Semantic Analysis

Scanner and parser accept programs with correct syntax.
- not all programs that are syntactically correct make sense
- the compiler may be able to recognize some of these
  - these programs are rejected and reported as erroneous
  - the language definition defines what erroneous means
- semantic analyses are necessary that, for instance:
  - check that identifiers are known and where they are defined
  - check the type-correct use of variables
Semantic Analysis

- Scanner and parser accept programs with correct syntax.
  - Not all programs that are syntactically correct make sense.
  - The compiler may be able to recognize some of these.
    - These programs are rejected and reported as erroneous.
    - The language definition defines what erroneous means.
  - Semantic analyses are necessary that, for instance:
    - Check that identifiers are known and where they are defined.
    - Check the type-correct use of variables.
  - Semantic analyses are also useful to:
    - Find possibilities to "optimize" the program.
    - Warn about possibly incorrect programs.

Semantic Analysis

- A semantic analysis annotates the syntax tree with attributes.

Attribute Grammars

- Many computations of the semantic analysis as well as the code generation operate on the syntax tree.
- What is computed at a given node only depends on the type of that node (which is usually a non-terminal).
- We call this a local computation:
  - Only accesses already computed information from neighbouring nodes.
  - Computes new information for the current node and other neighbouring nodes.

Chapter 1: Attribute Grammars
Attribute Grammars

- Many computations of the semantic analysis as well as the code generation operate on the syntax tree.
- What is computed at a given node only depends on the type of that node (which is usually a non-terminal).
- We call this a local computation:
  - Only accesses already computed information from neighbouring nodes.
  - Computes new information for the current node and other neighbouring nodes.

**Definition: Attribute Grammar**

An attribute grammar is a CFG extended by:

- A set of attributes for each non-terminal and terminal.
- Local attribute equations.

- In order to be able to evaluate the attribute equations, all attributes mentioned in that equation have to be evaluated already.
- The nodes of the syntax tree need to be visited in a certain sequence.

Example: Computation of the empty[r] Attribute

Consider the syntax tree of the regular expression (ab)*a(ab):

```
         *  
        /   
       *    
      /     
     l     l  
    /   
   0    1  
  /   
 2    a  
 /   
3    a  
/   
4    b
```

Example: Computation of the empty[r] Attribute

Consider the syntax tree of the regular expression (ab)*a(ab):

```
         *  
        /   
       *    
      /     
     l     l  
    /   
   0    1  
  /   
 2    a  
 /   
3    a  
/   
4    b
```
Example: Computation of the $\text{empty}^r$ Attribute

Consider the syntax tree of the regular expression $(ab)^*a(ab)$:

Implementation Strategy

- attach an attribute $\text{empty}$ to every node of the syntax tree
- compute the attributes in a depth-first post-order traversal:
  - at a leaf, we can compute the value of $\text{empty}$ without considering other nodes
  - the attribute of an inner node only depends on the attribute of its children
- the $\text{empty}$ attribute is a **synthetic** attribute
- The local dependencies between the attributes are dependent on the type of the node

Definition

An attribute is called
- synthetic if its value is always propagated upwards in the tree (in the direction leaf $\rightarrow$ root)
- inherited if its value is always propagated downwards in the tree (in the direction root $\rightarrow$ leaf)
Attribute Equations for \textit{empty}

In order to compute an attribute \textit{locally}, we need to specify attribute equations for each node. These equations depend on the \textit{type} of the node:

for leaves: \( r = x \) we define \( \text{empty}[r] = (x \equiv \epsilon) \).

otherwise:

\[
\begin{align*}
\text{empty}[r] &= \text{empty}[r_1] \lor \text{empty}[r_2] \\
\text{empty}[r_1] &= \text{empty}[r_1] \land \text{empty}[r_2] \\
\text{empty}[r_1] &= t \\
\text{empty}[r_2] &= t
\end{align*}
\]

