INTERPROCEDURAL SPECIALIZATION OF HIGHER-ORDER DYNAMIC LANGUAGES WITHOUT STATIC ANALYSIS

ECOOP
June 22, 2017

Baptiste Saleil
baptiste.saleil@umontreal.ca

Marc Feeley
feeley@iro.umontreal.ca
INTRODUCTION

- Research on JIT compilation
  - Dynamic languages
  - Dynamic techniques

- LC: Research oriented Scheme compiler
  - Scheme and Functional Programming Workshop 2014 & 2015
  - https://github.com/bsaleil/lc
INTRODUCTION

- **Dynamic languages**
  - Work done at compilation
  - Work done at execution

- **Dynamic type checking**
  - Ensures safety of the primitives :)  
  - Impact on performance :(  

Example

(define (sum-to-10 x)
  (if (> x 10)
      0
      (+ x (sum-to-10 (+ x 1)))))
Example

```
(define (sum-to-10 x)
  (if (> x 10)
      0
      (+ x (sum-to-10 (+ x 1))))))
```

4 type checks in this code
Dynamic Type Checking

Example

\[ \text{(define (sum-to-10 x)} \]
\[ \quad \text{(if (> x 10)} \]
\[ \quad \quad \quad 0 \]
\[ \quad \quad \quad (+ x \text{(sum-to-10 (+ x 1))}) \]\n
4 type checks in this code

How can we remove them?
(define (sum-to-10 x)
  (if (> x 10)
      0
      (+ x (sum-to-10 (+ x 1)))))

(print (sum-to-10 6))
(print (sum-to-10 7.5))
(define (sum-to-10 x)
  (if (> x 10)
      0
      (+ x (sum-to-10 (+ x 1)))))

(print (sum-to-10 6))
(print (sum-to-10 7.5))
(define (sum-to-10 x)
  (if (> x 10)
      0
      (+ x (sum-to-10 (+ x 1))))
)

(print (sum-to-10 6))
(print (sum-to-10 7.5))
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-umer 10))
(define sum-to-pi (make-umer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
RUNNING EXAMPLE: HIGHER ORDER FUNCTION

(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
RUNNING EXAMPLE: HIGHER ORDER FUNCTION

(define (make-sumer n)
  (letrec ((f (lambda (x)
                  (if (> x n) 0
                     (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))

→ Static analysis (e.g. 0-CFA)
RUNNING EXAMPLE: HIGHER ORDER FUNCTION

\[
\text{(define (make-sumer n)} \\
\text{ (letrec ( ((f (lambda (x) } \\
\text{ (if (> x n) } \\
\text{ 0 } \\
\text{ (+ x (f (+ x 1))))) ) ) } \\
\text{ f)) )}
\]

\[
\text{(define sum-to-10 (make-sumer 10))}
\]

\[
\text{(define sum-to-pi (make-sumer 3.14))}
\]

\[
; 6 + 7 + 8 + 9 + 10
\]

\[
\text{(print (sum-to-10 6))}
\]

\[
; 7.5 + 8.5 + 9.5
\]

\[
\text{(print (sum-to-10 7.5))}
\]

\[
; 1.10 + 2.10 + 3.10
\]

\[
\text{(print (sum-to-pi 1.10))}
\]

→ Static analysis (e.g. 0-CFA)
- JIT incompatible
- Lacks precision
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
INTERPROCEDUREAL SPECIALIZATION
INTERPROCEDURAL SPECIALIZATION OF HIGHER-ORDER LANGUAGES
INTERPROCEDURAL SPECIALIZATION OF HIGHER-ORDER DYNAMIC LANGUAGES
INTERPROCEDURAL SPECIALIZATION OF HIGHER-ORDER DYNAMIC LANGUAGES WITHOUT STATIC ANALYSIS
INTERPROCEDURAL SPECIALIZATION OF HIGHER-ORDER DYNAMIC LANGUAGES WITHOUT STATIC ANALYSIS

→ Basic Block Versioning (BBV)

- Simple and Effective Type Check Removal through Lazy Basic Block Versioning
  Maxime Chevalier-Boisvert and Marc Feeley, ECOOP 2015

- Lazy intraprocedural code specialization (JIT)

- No static analysis

- Dynamic languages (JavaScript, Scheme, ...)

