

Collapsing Heterogeneous Towers of Interpreters

Michael Buch – mb2244

University of Cambridge

June 7, 2019

Overview

Towers

Background

Project Aims

Collapsing a Tower

Heterogeneity

Experimental Tower

Summary

Table of Contents

Towers

Background

Project Aims

Collapsing a Tower

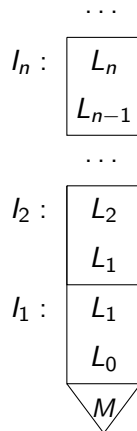
Heterogeneity

Experimental Tower

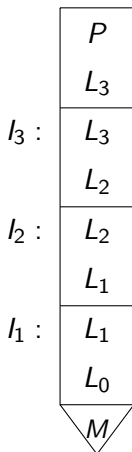
Summary

What are they?

- ▶ Reflective Towers
- ▶ Model for reflection
- ▶ Used for reflective languages, e.g., 3-LISP [1], Brown [2], Blond [3]
- ▶ **n-fold interpretative overhead**

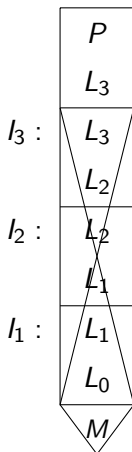


Removing Interpretative Overhead



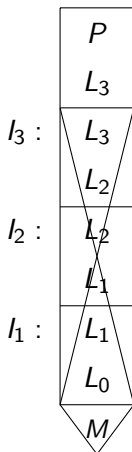
- ▶ Which combination of l_1 , l_2 , l_3 to remove?
 - ▶ Best case: simply run P on M

Removing Interpretative Overhead



- ▶ Which combination of l_1 , l_2 , l_3 to remove?
 - ▶ Best case: simply run P on M
- ▶ Idea: generate new tower without l_1 to l_3

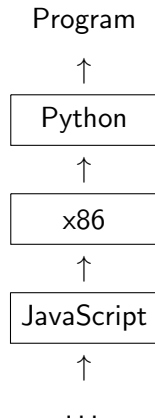
Removing Interpretative Overhead



- ▶ Which combination of l_1 , l_2 , l_3 to remove?
 - ▶ Best case: simply run P on M
- ▶ Idea: generate new tower without l_1 to l_3
 - ▶ Possible solution: turn interpretation into compilation

Motivating Example

- ▶ Python on x86 JavaScript emulator
- ▶ Compared to reflective towers:
 - ▶ *Different internal structure*, e.g., translation to bytecode
 - ▶ *Different language* at each level



Motivating Example

- ▶ Python on x86 JavaScript emulator
- ▶ Compared to reflective towers:
 - ▶ *Different internal structure*, e.g., translation to bytecode
 - ▶ *Different language* at each level
 - ▶ Need some way of expressing this **generalization**

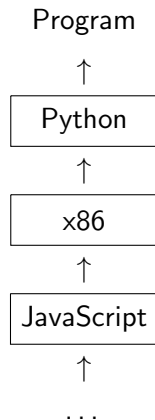


Table of Contents

Towers

Background

Project Aims

Collapsing a Tower

Heterogeneity

Experimental Tower

Summary

Staging

- ▶ How do we compile a tower of interpreters?

Staging

- ▶ How do we compile a tower of interpreters?
- ▶ interpreter $\stackrel{?}{=}$ compiler

Staging

- ▶ How do we compile a tower of interpreters?
- ▶ interpreter $\stackrel{?}{=}$ compiler
- ▶ Interpreter: directly executes source
- ▶ Compiler: translates source

Staging

- ▶ How do we compile a tower of interpreters?
- ▶ interpreter $\stackrel{?}{=}$ compiler
- ▶ Interpreter: directly executes source
- ▶ Compiler: translates source
- ▶ Staging: Split interpreter's execution into multiple stages, e.g., translation to some annotated source and then execute

Staging

- ▶ How do we compile a tower of interpreters?
- ▶ interpreter $\stackrel{?}{=}$ compiler
- ▶ Interpreter: directly executes source
- ▶ Compiler: translates source
- ▶ Staging: Split interpreter's execution into multiple stages, e.g., translation to some annotated source and then execute
- ▶ staged interpreter \simeq compiler

Staging

- ▶ How do we compile a tower of interpreters?
- ▶ $\text{interpreter} \stackrel{?}{=} \text{compiler}$
- ▶ Interpreter: directly executes source
- ▶ Compiler: translates source
- ▶ Staging: Split interpreter's execution into multiple stages, e.g., translation to some annotated source and then execute
- ▶ staged interpreter \simeq compiler
- ▶ 2nd Futamura Projection [4] shows we can use partial evaluator, *mix*, for compilation:

$$\begin{aligned} \text{target} &= \llbracket \text{mix} \rrbracket (\text{interpreter}, \text{source}) \\ &= \llbracket \text{compiler} \rrbracket (\text{source}) \end{aligned}$$



