CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
Phases of a compiler

Figure 1.6, page 5 of text

Semantic analysis

Symbol Table

Character stream
Lexical Analyzer
Token stream
Syntax Analyzer
Syntax tree
Semantic Analyzer
Syntax tree
Intermediate Code Generator
Intermediate representation
Machine-Independent Code Optimizer
Intermediate representation
Code Generator
target-machine code
Machine-Dependent Code Optimizer
target-machine code
Recursive records

A record type must allow a component to be of the same type as the type itself:

type Node: [ integer datum ; Node rest ]
Recursive records

A record type must allow a component to be of the same type as the type itself:

```
type Node: [ integer datum ; Node rest ]
```

Be careful how you process declaration: you need to ensure that the second occurrence of Node does not trigger an undefined name.
Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text
HLL 1 → IR → target 1
HLL 2 → IR → target 2
... → IR → ...
HLL m → IR → target n
Machine independent optimizations → IR

Machine dependent optimizations → target 1, target 2, ..., target n
HLL

Sprint 2 or 3

IR

target 1

Sprint 2, 3, or 4

machine independent optimizations

Milestone 4 (maybe some simple ones)
Intermediate Representations
Ex. 6.1 [p 359]

\[ a + a \times (b - c) + (b - c) \times d \]
Ex. 6.1 [p 359]

$$a + a \ast (b - c) + (b - c) \ast d$$
Directed Acyclic Graph (DAG)

- Similar to a syntax tree
- No repeated nodes: structure sharing
Ex. 6.1 [p 359]

\[ a + a \ast (b - c) + (b - c) \ast d \]
Ex. 6.1 [p 359]

\[ a + a \times (b - c) + (b - c) \times d \]

Things can be more complicated if expressions have side effects.
<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $E \rightarrow E_1 + T$</td>
<td>$E$.node = new Node('+', $E_1$.node, $T$.node)</td>
</tr>
<tr>
<td>2 $E \rightarrow E_1 - T$</td>
<td>$E$.node = new Node('-', $E_1$.node, $T$.node)</td>
</tr>
<tr>
<td>3 $E \rightarrow E_1 * T$</td>
<td>$E$.node = new Node('*', $E_1$.node, $T$.node)</td>
</tr>
<tr>
<td>4 $E \rightarrow T$</td>
<td>$E$.node = $T$.node</td>
</tr>
<tr>
<td>5 $T \rightarrow ( E )$</td>
<td>$T$.node = $E$.node</td>
</tr>
<tr>
<td>6 $T \rightarrow id$</td>
<td>$T$.node = new Leaf(id, id.entry)</td>
</tr>
<tr>
<td>7 $T \rightarrow num$</td>
<td>$T$.node = new Leaf(num, num.val)</td>
</tr>
</tbody>
</table>

Figure 6.4 in text (p. 360), corrected according to errata sheet.
SDT

Tree or DAG

- SDT produces a tree if each call to Node creates a new tree node.
- SDT produces a DAG if for each call to Node there is a check whether this node already exists, and if so it returns a reference to the existing node rather than returning a new node.
### Example

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>Leaf(id, entry-a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_2$</td>
<td>Leaf(id, entry-a) = $p_1$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>Leaf(id, entry-b)</td>
</tr>
<tr>
<td>$p_4$</td>
<td>Leaf(id, entry-c)</td>
</tr>
<tr>
<td>$p_5$</td>
<td>Node('-', $p_3$, $p_4$)</td>
</tr>
<tr>
<td>$p_6$</td>
<td>Node('*', $p_1$, $p_5$)</td>
</tr>
<tr>
<td>$p_7$</td>
<td>Node('+', $p_1$, $p_6$)</td>
</tr>
<tr>
<td>$p_8$</td>
<td>Leaf(id, entry-b) = $p_3$</td>
</tr>
<tr>
<td>$p_9$</td>
<td>Leaf(id, entry-c) = $p_4$</td>
</tr>
<tr>
<td>$p_{10}$</td>
<td>Node('-', $p_3$, $p_4$) = $p_5$</td>
</tr>
<tr>
<td>$p_{11}$</td>
<td>Leaf(id, entry-d)</td>
</tr>
<tr>
<td>$p_{12}$</td>
<td>Node('*', $p_5$, $p_{11}$)</td>
</tr>
<tr>
<td>$p_{13}$</td>
<td>Node('+', $p_7$, $p_{12}$)</td>
</tr>
</tbody>
</table>
Value-number method
Algorithm 6.3 [p. 361]

Input: label op, node l, node r

Output: The value number of a node in the array with signature <op,l,r>

Method: Search the array for a node M with signature <op,l,r>. If there is such a node, return the value number of M. If not, create in the array a new node N with signature <op,l,r> and return its value number.
Value-number method
Algorithm 6.3 [p. 361]

- **Input:** label op, node l, node r
- **Output:** The value number of a node in the array with signature \( <op,l,r> \)
- **Method:** Search the array for a node \( M \) with signature \( <op,l,r> \). If there is such a node, return the value number of \( M \). If not, create in the array a new node \( N \) with signature \( <op,l,r> \) and return its value number. Can use hash table for efficiency.
Revisiting 6.1
see construction steps in figure 6.5 [p. 360]

<table>
<thead>
<tr>
<th></th>
<th>id</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>*</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>+</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>*</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>+</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>