Efficient implementation of evaluation strategies via token-guided graph rewriting

Koko Muroya & Dan R. Ghica
(University of Birmingham)
TARGET

balancing space cost & time cost of program execution

abstract machines for lambda-calculus
Motivation: series of non-strict evaluation

- abstract machines of same end result
  - number of beta-reduction

- Krivine abstract machine: call by name
- Sestoft’s abstract machine: call by need
- Crégut’s lazy Krivine abstract machine
- Bologna optimal abstract machine
- Lévy’s optimal reduction
Motivation: series of non-strict evaluation

- abstract machines of same end result
  - number of beta-reduction
  - time cost

Accattoli et al.

- Krivine abstract machine: call by name
- Sestoft's abstract machine: call by need
- Crégut's lazy Krivine abstract machine
- Bologna optimal abstract machine
- Lévy's optimal reduction
Motivation: series of non-strict evaluation

- abstract machines of same end result
  - number of beta-reduction
  - time cost
  - space cost

- Krivine abstract machine
  - call by name

- Sestoft’s abstract machine
  - call by need

- Crégut’s lazy Krivine abstract machine

- Bologna optimal abstract machine

- Lévy’s optimal reduction

Accattoli et al.
Motivation: series of non-strict evaluation

- abstract machines of same end result
  - space cost

interaction
abstract
machine

[Kanos & Regnier ‘99]

- Gol-style token passing
- fixed graph

Krivine
abstract
machine

call by
name

ten term rewriting
Question

- abstract machines of same end result
  - space cost vs time cost… trade-off?

- Sestoft's abstract machine
- Krivine abstract machine
- Crégut's lazy Krivine abstract machine
- Bologna optimal abstract machine
- Lévy's optimal

interaction abstract machine

call by name
call by need
GOAL

unified framework that can balance space cost & time cost of program execution
GOAL

unified framework that can balance space cost & time cost of program execution

token-guided graph-rewriting abstract machine for lambda-calculus
Token-guided graph rewriting

Gol-style token passing, interleaved with graph rewriting
Token-guided graph rewriting

GoI-style token passing,
interleaved with graph rewriting
Token-guided graph rewriting

Gol-style token passing, interleaved with graph rewriting
Token-guided graph rewriting

Gol-style token passing,
interleaved with graph rewriting
Token-guided graph rewriting

Gol-style token passing, interleaved with graph rewriting
Token-guided graph rewriting

GoI-style token passing,
interleaved with graph rewriting
Token-guided graph rewriting

GoI-style token passing, interleaved with graph rewriting
Token-guided graph rewriting

Gol-style token passing,
interleaved with graph rewriting

redex detected
Token-guided graph rewriting

Gol-style token passing, interleaved with graph rewriting

(1) trigger rewriting
Token-guided graph rewriting

Gol-style token passing, interleaved with graph rewriting

redex detected
Token-guided graph rewriting

Gol-style token passing, interleaved with graph rewriting

(2) keep passing
Token-guided graph rewriting

Gol-style token passing, interleaved with graph rewriting

(1) trigger rewriting

(2) keep passing
Token-guided graph rewriting for lambda-calculus

**flexibility**, by choices of:

- graph rewriting system, with token passing
  - proof nets
- interleaving strategy
  - trigger rewriting vs. keep passing
- translation of lambda-terms
  - !(A ⊸ B), (!A) ⊸ B
Token-guided graph rewriting for lambda-calculus

**flexibility**, by choices of:

- graph rewriting system, with token passing
- interleaving strategy
- translation of lambda-terms

to…

- balance space cost & time cost
Non-strict evaluation: time cost improvement

- graph rewriting system with token passing
- interleaving strategy
- translation of lambda-terms

Token passing on fixed graph

Interaction abstract machine

Call by name

Call by need
Non-strict evaluation: time cost improvement

- token passing on fixed graph
- proof nets
- passes-only
- \( !(A \rightarrow B) \)

*call-by-name time cost*

- graph rewriting system with token passing
- interleaving strategy
- translation of lambda-terms
Non-strict evaluation: 
time cost improvement

token passing on fixed graph

- proof nets
- passes-only
- !\((A \Rightarrow B)\)

\textit{call-by-name time cost}

[- & Ghica, CSL ‘17]

- proof nets
- rewrites-first
- !\((A \Rightarrow B)\)

\textit{call-by-need time cost}

\textit{time cost analysis à la [Accattoli ’16]}

- graph rewriting system with token passing
- interleaving strategy
- translation of lambda-terms
Non-strict evaluation: space cost improvement?

- graph rewriting system with token passing
- interleaving strategy
- translation of lambda-terms

- token passing on fixed graph
- interaction abstract machine
- term rewriting
- Krivine abstract machine
- call by name
Non-strict evaluation: space cost improvement?

- graph rewriting system with token passing
- interleaving strategy
- translation of lambda-terms

- proof nets
- passes-only
- !(A \rightarrow B), (!A) \rightarrow B

\textit{call-by-name time cost}

\textit{token passing on fixed graph}

\textit{term rewriting}

\textit{Krivine abstract machine}

\textit{call by name}
Non-strict evaluation: space cost improvement?

- graph rewriting system with token passing
- interleaving strategy
- translation of lambda-terms

- token passing on fixed graph
- proof nets
- passes-only
- !(A → B), (!A) → B

*call-by-name time cost*

- graph rewriting
- proof nets
- rewrites-first
- (!A) → B

*call-by-name time cost*

- call by name
Strict evaluation

- graph rewriting system with token passing
- interleaving strategy
- translation of lambda-terms

[- & Ghica, WPTE ‘17]

- \( \lambda![@,@] \)-graphs
- rewrites-first
- \((A \rightsolve B)\)

*call-by-value* time cost

time cost analysis

à la [Accattoli ’16]
Token-guided graph rewriting for lambda-calculus

*flexibility*, by choices of:

- graph rewriting system, with token passing
- interleaving strategy
- translation of lambda-terms

...to...

- balance space cost & time cost

series of

- non-strict evaluation
- strict evaluation
Analyse token-guided graph rewriting via term rewriting + explicit redex searching [Sinot ‘05]

\[(\lambda x.x\ x)\ ((\lambda y.y)\ (\lambda z.z))\]
\[(\lambda x.x\ x)\ ((\lambda y.y)\ (\lambda z.z))\]
\[(\lambda x.x\ x)\ ((\lambda y.y)\ (\lambda z.z))\]
\[(\lambda x.x\ x)\ ((\lambda y.y)\ (\lambda z.z))\]
\[(\lambda x.x\ x)\ ((\lambda y.y)\ (\lambda z.z))\]
\[(\lambda x.x\ x)\ (y[y \leftarrow \lambda z.z])\]
\[(\lambda x.x\ x)\ ((\lambda z.z)[y \leftarrow \lambda z.z])\]
\[(\lambda x.x\ x)\ ((\lambda z.z)[y \leftarrow \lambda z.z])\]
\[(x\ x)[x \leftarrow \lambda z.z][y \leftarrow \lambda z.z]\]
\[(x\ x)[x \leftarrow \lambda z.z][y \leftarrow \lambda z.z]\]
\[(x\ x)[x \leftarrow \lambda z.z][y \leftarrow \lambda z.z]\]
\[(x\ (\lambda z.z))[x \leftarrow \lambda z.z][y \leftarrow \lambda z.z]\]
\[(x\ (\lambda z.z))[x \leftarrow \lambda z.z][y \leftarrow \lambda z.z]\]
\[((\lambda z'.z')\ (\lambda z.z))[x \leftarrow \lambda z.z][y \leftarrow \lambda z.z]\]
\[z'[z' \leftarrow \lambda z.z][x \leftarrow \lambda z.z][y \leftarrow \lambda z.z]\]
\[\lambda z.z[z' \leftarrow \lambda z.z][x \leftarrow \lambda z.z][y \leftarrow \lambda z.z]\]

[https://cwtsteven.github.io/GoI-Visualiser/CBV-with-CBV-embedding/index.html]
Token-guided graph rewriting for lambda-calculus

**flexibility**, by choices of:

- graph rewriting system, with token passing
- interleaving strategy
- translation of lambda-terms

series of

- non-strict evaluation
- strict evaluation

to...

- balance space cost & time cost

analysis via term rewriting