- ▶ Language Pink [5]
- ▶ Partial Evaluator
- ▶ Dynamic (Exp) vs. Static (Val)
- ▶ Normalization drives residualization
- ▶ *lift* operator wraps expressions which we want to keep in the residual program

Table of Contents

Towers

Background

Project Aims

Collapsing a Tower

Heterogeneity

Experimental Tower

Summary

Aims

- ▶ Construct and collapse heterogeneous towers of interpreters
- ▶ Explore effects of heterogeneity on collapse procedure

Table of Contents

Towers

Background

Project Aims

Collapsing a Tower

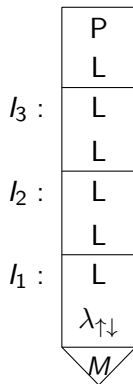
Heterogeneity

Experimental Tower

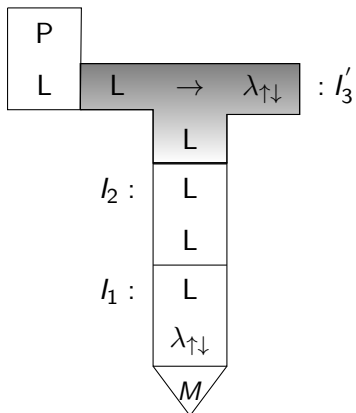
Summary

Ingredients

- ▶ Multi-level Language
- ▶ Stage Polymorphism
- ▶ TDPE-style Lift



Stage a Level



Residualize

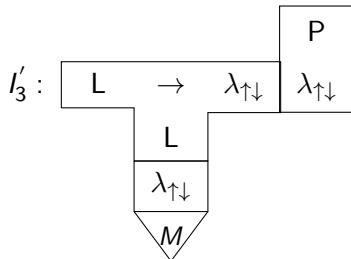


Table of Contents

Towers

Background

Project Aims

Collapsing a Tower

Heterogeneity

Experimental Tower

Summary

Heterogeneous Towers

- ▶ Generalization of reflective towers
- ▶ Permits:
 - ▶ non-meta-circular interpreters
 - ▶ *semantic gap* (i.e., difference in data representation and semantic between adjacent languages)

Methodology

- ▶ Construct a tower resembling Python-x86-JavaScript
- ▶ Collapse it (while staging at various heights)

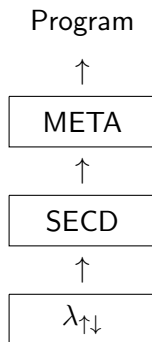


Table of Contents

Towers

Background

Project Aims

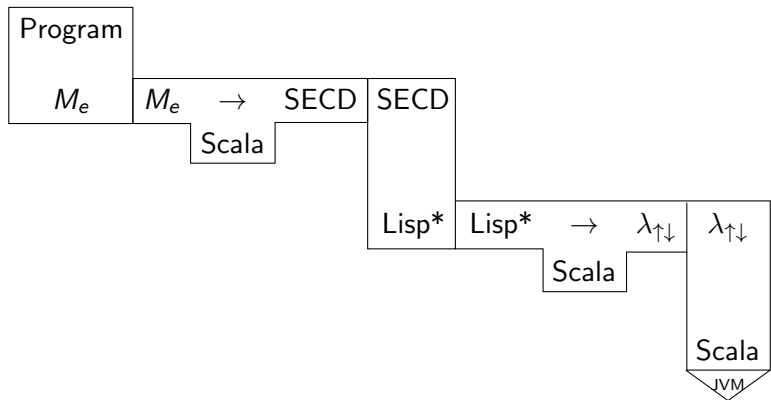
Collapsing a Tower

Heterogeneity

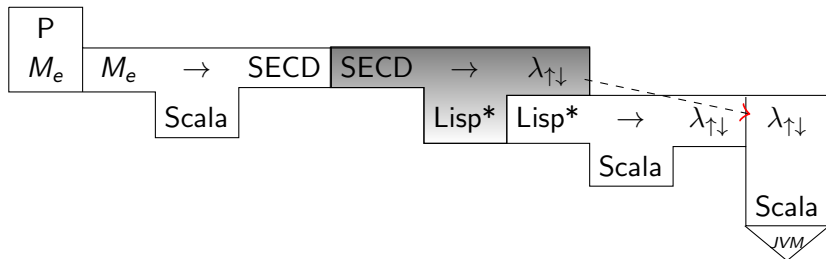
Experimental Tower

Summary

Experimental Tower



Staging SECD



Staging SECD

```
((lambda (fun)                                ;Y-combinator
  ((lambda (F)
    (F F))
   (lambda (F)
    (fun (lambda (x) ((F F) x)))))))

(lambda (factorial)                            ;Factorial
  (lambda (n)
    (if (eq? n 0)
        1
        (* n (factorial (- n 1)))))))
```