Specification of General Attribute Systems

\textbf{General Attribute Systems}

In general, for establishing attribute systems we need a flexible way to \textit{refer to parents and children}:

\textit{\rightarrow} We use consecutive indices to refer to neighbouring attributes

\[
\begin{align*}
\text{attribute}_r[0] : & \quad \text{the attribute of the current root node} \\
\text{attribute}_r[i] : & \quad \text{the attribute of the} \ i\text{-th child} \quad (i > 0)
\end{align*}
\]

Observations

- the \textit{local} attribute equations need to be evaluated using a \textit{global} algorithm that knows about the dependencies of the equations
- in order to construct this algorithm, we need
  \begin{itemize}
  \item a sequence in which the nodes of the tree are visited
  \item a sequence within each node in which the equations are evaluated
  \end{itemize}
- this \textit{evaluation strategy} has to be compatible with the \textit{dependencies} between attributes

... in the example:

\[
\begin{align*}
\text{empty}[0] & := (x \equiv \epsilon) \\
\text{empty}[1] & := \text{empty}[1] \lor \text{empty}[2] \\
\text{empty}[2] & := \text{empty}[1] \land \text{empty}[2] \\
\text{empty}[0] & := t \\
\text{empty}[0] & := t
\end{align*}
\]
Observations

- the local attribute equations need to be evaluated using a global algorithm that knows about the dependencies of the equations
- in order to construct this algorithm, we need a sequence in which the nodes of the tree are visited
- a sequence within each node in which the equations are evaluated
- this evaluation strategy has to be compatible with the dependencies between attributes

We visualize the attribute dependencies $D(p)$ of a production $p$ in a Local Dependency Graph:

\[ D(p) = \{ (\text{empty}[1], \text{empty}[0]), (\text{empty}[2], \text{empty}[0]) \} \]

\[ D(S \to E) = \{ (\text{empty}[1], \text{empty}[0]), (\text{first}[1], \text{first}[0]) \} \]

\[ D(E \to x) = \{ \} \]

Simultaneous Computation of Multiple Attributes

Computing empty, first, next from regular expressions:

\[ S \to E : \]

\[ \text{empty}[0] := \text{empty}[1] \]
\[ \text{first}[0] := \text{first}[1] \]
\[ \text{next}[1] := 0 \]

\[ E \to x : \]

\[ \text{empty}[0] := (x \equiv \varepsilon) \]
\[ \text{first}[0] := \{ x \mid x \neq \varepsilon \} \]
// (no equation for next )

Regular Expressions: Rules for Alternative

\[ E \to E | E : \]

\[ \text{empty}[0] := \text{empty}[1] \lor \text{empty}[2] \]
\[ \text{first}[0] := \text{first}[1] \lor \text{first}[2] \]
\[ \text{next}[1] := \text{next}[0] \]
\[ \text{next}[2] := \text{next}[0] \]

\[ D(E \to E | E) : \]

\[ D(E \to E | E) = \{ \{\text{empty}[1], \text{empty}[0]\}, \{\text{empty}[2], \text{empty}[0]\}, \{\text{first}[1], \text{first}[0]\}, \{\text{first}[2], \text{first}[0]\}, \{\text{next}[0], \text{next}[2]\}, \{\text{next}[0], \text{next}[1]\} \} \]

Regular Expressions: Rules for Concatenation

\[ E \to E.E : \]

\[ \text{empty}[0] := \text{empty}[1] \land \text{empty}[2] \]
\[ \text{first}[0] := \text{first}[1] \land (\text{empty}[1] \lor \text{empty}[2] \lor \text{next}[0] \lor \text{next}[1]) \]
\[ \text{next}[1] := \text{next}[0] \]
\[ \text{next}[2] := \text{next}[0] \]

\[ D(E \to E.E) : \]

\[ D(E \to E.E) = \{ \{\text{empty}[1], \text{empty}[0]\}, \{\text{empty}[2], \text{empty}[0]\}, \{\text{first}[1], \text{first}[0]\}, \{\text{first}[2], \text{first}[0]\}, \{\text{next}[0], \text{next}[2]\}, \{\text{next}[0], \text{next}[1]\} \} \]
Regular Expressions: Kleene-Star and ‘?’

