From Stack Traces to Lazy Rewriting Sequences

Stephen Chang, Eli Barzilay, John Clements*, Matthias Felleisen

Northeastern University

*California Polytechnic State University

10/5/2011

Debugging lazy programs is hard.

Freja (Nilsson and Fritzson 1992)

Hat (Sparud and Runciman, 1997)

Buddha (Pope, 1998)

HOOD (Gill, 2000)

New Hat (Wallace et al., 2001)

HsDebug (Ennals and Peyton Jones, 2003)

Rectus (Murk and Kolmoldin, 2006)

GHCi debugger (Marlow et al., 2007)

StackTrace (Allwood et al., 2009)

Freja (Nilsson and Fritzson 1992)

Hat (Sparud and Runciman, 1997)

Buddha (Pope, 1998)

HOOD (Gill, 2000)

New Hat (Wallace et al., 2001)

HsDebug (Ennals and Peyton Jones, 2003)

Rectus (Murk and Kolmoldin, 2006)

GHCi debugger (Marlow et al., 2007)

StackTrace (Allwood et al., 2009)

What do you think is Haskell's most glaring weakness / blind spot / problem? [Tibell, Knowlson 2011]

Inadequate Tools (50%)

[State of Haskell Survey 2011]

[State of Haskell Survey 2011]

"A debugger adjusted to the complexity of debugging lazily evaluated structures." *(weeks)*

[State of Haskell Survey 2011]

"A debugger adjusted to the complexity of debugging lazily evaluated structures." *(weeks)*

"Laziness is hard to come to grips with. It's powerful and good, but it also causes strange problems that a beginner often cannot diagnose." *(months)*

[State of Haskell Survey 2011]

"A debugger adjusted to the complexity of debugging lazily evaluated structures." *(weeks)*

"Laziness is hard to come to grips with. It's powerful and good, but it also causes strange problems that a beginner often cannot diagnose." *(months)*

"I think that a good debugger that lets me step through a program /quickly and comfortably/ would be a great help." *(1yr)*

[State of Haskell Survey 2011]

"A debugger adjusted to the complexity of debugging lazily evaluated structures." *(weeks)*

"Laziness is hard to come to grips with. It's powerful and good, but it also causes strange problems that a beginner often cannot diagnose." *(months)*

"I think that a good debugger that lets me step through a program /quickly and comfortably/ would be a great help." *(1yr)*

"I'd love to see some debugging (~step by step evaluation/run tracing) support." *(2yrs)*

[State of Haskell Survey 2011]

"A debugger adjusted to the complexity of debugging lazily evaluated structures." *(weeks)*

"Laziness is hard to come to grips with. It's powerful and good, but it also causes strange problems that a beginner often cannot diagnose." *(months)*

"I think that a good debugger that lets me step through a program /quickly and comfortably/ would be a great help." *(1yr)*

"I'd love to see some debugging (~step by step evaluation/run tracing) support." *(2yrs)*

"Debugging lazy code" *(4 yrs)*

[State of Haskell Survey 2011]

"A debugger adjusted to the complexity of debugging lazily evaluated structures." *(weeks)*

"Laziness is hard to come to grips with. It's powerful and good, but it also causes strange problems that a beginner often cannot diagnose." *(months)*

"I think that a good debugger that lets me step through a program /quickly and comfortably/ would be a great help." *(1yr)*

"I'd love to see some debugging (~step by step evaluation/run tracing) support." *(2yrs)*

"Debugging lazy code" *(4 yrs)*

Better lazy **step-based** tools are needed.

What's a "step"?

HsDebug

[Ennals and Peyton Jones 2003]

HsDebug

[Ennals and Peyton Jones 2003]

- Evaluate expressions optimistically.

HsDebug

[Ennals and Peyton Jones 2003]

- Evaluate expressions optimistically.
- To preserve lazy behavior, handle special cases:
  - non-termination
  - errors

HsDebug

[Ennals and Peyton Jones 2003]

- Evaluate expressions optimistically.
- To preserve lazy behavior, handle special cases:
  - non-termination
  - errors
- Too difficult to implement.

Idea #1:

Debugger shouldn't change the program evaluation model.

GHCi Debugger

[Marlow et al. 2007]

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

GHCi Debugger

[Marlow et al. 2007]

- Shows the effects of laziness.
- "having execution jump around can be distracting and confusing"

```
test1 x y = (test2 y) + x
test2 x = x * 2
test3 x = x + 1
main = print $ test1 (1 + 2) (test3 (3 + 4))
```

- Step semantics correspond to low-level implementation
  -- unfamiliar to programmers.

Idea #2:

Debugger should use a high-level semantics familiar to programmers.

A reduction semantics-based tool

[Gibbons and Wansbrough 1996, Ariola et al. 1995]

A reduction semantics-based tool

[Gibbons and Wansbrough 1996, Ariola et al. 1995]

- Persistent arguments clutter reductions.

