Concepts of Programming Languages

Stack based Paradimgs

Timo Luerweg

Universität zu Lübeck

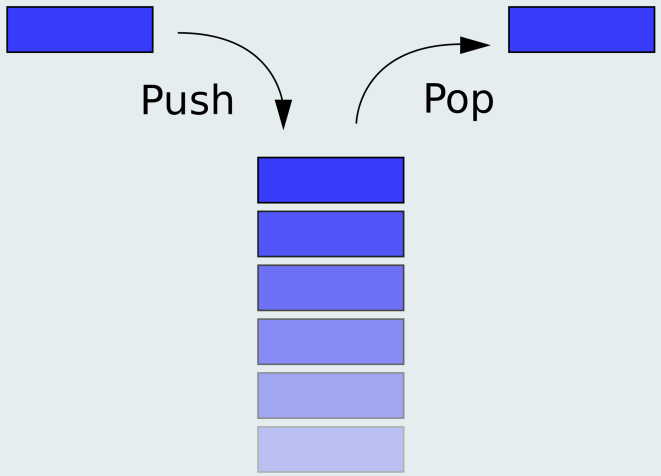
30. November 2015

Outline

Introduction

What are Stack-based Languages?

- Rely on one (or more) Stacks to execute programs
- Operations are executed on the top elements of the stack



Syntax

Stack based Languages use Reverse Polish Notation (Postfix)

Syntax

Stack based Languages use Reverse Polish Notation (Postfix)

- Notation reflects operation
- Doesn't need parentheses

Infix and Postfix

Example (Infix to Postfix)

$4 + 5 \cdot 8 \cdot (3 + 6) + 2$

Infix and Postfix

Example (Infix to Postfix)

$4 + 5 \cdot 8 \cdot (3 + 6) + 2$

$([4\ ([5\ 8\cdot][3\ 6\ +]\ \cdot\ ) +]2 +)$

Infix and Postfix

Example (Infix to Postfix)

$4 + 5 \cdot 8 \cdot (3 + 6) + 2$

$([4\ ([5\ 8\cdot][3\ 6\ +]\ \cdot\ ) +]2 +)$

$4\ 5\ 8 \cdot 3\ 6\ + \cdot + 2+$

Intermediate Languages

Intermediate Languages

Reasons for a frequent use are:

- Easy to generate code from
- Compact representation as Byte-Code
- Reduced set of instructions

Intermediate Languages

Reasons for a frequent use are:

- Easy to generate code from
- Compact representation as Byte-Code
- Reduced set of instructions
- Conversion with Shunting-yard algorithm

Intermediate Languages

Reasons for a frequent use are:

- Easy to generate code from
- Compact representation as Byte-Code
- Reduced set of instructions
- Conversion with Shunting-yard algorithm

Examples:

- Java Byte code (.class)
- Elisp byte-code
- PostScript

Meta Programming

Reasons for a possible use in this field are:

- Conceptually simple
- Extendible
- Code generation and modification at runtime possible

Forth

- Imperative Language
- Uses two stacks: Data and Return Stack
- Popular in Embedded Systems and Micro-controllers
- Different versions depending on “stack size”
- Untyped (5 + ”a” is possible)

Forth

Example (Hello World)

: HELLO ."Hello World " ;

Example ($4^2 \cdot 5^2$)

4 *dup* * 5 *dup* * * .

Joy

- Functional language
- Based on the composition of functions (not lambda calculus)
- Programs can pass whole structures as parameters

Joy

Example (Hello World)

"Hello world" (or)
"Hello" " world" concat

Example ($4^2 \cdot 5^2$)

4 *dup* $*$ 5 *dup* $**$ (or)
[4 5] [*dup* $*$]*map* 5 *get get* $*$

Variable Elimination

- Speed up execution
- Reduce **Load** and **Store** instructions

Variable Elimination

Requirements:

- Split able in blocks (of same size)
- Different branches leave same variables on the stack
- The stack is in the same state as before (Optional)

Variable Elimination

Requirements:

- Split able in blocks (of same size)
- Different branches leave same variables on the stack
- The stack is in the same state as before (Optional)
- Difference between **Local** and **Global** eliminaiton

Local Variable Elimination

- Algorithms exist(see example)
- Is performed on a single block of code
- only removes redundant variables

Example

```
a = b * c;
b = a / 8;
c = a - b;
```

Example

```
 1  (                  --) 76 LOCAL@    \ b @
 2  (          76      --) 77 LOCAL@    \ c @
 3  (76        77      --) *
 4  (          75      --) 75 LOCAL!    \ a !
 5  (                  --) 75 LOCAL@    \ a @
 6  (          75      --) 8            \ literal 8
 7  (75        88      --) /
 8  (          76      --) 76 LOCAL!    \ b !
 9  (                  --) 75 LOCAL@    \ a @ (DEAD)
10  (          75      --) 76 LOCAL@    \ b @
11  (75        76      --) -
12  (          77      --) 77 LOCAL!    \ c !
```

Example

```
 1  (                  --) 76 LOCAL@
 2  (          76      --) 77 LOCAL@
 3  (76        77      --) *
 4  (          75      --) DUP \ copy of 75
 5  (75        75      --) 75 LOCAL!
 6  (          75      --) NOP \ 75 LOCAL@
 7  (          75      --) 8
 8  (75        88      --) /
 9  (          76      --) 76 LOCAL!
10  (                  --) 75 LOCAL@
11  (          75      --) 76 LOCAL@
12  (75        76      --) -
13  (          77      --) 77 LOCAL!
```

