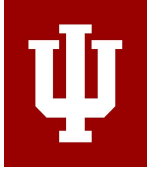


SELF-HOSTING FUNCTIONAL PROGRAMMING LANGUAGES ON META-TRACING JIT COMPILERS

Final Examination for Doctor of Philosophy in Computer Science

Caner Derici

Doctoral Committee:

- Sam Tobin-Hochstadt, Ph.D.
- Jeremy Siek, Ph.D.
- Daniel Leivant, Ph.D.
- Amr Sabry, Ph.D.

Thesis:

Efficient self-hosting of a full-scale functional language on a meta-tracing JIT compiler is achievable.

Thesis:

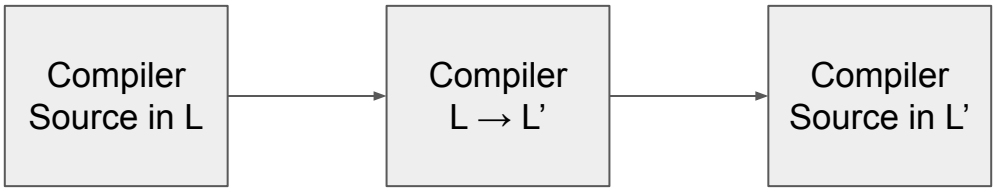
Efficient self-hosting of a full-scale functional language on a meta-tracing JIT compiler is achievable.

Plan:

1. Introduction: Unpack the Statement & Survey Related Work
2. Main Evidence: Racket on Pycket
3. Performance Evaluation: Cost of Self-hosting
4. Conclusion: Technical Contributions & Future Work

Unpack the Claim:

*Efficient **self-hosting** of a full-scale functional language on a meta-tracing JIT compiler is achievable.*



- Improved scalability
- Improved portability

Unpack the Claim:

*Efficient self-hosting of a full-scale functional language on a **meta-tracing JIT** compiler is achievable.*

Unpack the Claim:

Efficient *self-hosting of a full-scale functional language on a meta-tracing JIT compiler is achievable.*

Context: Literature Survey

- **Higher-order dynamic VMs:**
  - **GraalVM & Truffle** [Thomas Würthinger et al. “One VM to Rule Them All”]
  - **RPython Framework** [Davide Ancona et al. “RPython: A Step Towards Reconciling Dynamically and Statically Typed OO Languages”]

Context: Literature Survey

- **Higher-order dynamic VMs:**
  - **GraalVM & Truffle** [Thomas Würthinger et al. “One VM to Rule Them All”]
  - **RPython Framework** [Davide Ancona et al. “RPython: A Step Towards Reconciling Dynamically and Statically Typed OO Languages”]
- **Tracing JITs**
  - **Industry: Dynamo, Mozilla’s TraceMonkey for JS, Adobe’s Tamarin for ActionScript**
  - **Research: PyPy, LuaJIT, Converge, Lambdachine, PyHaskell**

Context: Literature Survey

- **Higher-order dynamic VMs:**
  - **GraalVM & Truffle** [Thomas Würthinger et al. “One VM to Rule Them All”]
  - **RPython Framework** [Davide Ancona et al. “RPython: A Step Towards Reconciling Dynamically and Statically Typed OO Languages”]
- **Tracing JITs**
  - **Industry: Dynamo, Mozilla’s TraceMonkey for JS, Adobe’s Tamarin for ActionScript**
  - **Research: PyPy, LuaJIT, Converge, Lambdachine, PyHaskell**
- **Self-hosting:**
  - **Tachyon for JS** [Maxime Chevalier-Boisvert et al. “Bootstrapping a Self-Hosted Research Virtual Machine for JavaScript: An Experience Report”]

Context: Literature Survey

- **Higher-order dynamic VMs:**
  - **GraalVM & Truffle** [Thomas Würthinger et al. “One VM to Rule Them All”]
  - **RPython Framework** [Davide Ancona et al. “RPython: A Step Towards Reconciling Dynamically and Statically Typed OO Languages”]
- **Tracing JITs**
  - **Industry: Dynamo, Mozilla’s TraceMonkey for JS, Adobe’s Tamarin for ActionScript**
  - **Research: PyPy, LuaJIT, Converge, Lambdachine, PyHaskell**
- **Self-hosting:**
  - **Tachyon for JS** [Maxime Chevalier-Boisvert et al. “Bootstrapping a Self-Hosted Research Virtual Machine for JavaScript: An Experience Report”]
- **Managing abstraction levels:**
  - **Type directed partial evaluation** [Andrzej Filinski. “A Semantic Account of Type-Directed Partial Evaluation”.]
  - **Multi-stage programming & Futamura projections** [Rui Ge and Ronald Garcia. “Refining Semantics for Multi-Stage Programming”]

Context: Fit & Ultimate Goal

Meta-tracing → fast prototyping efficient research interpreters for dynamic languages.

Self-hosting on meta-tracing → fast prototyping efficient full-scale runtimes for dynamic languages.

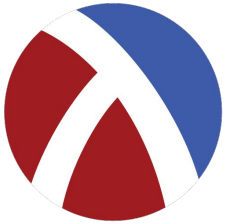
Unlocking research avenues:

- **Fast prototyping + same full language semantics on different VMs.**
- **Research space scales out with meta-tracing as well as self-hosting.**

Thesis:

Efficient self-hosting of a full-scale functional language on a meta-tracing JIT compiler is achievable.

Methodology: Tools

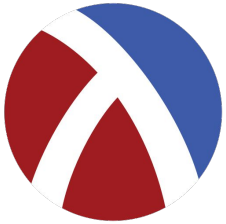


- Well-adopted functional dynamic language
- Facilitates PL research
- Self-hosting on Chez Scheme

Thesis:

Efficient self-hosting of a full-scale functional language on a meta-tracing JIT compiler is achievable.

Methodology: Tools



- Well-adopted functional dynamic language
- Facilitates PL research
- Self-hosting on Chez Scheme

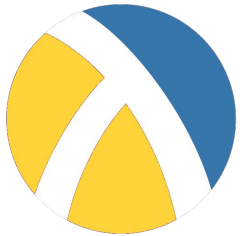
- Meta-tracing JIT compiler
- CEK machine at its core
- Rudimentary interpreter for #%kernel

Thesis:

Efficient self-hosting of a full-scale functional language on a meta-tracing JIT compiler is achievable.

Methodology:

Racket on Pycket



- Meta-tracing JIT compiler
- CEK machine at its core
- Full-scale Racket implementation

Racket on Chez

Rebuilding Racket on Chez Scheme (Experience Report)

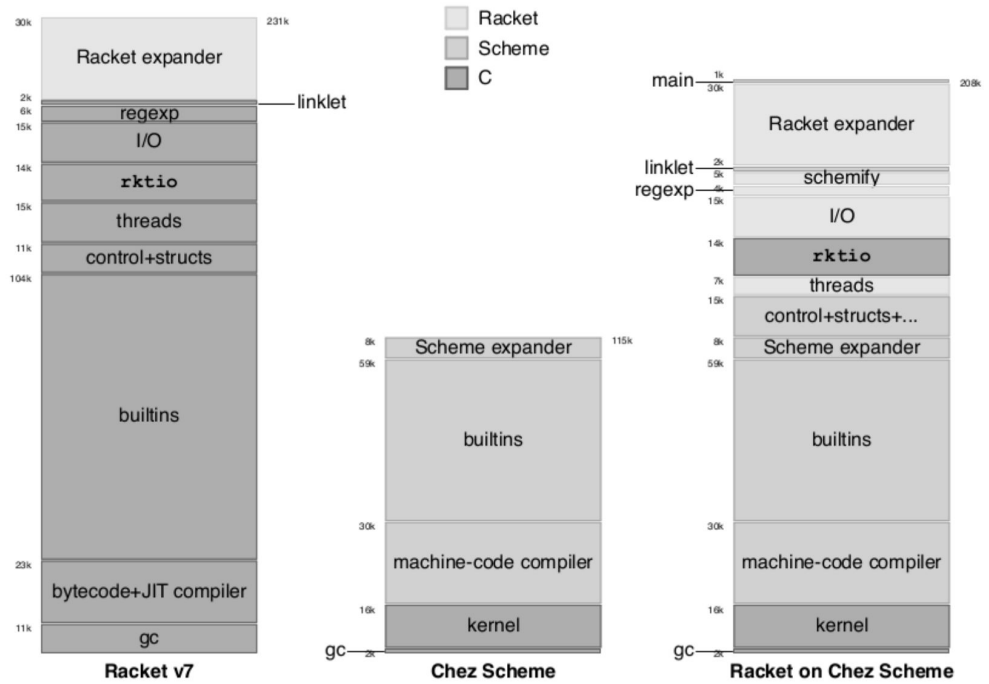
MATTHEW FLATT, University of Utah, USA
CANER DERICI, Indiana University, USA
R. KENT DYBVIG, Cisco Systems, Inc., USA
ANDREW W. KEEP, Cisco Systems, Inc., USA
GUSTAVO E. MASSACCESI, Universidad de Buenos Aires, Argentina
SARAH SPALL, Indiana University, USA
SAM TOBIN-HOCHSTADT, Indiana University, USA
JON ZEPPIERI, independent researcher, USA

We rebuilt Racket on Chez Scheme, and it works well—as long as we're allowed a few patches to Chez Scheme. DrRacket runs, the Racket distribution can build itself, and nearly all of the core Racket test suite passes. Maintainability and performance of the resulting implementation are good, although some work remains to improve end-to-end performance. The least predictable part of our effort was how big the differences between Racket and Chez Scheme would turn out to be and how we would manage those differences. We expect Racket on Chez Scheme to become the main Racket implementation, and we encourage other language implementers to consider Chez Scheme as a target virtual machine.

CCS Concepts: • **Software and its engineering → Functional languages; Software evolution**.

Methodology

Racket on Chez



Methodology

Linklets

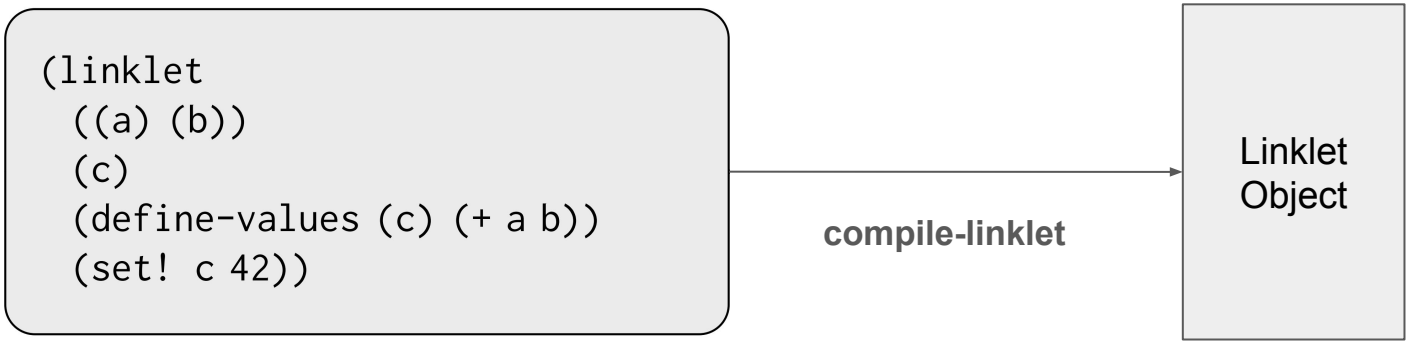
$$
\begin{aligned}
L &::= (\textbf{linklet}\ ((imp\ \ldots)\ \ldots)\ (exp\ \ldots)\ l\text{-}top\ \ldots\ e) \\
l\text{-}top &::= (\textbf{define-values}\ (x)\ e) \mid e \\
imp &::= x \mid (x\,x) \qquad \text{[external-name internal-name]} \\
exp &::= x \mid (x\,x) \qquad \text{[internal-name external-name]} \\
e &::= x \mid v \mid (e\ e\ \ldots) \mid (\textbf{if}\ e\ e\ e) \mid (o\ e\ e) \\
&\quad \mid (\textbf{begin}\ e\ e\ \ldots) \mid (\textbf{lambda}\ (x_{\_!\_}\ \ldots)\ e) \\
&\quad \mid (\textbf{set!}\ x\ e) \mid (\textbf{raises}\ e) \\
&\quad \mid (\textbf{var-ref}\ x) \mid (\textbf{var-ref/no-check}\ x) \\
&\quad \mid (\textbf{var-set!}\ x\ e) \\
&\quad \mid (\textbf{var-set/check-undef!}\ x\ e)
\end{aligned}
$$