A reduction semantics-based tool

[Gibbons and Wansbrough 1996, Ariola et al. 1995]

- Persistent arguments clutter reductions.

$$(\lambda x.\lambda y.x + y)\ (1 + 2)\ (3 + 4)$$

A reduction semantics-based tool

[Gibbons and Wansbrough 1996, Ariola et al. 1995]

- Persistent arguments clutter reductions.

$$(\lambda x.\lambda y.x + y)\ (1 + 2)\ (3 + 4)$$
$$\rightarrow (\lambda x.\lambda y.x + y)\ 3\ (3 + 4)$$

A reduction semantics-based tool

[Gibbons and Wansbrough 1996, Ariola et al. 1995]

- Persistent arguments clutter reductions.

$$(\lambda x.\lambda y.x + y)\ (1 + 2)\ (3 + 4)$$
$$\ldots \rightarrow (\lambda x.\lambda y.3 + y)\ 3\ (3 + 4)$$

A reduction semantics-based tool

[Gibbons and Wansbrough 1996, Ariola et al. 1995]

- Persistent arguments clutter reductions.

$$(\lambda x.\lambda y.x + y)\ (1 + 2)\ (3 + 4)$$
$$\ldots \rightarrow (\lambda x.\lambda y.3 + y)\ 3\ (3 + 4)$$

- Extraneous reductions.

A reduction semantics-based tool

[Gibbons and Wansbrough 1996, Ariola et al. 1995]

- Persistent arguments clutter reductions.

$$(\lambda x.\lambda y.x + y)\ (1 + 2)\ (3 + 4)$$
$$\ldots \rightarrow (\lambda x.\lambda y.3 + y)\ 3\ (3 + 4)$$

- Extraneous reductions.

$$\rightarrow (\lambda x.(\lambda y.3 + y)\ (3 + 4))\ 3\ (3 + 4)$$

A reduction semantics-based tool

[Gibbons and Wansbrough 1996, Ariola et al. 1995]

- Persistent arguments clutter reductions.

$$(\lambda x.\lambda y.x + y)\ (1 + 2)\ (3 + 4)$$
$$\ldots \rightarrow (\lambda x.\lambda y.3 + y)\ 3\ (3 + 4)$$

- Extraneous reductions.

$$\rightarrow (\lambda x.(\lambda y.3 + y)\ (3 + 4))\ 3\ (3 + 4)$$

$$\rightarrow (\lambda x.(\lambda y.3 + y)\ 7)\ 3$$

A reduction semantics-based tool

[Gibbons and Wansbrough 1996, Ariola et al. 1995]

- Persistent arguments clutter reductions.

$$(\lambda x.\lambda y.x + y)\ (1 + 2)\ (3 + 4)$$
$$\ldots \rightarrow (\lambda x.\lambda y.3 + y)\ 3\ (3 + 4)$$

- Extraneous reductions.

$$\rightarrow (\lambda x.(\lambda y.3 + y)\ (3 + 4))\ 3\ (3 + 4)$$

$$\ldots \rightarrow (\lambda x.(\lambda y.3 + 7)\ 7)\ 3$$

A reduction semantics-based tool

[Gibbons and Wansbrough 1996, Ariola et al. 1995]

- Persistent arguments clutter reductions.

$$(\lambda x.\lambda y.x + y)\ (1 + 2)\ (3 + 4)$$
$$\ldots \rightarrow (\lambda x.\lambda y.3 + y)\ 3\ (3 + 4)$$

- Extraneous reductions.

$$\rightarrow (\lambda x.(\lambda y.3 + y)\ (3 + 4))\ 3\ (3 + 4)$$

$$\ldots \rightarrow (\lambda x.(\lambda y.10)\ 7)\ 3$$

Idea #3:

A more "intuitive" lazy semantics is needed.

Our Work

Our Work

A step-based lazy debugging tool,

based on a high-level "intuitive" lazy semantics.

Our Work

- An algebraic stepper tool for Lazy Racket.

Our Work

- An algebraic stepper tool for Lazy Racket.
- A new, "intuitive" semantics for lazy languages, $\lambda_{need\|}$

Our Work

- An algebraic stepper tool for Lazy Racket.
- A new, "intuitive" semantics for lazy languages, $\lambda_{need\|}$
- Theory:
  - $\lambda_{need\|}$ corresponds to existing lazy semantics.

Our Work

- An algebraic stepper tool for Lazy Racket.
- A new, "intuitive" semantics for lazy languages, $\lambda_{need\|}$
- Theory:
  - $\lambda_{need\|}$ corresponds to existing lazy semantics.
  - Tool is correct with respect to $\lambda_{need\|}$

$\lambda_{need\|}$

"intuitive"

$\lambda_{need\|}$

"intuitive"

=

syntactic

$\lambda_{need\|}$

"intuitive"

=

syntactic

+

substitution-based

Demo!

Semantics

$\lambda_{need\|}$: Syntax

$$e = \lambda x.e \mid e\ e \mid \ldots \mid e^{\ell}$$

$$E = [\,] \mid E\ e \mid \ldots \mid E^{\ell}$$

$$\ell \in \text{labels}$$

$\lambda_{need\|}$: Two-phase Steps

$\lambda_{need\|}$: Two-phase Steps

1) Reduce next redex.

