Translator Architecture

Tokenizer → Parser → Code Generator

- string of characters (source code)
- string of tokens
- abstract program
- string of integers (object code)

Execute_Program

```plaintext
procedure_body Execute_Program ( preserves Program& p )
{

}
```
Execute_Statement

procedure_body Execute_Statement (  
    preserves Statement& s,  
    preserves Program& p  
)
{

}

BL Program Execution

- Execution of a BL program as Program object seems simple when done as one or more recursive operations cooperating to execute the whole program
- What about executing the program one “step” at a time?
**Program Execution Continued...**

- The problem seems to be related to the hierarchical, nested structure of Program objects.
- It is difficult to stop and resume execution, and to keep track of a specific “position” in the program.
- How can we execute a BL program in steps?

---

**Code Generation**

- Translate program to “linear” (i.e., non-nested) structure, e.g., string of low-level (assembly-like) instructions.
  - Primitive instructions are translated directly.
  - Control structures are replaced with jump instructions.
  - User-defined instructions are inlined.
An Example

IF next-is-enemy THEN
  infect
ELSE
  turnright
END IF

<table>
<thead>
<tr>
<th></th>
<th>JUMP_IF_NOT_NEXT_IS_ENEMY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>INFECT</td>
</tr>
<tr>
<td>3</td>
<td>JUMP</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>TURNRIGHT</td>
</tr>
<tr>
<td>6</td>
<td>...</td>
</tr>
</tbody>
</table>

Another Example

WHILE next-is-empty DO
  move
  infect
END WHILE

<table>
<thead>
<tr>
<th></th>
<th>JUMP_IF_NOT_NEXT_IS_EMPTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>MOVE</td>
</tr>
<tr>
<td>3</td>
<td>INFECT</td>
</tr>
<tr>
<td>4</td>
<td>JUMP</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>TURNLEFT</td>
</tr>
<tr>
<td>7</td>
<td>...</td>
</tr>
</tbody>
</table>
**BugsWorld Virtual Machine**

- Memory
- Instruction set (target language)
- Program counter

---

**Target Language**

- Primitive instructions
- Unconditional jump
- Conditional jumps
Primitive Instructions
- MOVE
- TURNLEFT
- TURNRIGHT
- INFECT
- SKIP
- HALT

Unconditional Jump
- JUMP addr

Conditional Jumps
- JUMP_IF_NOT_NEXT_IS_EMPTY addr
- JUMP_IF_NOT_NEXT_IS_NOT_EMPTY addr
- JUMP_IF_NOT_NEXT_IS_ENEMY addr
- JUMP_IF_NOT_NEXT_IS_NOT_ENEMY addr
- JUMP_IF_NOT_NEXT_IS_FRIEND addr
- JUMP_IF_NOT_NEXT_IS_NOT_FRIEND addr
- JUMP_IF_NOT_NEXT_IS_WALL addr
- JUMP_IF_NOT_NEXT_IS_NOT_WALL addr
- JUMP_IF_NOT_RANDOM addr
- JUMP_IF_NOT_TRUE addr
**Handling IF Statements**

IF condition THEN
  block
END IF

**Handling IF_ELSE Statements**

IF condition THEN
  if_block
ELSE
  else_block
END IF
### Handling WHILE Statements

```
WHILE condition DO
  block
END WHILE
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>k-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k+1</td>
<td>k+n+4</td>
<td></td>
</tr>
<tr>
<td>k+2</td>
<td>block</td>
<td></td>
</tr>
<tr>
<td>k+3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k+n+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k+n+2</td>
<td>JUMP</td>
<td></td>
</tr>
<tr>
<td>k+n+3</td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>k+n+4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Handling CALL Statements

```
INSTRUCTION my-instruction IS
  block
END my-instruction
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>k-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k+n-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k+n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>