Benchmarking a baseline fully-in-place functional language compiler

autor: Jaromír Procházka | supervisor: Tomáš Petříček, Ph.D. | MFF UK | 2025

Introduction

Functional programs are readable and elegant, but may be inefficient because of allocation requirements. Recent paper introduced a novel approach to address this, termed fully in-place functional programming (FIP), which reuses no longer needed memory described by reuse tokens $\Diamond_k$ where k is the value size. This approach has previously been benchmarked in the Koka language.

Our goal is to design a minimal functional language *StaFip* that implements FIP - minimal to establish a strong baseline for benchmarking without unrelated features from Koka confounding the testing results. Testing algorithms are used to compare performance of FIP and non-FIP approaches.

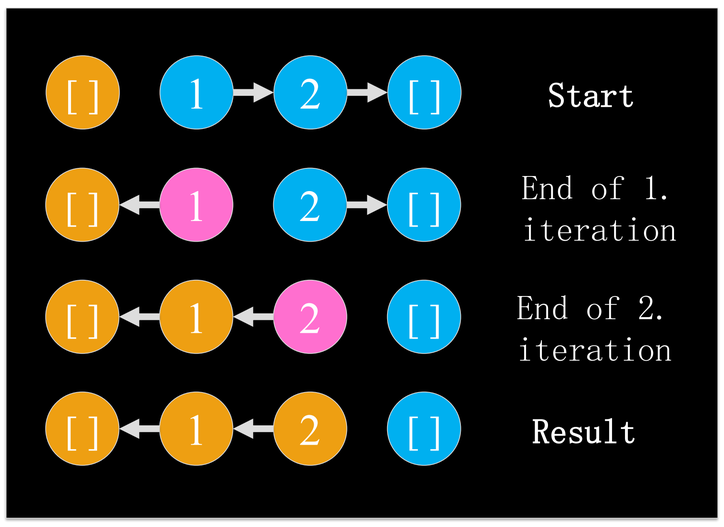
Materials and methods

The StaFip compiler is implemented using Flex and Bison tools.

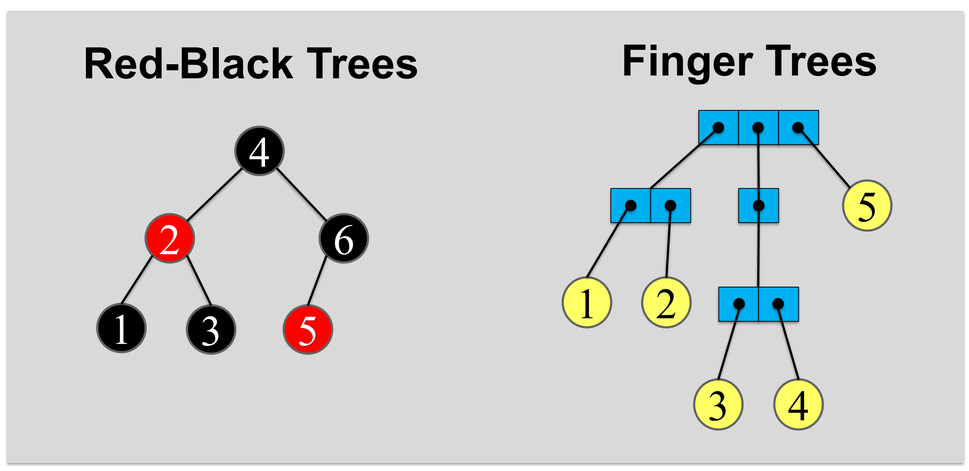
We use *Cecko Skeleton* framework for compiler implementation made by Bednárek D., Yaghob.

```
fip type list reverse [type list xs,
       type list acc]
    = match! xs -> type list {
        | Nil    -> acc
        | Cons(x, xx) //reuse token xs
        -> reverse(xx, Cons(x, acc) )
          //reuse token used by Cons
}
```

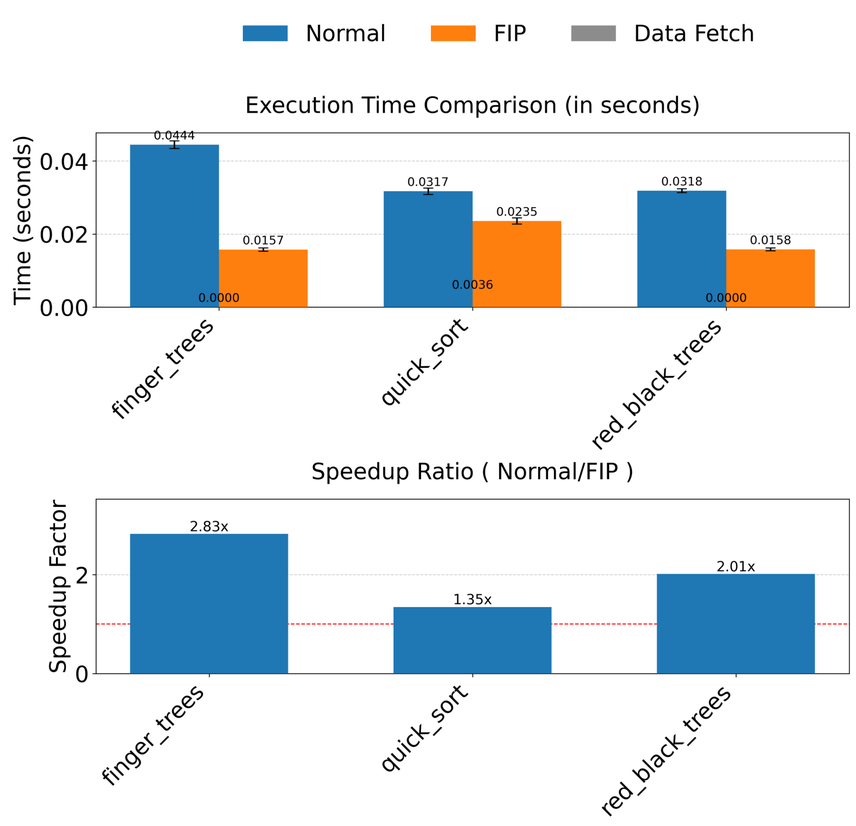
1. Step: design StaFip language (example of list reversal function in StaFip above)



Code run example. **Orange** is `acc`, **blue** is `xs`, **pink** is `x`. The `xs` head provides $\Diamond_2$ reuse token for `acc` head.



2. Step: implement *quicksort*, *red-black tree insertion*, and *finger tree insertion* algorithms in FIP and non-FIP forms of StaFip.



3. Step: Measure the difference between FIP and non-FIP approach. Results simillar to Koka.

Conclusions

We implemented a functional language with support for fully in-place updates inspired by Koka.

It demonstrates that the fully in-place (FIP) approach results in significant performance improvements. The performance profile of StaFip is similar to that reported for more complex Koka.

However, it also presents some difficulties in implementing the language compiler, such as the sensitivity of FIP to the implementation of data types and code bloat in the case of algorithm implementation.

Literature cited

Anton Lorenzen, Daan Leijen; Swierstra, Wouter. F P 2: Fully in-Place Functional Programming. Proceedings of the ACM on Programming Languages. 2023, 7:275–304.

Acknowledgments

I would like to thank my teachers, Tomáš Petříček, Ph.D., and doc. RNDr. Tomáš Dvořák, RNDr. David Bednárek, Ph.D. and RNDr. Jakub Yaghob, Ph.D.

Further information

GitHub: https://github.com/JaromirProchazka/FipCompiler

Cecko skeleton: https://gitlab.mff.cuni .cz/teaching/nswi098/cecko/skeleton