$\lambda_{need\|}$: Two-phase Steps

1) Reduce next redex.

$$E[\ (\lambda x.e_1)^{\vec{\ell}}\ e_2\ ] \to E[\ e_1\{x := e_2^{\ell_x}\}\ ]$$

ℓ_x fresh

$\lambda_{need\|}$: Two-phase Steps

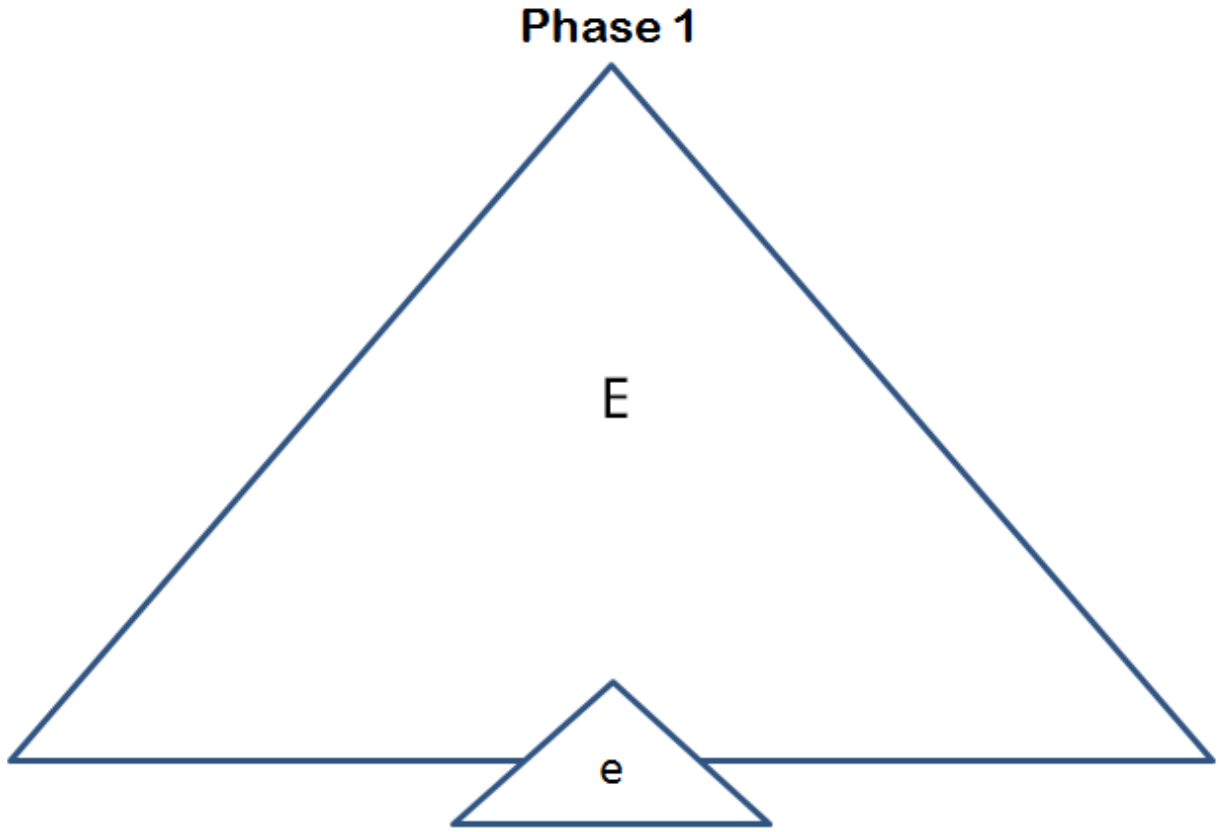
1) Reduce next redex.