Figure 3.3: Linklet Source Language

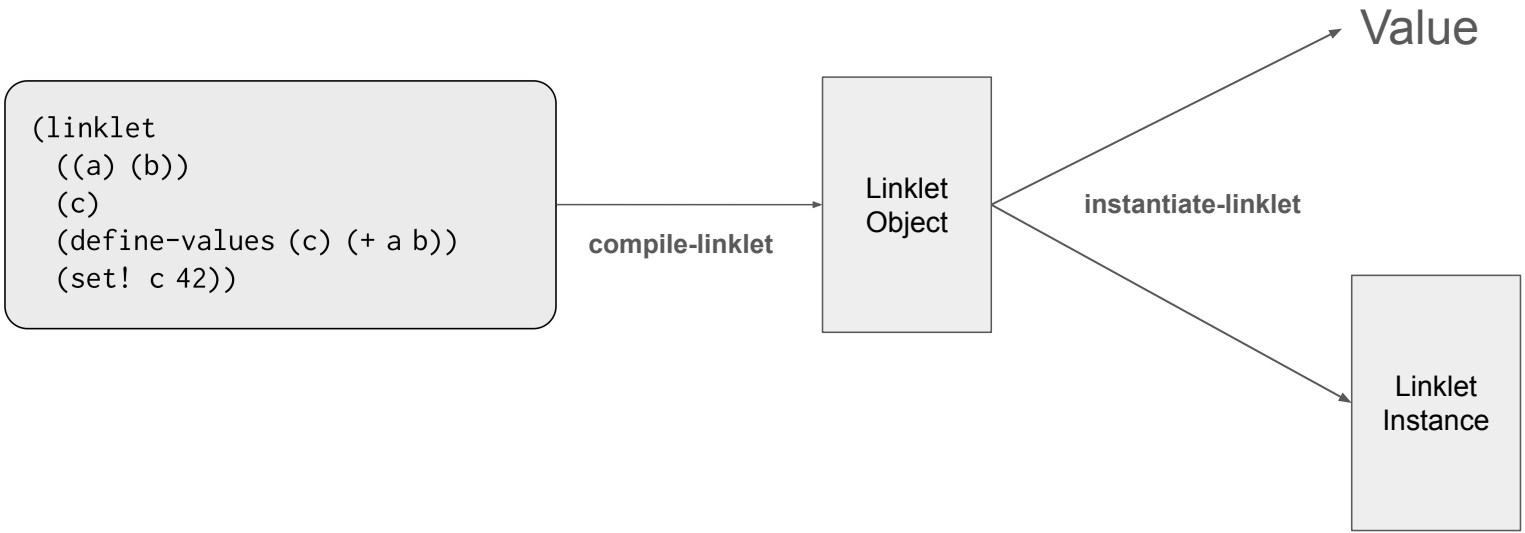
Linklets

```
(linklet
  ((a) (b))
  (c)
  (define-values (c) (+ a b))
  (set! c 42))
```

Linklets


(linklet
((a) (b))
(c)
(define-values (c) (+ a b))
(set! c 42))
compile-linklet
Linklet
Object

Linklets


(linklet
((a) (b))
(c)
(define-values (c) (+ a b))
(set! c 42))
compile-linklet
Linklet
Object
instantiate-linklet
Value
Linklet
Instance

Linklets

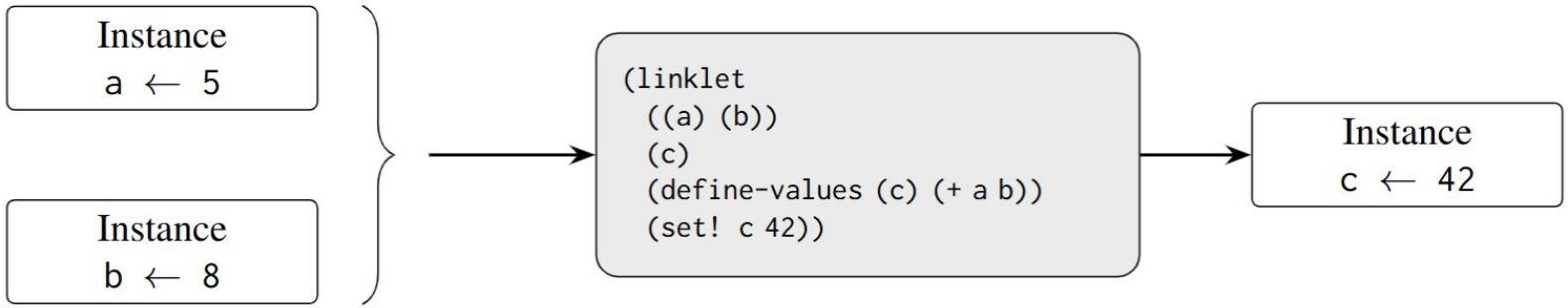


Figure 3.4: Regular Instantiation of a Linklet

Linklets

$$
\begin{aligned}
EP\ [\![\mathbf{\Phi^V}\ (x_{li},\ x)]\!], \rho, \sigma &\longrightarrow_{\beta_p} EP\ [\![v]\!], \rho, \sigma \ \textbf{where}\ v = \sigma[(\sigma[x_{li}])[x]] \\
EP\ [\![(\mathbf{\Phi^I}\ (\boldsymbol{L_\beta}\ x\ v\ \ldots\ v_l))]\!], \rho, \sigma &\longrightarrow_{\beta_p} EP\ [\![(v_l\ x)]\!], \rho, \sigma \\
EP\ [\![(\mathbf{\Phi^I}(\boldsymbol{L_\beta}\ x_t\ v_p\ \ldots\ (\textbf{define-values}\ (x)\ v)\ l\text{-}top\ \ldots))]\!], &\longrightarrow_{\beta_p} EP\ [\![\mathbf{\Phi^I}(\boldsymbol{L_\beta}\ x_t\ v_p\ \ldots\ l\text{-}top\ \ldots)]\!], \rho_1, \sigma_1 \\
&\quad \textbf{where}\ \rho_1 = \rho[x \to cell],\ \sigma_1 = \sigma[cell \to v] \\
EP\ [\![(\mathbf{\Phi^I}(\boldsymbol{L_\beta}\ x_t\ v_p\ \ldots\ l\text{-}top_1\ l\text{-}top\ \ldots))]\!], &\longrightarrow_{\beta_p} EP\ [\![\mathbf{\Phi^I}(\boldsymbol{L_\beta}\ x_t\ v_p\ \ldots\ v_1\ l\text{-}top\ \ldots)]\!], \rho_1, \sigma_1 \\
&\quad \textbf{where}\ l\text{-}top_1 \longrightarrow^*_{\beta_l} v_1 \\
EP\ [\![(\mathbf{\Phi^I}(\boldsymbol{L_\alpha}\ c\text{-}imps\ c\text{-}exps\ l\text{-}top\ldots)\ LI\ \ldots)]\!], \rho, \sigma &\longrightarrow_{\beta_p} EP\ [\![(\mathbf{\Phi^I}(\boldsymbol{L_\alpha}\ c\text{-}imps\ c\text{-}exps\ l\text{-}top\ \ldots)\ LI\ \ldots\ \textbf{\#:t}\ x_t)]\!], \rho, \sigma_1 \\
&\quad \textbf{where}\ x_t \notin dom(\sigma), \sigma_1 = \sigma[x_t \to (LI)] \\
EP\ [\![(\mathbf{\Phi^I}(\boldsymbol{L_\alpha}\ c\text{-}imps\ c\text{-}exps\ l\text{-}top\ldots)\ LI\ \ldots\ \textbf{\#:t}\ x_t)]\!], \rho, \sigma &\longrightarrow_{\beta_p} EP\ [\![(\mathbf{\Phi^I}(\boldsymbol{L_\beta}\ x_t\ l\text{-}top\ \ldots))]\!], \rho_2, \sigma_1 \\
&\quad \textbf{where}\ \rho_1 = V^I(c\text{-}imps, (LI\ \ldots), \rho) \\
&\qquad (\rho_2, \sigma_1) = V^E(c\text{-}exps, x_t, \rho_1, \sigma) \\
(\textbf{program}\ ((x\ L), (x_1\ L_1)\ \ldots)\ p\text{-}top), \rho, \sigma &\longrightarrow_{\beta_p} (\textbf{program}\ ((x_1\ L_1)\ \ldots)\ p\text{-}top[x := \mathbf{\Phi^C}(L)]), \rho, \sigma
\end{aligned}
$$

$$
eval^{\longrightarrow_{\beta_p}}(p) = v \ \textbf{if}\ p, (), () \twoheadrightarrow_{\beta_p \cup \beta_l} (\textbf{program}\ ()\ (v\ \_))
$$

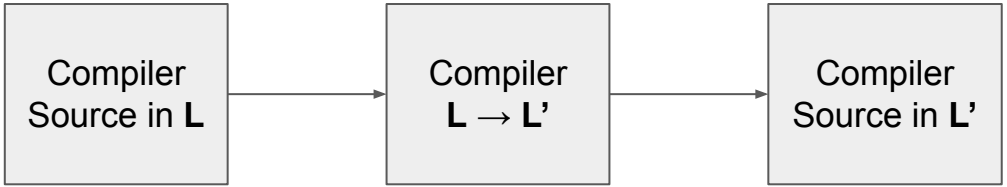
$$
\begin{aligned}
V^I &: c\text{-}imps \times (LI\ \ldots) \times \rho \longrightarrow \rho \\
V^I(((\overline{(\textbf{Import}\ x_{id}\ x_{int}\ x_{ext})})_n), (\overline{LI_n}), \rho) &= \rho[\overline{x_{id} \to LI_n[x_{ext}]}]
\end{aligned}
$$

$$
\begin{aligned}
V^E &: c\text{-}exps \times x \times \rho \times \sigma \longrightarrow \rho \times \sigma \\
V^E((exp\text{-}obj\ \ldots), x_t, \rho, \sigma) &= P(exp\text{-}obj, x_t, \sigma[x_t], \rho, \sigma)\ \ldots \\
P((\textbf{Export}\ x_{id}\ x_{int}\ x_{ext}), x_t, LI_t, \rho, \sigma) &= \begin{cases}
\rho_1, \sigma & \text{if } x_{ext} \in dom(LI_t), \textbf{where } \rho_1 = \rho[x_{id} \to LI_t[x_{ext}]] \\
\rho_1, \sigma_1 & \text{if } x_{ext} \notin dom(LI_t), \textbf{where } \rho_1 = \rho[x_{id} \to var_{new}] \\
 & \qquad var_{new} \notin \sigma \\
 & \qquad \sigma_1 = \sigma[var_{new} \to uninit, \\
 & \qquad\quad x_t \to LI_t[x_{ext} \to var_{new}]]
\end{cases}
\end{aligned}
$$

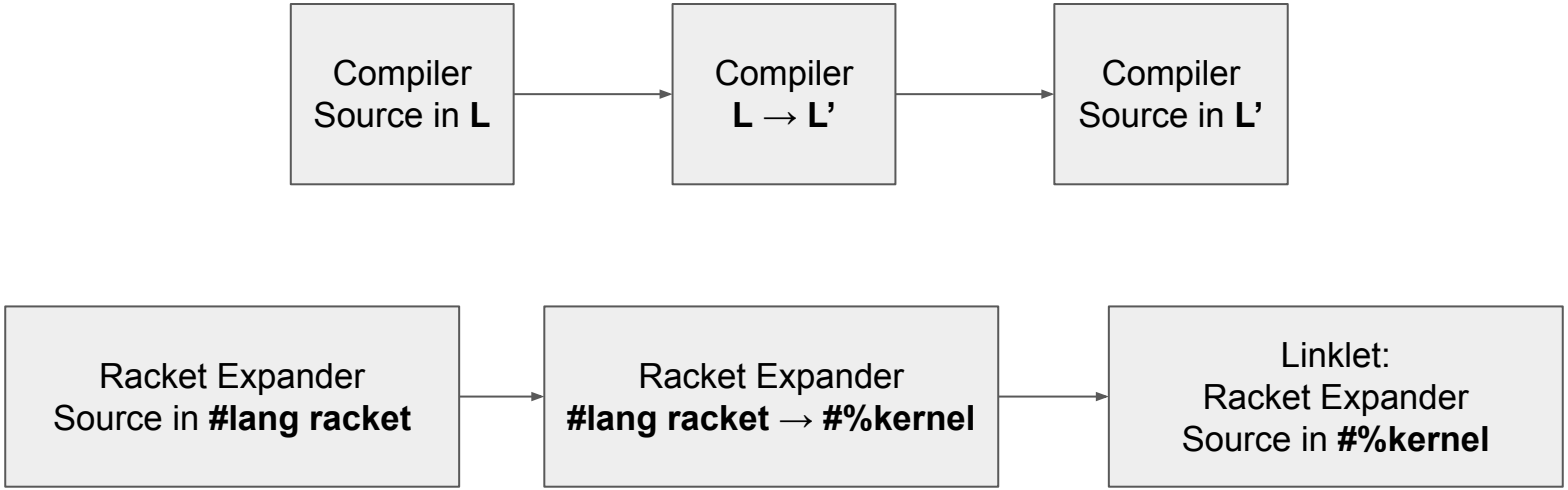
Φ^V : instance-variable-value, Φ^I : instantiate-linklet, Φ^C : compile-linklet
V^I : get import variables, V^E : create variables for exports

