A Reversible Crumbling Abstract Machine for Plotkin's Call-By-Value

Nicolò Pizzo, Claudio Sacerdoti Coen
University of Bologna
July 28, 2026

Introduction

λ-terms

We can see λ-terms as:

	Programs	**Proofs**
β-step		
β^{-1}-step		

The computational operation on λ-terms is **β-reduction**

λ-terms

We can see λ-terms as:

	Programs	**Proofs**
β-step	Computing function application	Simplifying a term of the proof
β^{-1}-step		

The computational operation on λ-terms is **β-reduction**

λ-terms

We can see λ-terms as:

	Programs	**Proofs**
β-step	Computing function application	Simplifying a term of the proof
β^{-1}-step		

The computational operation on λ-terms is **β-reduction**

What does it mean to revert a β-step?

λ-terms

We can see λ-terms as:

	Programs	Proofs
β-step	Computing function application	Simplifying a term of the proof
β^{-1}-step	Restoring the previous state	Backtracking simplification

The computational operation on λ-terms is **β-reduction**

What does it mean to revert a β-step?

The goal

We want to equip Plotkin's **weak**, **closed** CbV calculus with **reversibility**

Plotkin's CbV calculus (λ_{Plot})

Terms	$u, t ::= v \mid ut$
Values	$v ::= \lambda x.t \mid x$
Right v-context	$R ::= \langle\cdot\rangle \mid tR \mid Rv$
Reduction Rule	$R\langle(\lambda x.t)v\rangle \to_\beta R\langle t\{x \leftarrow v\}\rangle$

We fix the right-to-left evaluation order

Landauer's embedding

We introduce some auxiliary structure, storing the necessary information to revert each step of the computation.

Landauer's embedding

We introduce some auxiliary structure, storing the necessary information to revert each step of the computation.

Example

$(x, y) \mapsto x + y$

Landauer's embedding

We introduce some auxiliary structure, storing the necessary information to revert each step of the computation.

Example

$(x, y) \mapsto x + y$

$(x, y) \mapsto (x + y, x)$

Landauer's embedding for λ_{Plot}

λ_{Plot} Reduction rule

$$R\langle(\lambda x.t)v\rangle \to_\beta R\langle t\{x \leftarrow v\}\rangle$$

Examples

$t = (\lambda x.xx)(\lambda y.y)$	$\to_\beta (\lambda y.y)(\lambda y.y)$	Remember $\lambda x.xx$
$t = (\lambda x.y)M$	$\to_\beta y$	Remember M