7
Naive compilation

(define (make-sumer n)
  (letrec ((f (lambda (x)
               (if (> x n)
                   0
                   (+ x (f (+ x 1))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

5 checks needed

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
              (if (> x n)
                  0
                  (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
                 (if (> x n)
                     0
                     (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
      (if (> x n)
          0
          (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
              (if (> x n)
                  0
                  (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1))))))
     f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
        (if (> x n)
          0
          (+ x (f (+ x 1))))))
        f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
                 (if (> x n)
                     0
                     (+ x (f (+ x 1)))))
             f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
Basic Block Versioning

(define (make-sumer n)
  (letrec ((f (lambda (x)
                  (if (> x n)
                      0
                      (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
Basic Block Versioning

(define (make-sumer n)
  (letrec ((f (lambda (x)
                 (if (> x n)
                     0
                     (+ x (f (+ x 1)))))))
    f))

3 checks needed (2 removed)

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
OUR WORK: INTERPROCEDURAL EXTENSIONS

(define (make-sumer n)
  (letrec ((f (lambda (x)
                 (if (> x n)
                    0
                    (+ x (f (+ x 1))))))
            f))

2 checks needed
(3 removed)

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))

- Propagate the types through function calls → no check on x
OUR WORK: INTERPROCEDURAL EXTENSIONS

(define (make-sumer n)
  (letrec ((f (lambda (x)
                  (if (> x n)
                    0
                    (+ x (f (+ x 1)))))
            f))
    1 check needed
    (4 removed)

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))

• Propagate the types through function calls
  → no check on x

• Propagate the types through function returns
  → no check on the returned value
OUR WORK: INTERPROCEDURAL EXTENSIONS

- Propagate the types through function calls → no check on x
- Propagate the types through function returns → no check on the returned value
- Specialize the code using captured information → no check on n

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(no checks needed (5 removed))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```
OUR WORK: INTERPROCEDURAL EXTENSIONS

→ Several entry points, each specialized for a type combination
OUR WORK: INTERPROCEDURAL EXTENSIONS

→ Several entry points, each specialized for a type combination

1. Extend the closure representation
   • Allow storing multiple entry points instead of 1
OUR WORK: INTERPROCEDURAL EXTENSIONS

→ Several entry points, each specialized for a type combination

1. Extend the closure representation
   - Allow storing multiple entry points instead of 1

2. Dynamic dispatch to jump to the entry point
   - Each function call
   - Each function return (each continuation call)
OUR WORK: INTERPROCEDURAL EXTENSIONS

→ Several entry points, each specialized for a type combination

1. Extend the closure representation
   • Allow storing multiple entry points instead of 1

2. Dynamic dispatch to jump to the entry point
   • Each function call
   • Each function return (each continuation call)

3. Specialize using captured information
IMPLEMENTATION

Closure representation extension
FLAT CLOSURE REPRESENTATION

code address  captured variable 1 ... captured variable n
(define (make-sumer n)
  (letrec ((f (lambda (x)
               (if (> x n)
                   0
                   (+ x (f (+ x 1)))))))))

(f)

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1))))))
          f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
                   (if (> x n) 0
                       (+ x (f (+ x 1))))))
          f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
           (if (> x n)
               0
               (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
(define (make-sumer n)
  (letrec ((f (lambda (x)
               (if (> x n) 0
                   (+ x (f (+ x 1))))))
           f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
IMPLEMENTATION: FLAT CLOSURE EXTENSION

(\texttt{define (make-sumer n)}
 (\texttt{letrec ((f (lambda (x)
   (if (> x n)
     0
     (+ x (f (+ x 1))))))})
 f))

(\texttt{define sum-to-10 (make-sumer 10)})
(\texttt{define sum-to-pi (make-sumer 3.14)})

; 6 + 7 + 8 + 9 + 10
(\texttt{print (sum-to-10 6)})
; 7.5 + 8.5 + 9.5
(\texttt{print (sum-to-10 7.5)})
; 1.10 + 2.10 + 3.10
(\texttt{print (sum-to-pi 1.10)})
(define (make-sumer n)
  (letrec ((f (lambda (x)
            (if (> x n) 0
                (+ x (f (+ x 1))))))
           f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
Implementation: Flat Closure Extension

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
              (if (> x n)
                  0
                  (+ x (f (+ x 1)))))
             f))
    (define sum-to-10 (make-sumer 10))
    (define sum-to-pi (make-sumer 3.14))
)

(print (sum-to-10 6)) ; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 7.5)) ; 7.5 + 8.5 + 9.5
(print (sum-to-pi 1.10)) ; 1.10 + 2.10 + 3.10
```
IMPLEMENTATION
Dynamic dispatch
(define (make-sumer n)
  (letrec ((f (lambda (x)
            (if (> x n)
                0
                (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
Definition of `make-sumer` function:

```scheme
(define (make-sumer n)
  (letrec ((
    (f (lambda (x)
        (if (> x n) 0 (+ x (f (+ x 1)))))))
    f))
)
```

Examples:

```scheme
(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
**IMPLEMENTATION: DYNAMIC DISPATCH**

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
               (if (> x n)
                   0
                   (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1))))))
            f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
Implementation: Dynamic Dispatch

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
       (if (> x n)
           0
           (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-pi 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```
(define (make-sumer n)
  (letrec ((f (lambda (x)
               (if (> x n)
                0
                (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
IMPLEMENTATION
Captured information
**IMPLEMENTATION: CAPTURED INFORMATION**

- **Generated code**

- **Entry point tables**

- **Closure instances**

- **Stub memory space**

---

```scheme
(define (make-sumer n)
    (letrec ((f (lambda (x)
        (if (> x n)
            0
            (+ x (f (+ x 1))))))
        f))
)

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```
(define (make-sumer n)
  (letrec ((f (lambda (x)
    (if (> x n)
      0
      (+ x (f (+ x 1))))))
    f))

(define sum-to-10 (make-umer 10))
(define sum-to-pi (make-umer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
IMPLEMENTATION: CAPTURED INFORMATION

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                 (if (> x n)
                   0
                   (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```
IMPLEMENTATION: CAPTURED INFORMATION

Generated code

Entry point tables

Closure instances

Stub memory space

(make-sumer n)
(letrec ((f (lambda (x)
  (if (> x n)
    0
    (+ x (f (+ x 1))))))
  f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
IMPLEMENTATION: CAPTURED INFORMATION

(define (make-sumer n)
  (letrec ((f (lambda (x)
               (if (> x n) 0 (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-umer 10))
(define sum-to-pi (make-umer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
IMPLEMENTATION: CAPTURED INFORMATION

Generated code

Entry point tables

Closure instances

Stub memory space

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-pi 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```
IMPLEMENTATION: CAPTURED INFORMATION

(define (make-sumer n)
  (letrec ((f (lambda (x)
                 (if (> x n)
                     0
                     (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
What about continuations?
What about continuations?

→ Function return is a call to the continuation
→ Conceptually same implementation!
What about continuations?

→ Function return is a call to the continuation
→ Conceptually same implementation!

- Propagated types → Type of the returned value
- Captured types → Type of the local variables
RESULTS
RESULTS

35 benchmarks: Standard Scheme benchmarks

Configurations

- Intraprocedural BBV
- Interprocedural: function specialization
- Interprocedural: continuation specialization
- Interprocedural: function and continuation specialization

Metrics

- Number of type checks
- Generated code size
- Execution / Compilation / Total time
Number of type checks

- Number of executed type checks relative to pure intraprocedural specialization
- No checks for 9 benchmarks
- Significantly fewer checks for most of the benchmarks
CODE SIZE

- Generated code size relative to pure intraprocedural specialization

- Less code generated for half of the benchmarks
- Just 9% more code generated on average
Almost all benchmarks are faster
No benchmark is significantly slower
Up to 2x faster
### EXECUTION TIME (VS GAMBIT)

<table>
<thead>
<tr>
<th>Function</th>
<th>LC - Max=5, Interprocedural</th>
<th>LC - Max=5, Intraprocedural</th>
<th>Gambit - Safe mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>browser</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>sump</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>fibp</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>cpstkak</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>tak</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>graphs</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>divsiter</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>deriv</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>ack</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>nqueens</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>divrec</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>mazefin</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>conform</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>all</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>dderiv</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>destruct</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>fib</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>boyer</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>mboyer</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>sboyer</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>compiler</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>mibrot</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>peval</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>primes</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>sunloop</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>earley</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>triangl</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>lattice</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>array1</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>sum</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>simplex</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>permn</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>takl</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>ldpoly</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

- Execution time relative to the execution time of the code generated by the Gambit Scheme compiler with dynamic checks turned off (capped at 400%)

- Faster than Gambit with dynamic checks turned on

- Varies from 0.5x to 4x (6x slower with Gambit executing all the checks)
Compilation time relative to pure intraprocedural BBV

- Compilation time increase for most of the benchmarks
- Mostly small increase in compilation time, but up to 4x
Total time relative to pure intraprocedural BBV

- Large speedup for floating point (avoids boxing)
- Most of the benchmarks are faster
- From 0.14x to 1.33x
CONCLUSION
CONCLUSION

- Lazy interprocedural specialization

- Works well
  - Checks removed (up to 100%)
  - Faster code (up to 50%)

- Simple
  - Simple to implement
  - Does not require complex architecture
FUTURE WORK

- Propagating other properties
  - Value (e.g. \( x = 10 \))
  - Variable relationship (e.g. \( x < y \))

- Function identity and return address propagation
  - Dynamic function inlining

- Register allocation information
  - Arguments
  - Returned value