$$E[\ (\lambda x.e_1)^{\vec{\ell}}\ e_2\ ] \to E[\ e_1\{x := e_2^{\ell_x}\}\ ]$$

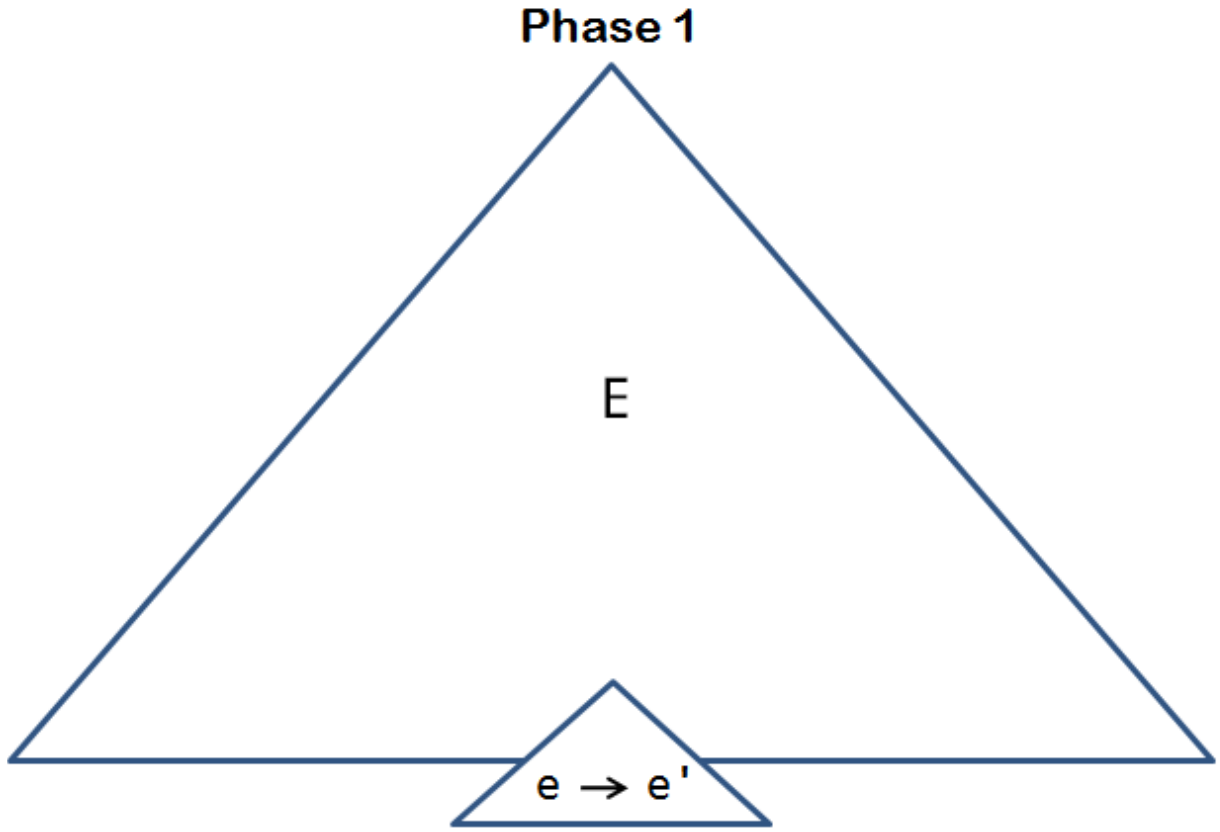
ℓ_x fresh

2) If redex is under a label, update all other identically labeled expressions to match.

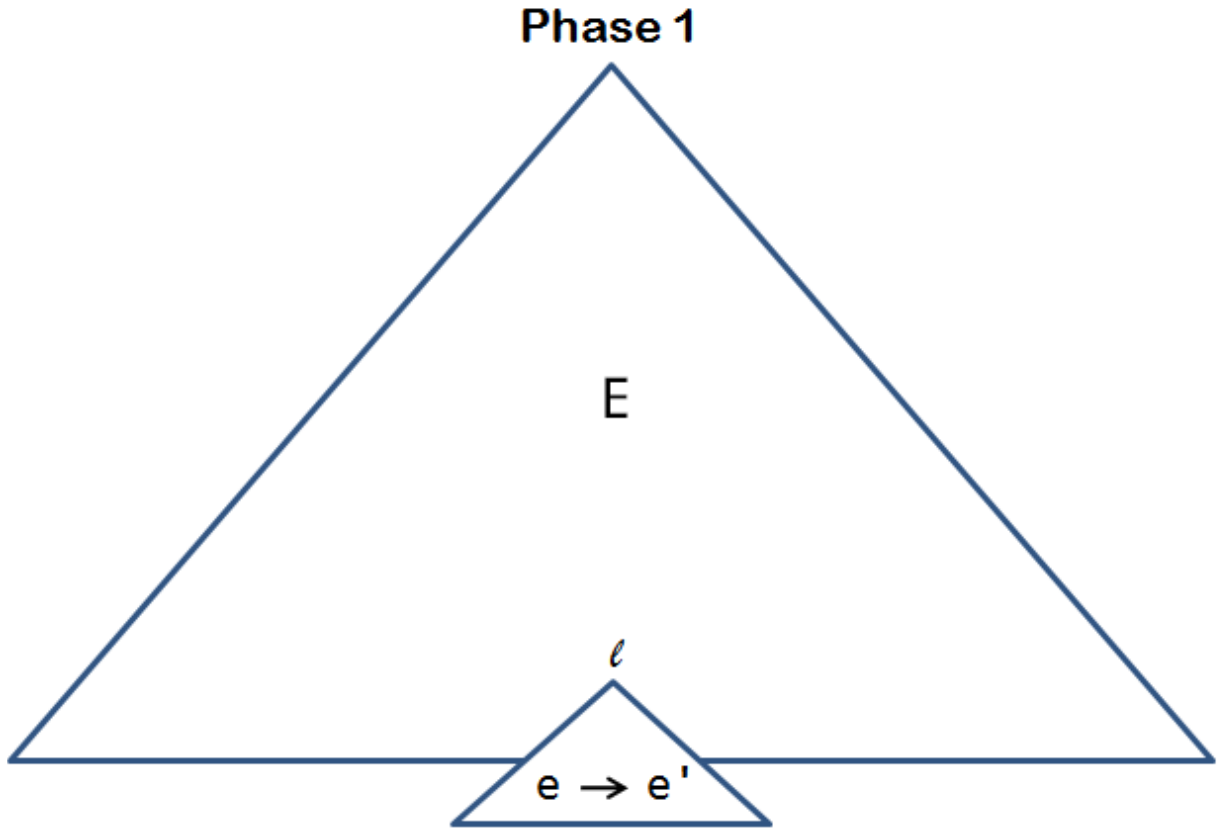
$\lambda_{need\|}$: Two-phase Steps, Pictorially



