Overview

• context handling
• annotating the AST
• attribute grammars
• manual methods

Attribute grammars

• formal attributes are associated with each grammar symbol
  • type
  • location
  • context

• inherited attributes
• synthesized attributes

Concise evaluation rules

Constant_definition(INH old symbol table, SYN new symbol table) → 'CONST' Defined_identifier '=' Expression ';'
ATTRIBUTE RULES:
  SET Expression . symbol table TO Constant_definition . old symbol table;
  SET Constant_definition . new symbol table TO Updated symbol table( Constant_definition . old symbol table, Defined_identifier . name, Expression . type, Expression . value);

Constant_definition(INH old symtab, SYN new symtab) → 'CONST' Defined_identifier(name) '=' Expression(old symtab, type, value) ';
ATTRIBUTE RULES:
  SET Expression . symbol table TO Constant_definition . old symtab;
  SET Constant_definition . new symbol table TO Updated symbol table( Constant_definition . old symtab, Defined_identifier . name, Expression . type, Expression . value);

Attribute grammars

• attribute evaluation rules are associated with each production

Constant_definition(INH old symbol table, SYN new symbol table) → 'CONST' Defined_identifier '=' Expression ';'
ATTRIBUTE RULES:
  SET Expression . symbol table TO Constant_definition . old symbol table;
  SET Constant_definition . new symbol table TO Updated symbol table( Constant_definition . old symbol table, Defined_identifier . name, Expression . type, Expression . value);

Attribute grammars

• dependency graph
Running example

integral numbers in decimal or octal notation

- 11D
- 234O
- 56O
- 789D

Attribute grammar for integral numbers

```
Number(SYN value) → Digit_Seq(base, value) Base_Tag(base)
```

```
ATTRIBUTE RULES
SET Digit_Seq.base TO Base_Tag.base;
```

```
Digit_Seq(INH base, SYN value) → Digit_Seq(base, value) Digit(base, value)
```

```
ATTRIBUTE RULES
SET value TO Digit_Seq.value * base + Digit.value;
```

```
Digit(INH base, SYN value) → DIGIT // token, '0'-'9'
```

```
ATTRIBUTE RULES
SET value TO Checked digit value(base, DIGIT.repr[0] – '0');
```

```
Base_Tag(SYN base) → 'O'
```

```
ATTRIBUTE RULES
SET base TO 8;
```

```
Base_Tag(SYN base) → 'D'
```

```
ATTRIBUTE RULES
SET base TO 10;
```

Dependency graphs for integral numbers

```
Number(SYN value) → Digit_Seq(base, value) Base_Tag(base)
```

```
ATTRIBUTE RULES
SET Digit_Seq.base TO Base_Tag.base;
```

Exercise (5 min.)

- draw the other dependency graphs for the integral-number attribute grammar

Answers

- allocate space for attributes in the nodes of the AST
- fill the attributes of the terminals in the AST (leaf nodes)
- execute the evaluation rules to assign values to attributes (interior nodes)
  - a rule may fire when all input attributes are defined
  - loop until no new values can be assigned

Attribute evaluation
Attribute evaluation

input: “13O”

Attribute evaluation by tree walking

• at each node:
  • try to perform all the assignments in the evaluation rules for that node
  • visit all children
  • again try to perform the attribute assignments
  • repeat walking until top-level attributes are assigned

Exercise (4 min.)

WHILE Number.value is not set:
  walk number(Number);

• how many tree walks are necessary to evaluate the attributes of the AST representing the octal number ‘13O’?
• how many for ‘1234D’?

Answers

Cycle handling

• dynamic cycle detection
  #walks > #attributes

• static cycle detection
  transitive closure of IS-SI graphs, see book
Multi-visit attribute grammars

- avoid interpretation overhead
- generate code for each visit that “knows” what attributes to assign and which children to visit

\[
\{(i_1, i_3, s_1), (i_2, s_2)\}
\]

- static attribute partition: \((IN_i, SN_i) = 1, ..., n\)

Ordered attribute grammars

- late evaluation partitioning heuristic
  - work backwards
  - find \((IN_{last}, SN_{last})\) of the last visit
    - \(SN_{last}\) has no outgoing edges in IS-SI graph
    - \(IN_{last}\) is the set of attributes \(SN_{last}\) depends on
  - remove \(IN_{last}\) and \(SN_{last}\) from the IS-SI graph
  - repeat until the IS-SI graph is empty

Exercise (7 min.)

- derive the late evaluation partitioning of the number attribute grammar using the IS-SI graphs below

\[
IN_1, SN_1, IN_2, SN_2
\]

Answers

<table>
<thead>
<tr>
<th></th>
<th>IN_1</th>
<th>SN_1</th>
<th>IN_2</th>
<th>SN_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digit_Seq</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base_Tag</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

L-attributed grammars

- combine attribute grammars and top-down parsing
- attributes may only depend on information from the parent node or siblings to the left

LLgen example

```
main : [line]+
line {int e;}
: expr(&e) '\n' { printf("%d\n", e);}

expr(int *e) {int t;}
: term(e)
[ '+' term(&t) { *e += t;}
]*

term(int *t) {int f;}
: factor(t)
[ '*' factor(&f) { *t *= f;}
]*

factor(int *f) : '(' expr(f) ')' | DIGIT { *f = yylval;}
```

```
Bottom-up parsing and attribute grammars

- stack of attributes
- problem with inherited attributes
  \[ A \to B \{ \text{C.inh_attr := } f(B.\text{syn_attr}); \} \quad \text{C} \]
  code can only be executed at the end
- solution: \( \epsilon \)-rules
  \[ A \to B \quad \text{Action1} \quad \text{C} \]
  \[ \text{Action1} \to \epsilon \quad \{ \text{C.inh_attr := } f(B.\text{syn_attr}); \} \]

S-attributed grammars

- combine attribute grammars and bottom-up parsing
- only synthesized attributes may be used

Summary

- attribute grammars
  - formal attributes per symbol: inherited & synthesized
  - attribute evaluation rule per production
- dependency graphs
- cycle detection
- evaluation order

Homework

- study sections:
  - 3.1.3.2 static cycle checking
- assignment 1:
  - replace yacc with LLgen
  - deadline April 9 08:59
- print handout for next week [blackboard]