Compiler II: Code Generation

Shimon Schocken

Spring 2005

The big picture

- Syntax analysis: understanding the code
- Code generation: constructing semantics
Syntax analysis (review)

```plaintext
Class Bar {
    method Fraction foo(int y) {
        var int temp; // a variable
        let temp = (xxx+12)*-63;
        ...
    }
}
```

The code generation challenge:
- Extend the syntax analyzer into a full-blown compiler
- Program = a series of operations that manipulate data
- The compiler should convert each “understood” (parsed) source operation and data item into corresponding operations and data items in the target language
- So we have to generate code for:
  - handling data
  - handling operations.

Handling variables

- **What is x’s data type?**
  - Primitive, ADT (class name)?
    - (Need to know in order to allocate it to the RAM properly)

- **What kind of variable is x?**
  - local, static, field, argument?
    - (Need to know in order to manage its life cycle properly)
Symbol table

class BankAccount {
    // Class variables
    static int accounts;
    static int bankCommission;
    // account properties
    field int id;
    field String owner;
    field int balance;
}

method int commission(int x) {
    // Code omitted
}

method void transfer(int sum, BankAccount from, Date when) {
    var int i, j; // Some local variables
    var Date due; // Date is a user-defined type
    let balance = (balance + sum) - commission(sum * 5);
    // More code...
}

Classical implementation: a list of hash tables, each reflecting a single scope nested within the next one in the list
- The identifier lookup works from the current table upwards
- The depth of the list depends on the language.

The Elements of Computing Systems 5 Compiler II (Ch. 11)

Life cycle

Class-scope symbol table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Kind</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>accounts</td>
<td>int</td>
<td>static</td>
<td>0</td>
</tr>
<tr>
<td>bankCommission</td>
<td>int</td>
<td>static</td>
<td>1</td>
</tr>
<tr>
<td>id</td>
<td>int</td>
<td>field</td>
<td>0</td>
</tr>
<tr>
<td>owner</td>
<td>String</td>
<td>field</td>
<td>1</td>
</tr>
<tr>
<td>balance</td>
<td>int</td>
<td>field</td>
<td>2</td>
</tr>
</tbody>
</table>

Method-scope (transfer) symbol table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Kind</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>this BankAccount</td>
<td>argument</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>sum</td>
<td>int</td>
<td>argument</td>
<td>1</td>
</tr>
<tr>
<td>from BankAccount</td>
<td>argument</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>when Date</td>
<td>argument</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>int</td>
<td>var</td>
<td>0</td>
</tr>
<tr>
<td>j</td>
<td>int</td>
<td>var</td>
<td>1</td>
</tr>
<tr>
<td>due Date</td>
<td>Date</td>
<td>var</td>
<td>2</td>
</tr>
</tbody>
</table>

- Static: single copy must be kept alive throughout the program duration
- Field: different copies must be kept for each object
- Local: created on subroutine entry, killed on exit
- Argument: similar to local
- Good news: the VM handles all these details !!!
Handling arrays: memory allocation and access

Java code

```java
void foo(int k) {
    int x, y;
    int[] bar; // Declare an array
    ...
    // Construct the array:
    bar = new int[10];
    ...
    bar[k] = 19;
    ...
    Main.foo(2); // Call the foo method
    ...
}
```

Pseudo VM Code

```java
// bar[k]=19, or *(bar+k)=19
push bar
push k
add
// Use a pointer to access x[k]
pop addr // addr points to bar[k]
push 19
pop *addr // Set bar[k] to 19
```

Final VM Code

```
bar=new int(n)
Is typically handled by generating code that affects
bar=alloc(n)
```

The RAM state is shown just after executing `bar[k]=19`.