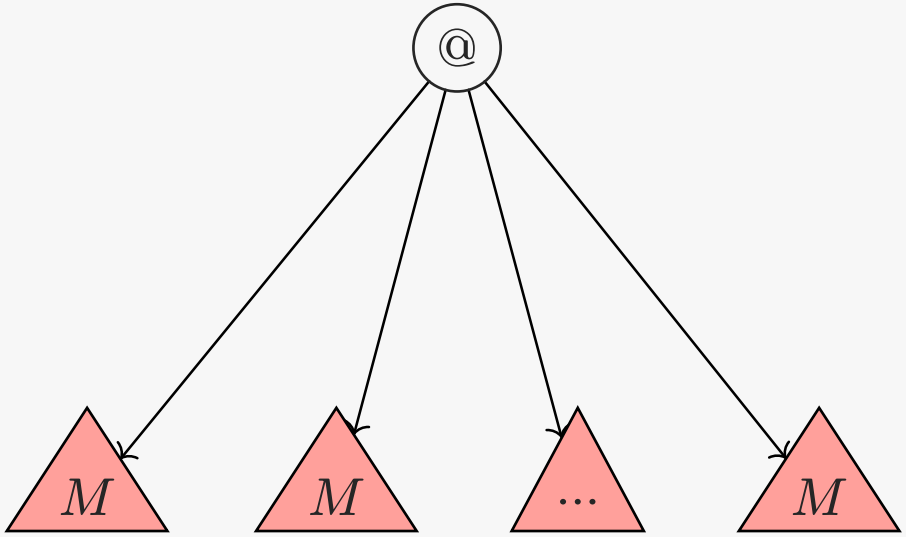
The cost is $O(|t|)$ **for each step**

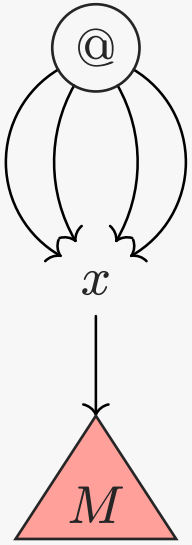
The size-explosion problem

$$(\lambda x.xx \cdots x)M \quad \rightarrow_\beta \quad MM \cdots M$$

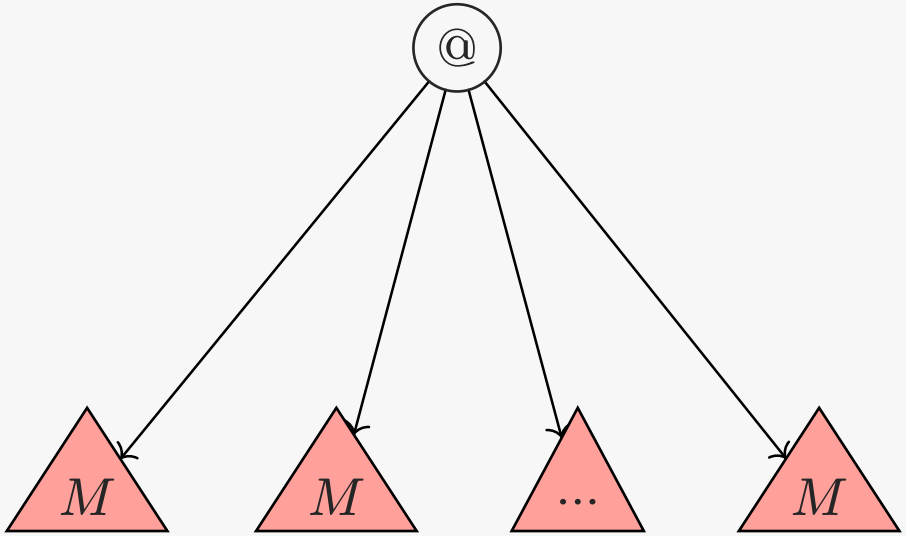
If M is similar in size to $(\lambda x.xx \cdots x)M$, the term size explodes!

Sharing



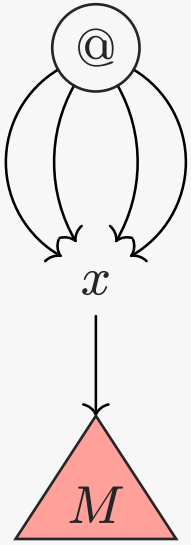


Sharing



Let-in

```
let x = M in x x ... x
```



Explicit Substitutions

$(xx \cdots x)[x \leftarrow M]$

Implementing λ-calculi

Abstract machines allow for efficent implementations of λ-calculi

-
-

Implementing λ-calculi

Abstract machines allow for efficent implementations of λ-calculi

- SECD Machine
  - “A Reversible SE(M)CD Machine”, W. Kluge 1999
-

Implementing λ-calculi

Abstract machines allow for efficent implementations of λ-calculi

- SECD Machine
  - "A Reversible SE(M)CD Machine", W. Kluge 1999
- **Crumbling Machines**

Crumbling

With **crumbling** we avoid nested applications

Example

$$(\lambda x.x(xx))(\lambda y.y)$$

Sharing: $x(xx)[x \leftarrow \lambda y.y]$

Crumbling: $(z_1 z_2)[z_1 \leftarrow \lambda x.(xz_3)[z_3 \leftarrow xx]][z_2 \leftarrow \lambda y.y]$

Crumbling Abstract Machines

Crumbling Abstract Machines encode both the *evaluation order* and the *evaluation context*!

Example

λ-term	$II \quad (v_1 v_2) \quad v_3$
Right-to-left	$(z_1 z_2)[z_1 \leftarrow \underline{I}][z_2 \leftarrow z_3 z_4][z_3 \leftarrow \underline{I}][z_4 \leftarrow z_5 z_6] \quad [z_5 \leftarrow z_7 z_8] \quad \left[z_7 \leftarrow \underline{v_1}\right]\left[z_8 \leftarrow \underline{v_2}\right]\left[z_9 \leftarrow \underline{v_3}\right]$
Left-to-right	$(z_1 z_2)[z_2 \leftarrow z_3 z_4]\left[z_4 \leftarrow \underline{v_3}\right][z_3 \leftarrow z_5 z_6]\left[z_6 \leftarrow \underline{v_2}\right]\left[z_5 \leftarrow \underline{v_1}\right] \quad [z_1 \leftarrow z_7 z_8] \quad [z_8 \leftarrow \underline{I}][z_7 \leftarrow \underline{I}]$

Crumbling Abstract Machines

Crumbling Abstract Machines encode both the *evaluation order* and the *evaluation context*!