Figure 3.9: Standard Reduction Relations for Operational Semantics of Linklets

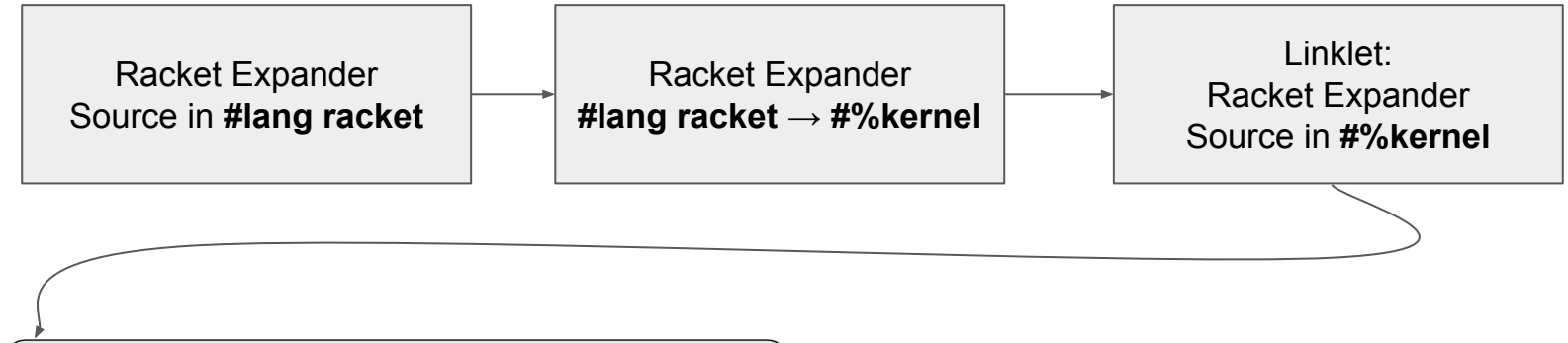
Racket on Pycket



Racket on Pycket

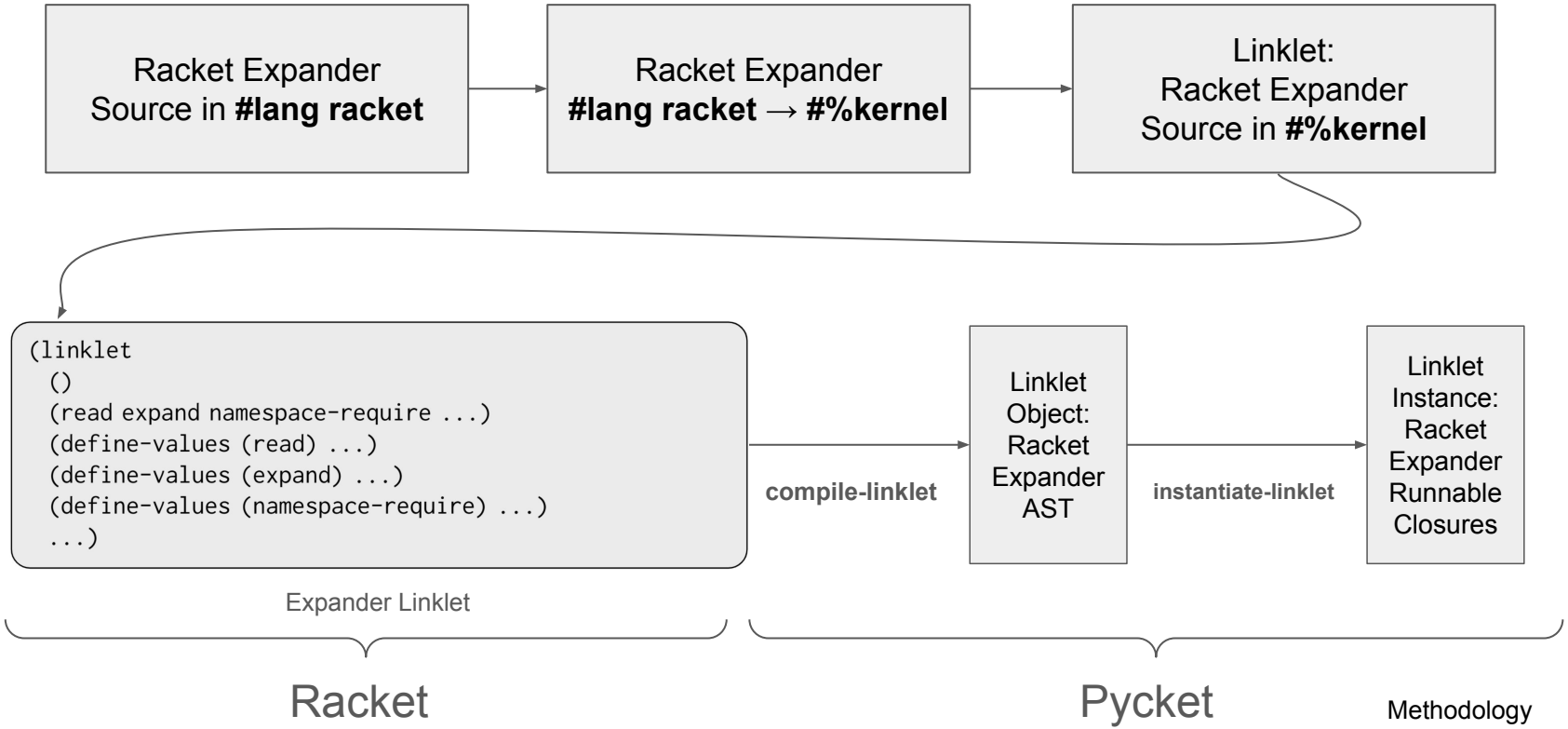


Racket on Pycket



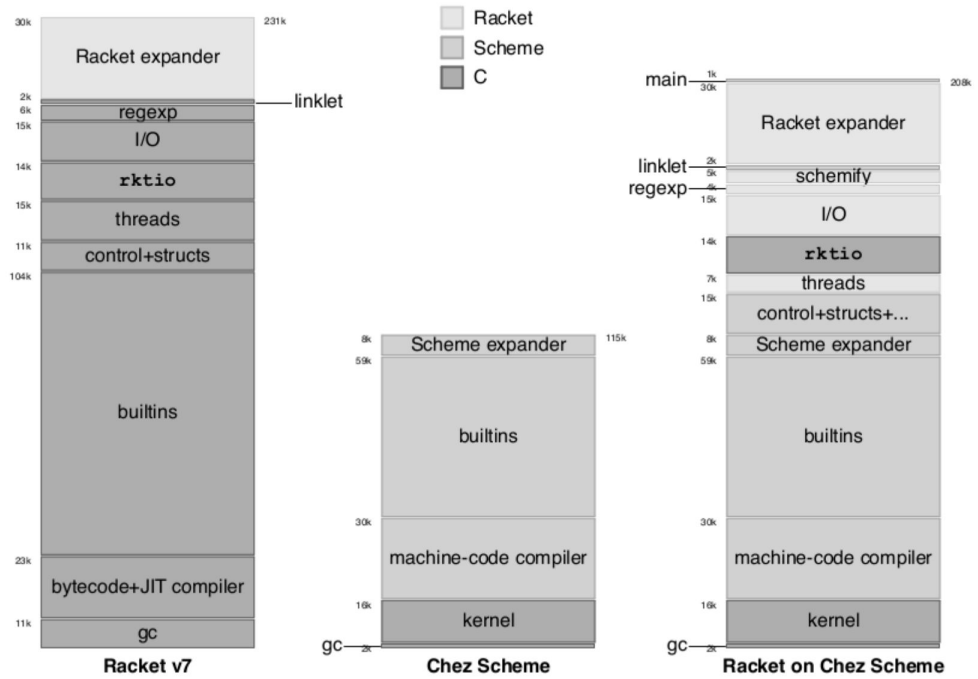
Expander Linklet

Racket on Pycket



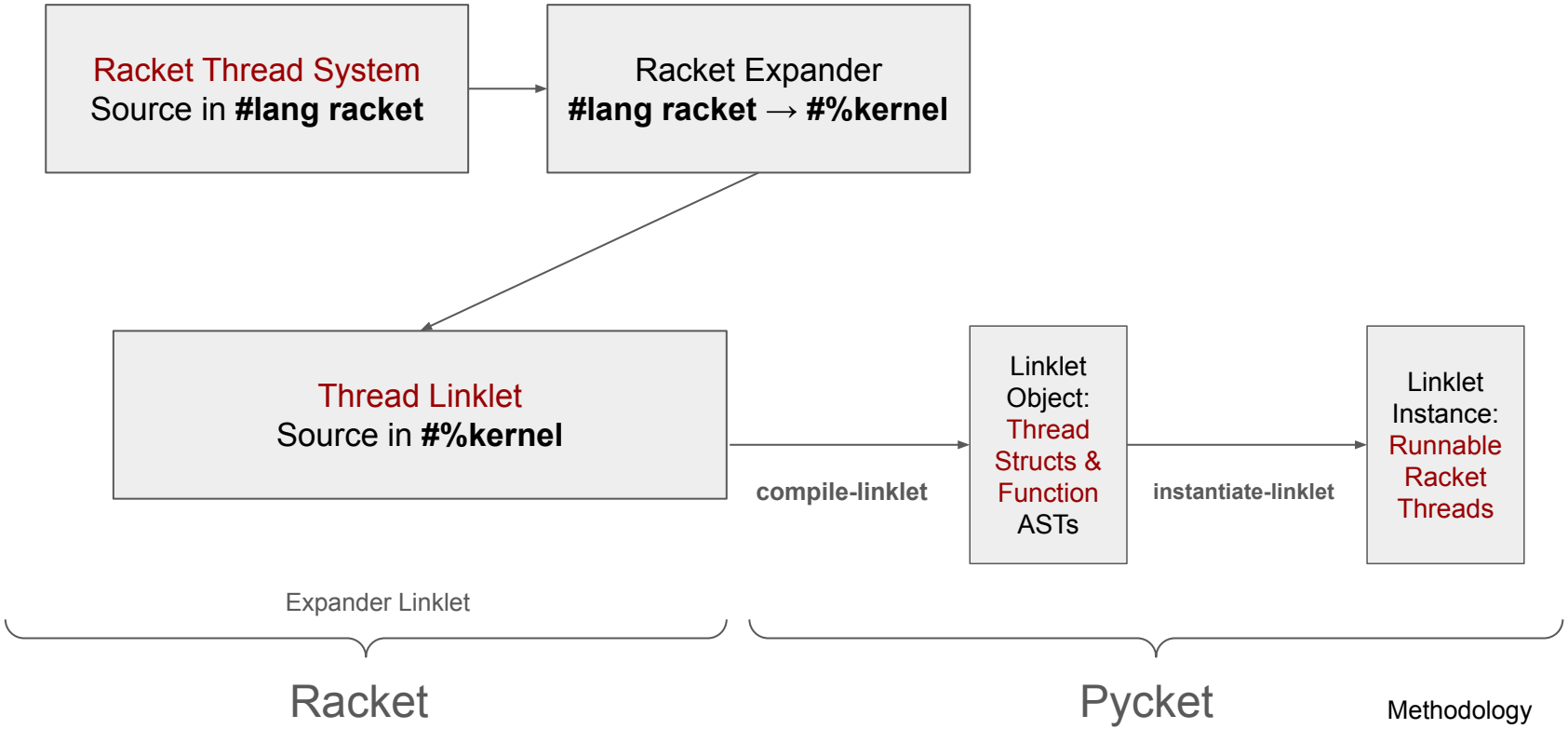
Methodology

Racket on Chez

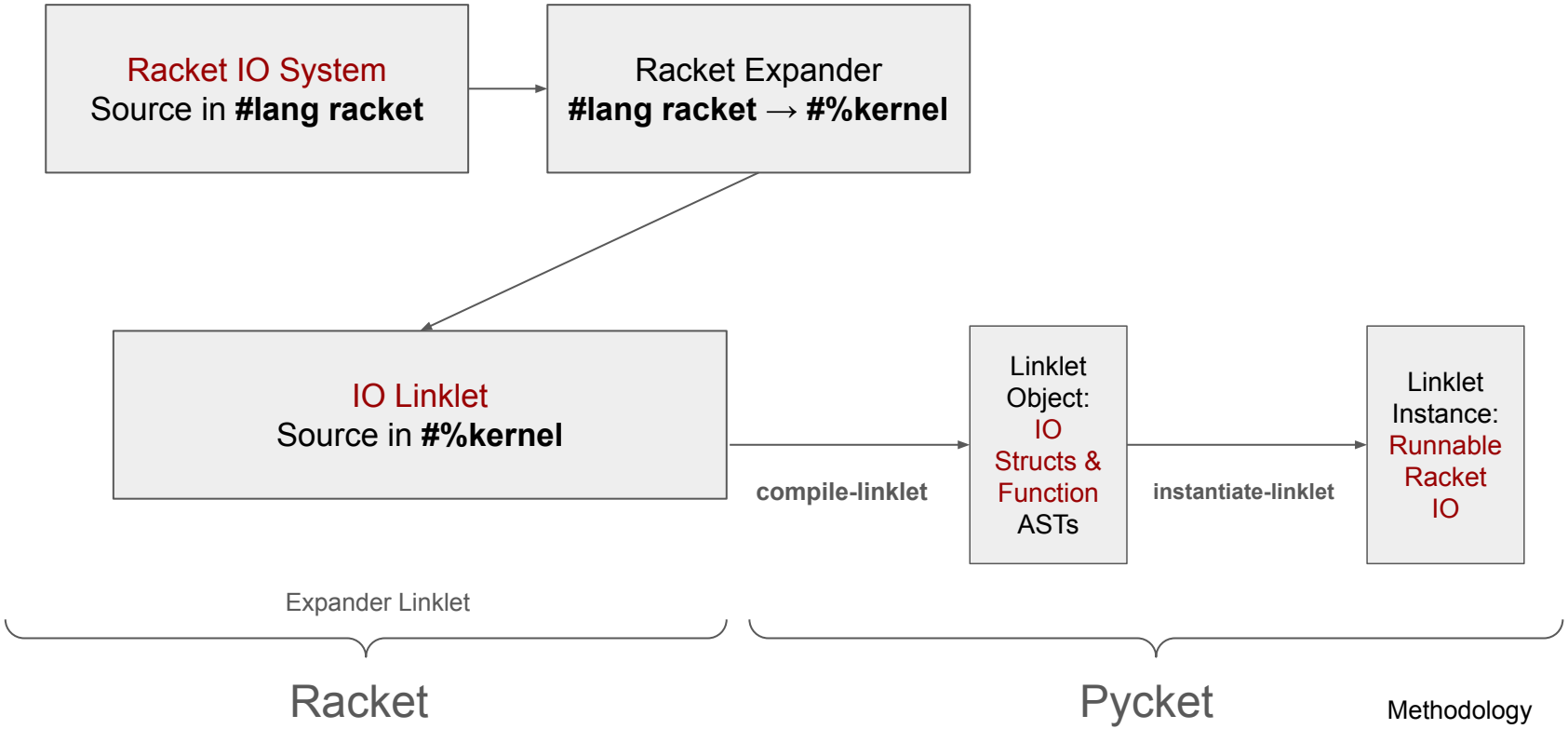


Methodology

Racket on Pycket



Racket on Pycket



From Rudimentary Interpreter → Full Runtime

Expander Linklet

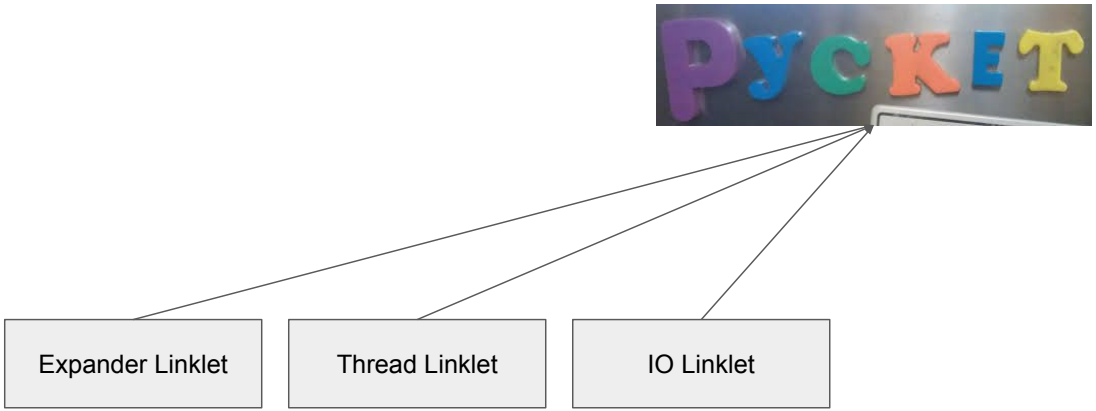
From Rudimentary Interpreter → Full Runtime

Expander Linklet

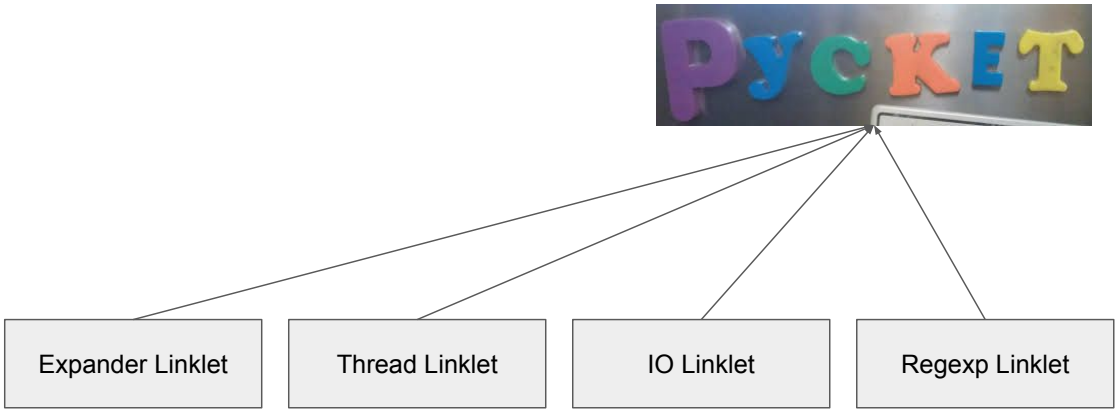
Thread Linklet

engines
meta-continuations
lock support
…

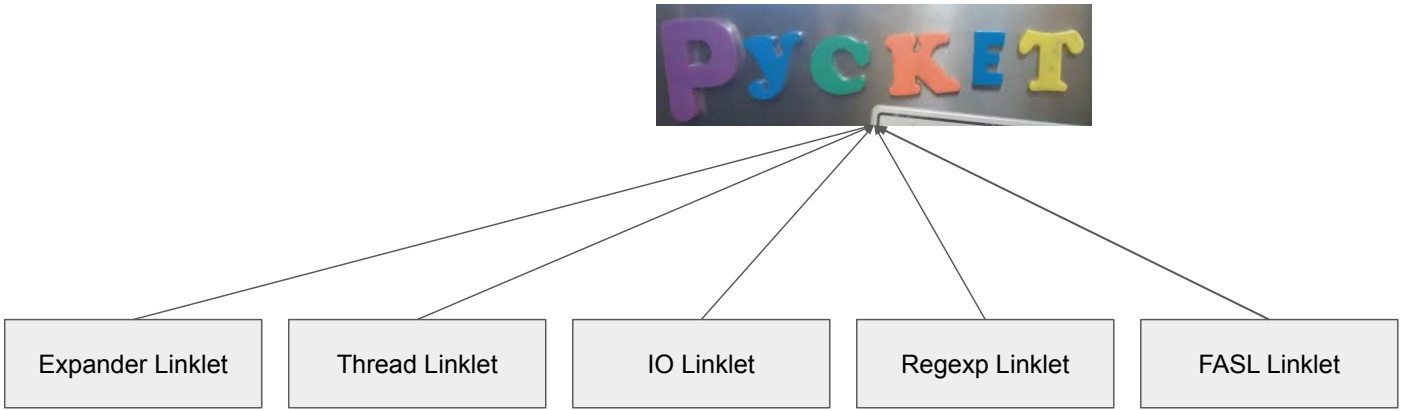
From Rudimentary Interpreter → Full Runtime



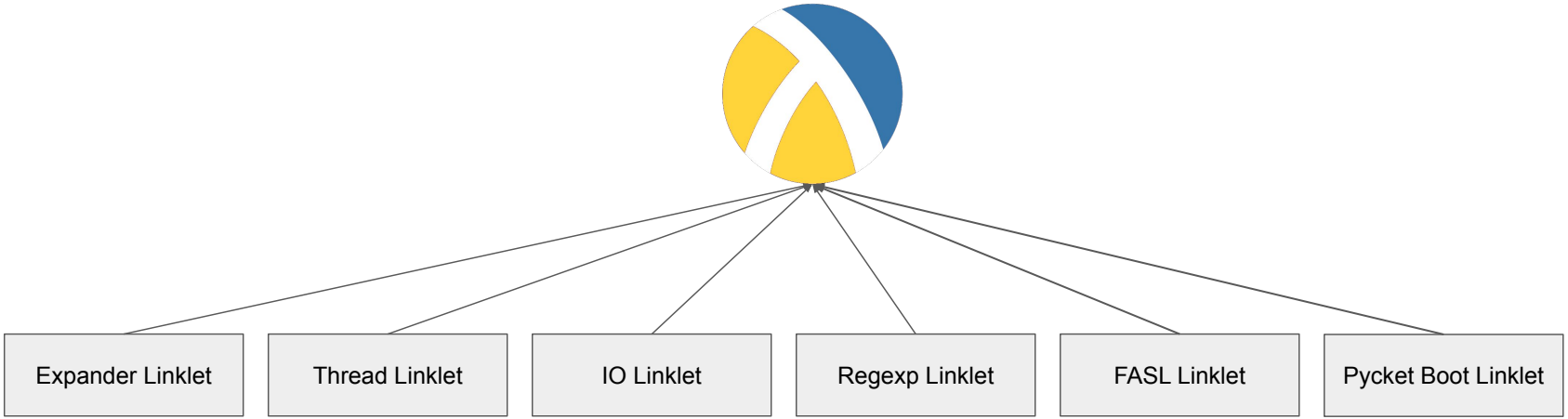
From Rudimentary Interpreter → Full Runtime



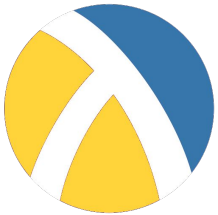
From Rudimentary Interpreter → Full Runtime



From Rudimentary Interpreter → Full Runtime

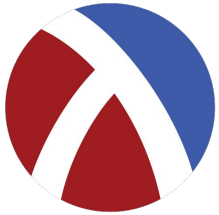