Example

```
 1  (                                  --) 76 LOCAL@
 2  (                  76              --) 77 LOCAL@
 3  (          76      77              --) *
 4  (                  75              --) DUP
 5  (          75      75              --) 75 LOCAL!(DEAD)
 6  (                  75              --) NOP
 7  (                  75              --) 8
 8  (          75      88              --) UNDER
 9  (75        75      88              --) /
10  (          75      76              --) 76 LOCAL!
11  (                  75              --) NOP \ 75 LOCAL@
12  (                  75              --) 76 LOCAL@
13  (          75      76              --) -
14  (                  77              --) 77 LOCAL!
```

Example

```
 1  (                                  --) 76 LOCAL@
 2  (                  76              --) 77 LOCAL@
 3  (          76      77              --) *
 4  (                  75              --) DUP
 5  (          75      75              --) 75 LOCAL!(DEAD)\ *
 6  (                  75              --) NOP
 7  (                  75              --) 8
 8  (          75      88              --) UNDER
 9  (75        75      88              --) /
10  (          75      76              --) TUCK \ copy of 76
11  (76        75      76              --) 76 LOCAL!
12  (          76      75              --) NOP
13  (          76      75              --) SWAP \ 76 LOCAL@
14  (          75      76              --) -
15  (                  77              --) 77 LOCAL!
```

Example

```
 1  (                                  --) 76 LOCAL@
 2  (                    76            --) 77 LOCAL@
 3  (          76        77            --) *
 4  (                    75            --) 8
 5  (          75        88            --) UNDER
 6  (75        75        88            --) /
 7  (          75        76            --) DUP
 8  (76        75        76            --) 76 LOCAL!
 9  (          75        76            --) -
10  (                    77            --) 77 LOCAL!
```

Global Variable Elimination

- No algorithm at this point
- Can be applied manually

Results of Variable Elimination

- Results differ with different **Stack depths**
- Some programs are hard to optimize

Results of Variable Elimination

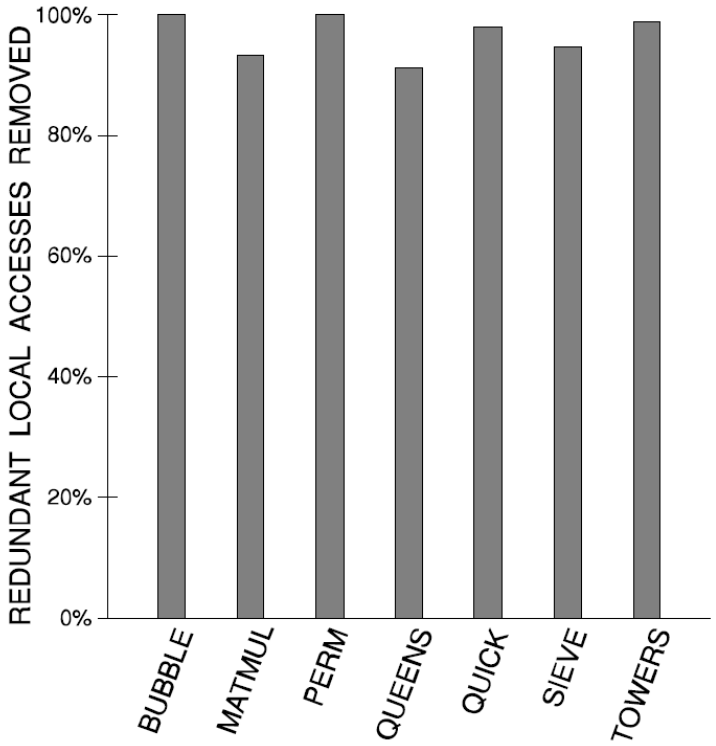


Figure: 1 Intra-block stack scheduling removes most redundant accesses to local variables.

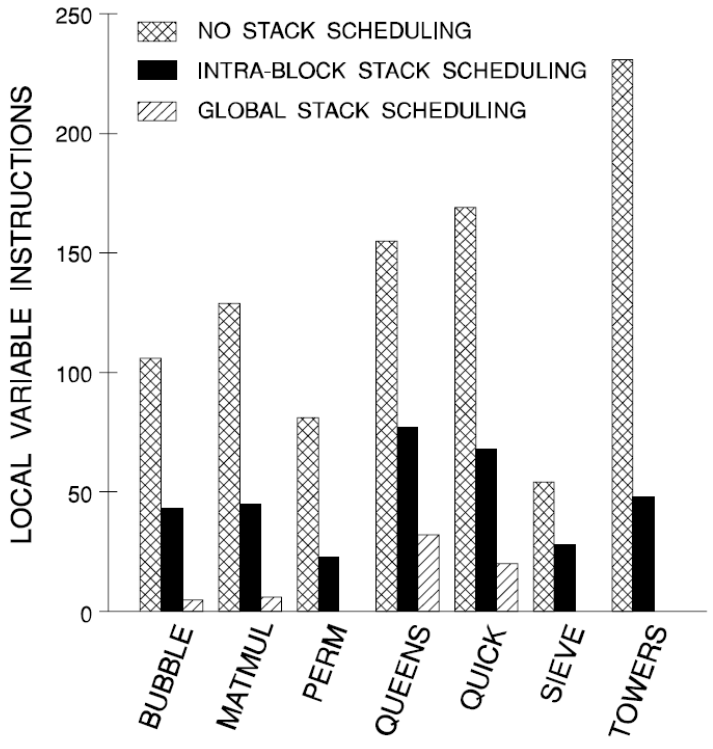


Figure: 2 The number of local variable instructions in the compiled code reduces dramatically with stack scheduling.

Thank you for your attention

Questions?

Conclusion

Stack based Languages are:

- Easy to extend
- Fast at execution
- Conceptually simple
- Compactly represented as Byte-code

Conclusion

However they are not frequently used because:

- Specialized on a certain field
- Implicit passing
- Unusual syntax