\[ E \rightarrow E^* : \]
- empty[0] := \text{t}
- first[0] := first[1] ∪ next[0]
- next[1] := next[0]

\[ E \rightarrow E? : \]
- empty[0] := \text{t}
- first[0] := first[1]
- next[1] := next[0]

\[ D(E \rightarrow E^*) : \]
- D(E) = \{ (first[1], first[0]), (first[1], next[2]), (next[0], next[1]) \}

\[ D(E \rightarrow E?) : \]
- D(E) = \{ (first[1], first[0]), (next[0], next[1]) \}

Challenges for General Attribute Systems

Static evaluation
Is there a static evaluation strategy, which is generally applicable?

- an evaluation strategy can only exist, if for any derivation tree the dependencies between attributes are acyclic
- it is \textit{DEXTIME}-complete to check for cyclic dependencies
  [Jazayeri, Odgen, Rounds, 1975]

Subclass: Strongly Acyclic Attribute Dependencies

Idea: For all nonterminals \( X \) compute a set \( \mathcal{R}(X) \) of relations between its attributes, as an \textit{overapproximation of the global dependencies} between root attributes of every production for \( X \).

Describe \( \mathcal{R}(X) \) as sets of relations, similar to \( D(p) \) by

- setting up each production \( X \rightarrow X_1 \ldots X_k \)'s effect on the relations of \( \mathcal{R}(X) \)
- compute effect on all so far accumulated evaluations of each rhs \( X_i \)'s \( \mathcal{R}(X_i) \)
- iterate until stable

Ideas
- Let the User specify the strategy
- Determine the strategy dynamically
- Automate subclasses only
**Subclass: Strongly Acyclic Attribute Dependencies**

The 3-ary operator $L[p,i]$ re-decorates relations from $L$

$$L[p,i] = \{ (p.a[i], p.b[i]) \mid (a, b) \in L \}$$

$\pi_0$ projects only onto relations between root elements only

$$\pi_0(S) = \{ (a, b) \mid (p.a[0], p.b[0]) \in S \}$$

**Subclass: Strongly Acyclic Attribute Dependencies**

The 3-ary operator $L[p,i]$ re-decorates relations from $L$

$$L[p,i] = \{ (p.a[i], p.b[i]) \mid (a, b) \in L \}$$

$\pi_0$ projects only onto relations between root elements only

$$\pi_0(S) = \{ (a, b) \mid (p.a[0], p.b[0]) \in S \}$$

root-projects the transitive closure of relations from the $L,s$ and $D(p)$

$$[p]^{\pi_0}(L_1, \ldots, L_k) = \pi_0((D(p) \cup L_1[p,1] \cup \ldots \cup L_k[p,k])^+)$$

**Subclass: Strongly Acyclic Attribute Dependencies**

The 3-ary operator $L[p,i]$ re-decorates relations from $L$

$$L[p,i] = \{ (p.a[i], p.b[i]) \mid (a, b) \in L \}$$

$\pi_0$ projects only onto relations between root elements only

$$\pi_0(S) = \{ (a, b) \mid (p.a[0], p.b[0]) \in S \}$$

root-projects the transitive closure of relations from the $L,s$ and $D(p)$

$$[p]^{\pi_0}(L_1, \ldots, L_k) = \pi_0((D(p) \cup L_1[p,1] \cup \ldots \cup L_k[p,k])^+)$$

$\mathcal{R}$ maps symbols to relations (global attributes dependencies)

$$\mathcal{R}(X) = \bigcup \{ [p] \mid \mathcal{R}(X_1), \ldots, \mathcal{R}(X_k) \mid p \rightarrow X_1 \ldots X_k \mid X \in N \}$$

$$\mathcal{R}(X) \supseteq \emptyset \quad \forall X \in N \quad \land \quad \mathcal{R}(a) = \emptyset \quad \forall a \in T$$
Subclass: Strongly Acyclic Attribute Dependencies

