Th 3/26/2013: Syntax-directed translation

Lecture topics:
1. Abstract syntax trees
2. Infix → postfix example
3. Translation schemes
4. Object-oriented implementation

I. Abstract syntax trees

Example grammar:
E → E + E | E - E | E * E | E / E | E

Parse trees (concrete syntax trees):

```
E
/   
|    |
E    E
/     |
E    E
/     |
|    N
N
```

* has higher precedence than +
- is left associative

Abstract syntax trees (ASTs):

```
+       *
\      /  \\
|    |   |
E    E
/     |
|   N |
|   N |
```

<table>
<thead>
<tr>
<th>kind</th>
<th>concrete</th>
<th>abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>punctuation</td>
<td>E := E_1</td>
<td>E_1 E_2</td>
</tr>
<tr>
<td>pass-through</td>
<td>E</td>
<td>N</td>
</tr>
<tr>
<td>production</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>implemented</td>
<td>implicit</td>
<td>allocate objects in memory</td>
</tr>
</tbody>
</table>

II. Infix → postfix example

<table>
<thead>
<tr>
<th>infix</th>
<th>postfix</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 + 3</td>
<td>3 3 +</td>
</tr>
<tr>
<td>1 + 2 * 3</td>
<td>1 2 3 * +</td>
</tr>
<tr>
<td>1 - 2 - 3</td>
<td>1 2 3 -</td>
</tr>
</tbody>
</table>

task: given AST for infix expression, build postfix string

Translation scheme:

```
E → E + E_2 | E_1 - E_2 | E_1 * E_2 | E_1 / E_2 | N
```

```
E_p = E_p + E_2_p "+"
E_p = E_p + E_2_p "-"
E_p = E_p + E_2_p "*"
E_p = E_p + E_2_p "/"
E_p = N_p
```

Simple compiler:

```
character stream → Lexical analyzer

stream of tokens with "v" attributes

Syntax analyzer

AST

Code generator

postfix code

Translation example:

```
E_p = 12 -
```

```
E_p = 1
E_p = 2
E_p = 3
N_v = 1
N_v = 2
N_v = 3
1 2 3
```

III. Translation schemes

SDD = syntax-directed definition
      = grammar + rules defining attributes

Synthesized attribute = 

parent computed from children
(can be done during parsing or using tree traversal)

Inherited attribute = 

child computed from parent or siblings
(requires tree traversal)

S-attribute definition = 

SDD with only synthesized attributes

L-attribute definition = 

SDD where all attributes are either synthesized or, if inherited, depend only on parent or left siblings

p.1/2
IV Object-oriented implementation

```
AstNode
  abstract Object accept(Visitor v)
```

```
Expr
  String e
```

```
InfixExpr
  Object accept(Visitor v) {
    return v.visit(this);
    Expr e1
    Expr e2
  }
```

```
Number
  Object accept(Visitor v) {
    return v.visit(this);
    Token tok
  }
```

```
Visitor
```

```
InfixToPostfixTranslator
  Object visit(InfixExpr e) {
    e.e1.accept(this);
    e.e2.accept(this);
    e.op = e.e1.op +
    e.e2.p + e.op ;
  }
```

```
Interaction:
```

```
Main
  1. accept
```

```
InfixExpr
  2. visit
```

```
Number
  3. accept
```

```
InfixToPostfixTranslator
  4. visit
```

```
InfixExpr
  5. accept
```

```
InfixExpr
  6. visit
```

Note: HACS hides all these implementation details, allows you to specify translation schemes directly.

Reminder:
- Pr 1 due tomorrow
-.hw 3 due Fri 10/4