Context-Sensitive Analysis

Fanchao Meng [1]

[1]Department of Computer and Information Sciences
University of Delaware
fcmeng@udel.edu

Outline

1. Motivation
2. Attribute Grammar

Recap

Can CFG see what wrong is here?

```
m_A_member1 : Int <- "a_string";
```

Recap

Can CFG see what wrong is here?

```
m_A_member1 : Int <- "a_string";
```

Let us try...

Recap (Cont.)

Then which level does this problem exist?

- Lexcial?
- Syntax?
- Semantic?

Recap (Cont.)

Conclusion

- CFG cannot interpret what a parsing result (e.g. a parse tree) means.

Recap (Cont.)

Conclusion

- CFG cannot interpret what a parsing result (e.g. a parse tree) means.
- Semantic analysis judges whether the syntax structure of a program makes sense in terms of its meaning.

Recap (Cont.)

What should we check in a semantic analysis? (not a thorough list)

Recap (Cont.)

What should we check in a semantic analysis? (not a thorough list)

- Type.

Recap (Cont.)

What should we check in a semantic analysis? (not a thorough list)

- Type.
- Scope.

Recap (Cont.)

Some typical semantic errors:

Recap (Cont.)

Some typical semantic errors:

- Type mismatch.

Recap (Cont.)

Some typical semantic errors:

- Type mismatch.
- Undeclared variable.

Recap (Cont.)

Some typical semantic errors:

- Type mismatch.
- Undeclared variable.
- Reserved identifier misuse.

Recap (Cont.)

Some typical semantic errors:

- Type mismatch.
- Undeclared variable.
- Reserved identifier misuse.
- Multiple declaration of variable in a scope.

Recap (Cont.)

Some typical semantic errors:

- Type mismatch.
- Undeclared variable.
- Reserved identifier misuse.
- Multiple declaration of variable in a scope.
- Accessing an out of scope variable.

Recap (Cont.)

Some typical semantic errors:

- Type mismatch.
- Undeclared variable.
- Reserved identifier misuse.
- Multiple declaration of variable in a scope.
- Accessing an out of scope variable.
- Actual and formal parameter mismatch.

Recap (Cont.)

Question

How do we do semantic analysis?

Recap (Cont.)

Question

How do we do semantic analysis?

Answer

One way to do it is to utilize **attribute grammars**.

Concepts

Attribute Grammar An **attribute grammar** consists of a context-free grammar augmented by a set of rules that specify computations.

Concepts

Attribute Grammar An **attribute grammar** consists of a context-free grammar augmented by a set of rules that specify computations.

Synthesized Attribute An attribute defined wholly in terms of the attributes of the node, its children, and constants. They are used or passing information downwards in the tree.

Concepts

Attribute Grammar An **attribute grammar** consists of a context-free grammar augmented by a set of rules that specify computations.

Synthesized Attribute An attribute defined wholly in terms of the attributes of the node, its children, and constants. They are used or passing information downwards in the tree.

Inherited Attribute An attribute defined wholly in terms of the nodes own attributes and those of its siblings or its parent in the parse tree (plus constants). They are used to pass information upwards.

Ingredients

An attribute grammar consists of:

Ingredients

An attribute grammar consists of:

- A underlying context free grammar.

Ingredients

An attribute grammar consists of:

- A underlying context free grammar.
- A description of which nonterminals have which attributes (synthesized or inherited).

Ingredients

An attribute grammar consists of:

- A underlying context free grammar.
- A description of which nonterminals have which attributes (synthesized or inherited).
- For each production, a description how to compute the:
  - Inherited attributes of the non-terminals in the RHS.
  - Synthesized attributes of the non-terminal at the LHS.

Examples

An example of synthesized attributes and how to do the evaluation:

PRODUCTION	SEMANTIC RULE
$L \to E\text{n}$	$print(E.val)$
$E \to E_1 + T$	$E.val := E_1.val + T.val$
$E \to T$	$E.val := T.val$
$T \to T_1 * F$	$T.val := T_1.val * F.val$
$T \to F$	$T.val := F.val$
$F \to (E)$	$F.val := E.val$
$F \to \text{digit}$	$F.val := \text{digit}.lexval$

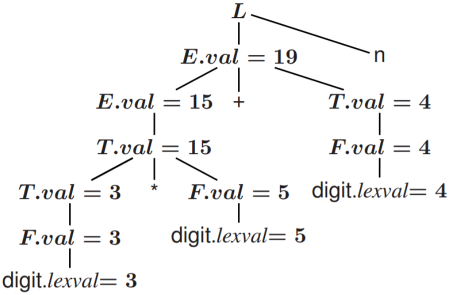

Figure 1: $3 * 5 + 4n$.

Examples (Cont.)

An example of inherited attributes and how to do the evaluation:

PRODUCTION	SEMANTIC RULE
$D \to TL$	$L.in := T.type$
$T \to$int	$T.type :=integer$
$T \to$real	$T.type :=real$
$L \to L_1$, id	$L_1.in := L.in$; *addtype(*id*.entry, L.in)*
$L \to$ id	*addtype(*id*.entry, L.in)*

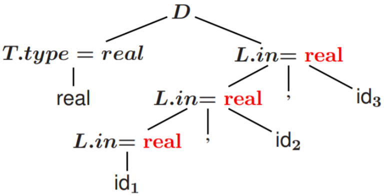

Figure 2: real id_1, id_2, id_3.

Evaluation

Goal

Each attribute value must be available when a computation is performed.

Evaluation

Goal

Each attribute value must be available when a computation is performed.

How?

Dependency graphs for representing attribute dependencies, and **topological sort** for obtaining the evaluation order.

Evaluation (Cont.)

Dependency Graph A **dependency graph** shows the interdependencies among the attributes of the various nodes of a parse-tree.

- A node for each attribute.
- An attribute b depending on anther attribute c is denoted by $b \leftarrow c$.

Evaluation (Cont.)

Dependency Graph A **dependency graph** shows the interdependencies among the attributes of the various nodes of a parse-tree.

- A node for each attribute.
- An attribute b depending on anther attribute c is denoted by $b \leftarrow c$.

Topological Sort Any ordering $s_1, \ldots, s_k$ such that if $s_i \leftarrow s_j$ is a link in the dependency graph then $s_i > s_j$.

Evaluation (Cont.)

Examples

Can you draw the dependency graphs for the previous two examples?

Evaluation (Cont.)

Examples

Can you draw the dependency graphs for the previous two examples?

Something Real

Let us check out a real example for our project.