From Rudimentary Interpreter → Full Runtime



```
Welcome to Pycket v8.17.0.3
>
```

Racket on Pycket



```
Welcome to Racket v8.17.0.3
>
```

Racket on Chez

Thesis:

Efficient self-hosting of a full-scale functional language on a meta-tracing JIT compiler is achievable.

Plan:

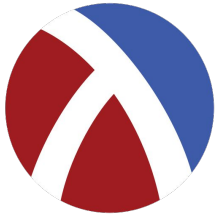
1. ✓ Introduction: Unpack the Statement & Survey Related Work
2. ✓ Main Evidence: Racket on Pycket
3. Performance Evaluation: Cost of Self-hosting
4. Conclusion: Significance & Future Work

Investigating the Cost of Self-Hosting on Meta-tracing

Actors:

Original Pycket

Racket on Pycket



Racket on Chez

Investigating the Cost of Self-Hosting on Meta-tracing

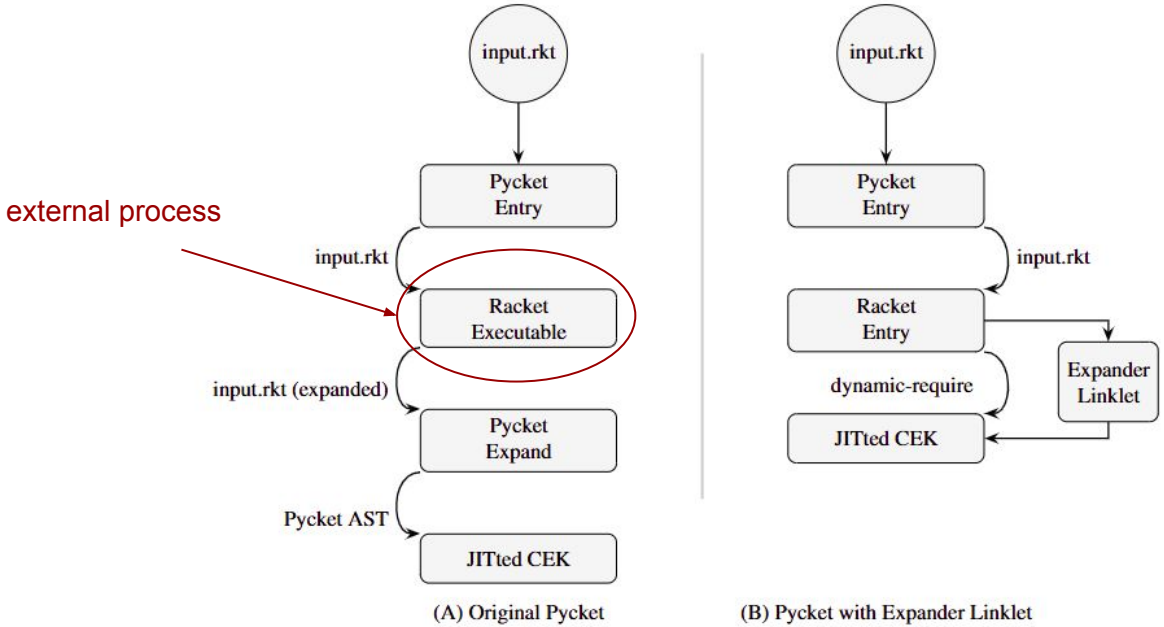


Figure 4.3: Comparison of two Pycket front-ends. (A) The original front-end with offline expansion; (B) The new front-end with expansion in run-time.

Investigating the Cost of Self-Hosting on Meta-tracing

Back-end

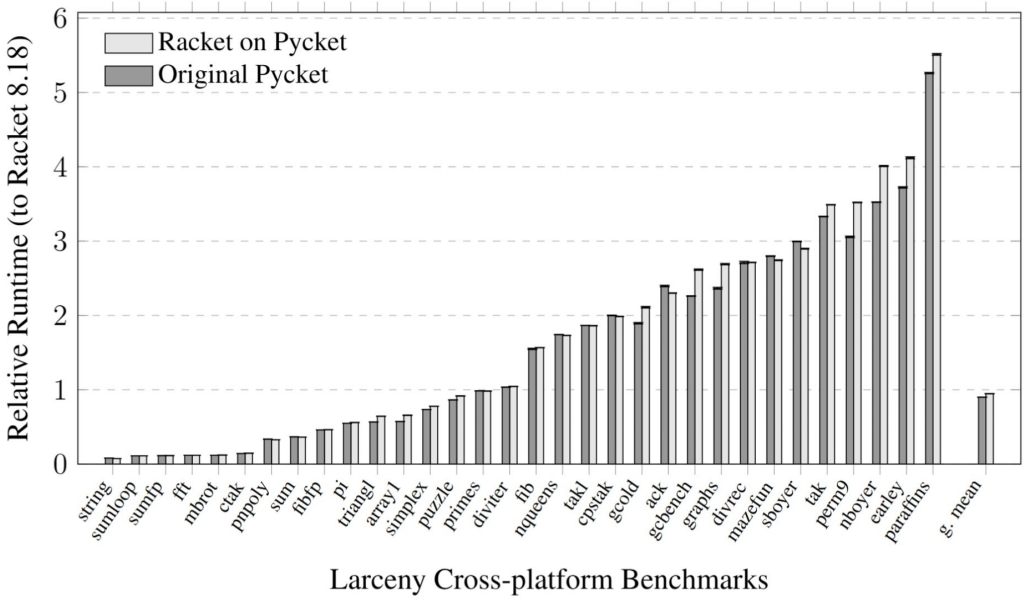


Figure 6.1: Fully-expanded program evaluation performance. Racket on Pycket vs Original Pycket, relative to Racket 8.18 – lower is better.