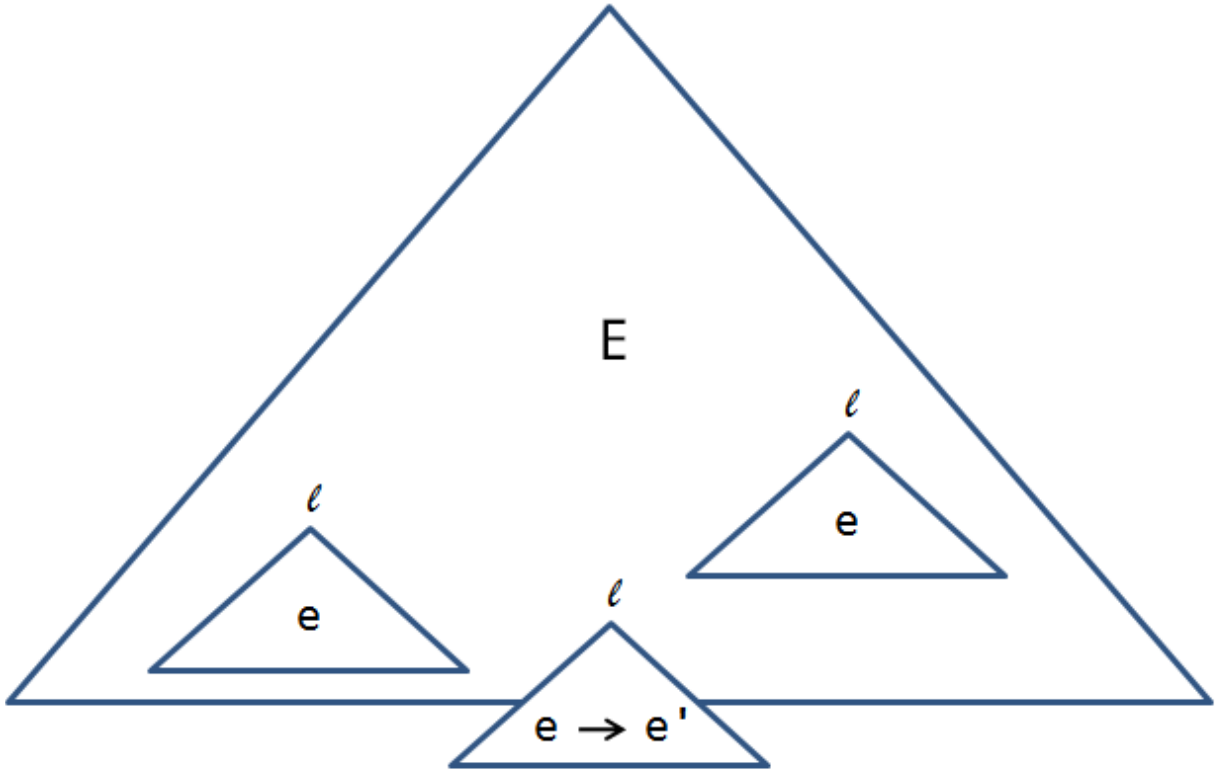
$\lambda_{need\|}$: Two-phase Steps, Pictorially



$\lambda_{need\|}$: Two-phase Steps, Pictorially

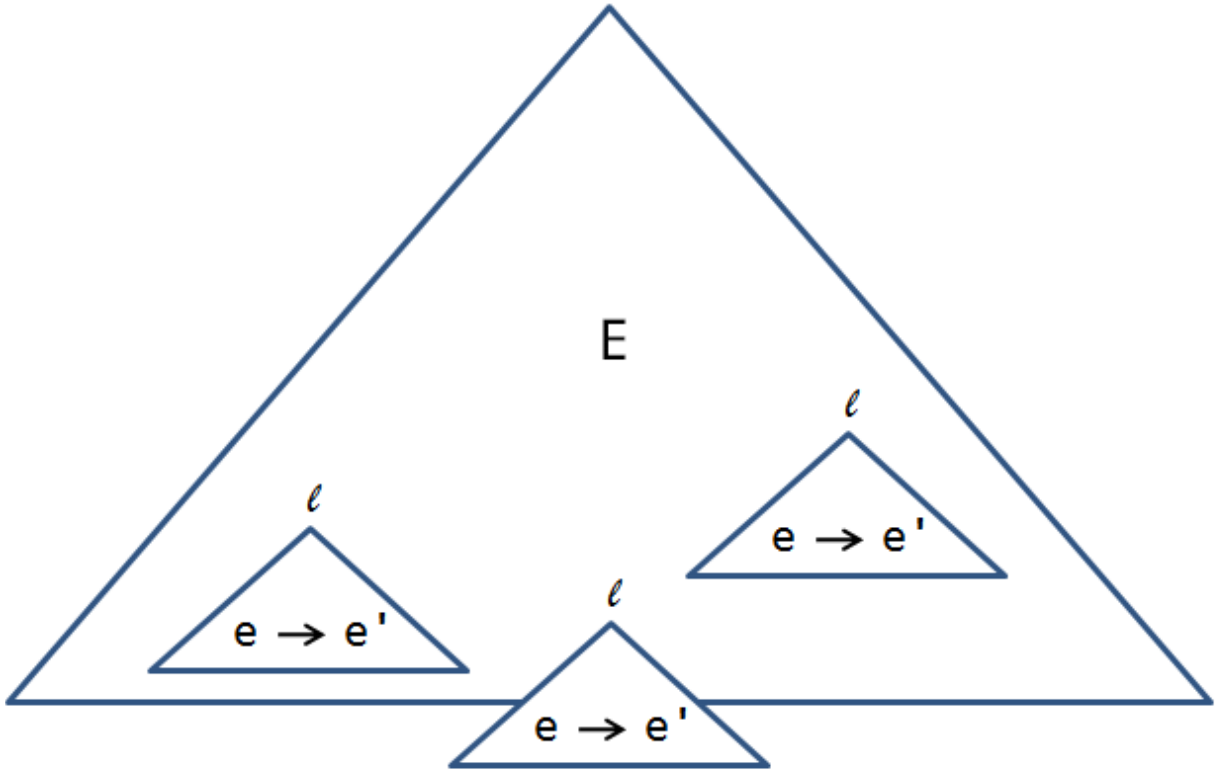


$\lambda_{need\|}$: Two-phase Steps, Pictorially



Phase 2

$\lambda_{need\|}$: Two-phase Steps, Pictorially

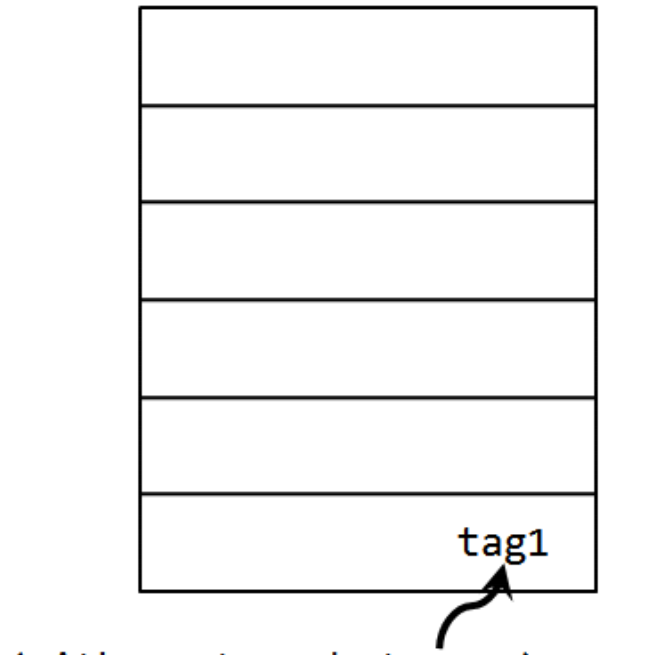


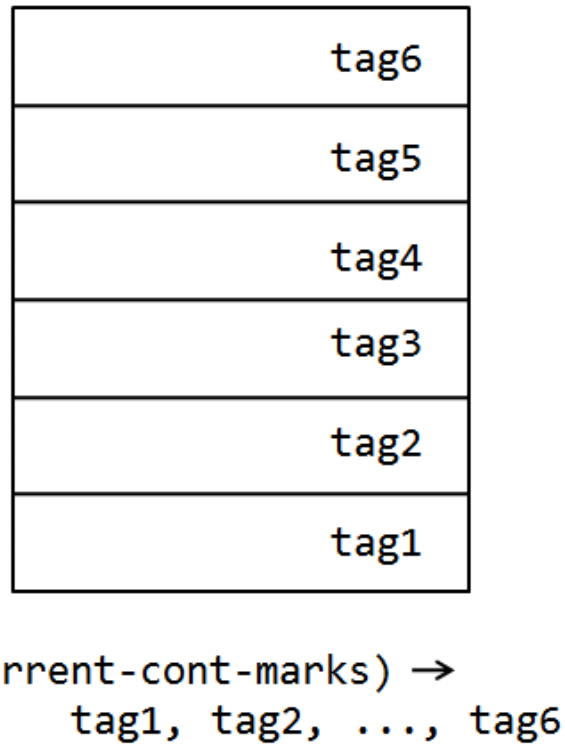
Phase 2

Implementation

Continuation Marks

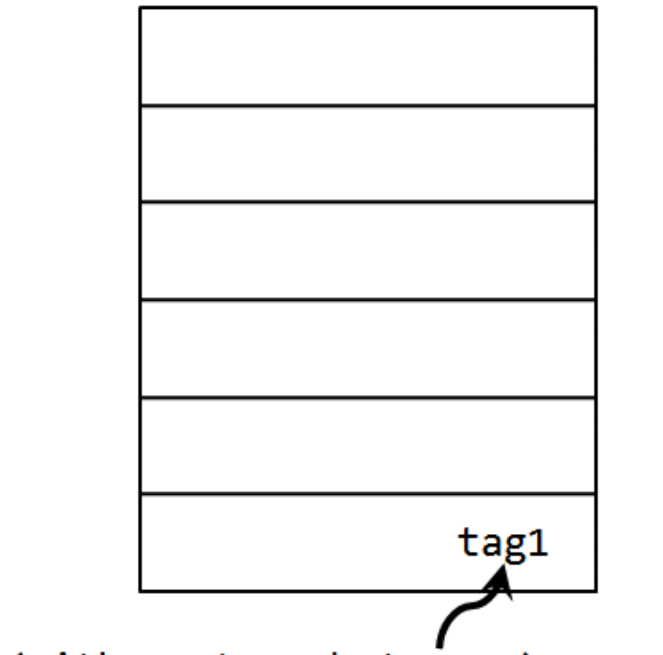
Mechanism for lightweight stack access. [Clements 2001]

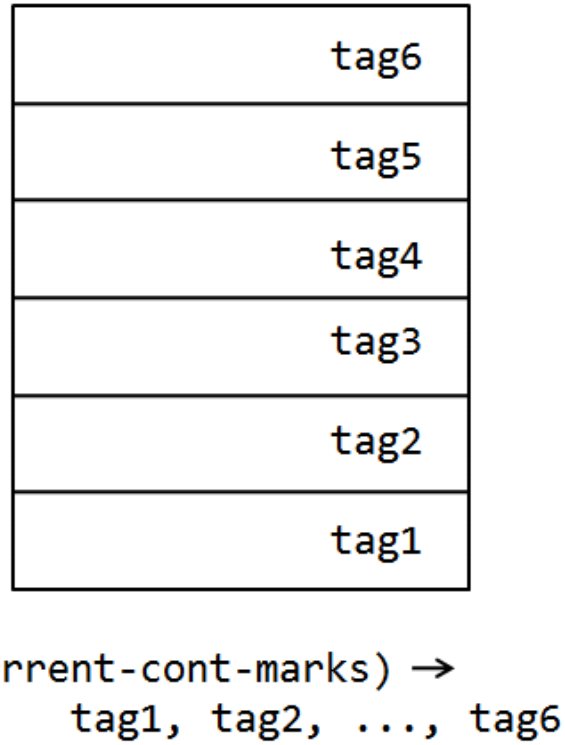




Continuation Marks

Mechanism for lightweight stack access. [Clements 2001]

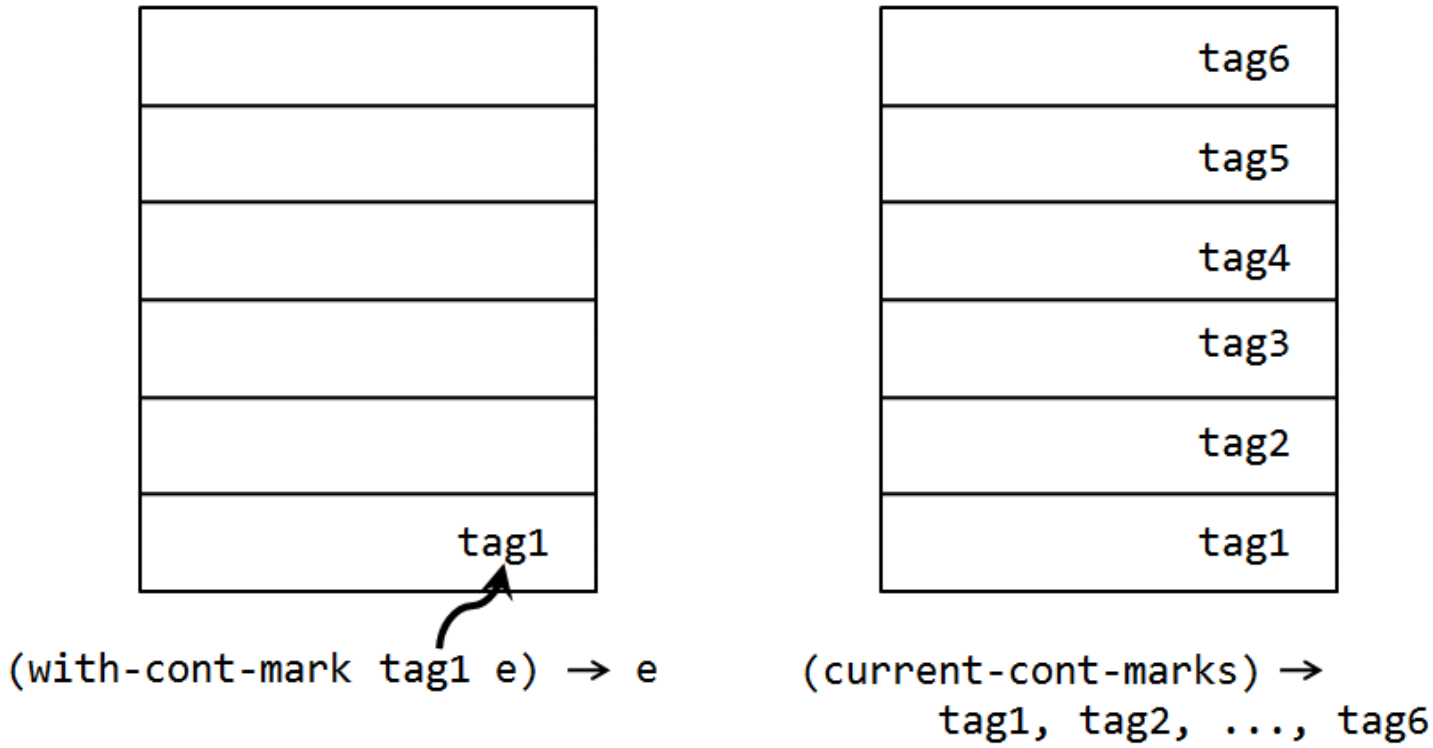




Continuation marks used in Racket implementation of:

Continuation Marks

Mechanism for lightweight stack access. [Clements 2001]

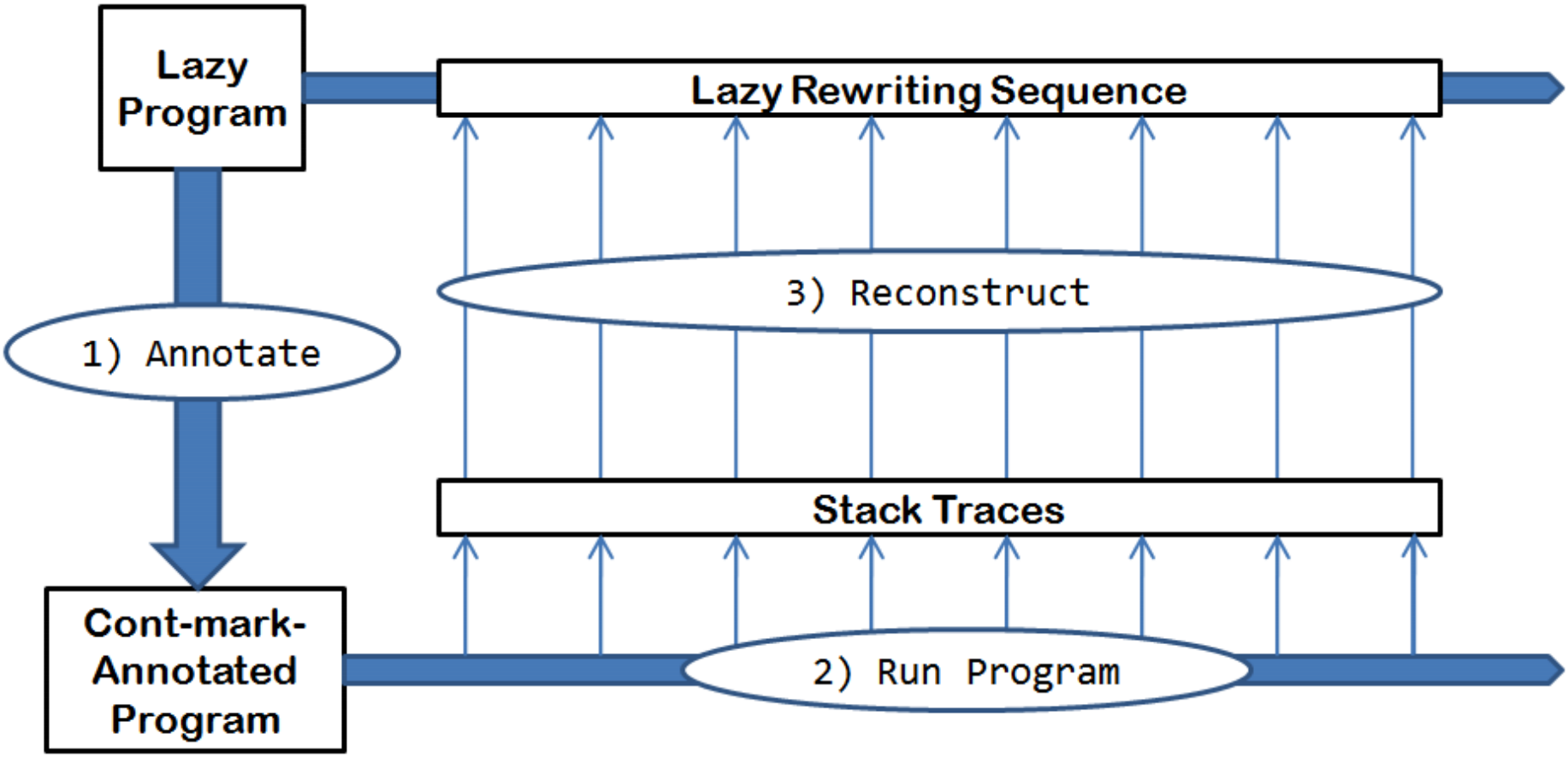


Continuation marks used in Racket implementation of:

stack tracer, stepper, debugger, profiler, exception handling,

dynamic binding, delimited continuations, web server

Stepper Architecture



Continuation marks are easily added to any language.

Continuation marks are easily added to any language.

["Implementing continuation marks in JavaScript" (Clements et al., 2008)]

Continuation marks are easily added to any language.

["Implementing continuation marks in JavaScript" (Clements et al., 2008)]

["Finding the needle: stack traces for GHC" (Allwood et al., 2009)]

$\lambda_{need\|}$: Correctness

Correspondence exists between $\lambda_{need\|}$ and:

$\lambda_{need\|}$: Correctness

Correspondence exists between $\lambda_{need\|}$ and:

- Low-level semantics (i.e., Launchbury)

$\lambda_{need\|}$: Correctness

Correspondence exists between $\lambda_{need\|}$ and:

- Low-level semantics (i.e., Launchbury)
- Reduction semantics (i.e., Ariola et al.)

To Do

To Do

- Advanced navigation features, breakpointing

To Do

- Advanced navigation features, breakpointing
- Additional inspection of program state

To Do

- Advanced navigation features, breakpointing
- Additional inspection of program state
- Scaling to large programs

Summary

- New semantics for lazy evaluation: $\lambda_{need\|}$
  - Easy to understand and suitable for use in a debugger.
  - Equivalent to existing lazy semantics.
- Algebraic stepper for Lazy Racket, based on $\lambda_{need\|}$
  - Proven correct.
  - Easily ported to any lazy language via continuation marks.

Summary

- New semantics for lazy evaluation: $\lambda_{need\|}$
  - Easy to understand and suitable for use in a debugger.
  - Equivalent to existing lazy semantics.
- Algebraic stepper for Lazy Racket, based on $\lambda_{need\|}$
  - Proven correct.
  - Easily ported to any lazy language via continuation marks.

Thanks!

`http://racket-lang.org/`