Example

λ-term	$I\,I\;(v_1 v_2)\;v_3$
Right-to-left	$(z_1 z_2)[z_1 \leftarrow \underline{I}][z_2 \leftarrow z_3 z_4][z_3 \leftarrow \underline{I}][z_4 \leftarrow z_5 z_6]\;[z_5 \leftarrow z_7 z_8]\;\left[z_7 \leftarrow \underline{v_1}\right]\left[z_8 \leftarrow \underline{v_2}\right]\left[z_9 \leftarrow \underline{v_3}\right]$
Left-to-right	$(z_1 z_2)[z_2 \leftarrow z_3 z_4]\left[z_4 \leftarrow \underline{v_3}\right][z_3 \leftarrow z_5 z_6]\left[z_6 \leftarrow \underline{v_2}\right]\left[z_5 \leftarrow \underline{v_1}\right]\;[z_1 \leftarrow z_7 z_8]\;[z_8 \leftarrow \underline{I}][z_7 \leftarrow \underline{I}]$

Crumbling Abstract Machines

Crumbling Abstract Machines encode both the *evaluation order* and the *evaluation context*!

Example

λ-term $\quad II \; (v_1 v_2) \;\; v_3$

Right-to-left $\quad (z_1 z_2)[z_1 \leftarrow \underline{I}][z_2 \leftarrow z_3 z_4][z_3 \leftarrow \underline{I}][z_4 \leftarrow z_5 z_6] \;\; [z_5 \leftarrow z_7 z_8] \;\; \left[z_7 \leftarrow \underline{v_1}\right]\left[z_8 \leftarrow \underline{v_2}\right]\left[z_9 \leftarrow \underline{v_3}\right]$

Left-to-right $\quad (z_1 z_2)[z_2 \leftarrow z_3 z_4]\left[z_4 \leftarrow \underline{v_3}\right][z_3 \leftarrow z_5 z_6]\left[z_6 \leftarrow \underline{v_2}\right]\left[z_5 \leftarrow \underline{v_1}\right] \; [z_1 \leftarrow z_7 z_8] \; [z_8 \leftarrow \underline{I}][z_7 \leftarrow \underline{I}]$

The RCAM

Crumbled calculi

- “Crumbling Abstract Machines”, *B. Accattoli et al., 2019*
- “Positive Sharing and Abstract Machines”, *B. Accattoli et al., 2025*
- Finest Crumbling,

Crumbled calculi

- “Crumbling Abstract Machines”, *B. Accattoli et al., 2019*
- “Positive Sharing and Abstract Machines”, *B. Accattoli et al., 2025*
- Finest Crumbling, **N. Pizzo and C. Sacerdoti Coen, 2026**

Finest Crumbling and RCAM

Plotkin Terms

Terms	Values
$u, t ::= v \mid ut$	$v ::= \lambda x.t \mid x$

Finest Crumbling

Bites	Crumbled Values	Environments
$b ::= xy \mid v$	$v ::= \lambda x.[* \leftarrow y] \mid \lambda x.E$	$E ::= \varepsilon \mid E[z \leftarrow b]$

Finest Crumbling and RCAM

Plotkin Terms

Terms	Values
$u, t ::= v \mid ut$	$v ::= \lambda x.t \mid x$

$\dot{\downarrow}$ $\uparrow_{\downarrow}$

Finest Crumbling

Bites	Crumbled Values	Environments
$b ::= xy \mid v$	$v ::= \lambda x.[* \leftarrow y] \mid \lambda x.E$	$E ::= \varepsilon \mid E[z \leftarrow b]$

Finest Crumbling and RCAM

Plotkin Terms

Terms	Values
$u, t ::= v \mid ut$	$v ::= \lambda x.t \mid x$

$\dot{\downarrow}$ $\uparrow_{\downarrow}$

Finest Crumbling

Bites	Crumbled Values	Environments
$b ::= xy \mid v$	$v ::= \lambda x.[* \leftarrow y] \mid \lambda x.E$	$E ::= \varepsilon \mid E[z \leftarrow b]$

RCAM

State	History Stack
$S ::= (E, E_v, \mathcal{H})$	$\mathcal{H} ::= \varepsilon \mid \langle\rangle : \mathcal{H} \mid \langle x, y\rangle : \mathcal{H}$

