# Optimizing Functional Language Compilers

Manvi Gupta

Indian Institute of Technology Mandi

*b17092@students.iitmandi.ac.in*

MiniProject - May 2021

# Overview

# Functional Programming Languages

- When the program "only" consists of functions and their composition. Features:
  - Functions are first class citizens.
  - Function Purity
  - Referential Transparency

Let's compare the performance of Scheme (FPL) and C++ (IPL) in computing the $n^{th}$ Fibonacci Number

| $n$ | **C++** | **Scheme** |
|---|---|---|
| 20 | 0.002 | 0.002 |
| 30 | 0.039 | 0.090 |
| 40 | 0.925 | 10.968 |
| 45 | 9.720 | 118.916 |

Table: Time taken (in seconds) for fib(n) on Ubuntu x86 machine.

# Optimising FPLs

**Why is it hard?**

- Existing Von Neumann Architectures.
- Everything is a procedure $\rightarrow$ less opportunities to optimize.
- Too many functions and small function body.

**Why is it easy?**

- Purity: No side Effects.
- Immutable data structures, state-free.

*Let's look at some methods.*

# Short Cut Fusion in Haskell

How to multiply numbers from 1 to 10?

- **In Imperative Languages:**

```
int product = 1;
for (int i = 1; i <= 10; i++) {
  product *= i;
}
```

- **In Haskell:**

```
foldr (*) 1 [1..10]
```

Haskell code looks smaller, but is it?

*The number of intermediate lists created decreases efficiency!*

Solution: **Short Cut Fusion**

# Short Cut Fusion (Contd.)

- Method to avoid intermediate data structures. Saves memory.

We know that [1..10] is the same as `from 1 10`.

```
from a b = if a > b
           then []
           else a : from (a + 1) b
```

We can abstract out the definition to incorporate *cons* and *nil*:

```
from' a b c n = if a > b
                then n
                else c a (from' (a + 1) b c n)
```

# Short Cut Fusion Result

## Example

```
foldr (*) 1 (from 1 10)
    => from' 1 10 (*) 1
    => if 1 > 10
       then 1
       else 1 * (from' 2 10 (*) 1)
    => 1 * 2 * ... * 9 * 10 * 1
    => 3628800
```

- List elements are produced one by one and consumed immediately.
- **Result:** No intermediate lists created.

[0] [Seo, 2016]

# Globalization

## Globalization

"Safely" replacing function parameters by global variables.

- **Goal:** Reduce cost of stack allocation
- Currently for strict FPLs with "call by value"

# Globalization (Contd.)

## *Example*[1]

```
(define (f r n)
    (if (= n 0) r
        (f (* 2 r) (- n 1))))

(f 1 x)
```

If `r` and `n` do not interfere, it can be modified as:

```
(define R 1)  (define N x)

(define (f)
    (if (= N 0) R
        (begin (set! R (* 2 R)) (set! N (- N 1)) (f))))

(f)
```

[1] [Sestoft, 1988]

# Globalization: Checking for Interference

- Can a given parameter be safely globalised?
  $\rightarrow$ **Interference Analysis** based on
  - Def-Use Path
  - Path Semantics
  - Interference in a path
    (When value of a variable is modified before its last use.)
  - Def-Use Grammar
    (Program + Path Semantics)
- **A non-interfering variable is globalizable.**

# Conclusion

- Why is it difficult to optimize Functional Languages?
- Optimizing Techniques:
  - **Short Cut Fusion:** Method to avoid intermediate data structures.
  - **Globalization:** Safely replacing function parameters by global variables.
- Other Techniques:
  - Call-Arity Optimization
  - Eta-Conversion
  - Array bound Optimization, etc.

# References


Peter Sestoft (1988)

Replacing Function Parameters by Global Variables

*M.Sc. Thesis* 88-7-2, 107 pages

Andrew Gill, John Launchbury, Simon L Peyton Jones (1993)

A Short Cut to Deforestation

*FPCA 1993*

Kwang Yul Seo (2016)

Short Cut Fusion

https://kseo.github.io/posts/2016-12-18-short-cut-fusion.html

# The End