Compiler construction in4303 – lecture 6

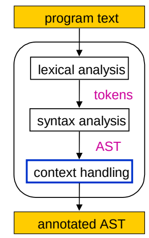
AST processing
attribute grammars

Chapter 3.1

Overview

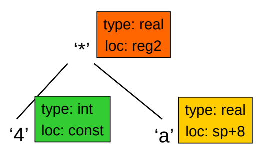
- context handling
- annotating the AST
  - attribute grammars
  - manual methods

```
expr : expr '+' expr   { $$ = $1 + $3;}
     | expr '*' expr   { $$ = $1 * $3;}
     | '(' expr ')'    { $$ = $2;}
     | DIGIT
     ;
```



Attribute grammars

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



- inherited attributes
- synthesized attributes

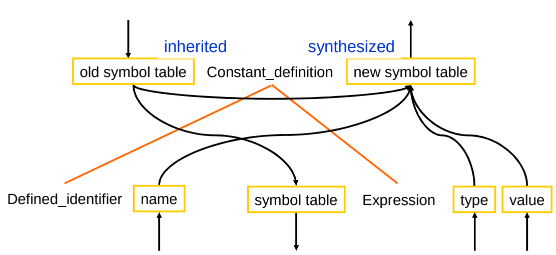
Attribute grammars

- 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);

Dependency graph

Constant_definition(INH old symbol table, SYN new symbol table) →
  'CONST' Defined_identifier '=' Expression ';'

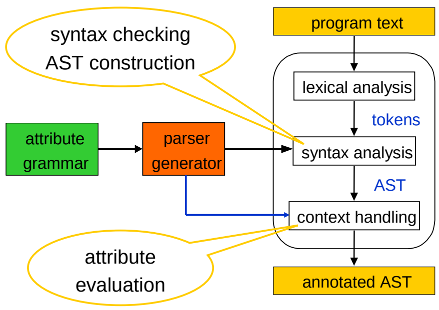


Concise evaluation rules

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 new symtab TO Updated symbol table(old symtab, name, type, value);

Extended parser generator



Running example

integral numbers in decimal or octal notation

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

Number	→	Digit_Seq Base_Tag
Digit_Seq	→	Digit_Seq Digit | Digit
Digit	→	DIGIT // token, ‘0’-’9’
Base_Tag	→	‘O’ | ‘D’

Attribute grammar for integral numbers

Number(SYN value) → Digit_Seq(base, value) 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_Seq(INH base, SYN value) → Digit(base, value)

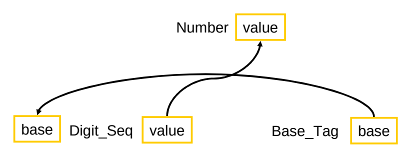
Digit(INH base, SYN value) → DIGIT // token, ‘0’-’9’
ATTRIBUTE RULES
SET value TO Checked digit value(base, DIGIT.repr[0] – ‘0’);

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 Number.value TO Digit_Seq.value;
SET Digit_Seq.base TO Base_Tag.base;



Exercise (5 min.)

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

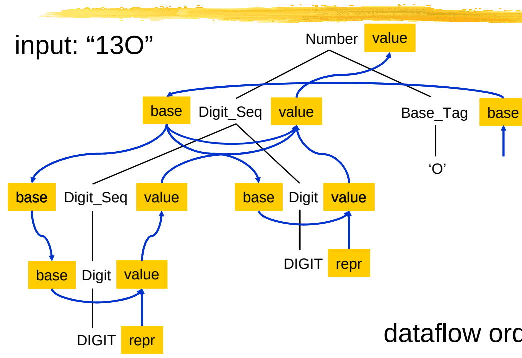
Answers

Attribute evaluation

- 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

input: “13O”



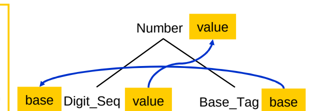
dataflow order

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

Attribute evaluation by tree walking

```
walk number(node)
    evaluate number(node)
    walk digit_seq(node.digit_seq)
    walk base_tag(node.base_tag)
    evaluate number(node)
```




```
evaluate number(node)
  IF node.value is not set AND node.digit_seq.value is set THEN
      SET node.value TO node.digit_seq.value
  IF node.digit_seq.base is not set AND node.base_tag.base is set THEN
      SET node.digit_seq.base TO node.base_tag.base
```

Exercise (6 min.)