Investigating the Cost of Self-Hosting on Meta-tracing

```
#lang racket/base

(require "conf.rkt")

(define (ack m n)
  (cond ((= m 0) (+ n 1))
        ((= n 0) (ack (- m 1) 1))
        (else (ack (- m 1) (ack m (- n 1))))))

(for ([i (in-range outer)])
  (time (for ([j (in-range ack-iters)])
          (ack 3 9))))
```

(A) Back-end benchmark

```
#lang racket/base

(require racket/syntax "conf.rkt")

(define stx
  #'(module ack racket/base
      (define outer 100)
      (define (ack m n)
        (cond ((= m 0) (+ n 1))
              ((= n 0) (ack (- m 1) 1))
              (else (ack (- m 1) (ack m (- n 1))))))
      ))))

(for ([i (in-range expand-outer)])
  (time (for ([j (in-range expand-inner)])
          (expand stx)))))
```

(B) Front-end benchmark

Figure 6.2: Example benchmark used in measuring (A) Back-end and (B) Front-end performance.

Investigating the Cost of Self-Hosting on Meta-tracing

Front-end

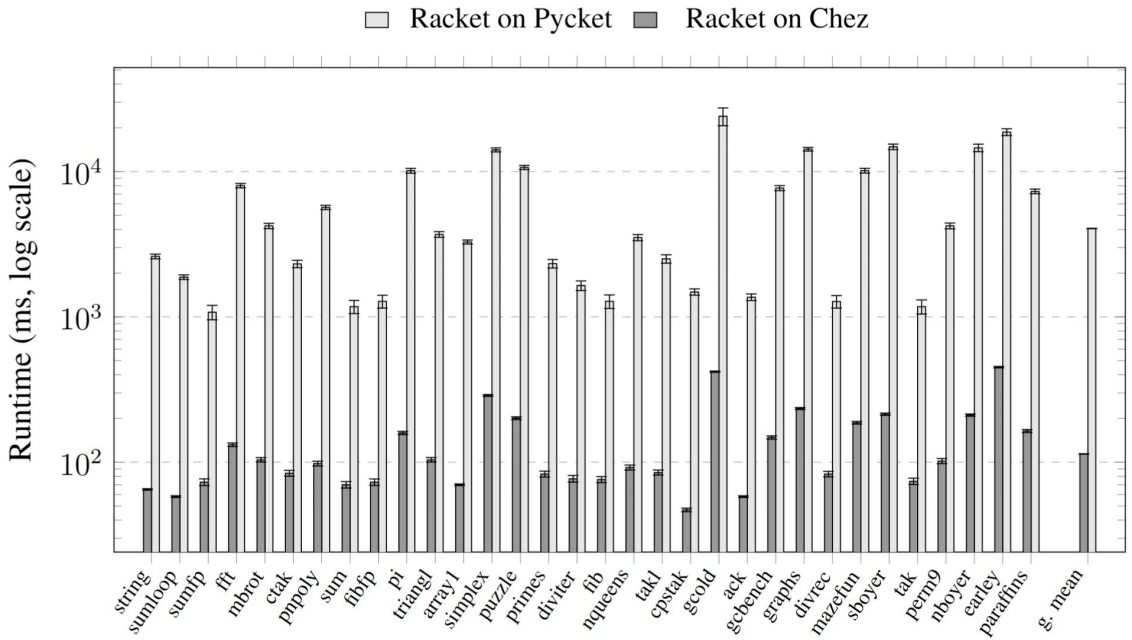


Figure 6.3: Program-expansion performance. Racket on Pycket vs. Racket on Chez – lower is better.

Two Performance Issues

1. Self-hosting seems incompatible with the nature of tracing

2. Memory pressure caused by large long-lived objects on the heap

Self-hosting seems incompatible with the nature of tracing

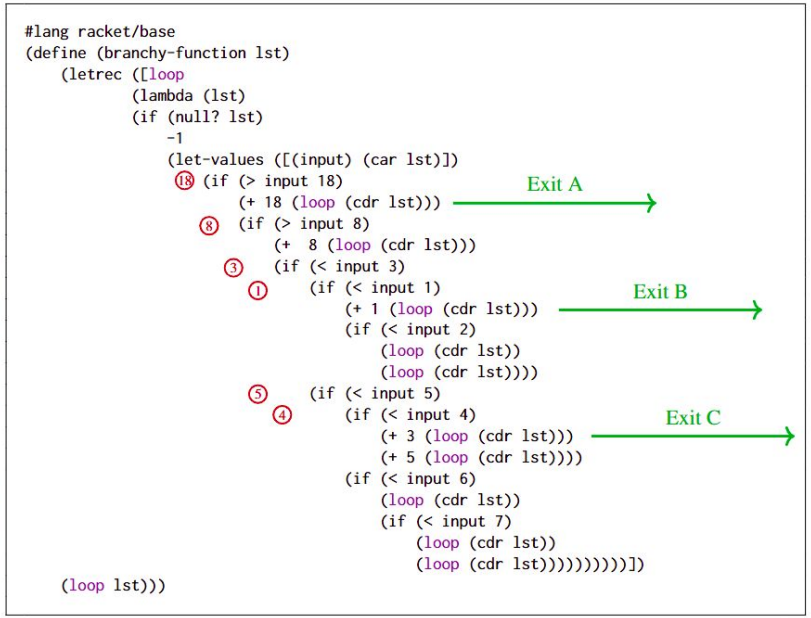


Figure 6.4: Branchy: a branch-heavy program designed to exhibit data-dependent behavior.

Self-hosting seems incompatible with the nature of tracing

Tracing JITs operate under two fundamental assumptions:

1. Programs spend most of their execution time in loops.
2. Iterations of the same loop often follow similar code paths

Self-hosting seems incompatible with the nature of tracing

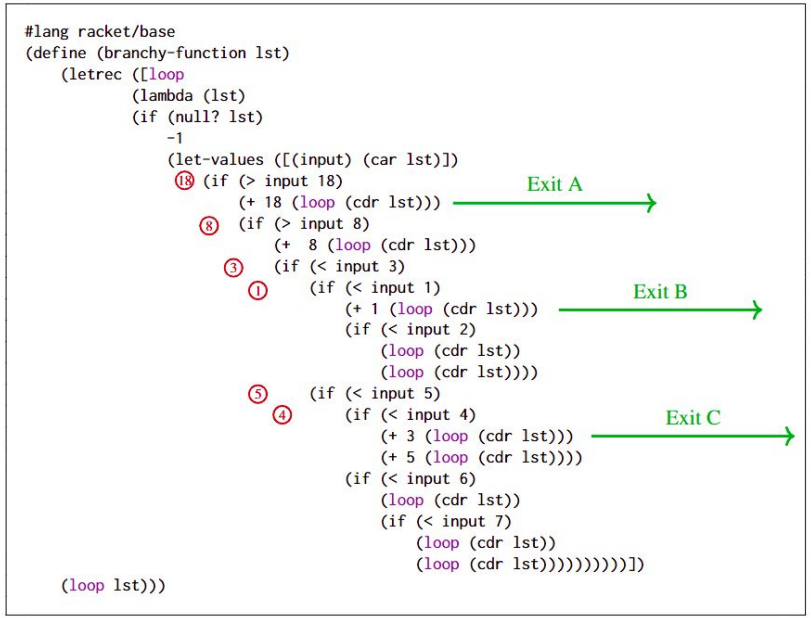


Figure 6.4: Branchy: a branch-heavy program designed to exhibit data-dependent behavior.

Self-hosting seems incompatible with the nature of tracing

```
# Inner loop with 18 ops
label(p19, p24, p8, p21, p1, p22, p30)                54
i31 = getfield_gc_i(p24)      ; FixnumCons._car       55
guard_not_invalidated()                               56

i33 = int_gt(i31, 18)                                 57
guard_false(i33)                                      58

i35 = int_gt(i31, 8)                                  59
guard_false(i35)                                      60

i37 = int_lt(i31, 3)                                  61
guard_true(i37)                                       62

i39 = int_lt(i31, 1)                                  63
guard_true(i39)                                       64

p40 = getfield_gc_r(p24)      ; FixnumCons._cdr       65
guard_nonnull_class(p40)                              66
p42 = new_with_vtable()                               67
setfield_gc(p42, p8)                                  68
setfield_gc(p42, p21)                                 69
setfield_gc(p42, p1)                                  70
setfield_gc(p42, ConstPtr(ptr43))                     71
jump(p19, p40, p8, p30, p42, p22, p30)                72
```

Figure 6.5: RPython trace inner loop for Branchy running all-zero input taking a long 18-8-3-1 path.

Internal structure of input influencing tracing

	Repeating Branches	**Random Branches**
Tracing (s)	**0.004566**	**0.153118**
Backend (s)	0.001239	0.044965
TOTAL (s)	1.185808	2.242329
ops	2130	147895
heapcached ops	572	41615
recorded ops	**462**	**31202**
calls	12	380
guards	128	7391
opt ops	243	11448
opt guards	75	3768
opt guards shared	46	2553
nvirtuals	74	1982
nvholes	8	126
nvreused	36	907
Total # loops	**2**	**39**
Total # bridges	**3**	**248**

Self-hosting seems incompatible with nature of tracing

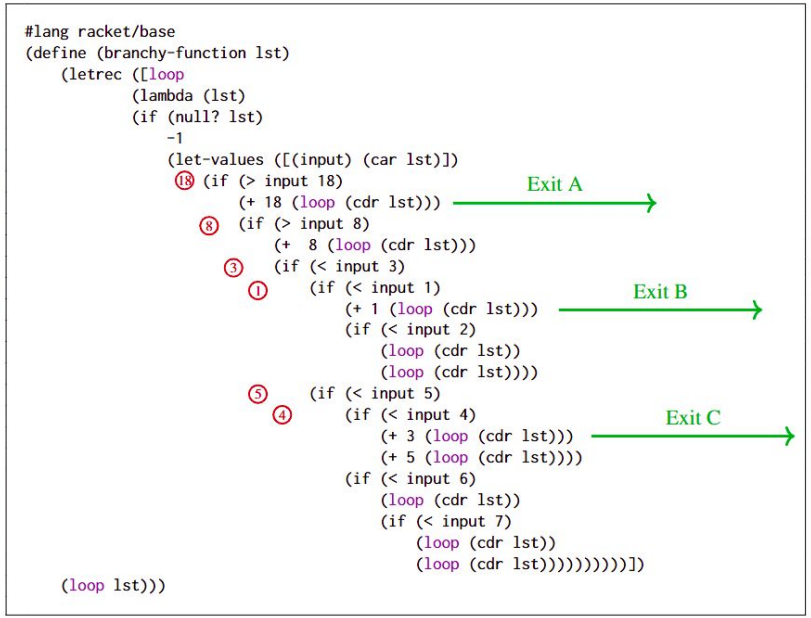


Figure 6.4: Branchy: a branch-heavy program designed to exhibit data-dependent behavior.

Two Performance Issues

1. Self-hosting seems incompatible with the nature of tracing

2. Memory pressure caused by large long-lived objects on the heap

Memory Pressure of Self-hosting

	Collections	Bytes copied (KiB)	Old-gen growth (MiB)	GC wall-time (ms)
Original Pycket	15	1367.1	19.2	40.6
Racket on Pycket	500	546.4	148.0	890.6

Table 6.4: Memory footprint of original Pycket vs Pycket hosting Racket, both loading `racket/base`. Nursery size: 32M

Memory Pressure of Self-hosting

Minimark GC

- RPython installs a built-in GC.
- Generational semi-space collector with nursery (~ cpu L2 size).
- Objects born in nursery → survivors go to older generations.
- Most objects are small and die young.
- Large and old objects are moved to an external generation → mark-and-sweep.

Memory Pressure of Self-hosting

Minimark GC

- RPython installs a built-in GC.
- Generational semi-space collector with nursery (~ cpu L2 size).
- Objects born in nursery → survivors go to older generations.
- Most objects are small and die young.
- Large and old objects are moved to an external generation → mark-and-sweep.

- Self-hosting brings large objects that live very long.
  - Expander linklet instance
  - Long continuation chains in CEK due to deep control-flow in real-world Racket code.

Investigating the Cost of Self-Hosting on Meta-tracing

Is efficient self-hosting on meta-tracing JIT possible?

