CS 536 Announcements for Monday, February 19, 2024

Programming Assignment 2
• due Tuesday, February 20

Last Time
• syntax-directed translation
• abstract syntax trees

Today
• implementing ASTs

Next Time
• Java CUP

SDT review

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

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

eexpr \rightarrow \text{expr} + \text{term} \\
| \text{term} \\

term \rightarrow \text{term} \times \text{factor} \\
| \text{factor} \\
factor \rightarrow \text{INTLIT} \\
| \text{( expr )}

AST for parsing

We've been showing the translation in two steps:

In practice we'll do

Why have an AST?
AST implementation

Define a class for each kind of AST node

Create a new node object in some rules
- new node object is the value of LHS.trans
- fields of node object come from translations of RHS non-terminals
### Translation rules to build ASTs for expressions

<table>
<thead>
<tr>
<th>CFG</th>
<th>Translation rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expr → expr + term</code></td>
<td><code>expr_{1}.trans =</code></td>
</tr>
<tr>
<td><code>term                  </code></td>
<td><code>expr.trans =</code></td>
</tr>
<tr>
<td><code>term → term * factor</code></td>
<td><code>term_{1}.trans =</code></td>
</tr>
<tr>
<td><code>factor                </code></td>
<td><code>term.trans =</code></td>
</tr>
<tr>
<td><code>factor → INTLIT</code></td>
<td><code>factor.trans =</code></td>
</tr>
<tr>
<td><code>factor → ( expr )</code></td>
<td><code>factor.trans =</code></td>
</tr>
</tbody>
</table>
Example

```cpp
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**: regular expressions
- **Output**: token stream
- **Tool**: JLex
- **Implementation**: interpret DFA using table (for δ), recording most_recent_accepted_position & most_recent_token

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