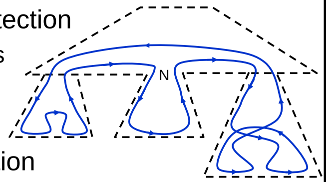
```
WHILE Number.value is not set:
      walk number(Number);
```

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

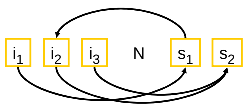
Answers

Cycle handling

- dynamic cycle detection
  #walks > #attributes

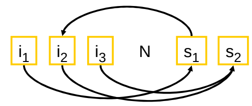


- static cycle detection
  transitive closure of IS-SI graphs, see book



IS-SI graphs

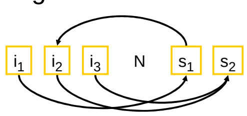
Synthesized-Inherited dependencies capture the dependencies in all trees above N



Inherited-Synthesized dependencies capture the dependencies in all trees below N

Multi-visit attribute grammars

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

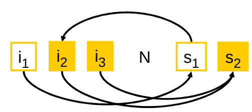


- static attribute partition: $(IN_i,SN_i)_{i = 1..n}$

$(\{i_1, i_3\},\{s_1\}), (\{i_2\},\{s_2\})$

Ordered attribute grammars

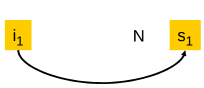
- late evaluation partitioning heuristic
  - work backwards
  - find (IN_{last},SN_{last}) of the last visit
    - SN_{last} has no outgoing edges in IS-SI graph
    - IN_{last} is the set of attributes SN_{last} depends on
  - remove IN_{last} and SN_{last} from the IS-SI graph
  - repeat until the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



$(\{i_2, i_3\},\{s_2\})$

Ordered attribute grammars

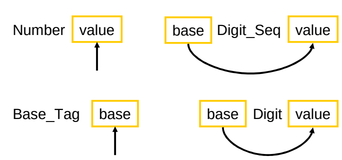
- late evaluation partitioning heuristic
  - work backwards
  - find (IN_{last},SN_{last}) of the last visit
    - SN_{last} has no outgoing edges in IS-SI graph
    - IN_{last} is the set of attributes SN_{last} depends on
  - remove IN_{last} and SN_{last} from the IS-SI graph
  - repeat until the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



$(\{i_1\},\{s_1\})$ $(\{i_2, i_3\},\{s_2\})$

Example: integral numbers

- IS-SI graphs:



- attribute partitioning: (late evaluation)

	IN_1	SN_1
Number		value
Digit_Seq	base	value
Digit	base	value
Base_Tag		base

Example: integral numbers

- attribute partitioning: (late evaluation)

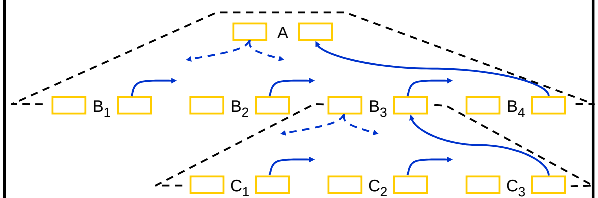
	IN_1	SN_1
Number		value
Digit_Seq	base	value
Digit	base	value
Base_Tag		base

- code:

```
visit_1 number(node)
    visit_1 base_tag(node.base_tag)
    SET node.digit_seq.base TO node.base_tag.base
    visit_1 digit_seq(node.digit_seq)
    SET node.value TO node.digit_seq.value
```

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



LLnextgen example

```
main    : line+
        ;
line
        : expr '\n'          { printf("%d\n", expr);}
        ;
expr<int>
        : term               { LLretval = term;}
          [ '+' term<right>  { LLretval += right;}
          ]*
        ;
term<int>
        : factor             { LLretval = factor;}
          [ '*' factor       { LLretval *= factor;}
          ]*
        ;
factor<int>
        : '(' expr ')'       { LLretval = expr;}
        | DIGIT              { LLretval = yylval;}
        ;
```

attributes

evaluation rules

Bottom-up parsing and attribute grammars

- stack of attributes
- problem with inherited attributes

$A \rightarrow B$ {C.inh_attr := f(B.syn_attr);} C

code can only be executed at the end

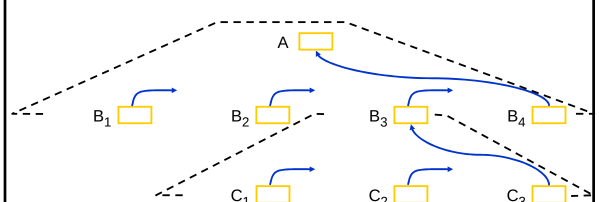
- solution: ε-rules

$A \rightarrow B$ Action1 C

Action1 $\rightarrow \varepsilon$ {C.inh_attr := f(B.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