Thesis:

Efficient *self-hosting of a full-scale functional language on a meta-tracing JIT compiler is achievable.*

1. Self-hosting seems incompatible with the nature of tracing

2. Memory pressure caused by large long-lived objects on the heap

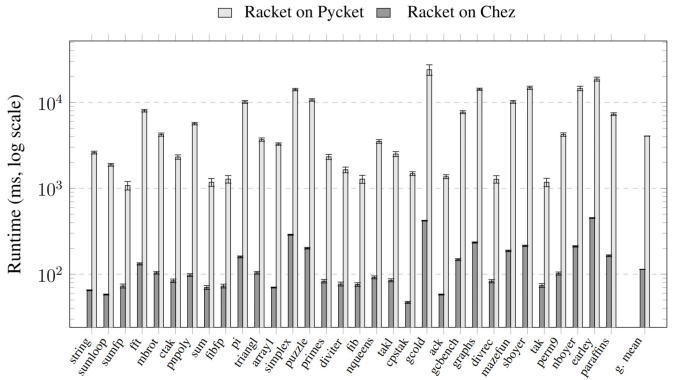


Figure 6.3: Program-expansion performance. Racket on Pycket vs. Racket on Chez – lower is better.

Targeted Approaches for Improving Performance

1. Self-hosting seems incompatible with the nature of tracing
   - ➢ Guiding Tracer Away from Branch-Heavy Computation
     - Using *meta-hints* to guide the tracer
     - Extend Pycket *green variables* with nested if depth

2. Memory pressure caused by large long-lived objects on the heap
   - ➢ CEK & Stackful Hybrid Computational Model

Targeted Approaches for Improving Performance

- Guiding Tracer Away from Branch-Heavy Computation

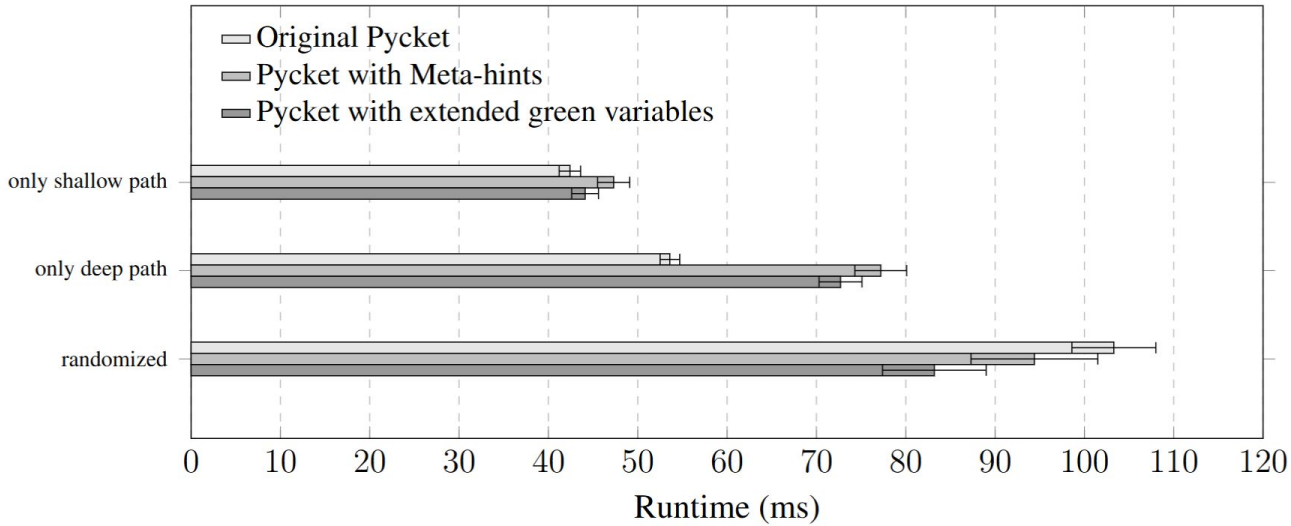


Figure 7.3: Experiment results for targeted improvement approaches running Branchy. Lower is better.

Targeted Approaches for Improving Performance

- Guiding Tracer Away from Branch-Heavy Computation

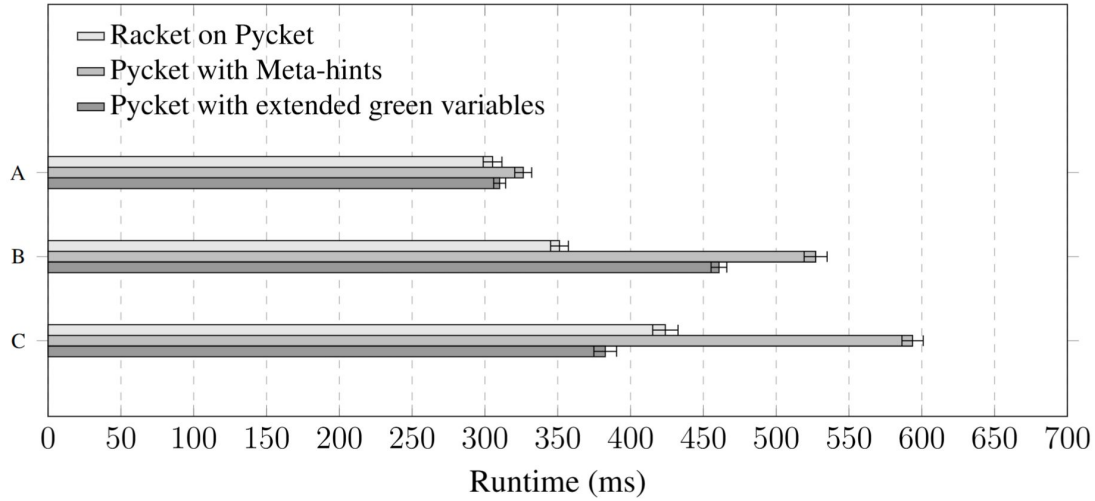


Figure 7.4: Experiment results of regular expression matching with targeted improvement approaches. All interpreters use the regexp linklet. Lower is better.

Targeted Approaches for Improving Performance

- Guiding Tracer Away from Branch-Heavy Computation

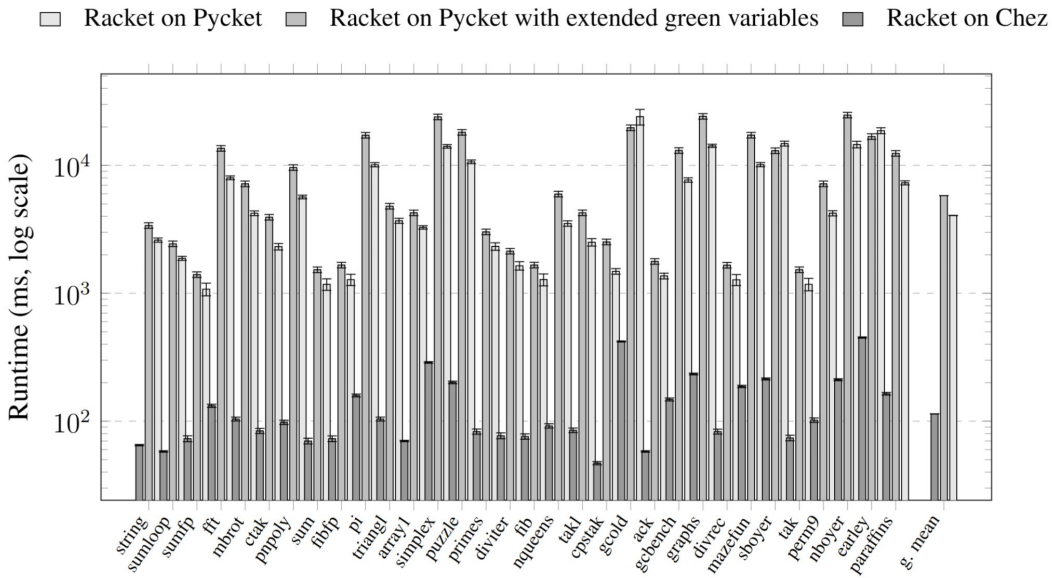


Figure 7.5: Program-expansion performance. Racket on Pycket with and without the extended green variables optimization vs. Racket on Chez – lower is better.

Targeted Approaches for Improving Performance

- CEK & Stackful Hybrid Computational Model

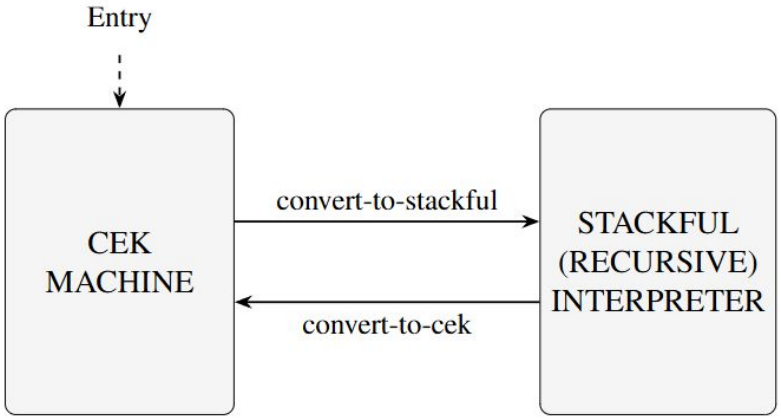


Figure 7.4: Stackful & CEK Switch

Targeted Approaches for Improving Performance

- CEK & Stackful Hybrid Computational Model

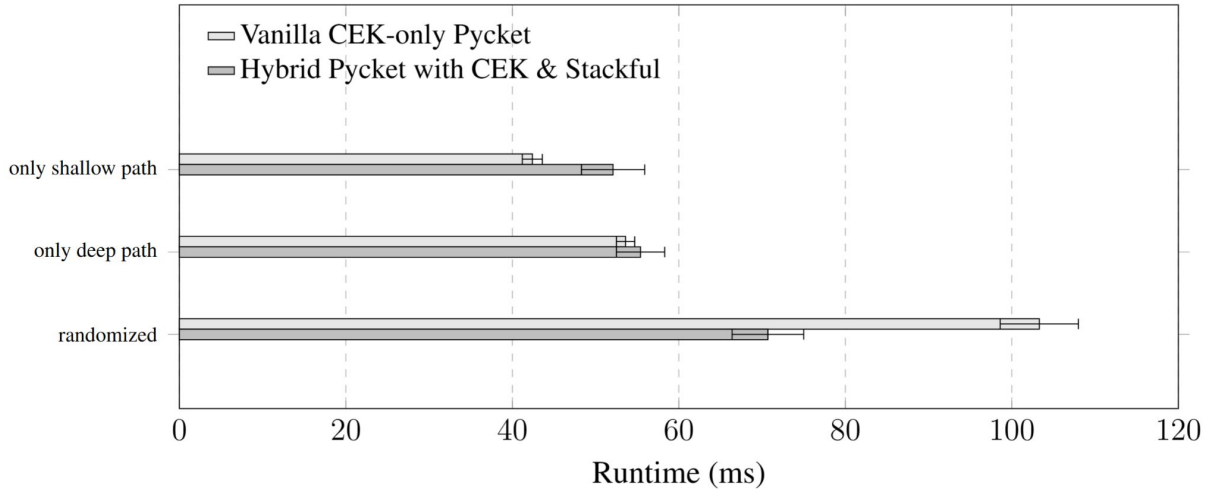


Figure 7.7: Experiment results of CEK-only Pycket vs CEK+Stackful Pycket running Branchy. Lower is better.

Targeted Approaches for Improving Performance

- CEK & Stackful Hybrid Computational Model

	Collections	Bytes copied (KiB)	Old-gen growth (MiB)	GC wall-time (ms)
Original Pycket	32	1161.3	17.8	70.3
Hybrid Pycket	10	2713.4	2.7	20.2

Table 7.1: Memory footprint of CEK-only Pycket, Hybrid Pycket with CEK & Stackful, running Branchy with randomized input. Nursery size: 32M

Targeted Approaches for Improving Performance

- CEK & Stackful Hybrid Computational Model

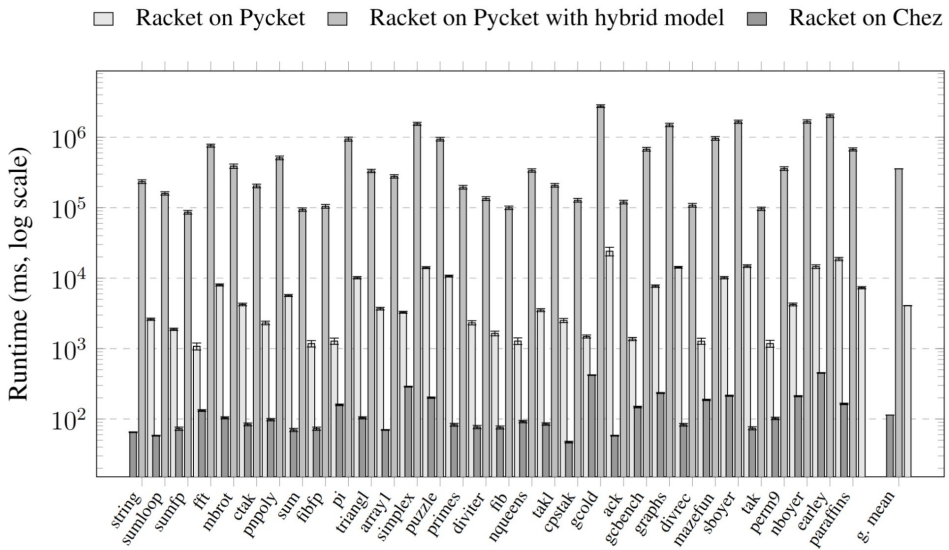


Figure 7.8: Program-expansion performance. CEK-only Racket on Pycket vs Racket on Pycket with CEK & Stackful hybrid model vs. Racket on Chez — lower is better.