The 3-ary operator \( L[p,i] \) re-decorates relations from \( L \)

\[ L[p,i] = \{(p,a[i],p,b[i]) \mid (a,b) \in L\} \]

\( \pi_0 \) projects only onto relations between root elements only

\[ \pi_0(S) = \{(a,b) \mid (a,a[0],p,b[0]) \in S\} \]

root-projects the transitive closure of relations from the \( L_i \)s and \( D(p) \)

\[ [p]^\dagger(L_1, \ldots, L_k) = \pi_0((D(p) \cup L_1[p,1] \cup \ldots \cup L_k[p,k])^+) \]

\( \mathcal{R} \) maps symbols to relations (global attributes dependencies)

\[ \mathcal{R}(X) = \bigcup\{(p)^\dagger(\mathcal{R}(X_1), \ldots, \mathcal{R}(X_k)) \mid p : X \rightarrow X_1 \ldots X_k\} \mid X \in N \]

\[ \mathcal{R}(X) \supseteq \emptyset \quad \land \quad \mathcal{R}(a) = \emptyset \quad \land \quad a \in T \]

Strongly Acyclic Grammars

The system of inequalities \( \mathcal{R}(X) \)

- characterizes the class of strongly acyclic Dependencies
- has a unique least solution \( \mathcal{R}^*(X) \) (as \( [\cdot]^\dagger \) is monotonic)

Example: Strong Acyclic Test

Given grammar \( S \rightarrow L \), \( L \rightarrow a \mid b \), Dependency graphs \( D_p \):

Start with computing \( \mathcal{R}(L) = ([L \rightarrow a]^\dagger) \cup [L \rightarrow b]^\dagger \) :

\[ \begin{align*}
& h \quad j \quad L \\
& a
\end{align*} \]

\[ \begin{align*}
& h \quad j \quad L \\
& b
\end{align*} \]

\( \bullet \) terminal symbols do not contribute dependencies
Subclass: Strongly Acyclic Attribute Dependencies

The 3-ary operator \( L[p,i] \) re-decorates relations from \( L \)

\[ L[p,i] = \{ (p.a[i], p.b[i]) | (a, b) \in L \} \]

\( \pi_0 \) projects only onto relations between root elements only

\[ \pi_0(S) = \{ (a, b) | (p.a[0], p.b[0]) \in S \} \]

root-projects the transitive closure of relations from the \( L_i \)s and \( D(p) \)

\[ [p]^+(L_1, \ldots, L_k) = \pi_0((D(p) \cup L_1[p,1] \cup \ldots \cup L_k[p,k])^+) \]

\( \mathcal{R} \) maps symbols to relations (global attributes dependencies)

\[ \mathcal{R}(X) = \bigcup \{ [p]^+(\mathcal{R}(X_1), \ldots, \mathcal{R}(X_k)) \mid p : X \rightarrow X_1 \ldots X_k \mid X \in N \} \]

\[ \mathcal{R}(X) \supseteq \emptyset \quad \text{and} \quad \mathcal{R}(a) = \emptyset \mid a \in T \]

Strongly Acyclic Grammars

The system of inequalities \( \mathcal{R}(X) \)

- characterizes the class of strongly acyclic Dependencies
- has a unique least solution \( \mathcal{R}^*(X) \) (as \([1]^+\) is monotonic)

Example: Strong Acyclic Test

Start with computing \( \mathcal{R}(L) = [L\rightarrow a]^+ \cup [L\rightarrow b]^+ \):

- terminal symbols do not contribute dependencies
- transitive closure of all relations in \((D(L\rightarrow a))^+\) and \((D(L\rightarrow b))^+\)