Finest Crumbling and RCAM

Plotkin Terms

Terms	Values
$u, t ::= v \mid ut$	$v ::= \lambda x.t \mid x$

$\dot{\downarrow}$ $\uparrow_{\cdot\downarrow}$

Finest Crumbling

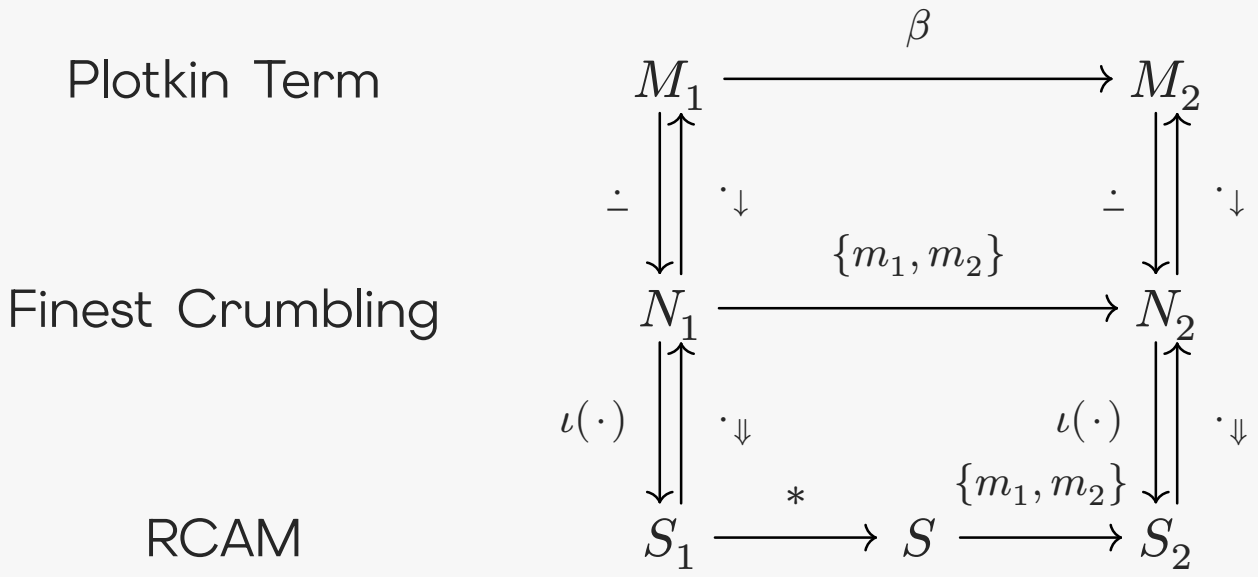
Bites	Crumbled Values	Environments
$b ::= xy \mid v$	$v ::= \lambda x.[* \leftarrow y] \mid \lambda x.E$	$E ::= \varepsilon \mid E[z \leftarrow b]$

$\iota(\cdot)\downarrow$ $\uparrow_{\cdot\Downarrow}$

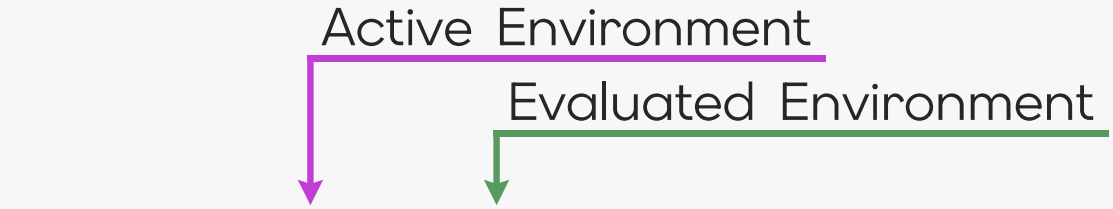
RCAM

State	History Stack
$S ::= (E, E_v, \mathcal{H})$	$\mathcal{H} ::= \varepsilon \mid \langle\rangle : \mathcal{H} \mid \langle x, y\rangle : \mathcal{H}$

Finest Crumbling and RCAM



RCAM Transitions



$$(E_1[z \leftarrow xy], E_v, \mathcal{H}) \;{}_{m_1^b}\!\!\leftrightsquigarrow_{m_1}\; (E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H})$$

$$\text{Where } E_v(x) = \lambda x.[* \leftarrow b]E, \quad [* \leftarrow b]E\{x \leftarrow y\} = [* \leftarrow b']E'$$

$$(E_1[z \leftarrow xy], E_v, \mathcal{H}) \;{}_{m_2^b}\!\!\leftrightsquigarrow_{m_2}\; (E_1, [z \leftarrow E_v(y')]\; E_v, \langle x, y\rangle : \mathcal{H})$$

$$\text{Where } E_v(x) = \lambda x.[* \leftarrow x'], \quad [* \leftarrow x']\{x' \leftarrow y\} = [* \leftarrow y']$$

$$(E_1[z \leftarrow v], E_v, \mathcal{H}) \;{}_{\text{sea}^b}\!\!\leftrightsquigarrow_{\text{sea}}\; (E_1, [z \leftarrow v]E_v, \langle\rangle : \mathcal{H})$$