Investigating the Cost of Self-Hosting on Meta-tracing

Is efficient self-hosting on meta-tracing JIT possible?

Thesis:

Efficient *self-hosting of a full-scale functional language on a meta-tracing JIT compiler is achievable.*

1. Self-hosting seems incompatible with the nature of tracing
   - ➢ Guiding Tracer Away from Branch-Heavy Computation

2. Memory pressure caused by large long-lived objects on the heap
   - ➢ CEK & Stackful Hybrid Computational Model

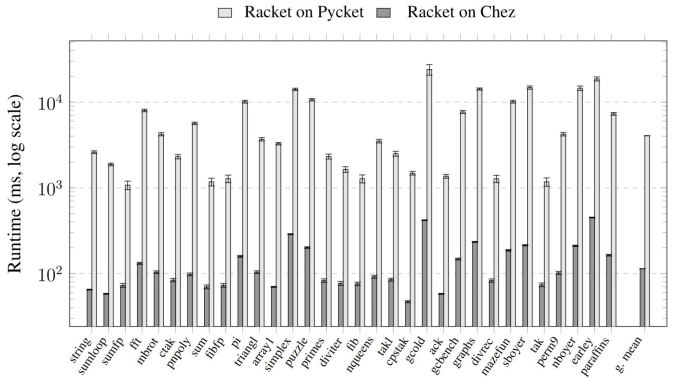


Figure 6.3: Program-expansion performance. Racket on Pycket vs. Racket on Chez – lower is better.

Thesis:

Efficient self-hosting of a full-scale functional language on a meta-tracing JIT compiler is achievable.

Plan:

1. ✓ Introduction: Unpack the Statement & Survey Related Work
2. ✓ Main Evidence: Racket on Pycket
3. ✓ Performance Evaluation: Cost of Self-hosting
4. Conclusion: Technical Contributions & Future Work

Technical Contributions

The primary technical contributions of this dissertation are:

- Pycket, a complete run-time for Racket.
- Operational semantics of linklets - formalism model in PLT Redex.
- Operational semantics for a hybrid computational model (CEK & Stackful) - formalism model in PLT Redex.
- Discovery and analysis of performance issues fundamental to self-hosting on meta-tracing. Solution approaches with prototype implementations.

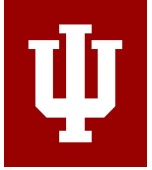
Technical Contributions

The primary technical contributions of this dissertation are:

- Pycket:
  - https://github.com/pycket/pycket
- Linklet Semantics PLT Redex Model:
  - https://github.com/cderici/linklets-redex-model
- CEK & Stackful Hybrid PLT Redex Model:
  - https://github.com/cderici/abstract-machine-interp
- All performance experiment setup, k8s script generators, custom plotting library:
  - https://github.com/cderici/pycket-performance
- RPython Trace visual analysis tool:
  - https://github.com/cderici/trace-draw

Future Work

- Unlock full potential by ironing out performance issues
- Racket-on-Pycket & Racket-on-Chez → 1:1 comparison across diverse computational patterns.
- Self-hosting on method-based JIT → GraalVM
- Would you like to be globally famous? Tracing branch-heavy code



SELF-HOSTING FUNCTIONAL PROGRAMMING LANGUAGES ON META-TRACING JIT COMPILERS

Final Examination for Doctor of Philosophy in Computer Science

Caner Derici

Doctoral Committee:

- Sam Tobin-Hochstadt, Ph.D.
- Jeremy Siek, Ph.D.
- Daniel Leivant, Ph.D.
- Amr Sabry, Ph.D.

Thank You!

— EXTRA SLIDES —

Trace: linear sequence of machine code instructions

```
# start of the trace (preamble)
label(p0, p1)                                   1
guard_not_invalidated()                         2
guard_class(p0, ConsEnv)                        3
p3 = getfield_gc_r(p0, ConsEnv.prev)            4
guard_class(p3, ConsEnv)                        5
i5 = getfield_gc_i(p3, Fixnum)                  6
i6 = getfield_gc_i(p0, Fixnum)                  7
i7 = int_add_ovf(i5, i6)                        8
guard_no_overflow()                             9
guard_class(p1, LetCont)                        10
p9 = getfield_gc_r(p1, LetCont.ast)             11
guard_value(p9, ConstPtr(ptr10))                12
p11 = getfield_gc_r(p1, LetCont.env)            13
p12 = getfield_gc_r(p1, LetCont.prev)           14

# peeled-iteration (inner loop)
label(p11, i7, p12, "64723392")                 15
guard_not_invalidated()                         16
guard_class(p11, ConsEnv)                       17
i14 = getfield_gc_i(p11, Fixnum)                18
i15 = int_add_ovf(i14, i7)                      19
guard_no_overflow()                             20
guard_class(p12, LetCont)                       21
p17 = getfield_gc_r(p12, LetCont.ast)           22
guard_value(p17, ConstPtr(ptr18))               23
p19 = getfield_gc_r(p12, LetCont.env)           24
p20 = getfield_gc_r(p12, LetCont.prev)          25
jump(p19, i15, p20, "64723392")                 26
```

Figure 2.1: A trace in a tracing JIT compiler is a linear sequence of machine code instructions.

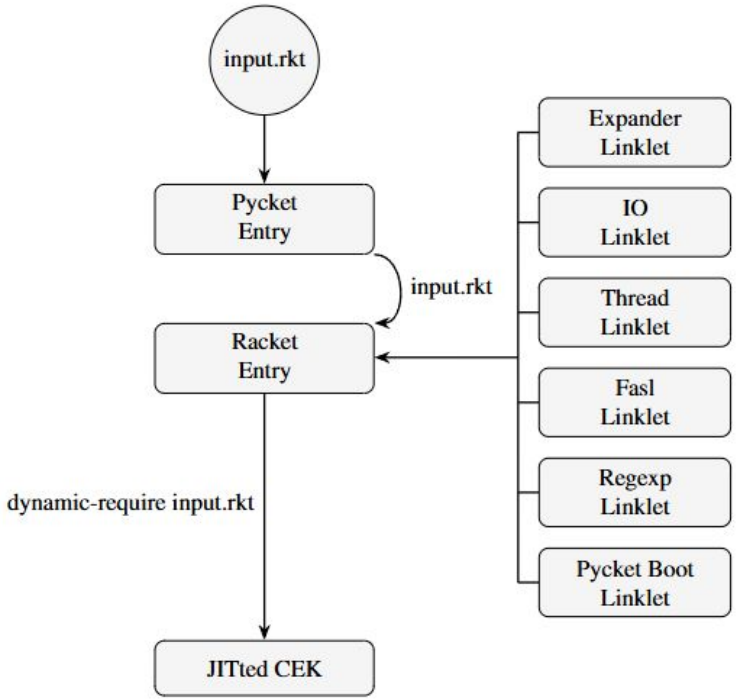
Meta-tracing & Pycket Backend

Meta-tracing & Pycket Backend

$$
\begin{array}{lcl}
e & ::= & x \mid \lambda x.\, e \mid e\, e \\
\kappa & ::= & [\,] \mid \mathsf{arg}(e, \rho) :: \kappa \mid \mathsf{fun}(v, \rho) :: \kappa
\end{array}
$$

$$
\begin{array}{rcl}
\langle x, \rho, \kappa \rangle & \longmapsto & \langle \rho(x), \rho, \kappa \rangle \\
\langle (e_1\, e_2), \rho, \kappa \rangle & \longmapsto & \langle e_1, \rho, \mathsf{arg}(e_2, \rho) :: \kappa \rangle \\
\langle v, \rho, \mathsf{arg}(e, \rho') :: \kappa \rangle & \longmapsto & \langle e, \rho', \mathsf{fun}(v, \rho) :: \kappa \rangle \\
\langle v, \rho, \mathsf{fun}(\lambda x.\, e, \rho') :: \kappa \rangle & \longmapsto & \langle e, \rho'[x \mapsto v], \kappa \rangle
\end{array}
$$

Full Pycket Front-end with Bootstrapping Linklets



Effects of Nursery Size on Total GC Time in Pycket

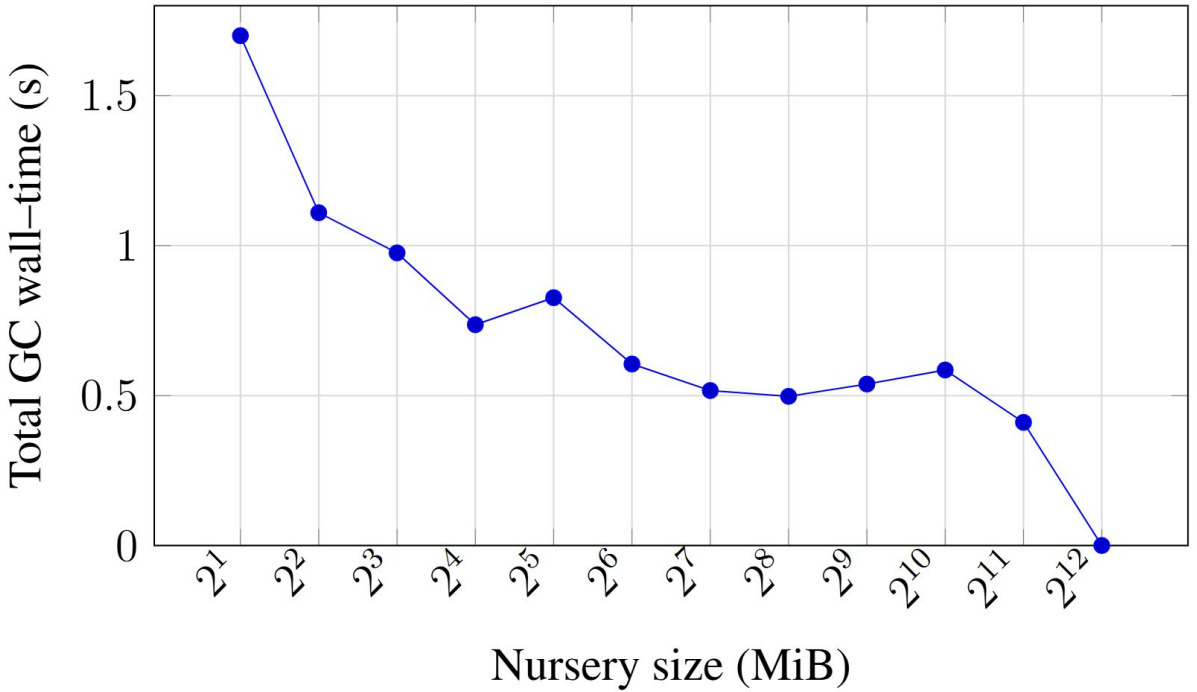


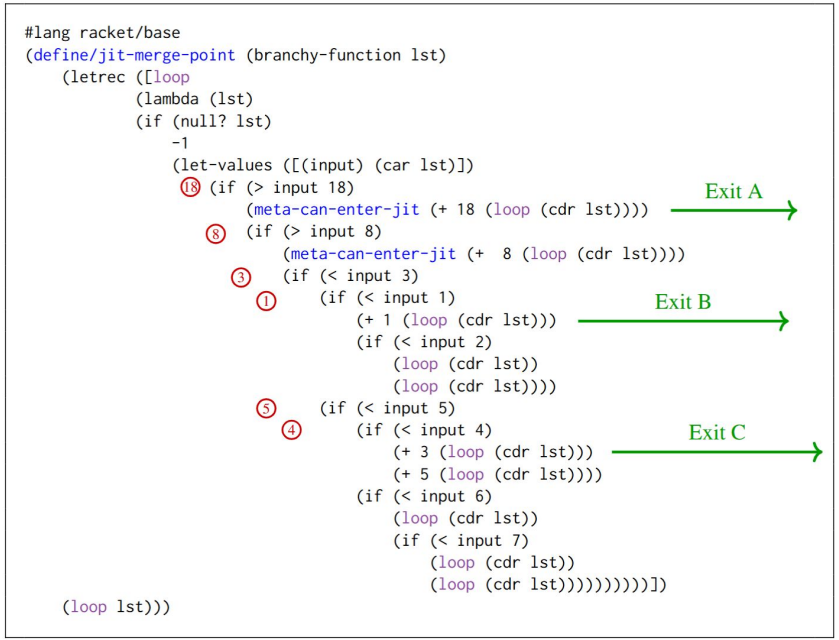
Figure 6.11: Effect of nursery size on total GC wall–time for Pycket with self-hosting.