Example: Strong Acyclic Test

Start with computing \( \mathcal{R}(L) = [L\rightarrow a]^+ \cup [L\rightarrow b]^+ \):

- terminal symbols do not contribute dependencies
- transitive closure of all relations in \((D(L\rightarrow a))^+\) and \((D(L\rightarrow b))^+\)
- apply \( \pi_0 \)
Example: Strong Acyclic Test

Start with computing $\mathcal{R}(L) = [L \rightarrow a]^+() \cup [L \rightarrow b]^+()$:

- terminal symbols do not contribute dependencies
- transitive closure of all relations in $(D(L \rightarrow a))^+$ and $(D(L \rightarrow b))^+$
- apply $\pi_0$
- $\mathcal{R}(L) = \{(k, j), (i, h)\}$

Example: Strong Acyclic Test

Continue with $\mathcal{R}(S) = [S \rightarrow L]^+(\mathcal{R}(L))$:

- re-decorate $\mathcal{R}(L)$ via $L[S \rightarrow L, 1]$

Example: Strong Acyclic Test

Continue with $\mathcal{R}(S) = [S \rightarrow L]^+(\mathcal{R}(L))$:

- re-decorate $\mathcal{R}(L)$ via $L[S \rightarrow L, 1]$
- transitive closure of all relations $(D(S \rightarrow L) \cup \{(p, k[1], p, j[1])\}) \cup \{(p, i[1], p, h[1])\})^+$

Example: Strong Acyclic Test

Continue with $\mathcal{R}(S) = [S \rightarrow L]^+(\mathcal{R}(L))$:

- re-decorate $\mathcal{R}(L)$ via $L[S \rightarrow L, 1]$
- transitive closure of all relations $(D(S \rightarrow L) \cup \{(p, k[1], p, j[1])\}) \cup \{(p, i[1], p, h[1])\})^+$

Example: Strong Acyclic Test

Continue with $\mathcal{R}(S) = [S \rightarrow L]^+(\mathcal{R}(L))$:

- re-decorate $\mathcal{R}(L)$ via $L[S \rightarrow L, 1]$
- transitive closure of all relations $(D(S \rightarrow L) \cup \{(p, k[1], p, j[1])\}) \cup \{(p, i[1], p, h[1])\})^+$
Example: Strong Acyclic Test

Start with computing $\mathcal{R}(L) = [L \rightarrow a]^\dagger \cup [L \rightarrow b]^\dagger$:

- Terminal symbols do not contribute dependencies
- Transitive closure of all relations in $(D(L \rightarrow a))^+$ and $(D(L \rightarrow b))^+$
- Apply $\pi_0$
- $\mathcal{R}(L) = \{(k, j), (i, h)\}$

Example: Strong Acyclic Test

Continue with $\mathcal{R}(S) = [S \rightarrow L]^\dagger(\mathcal{R}(L))$:

- Re-decorate $\mathcal{R}(L)$ via $L[S \rightarrow L, 1]$
- Transitive closure of all relations $(D(S \rightarrow L) \cup \{(p, k[1], p, j[1])\} \cup \{(p, i[1], p, h[1])\})^+$

Example: Strong Acyclic Test

Continue with $\mathcal{R}(S) = [S \rightarrow L]^\dagger(\mathcal{R}(L))$:

- Re-decorate $\mathcal{R}(L)$ via $L[S \rightarrow L, 1]$
- Transitive closure of all relations $(D(S \rightarrow L) \cup \{(p, k[1], p, j[1])\} \cup \{(p, i[1], p, h[1])\})^+$
- Apply $\pi_0$

Strong Acyclic and Acyclic

The grammar $S \rightarrow L, L \rightarrow a \mid b$ has only two derivation trees which are both acyclic:

It is not strongly acyclic since the dependence graph for the non-terminal $L$ contribute to a cycle when computing $\mathcal{R}(S)$:
From Dependencies to Evaluation Strategies

Possible strategies: