed form of parse tree} \)

- 
- 
- 
- 
- 

Example:
Example 2

CFG

\[ \text{expr} \rightarrow \text{expr} + \text{term} \]
| \text{term} \\
\[ \text{term} \rightarrow \text{term} * \text{factor} \]
| \text{factor} \\
\[ \text{factor} \rightarrow \text{INTLIT} \]
| ( \text{expr} ) \\

AST for parsing
AST implementation
Translation rules to build ASTs for expressions

\[
\begin{align*}
\text{expr} & \rightarrow \text{expr} + \text{term} \\
& \quad | \quad \text{term} \\
\text{term} & \rightarrow \text{term} * \text{factor} \\
& \quad | \quad \text{factor} \\
\text{factor} & \rightarrow \text{INTLIT} \\
& \quad | \quad ( \text{expr} )
\end{align*}
\]
ASTs for non-expressions

Example

```c
void foo(int x, int y) {
    if (x == y) {
        return;
    }
    while (x < y) {
        cout << "hello";
        x = x + 1;
    }
    return;
}
```
ASTs for lists

CFG

idList → idList COMMA ID
|   ID
The bigger picture

Scanner

- **Language abstraction**: regex
- **Output**: token stream
- **Tool**: JLex
- **Implementation**: interpret DFA using table (for $\delta$), recording
  `most_recent_accepted_position` & `most_recent_token`

Parser

- **Language abstraction**: 
- **Output**: 
- **Tool**: 
- **Implementation**: 

---

The bigger picture

Scanner

- **Language abstraction**: regex
- **Output**: token stream
- **Tool**: JLex
- **Implementation**: interpret DFA using table (for $\delta$), recording
  `most_recent_accepted_position` & `most_recent_token`