Using meta-hints to guide the tracer

```
driver_two_state = jit.JitDriver(reds=["env", "cont"],                                  1
                                 greens=["ast", "came_from"])                           2
                                                                                        3
def inner_interpret_two_state(ast, env, cont):                                          4
    came_from = ast                                                                     5
    while True:                                                                         6
        driver_two_state.jit_merge_point(ast=ast, came_from=came_from, env=env, cont=cont)  7
        came_from = ast if isinstance(ast, App) else came_from                          8
        t = type(ast)                                                                   9
                                                                                        10
        if t is Let:                                                                    11
            ast, env, cont = ast.interpret(env, cont)                                   12
        elif t is If:                                                                   13
            ast, env, cont = ast.interpret(env, cont)                                   14
        elif t is Begin:                                                                15
            ast, env, cont = ast.interpret(env, cont)                                   16
        else:                                                                           17
            ast, env, cont = ast.interpret(env, cont)                                   18
        if ast.should_enter:                                                            19
            driver_two_state.can_enter_jit(ast=ast, came_from=came_from, env=env, cont=cont)  20
```

Figure 2.2: Pycket's CEK loop uses interpreter hints to provide runtime feedback to the meta-tracer.

Using meta-hints to guide the tracer




```
1  #lang racket/base
2  (define/jit-merge-point (branchy-function lst)
3      (letrec ([loop
4              (lambda (lst)
5              (if (null? lst)
6                  -1
7                  (let-values ([(input) (car lst)])
8                   (18 (if (> input 18)
9                          (meta-can-enter-jit (+ 18 (loop (cdr lst))))
10                     (8  (if (> input 8)
11                             (meta-can-enter-jit (+  8 (loop (cdr lst))))
12                         (3   (if (< input 3)
13                                 (1     (if (< input 1)
14                                            (+ 1 (loop (cdr lst)))
15                                            (if (< input 2)
16                                                (loop (cdr lst))
17                                                (loop (cdr lst))))
18                                    (5     (if (< input 5)
19                                               (4     (if (< input 4)
20                                                            (+ 3 (loop (cdr lst)))
21                                                            (+ 5 (loop (cdr lst))))
22                                                        (if (< input 6)
23                                                            (loop (cdr lst))
24                                                            (if (< input 7)
25                                                                (loop (cdr lst))
26                                                                (loop (cdr lst))))))))))]))
27     (loop lst)))
```

Figure 7.1: Branchy; from Figure 6.4 annotated with meta-hints.

Using meta-hints to guide the tracer

```
1  (define/jit-merge-point (match-pat pattern p-pos str s-pos m)
2    (cond
3      ;; Done with the pattern
4      [(>= p-pos (string-length pattern)) m]
5      ;; $ – Match the end of a string  a$
6      [(char=? (string-ref pattern p-pos) #\$) (and (>= s-pos (string-length str)) m)]
7      ;; ? – Match 0 or 1 of the previous character   a?  "", a
8      [(and (< (+ p-pos 1) (string-length pattern)) (char=? (string-ref pattern (+
       p-pos 1)) #\?))
9       (match-huh pattern p-pos str s-pos (+ p-pos 1) m)]
10     ;; * – Match 0 or more of the previous character  a*  "", a, aa, aaa
11     [(and (< (+ p-pos 1) (string-length pattern)) (char=? (string-ref pattern (+
       p-pos 1)) #\*))
12      (match-star pattern p-pos str s-pos (+ p-pos 1) m)]
13     ;; . – Match any character literal  .   a, b, c, d, e ...
14     [(char=? (string-ref pattern p-pos) #\.)
15      (and (< s-pos (string-length str))
16           (meta-can-enter-jit (match-pat pattern (add1 p-pos) str (add1 s-pos)
17                                          (cons (string-ref str s-pos) m))))]
18     ;; literal – Match the specified character literal  x, q
19     [else
20      (and (char=? (string-ref pattern p-pos) (string-ref str s-pos))
21           (meta-can-enter-jit (match-pat pattern (add1 p-pos) str (add1 s-pos)
22                                          (cons (string-ref str s-pos) m))))]))
23
```

Using meta-hints to guide the tracer

```
23
24  (define (match-huh pattern p-pos str s-pos huh-pos m)
25    (or (and (< s-pos (string-length str))
26             (char=? (string-ref pattern p-pos) (string-ref str s-pos))
27             (match-pat pattern (add1 huh-pos) str (add1 s-pos)
28                        (cons (string-ref str s-pos) m)))
29        (match-pat pattern (add1 huh-pos) str s-pos m)))
30
31  (define (match-star pattern p-pos str s-pos star-pos m)
32    (or (and (< s-pos (string-length str))
33             (char=? (string-ref pattern p-pos) (string-ref str s-pos))
34             (match-pat pattern p-pos str (add1 s-pos)
35                        (cons (string-ref str s-pos) m)))
36        (match-pat pattern (add1 star-pos) str s-pos m)))
```

Figure 7.2: A simple regular expression matcher annotated with meta-hints

Extend current Pycket green variables with nested if depth

```
driver_two_state = jit.JitDriver(reds=["env", "cont"],
                                 greens=["ast", "came_from", "if_depth"])
```

- Depth scores at A-normalization
- Deeper branches increase if_depth to change green variables
- Shallow branches greens stay the same so counter revs up

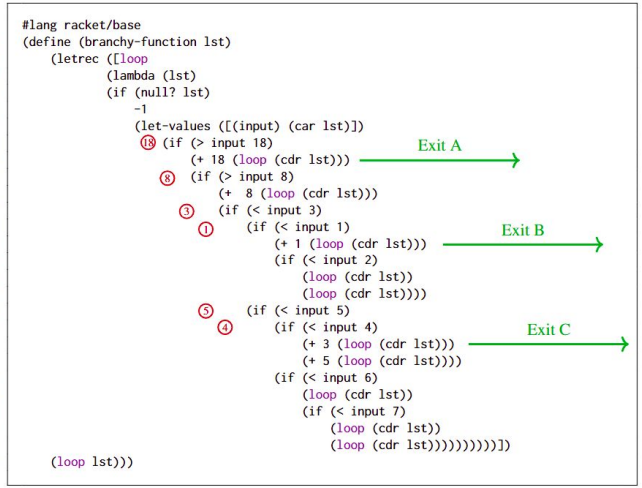


Figure 6.4: Branchy: a branch-heavy program designed to exhibit data-dependent behavior.

Linklet Toplevel Example - 1

$$
\begin{aligned}
L &::= (\textbf{linklet}\ ((imp\ \ldots)\ \ldots)\ (exp\ \ldots)\ l\text{-}top\ \ldots\ e) \\
l\text{-}top &::= (\textbf{define-values}\ (x)\ e) \mid e \\
imp &::= x \mid (x\,x) \qquad \text{[external-name internal-name]} \\
exp &::= x \mid (x\,x) \qquad \text{[internal-name external-name]} \\
e &::= x \mid v \mid (e\ e\ \ldots) \mid (\textbf{if}\ e\ e\ e) \mid (o\ e\ e) \\
&\mid (\textbf{begin}\ e\ e\ \ldots) \mid (\textbf{lambda}\ (x\_!\_\ \ldots)\ e) \\
&\mid (\textbf{set!}\ x\ e) \mid (\textbf{raises}\ e) \\
&\mid (\textbf{var-ref}\ x) \mid (\textbf{var-ref/no-check}\ x) \\
&\mid (\textbf{var-set!}\ x\ e) \\
&\mid (\textbf{var-set/check-undef!}\ x\ e)
\end{aligned}
$$

Figure 3.3: Linklet Source Language

$$
\begin{aligned}
CL &::= \boldsymbol{\Phi^C}(L) \\
L\text{-}obj &::= (\boldsymbol{L_\alpha}\ c\text{-}imps\ c\text{-}exps\ l\text{-}top \ldots) \mid (\boldsymbol{L_\beta}\ x\ l\text{-}top\ \ldots) \\
LI &::= (\textbf{linklet-instance}\ (x\ var)\ \ldots) \\
c\text{-}imps &::= ((imp\text{-}obj\ \ldots)\ \ldots) \\
c\text{-}exps &::= (exp\text{-}obj\ \ldots) \\
imp\text{-}obj &::= (\textbf{Import}\ x\ x\ x) \qquad \text{[id internal external]} \\
exp\text{-}obj &::= (\textbf{Export}\ x\ x\ x) \qquad \text{[id internal external]}
\end{aligned}
$$

Φ^C : **compile-linklet**

Figure 3.7: Linklet Runtime Language

Linklet Toplevel Example - 2

```
1  > (define k (lambda () a))
2  > (define a 10)
3  > (k)
4  10
```

Linklet Toplevel Example - 3

	program	ρ	σ

	`(program`
	`  ([l1 (linklet () (a k)`
	`  (define-values (k) (lambda () a)) (void))]`
	`  [l2 (linklet () (a) (define-values (a) 10) (void))]`
	`  [l3 (linklet () (k) (k))])`
	`  (let-inst t (make-instance)`
	`  (seq ($\phi^I$ l1 #:t t) ($\phi^I$ l2 #:t t) ($\phi^I$ l3 #:t t))))`
		[]	[]

$\longrightarrow^*_{\beta p}$	`(program ()`
	`(let-inst t (make-instance)`
	`(seq`
	`  ($\phi^I$`
	`  (L$\alpha$ () ((Export a1 a a) (Export k1 k k))`
	`  (define-values (k) (lambda () (var-ref a1)))`
	`  (var-set! k1 k) (void)) #:t t)`
	`  ($\phi^I$`
	`  (L$\alpha$ () ((Export a1 a a))`
	`  (define-values (a) 10) (var-set! a1 a) (void)) #:t t)`
	`  ($\phi^I$`
	`  (L$\alpha$ () ((Export k1 k k))`
	`  ((var-ref k1))) #:t t))))`
		[]	[]

Linklet Toplevel Example - 4

$\longrightarrow^{*}_{\beta p}$

```
(program ()
(seq
  (ϕ^I (Lβ t (define-values (k) (lambda () (var-ref a1)))
               (var-set! k1 k) (void)))
  (ϕ^I
    (Lα () ((Export a1 a a))
     (define-values (a) 10) (var-set! a1 a) (void)) #:t t)
  (ϕ^I
    (Lα () ((Export k1 k k))
     ((var-ref k1))) #:t t)))
```

$[\text{k1} \to var_k,$
$\text{a1} \to var_a]$

$[var_a, var_k \to \text{uninit},$
$\text{t} \to (LI$
$(\text{a } var_a)\ (\text{k } var_k))]$

Linklet Toplevel Example - 5

	program	ρ	σ
$\longrightarrow^*_{\beta p}$	(program () (seq (ϕ^I (Lβ t (var-set! k1 k) (void))) (ϕ^I (Lα () ((Export a1 a a)) (define-values (a) 10) (var-set! a1 a) (void)) #:t t) (ϕ^I (Lα () ((Export k1 k k)) ((var-ref k1))) #:t t)))	[k $\rightarrow$ $cell_1$, k1 $\rightarrow$ var_k, a1 $\rightarrow$ var_a]	[$cell_1$ $\rightarrow$ closure, var_a, var_k $\rightarrow$ uninit, t $\rightarrow$ (LI (a var_a) (k var_k))]
$\longrightarrow^*_{\beta p}$	(program () (seq (void) (ϕ^I (Lα () ((Export a1 a a)) (define-values (a) 10) (var-set! a1 a) (void)) #:t t) (ϕ^I (Lα () ((Export k1 k k)) ((var-ref k1))) #:t t)))	[]	[$cell_1$ $\rightarrow$ closure, var_a $\rightarrow$ uninit, var_k $\rightarrow$ $cell_1$ t $\rightarrow$ (LI (a var_a) (k var_k))]

Linklet Toplevel Example - 6

$\longrightarrow^*_{\beta p}$

```
(program ()
(seq
  (void)
  (ϕ^I
    (Lβ t (define-values (a) 10) (var-set! a1 a) (void)))
  (ϕ^I
    (Lα () ((Export k1 k k))
     ((var-ref k1))) #:t t)))
```

$[\text{a1} \to var_a]$

$[cell_1 \to \text{closure},$
$var_a \to \text{uninit},$
$var_k \to cell_1$
$\text{t} \to (LI$
$(\text{a } var_a)\ (\text{k } var_k))]$

$\longrightarrow^*_{\beta p}$

```
(program ()
(seq
  (void)
  (ϕ^I
    (Lβ t (var-set! a1 a) (void)))
  (ϕ^I
    (Lα () ((Export k1 k k))
     ((var-ref k1))) #:t t)))
```

$[\text{a} \to 10,$
$\text{a1} \to var_a]$

$[cell_1 \to \text{closure},$
$var_a \to \text{uninit},$
$var_k \to cell_1$
$\text{t} \to (LI$
$(\text{a } var_a)\ (\text{k } var_k))]$

Linklet Toplevel Example - 7

$\longrightarrow^*_{\beta p}$

```
(program ()
(seq
  (void)
  (void)
  (ϕ^I
    (Lα () ((Export k1 k k))
     ((var-ref k1))) #:t t)))
```

[]

$[cell_1 \rightarrow \text{closure},$
$var_a \rightarrow 10,$
$var_k \rightarrow cell_1$
$\text{t} \rightarrow (LI$
$(\text{a } var_a)\ (\text{k } var_k))]$

$\longrightarrow^*_{\beta p}$

```
(program ()
(seq
  (void)
  (void)
  (ϕ^I
    (Lβ t ((var-ref k1))))))
```

$[\text{k1} \rightarrow var_k]$

$[cell_1 \rightarrow \text{closure},$
$var_a \rightarrow 10,$
$var_k \rightarrow cell_1$
$\text{t} \rightarrow (LI$
$(\text{a } var_a)\ (\text{k } var_k))]$

Linklet Toplevel Example - 8