---

Handling objects: memory allocation

Java code

```java
class Complex {
    // Properties (fields):
    int re;  // Real part
    int im;  // Imaginary part
    ...
    /** Constructs a new Complex object. */
    public Complex(int aRe, int aIm) {
        re = aRe;
        im = aIm;
        ...
    }
    ...
    // The following code can be in any class:
    public void bla() {
        Complex a, b, c;
        ...
        a = new Complex(5,17);
        b = new Complex(12,192);
        ...
        c = a; // Only the reference is copied
        ...
    }
}
```

```
foo = new ClassName(...)
Is typically handled by generating code that affects
foo=alloc(n)
```

The RAM state is shown just after executing `foo = new ClassName(...)`.
Handling objects: operations

Java code

```java
class Complex {
    // Properties (fields):
    int re; // Real part
    int im; // Imaginary part
    ... // properties
    /** Constructs a new Complex object. */
    public Complex(int aRe, int aIm) {
        re = aRe;
        im = aIm;
    }
    ... // methods
    public void mult (int c) {
        re = re * c;
        im = im * c;
    }
    ...
}
```

Translating

```java
im = im * c:
*(this+1) = *(this+1)
times (argument 0)
```

Resulting semantics:

- Look up the symbol table
- Of course this should be written in the target language.

Handling objects: method calls

Java code

```java
class Complex {
    // Properties (fields):
    int re; // Real part
    int im; // Imaginary part
    ... // properties
    /** Constructs a new Complex object. */
    public Complex(int aRe, int aIm) {
        re = aRe;
        im = aIm;
    }
    ... // methods
    class Foo {
        ...
        public void foo() {
            Complex x;
            ...
            x = new Complex(1,2);
            x.mult(5);
        }
        ...
    }
}
```

Translating

```java
x.mult(5):
```

- Look up the symbol table
- Generated code:

```java
// x.mult(5):
push x
push 5
call mult
```

General rule: each method call

```java
foo.bar(v1,v2,...)
```

can be translated into

```java
push foo, push v1, push v2, ...
call bar
```
Generating code for expressions

\[
x + g(2, y, -z) \times 5
\]

Syntax analysis

Code generation

```
codeWrite(exp):
    if exp is a number \( n \) then output "push \( n \)"
    if exp is a variable \( v \) then output "push \( v \)"
    if \( \text{exp} = (\text{exp1} \ op \ \text{exp2}) \) then codeWrite(\text{exp1}), codeWrite(\text{exp2}), output "\text{op}".
    if \( \text{exp} = \text{op}(\text{exp1}) \) then codeWrite(\text{exp1}), output "\text{op}".
    if \( \text{exp} = f(\text{exp1} \ldots \text{expN}) \) then codeWrite(\text{exp1}) \ldots codeWrite(\text{expN}), output "call \( f \)".
```

Handling control flow (e.g. IF, WHILE)

<table>
<thead>
<tr>
<th>Source code</th>
<th>Generated code</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (cond) s1 else s2 ...</td>
<td>code for computing ~cond if-goto L1 code for executing s1 goto L2 label L1 code for executing s2 label L2 ...</td>
</tr>
<tr>
<td>while (cond) s1 ...</td>
<td>label L1 code for computing ~cond if-goto L2 code for executing s1 goto L1 label L2 label L2 ...</td>
</tr>
</tbody>
</table>
Final example

Perspective

- "Hard" Jack simplifications:
  - Primitive type system
  - No inheritance
  - No public class fields (e.g. must use `r.c.getRadius()` rather than `r.c.radius`)

- "Soft" Jack simplifications:
  - Limited control structures (no for, switch, ...)
  - Cumbersome handling of char types (cannot use `let x='c'`)

- Optimization
  - For example, `c++` will be translated into push `c`, push 1, add, pop `c`.
  - Many other examples of possible improvements ...
Susan asks you to invite some bachelors to her party. A “bachelor” is “an adult human male who has never been married.” How can this category definition help decide who of the following people should be invited?

- Arthur has been living happily with Alice for the last five years. They have a two-year-old daughter and have never officially married.
- Bruce was going to be drafted, so he arranged with his friend Barbara to have a justice of the peace marry them so he would be exempt. They have never lived together. He dates a number of women, and plans to have the marriage annulled as soon as he finds someone he wants to marry.
- Charlie is 17 years old. He lives at home with his parents and is in high school.
- David is 17 years old. He left home at 13, started a small business, and is now a successful young entrepreneur leading a playboy’s lifestyle in his penthouse apartment.
- Eli and Edgar are homosexual lovers who have been living together for many years.
- Faisal is allowed by the law of his native Abu Dhabi to have three wives. He currently has two and is interested in meeting another potential fiancée.
- Father Gregory is the bishop of the Catholic cathedral at Groton upon Thames.

Natural language is far more interesting than programming languages!