Staging SECD

```

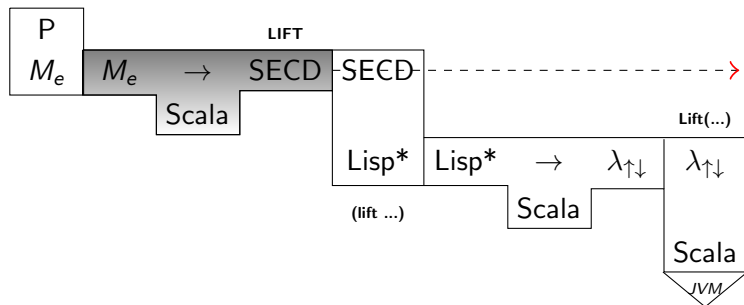
DUM NIL LDF
(LD (1 1) SYM? SEL
  (NIL LD (1 1) CONS LD (1 2) AP JOIN) (LD (1 1) NUM? SEL
(NIL LD (1 1) CONS LDF
  (LD (1 1) RTN) AP JOIN) (LDC + LD (1 1) CAR EQ SEL
(NIL LD (1 2) CONS LD (1 1) CADDR CONS LDR (1 1) AP NIL
  LD (1 2) CONS LD (1 1) CADR CONS LDR (1 1) AP ADD JOIN)
  (LDC - LD (1 1) CAR EQ SEL
  ...
  LDC 1 CONS LDC n CONS LDC - CONS CONS LDC factorial
  CONS CONS LDC n CONS LDC * CONS CONS LDC 1 CONS
  LDC . LDC 0 CONS LDC n CONS LDC eq? CONS CONS LDC if
  CONS CONS LDC . LDC n CONS CONS LDC lambda
  ...
  LDR (1 1) AP RTN ) RAP STOP

```

Staging SECD

```
(let x0
  (lambda f0 x1
    (let x2
      (lambda f2 x3
        (let x4 (car x3)
          (let x5 (car x4)
            (let x6 (sym? x5)
              (if x6
                ...
                (let x8 (car x7)
                  (let x9 (num? x8)
                    (if x9
                      ...
                      (let x12 (car x11)
                        (let x13 (eq? x12 '+)
                          ...

```


Staging M_e 

Staging M_e

```

DUM NIL LDF
  (LD (1 1) SYM? SEL ; $M_e$  dispatch Logic
    (NIL LD (1 1) CONS LD (1 2) AP JOIN )
  (LD (1 1) NUM? SEL
    (LD (1 1) LIFT JOIN ) ;Lift literals
  ...
(LDC letrec LD (1 1) CAR EQ SEL
  (NIL NIL LDF
    (LD (2 1) CADR CAR LD (1 1) EQ SEL
      (LD (12 1) LIFT JOIN) ;Lift recursive lambdas
    ...
(LDC lambda LD (1 1) CAR EQ SEL
  ...
  CONS LDR (1 1) AP RTN) LIFT JOIN) ;Lift lambdas
  ...

```

Staging M_e

```
(lambda f0 x1
  (let x2
    (lambda f2 x3
      (let x4
        (lambda f4 x5 ;Factorial
          (let x6 (eq? x3 0)
            (let x7
              (if f4 1
                (let x7 (- x3 1)
                  (let x8 (x1 x5)
                    (let x9 (* x3 x6) x7)))) x5))) f2)))
  (let x3
    (lambda f3 x4 ;Y-combinator
      (let x5
        (lambda f5 x6
          (let x7
            ...
```

Table of Contents

Towers

Background

Project Aims

Collapsing a Tower

Heterogeneity

Experimental Tower

Summary

Conclusions

- ▶ Heterogeneity raises issues: likely to include translation layers and binding-time info not propagatable
- ▶ *lift* is able to eliminate interpretative overhead even in the presence of translation layers in a tower
- ▶ We propagate annotations by implementing lift (but it requires reverse engineering and transformation of program constructs between interpreter boundaries)

Future Work

- ▶ Bringing methodology even closer to practice
- ▶ Does our methodology extend to side-effects?
- ▶ Interpreter of different paradigms? E.g., WAM for logic programming
- ▶ Less-intrusive collapse

References

- [1] B. C. Smith, "Reflection and semantics in Lisp," in *Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 1984, pp. 23–35.
- [2] M. Wand and D. P. Friedman, "The mystery of the tower revealed: A nonreflective description of the reflective tower," *Lisp and Symbolic Computation*, vol. 1, no. 1, pp. 11–38, 1988.
- [3] O. Danvy and K. Malmkjaer, "Intensions and extensions in a reflective tower," in *Proceedings of the 1988 ACM conference on LISP and functional programming*. ACM, 1988, pp. 327–341.
- [4] Y. Futamura, "Partial evaluation of computation process—an approach to a compiler-compiler," *Higher-Order and Symbolic Computation*, vol. 12, no. 4, pp. 381–391, 1999.
- [5] N. Amin and T. Rompf, "Collapsing towers of interpreters," *Proceedings of the ACM on Programming Languages*, vol. 2, no. POPL, p. 52, 2017.