Prof. O. Nierstrasz
Marcus Denker, Philipp Bunge

11. Bytecode

Generating bytecode is the job of *IRBuilder*. This class is part of the *NewCompiler* package. There is a pre-build image available at

`http://www.iam.unibe.ch/~scg/Research/NewCompiler/`.

Here one example as seen in the lecture:

```
irMethod:= IRBuilder new
  numRargs: 1;
  addTemps: #(self);
  pushTemp: #self;
  send: #truncated;
  returnTop;
  ir.

aCompiledMethod  := irMethod compiledMethod.
```

The variable *aCompiledMethod* now contains the generated compiled method. This method can be executed:

```
aCompiledMethod valueWithReceiver:3.5 arguments: #()
```

Exercise 11.1: Expressions

With the help of IRBuilder, generate a method that calculates the expression `(3 + 2*2) factorial` and returns the result.

Exercise 11.2: Parameters

Change the code that you wrote for the first exercise to use parameters instead of hard coded numbers: `(a + b) factorial`. If executed with

```
aCompiledMethod valueWithReceiver: nil arguments: #(3 4)
```

it should print 5040.

Exercise 11.3: Instance Variables

Generate a method that adds two instance variables and returns the result. Test the code by running it on a Point, e.g., 3@4.

Exercise 11.4: Loops *[1]

The Squeak bytecode has support for jumps. Jumps are used to implement conditionals and loops in an efficient way. Generate a compiledMethod with `IRBuilder` that outputs the numbers 1 to 10 on the Transcript window.

Static Bytecode Analysis

With the help of InstructionStream and InstructionClient, we can easily interpret the code of a method. As an example, see class InstructionPrinter.

[1]The starred exercises are optional

Exercise 11.5: Counting Bytecodes

Implement a class that counts the message send bytecodes of a method. The code

```
SendCounter on: (Object compiledMethodAt: #halt)
```

should return 1.

Runtime Bytecode Analysis

Look at the method tallyInstructions: in the class ContextPart (class-side):

```
"This method uses the simulator to count the number of occurrences of
 each of the Smalltalk instructions executed during evaluation of aBlock.
 Results appear in order of the byteCode set."
| tallies |
 tallies := Bag new.
 thisContext sender
     runSimulated: aBlock
     contextAtEachStep: [:current | tallies add: current nextByte].
ˆtallies sortedElements
```

The method runSimulated: aBlock contextAtEachStep: [:current ...] executes aBlock and for each bytecode executed, the second argument block is evaluated with an instance of one of the subclasses of ContextPart as the argument.

```
ContextPart tallyInstructions: [3.14159 printString]
```

Exercise 11.6: Counting Number of Executed Bytecodes

Write a similar method named numberOfBytecodeExecuted: aBlock that returns the number of bytecode executed when evaluating the provided block. For instance:

```
ContextPart numberOfBytecodeExecuted: [3.14159 printString]
```

In total, the expression 3.14159 printString is evaluated by executing around 1000 bytecodes.

Exercise 11.7: Bytecode Covered *

When a method is invoked, not all the bytecode contained in this method are executed. For instance, when executing 3.14159 printString the method on: defined in the class WriteStream is executed, but only 90% of its bytecode are executed.

```
ContextPart bytecodeCovered: [3.14159 printString]
==>  #(#('WriteStream>>on:' 90) #('LimitedWriteStream>>nextPut:' 69)
#('Object>>species' 100) ...)
```

Please save the code and send it by mail to st-staff@iam.unibe.ch. Attach your written solutions that are not part of the source-code to the mail or hand them in as hardcopy at the beginning of the next exercise session. Your mail and solutions should be clearly marked with names and matrikel numbers of the solution authors.