The Design of an Experimental Programming Language and its Translator

The Nuua Programming Language

Èrik Campobadal Forés

June 17, 2019

Universitat Politècnica de Catalunya

Code Generator: Bytecode and Source File

PROBLEM

- Runtime exceptions require the source file information.
- Requires the file name, the line and the column.
- Each opcode require this information because it can fail.
- Very memory inefficient.

SOLUTION

- Register only the changes on the file, line or column.
- Guess the file, line and column based on the current opcode.

Code Generator: Bytecode and Source File

```
1  a: int = 10
2  b: int = a - 10
3  print a / b
```

(i) Input program

```
LOAD_C   R-00000 C-00000
LOAD_C   R-00002 C-00001
SUB_INT  R-00001 R-00000 R-00002
DIV_INT  R-00001 R-00000 R-00001
PRINT    R-00001
```

(ii) Bytecode generated

CONDITION

Highest index that is lower or equal to the crash index.

Opcode Index	**Line number**
0	1
3	2
10	3

(iii) Registered line changes

Figure 1 – Runtime exception

Code Generator Optimizations: Basic Constant Folding

IMPLEMENTED

- Very basic constant folding on lists and dictionaries.

IMPROVEMENT

- Reduce the number of opcodes.
- Improve runtime performance.

Code Generator Optimizations: Basic Constant Folding

```
print [1, 2, 3]
a: int = 3
print [1, 2, a]
```

(i) Input program

```
PRINT_C C-00000
LOAD_C  R-00000 C-00001
LOAD_C  R-00001 C-00002
LPUSH_C R-00001 C-00003
LPUSH_C R-00001 C-00004
LPUSH   R-00001 R-00000
PRINT   R-00001
```

(ii) Optimized bytecode generated

Figure 2 – List constant folding

Code Generator Optimizations: Register Allocation

PROBLEM

- How long is the value of a register needed?
- Can we re-use the registers?

IMPLEMENTED

- Linear scan register allocation.

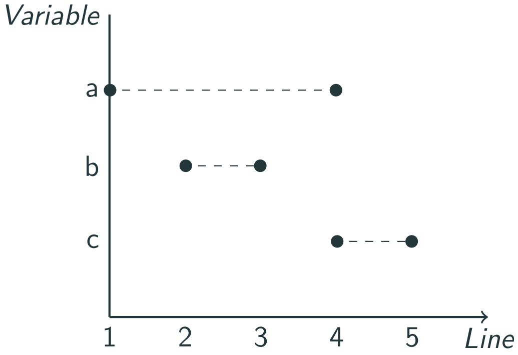
IMPROVEMENT

- Significant memory reduction.

Code Generator Optimizations: Register Allocation

```
a: int = 10
b: int = 20
print a + b
c: int = a
print c
```

(i) Input program



(ii) Variable lifetime

Figure 3 – Variable lifetime of a program

Code Generator Optimizations: Register Allocation

```
a: int = 10
b: int = 20
print a + b
c: int = a
print c
```

(i) Input program

```
LOAD_C  R-00000 C-00000
LOAD_C  R-00001 C-00001
ADD_INT R-00001 R-00000 R-00001
PRINT   R-00001
MOVE    R-00001 R-00000
PRINT   R-00001
```

(ii) Optimized bytecode generated

Figure 4 – Register allocation optimization of a program

Virtual Machine: Instruction Dispatch

PROBLEM

- Threaded dispatch is not ANSI C compilant.

SOLUTION

- Use a switch dispatch

TRADEOFF

- ANSI C compilant.
- Less efficient than a threaded dispatch.
- Simple to implement.

Virtual Machine: Global Frame and Call stack

IMPLEMENTED

- A global frame.
- A call stack.
- A shared value stack.
- A constant pool.

Virtual Machine: Global Frame and Call stack

```
print [1, 2, 3]
a: int = 3
print [1, 2, a]
```

(i) Input program

```
PRINT_C C-00000
LOAD_C  R-00000 C-00001
LOAD_C  R-00001 C-00002
LPUSH_C R-00001 C-00003
LPUSH_C R-00001 C-00004
LPUSH   R-00001 R-00000
PRINT   R-00001
```

(ii) Optimized bytecode generated

Figure 5 – Optimized register allocation of a program