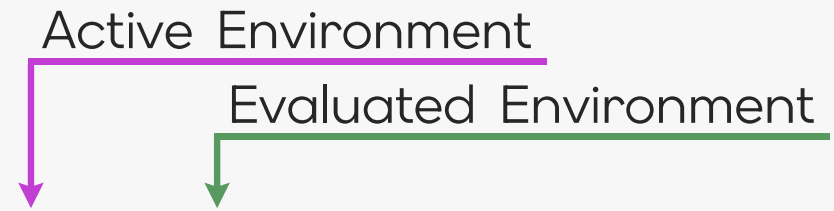
Reverting a step: the idea

$$(E_1[z \leftarrow xy], E_v, \mathcal{H}) \rightsquigarrow_{m_1} (E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H})$$

$$\text{Where } E_v(x) = \lambda x.[* \leftarrow b]E, \quad [* \leftarrow b]E\{x \leftarrow y\} = [* \leftarrow b']E'$$

$$(E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H}) \rightsquigarrow_{m_1^b} (E_1[z \leftarrow xy], E_v, \mathcal{H})$$

Reverting a step: the idea

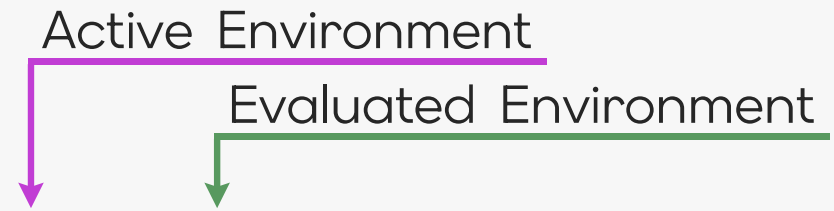


$$(E_1[z \leftarrow xy], E_v, \mathcal{H}) \rightsquigarrow_{m_1} (E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H})$$

$$\text{Where } E_v(x) = \lambda x.[* \leftarrow b]E, \quad [* \leftarrow b]E\{x \leftarrow y\} = [* \leftarrow b']E'$$

$$(E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H}) \rightsquigarrow_{m_1^b} (E_1[z \leftarrow xy], E_v, \mathcal{H})$$

Reverting a step: the idea



$$(E_1[z \leftarrow xy], E_v, \mathcal{H}) \rightsquigarrow_{m_1} (E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H})$$

$$\text{Where } E_v(x) = \lambda x.[* \leftarrow b]E, \quad [* \leftarrow b]E\{x \leftarrow y\} = [* \leftarrow b']E'$$

$$(E_1[z \leftarrow \not{b}']E', E_v, \langle x, y\rangle : \mathcal{H}) \rightsquigarrow_{m_1^b} (E_1[z \leftarrow xy], E_v, \mathcal{H})$$

xy

Reverting a step: the idea

Active Environment

Evaluated Environment

$$(E_1[z \leftarrow xy], E_v, \mathcal{H}) \rightsquigarrow_{m_1} (E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H})$$

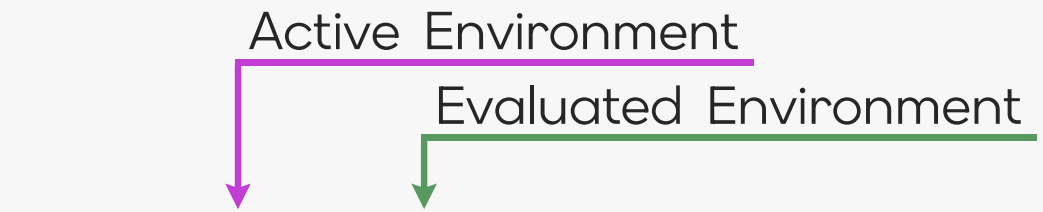
Where $E_v(x) = \lambda x.[* \leftarrow b]E, \quad [* \leftarrow b]E\{x \leftarrow y\} = [* \leftarrow b']E'$

$$(E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H}) \rightsquigarrow_{m_1^b} (E_1[z \leftarrow xy], E_v, \mathcal{H})$$

xy

$E_v(x)$ already holds this information

Reverting a step: the idea



$$(E_1[z \leftarrow xy], E_v, \mathcal{H}) \rightsquigarrow_{m_1} (E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H})$$

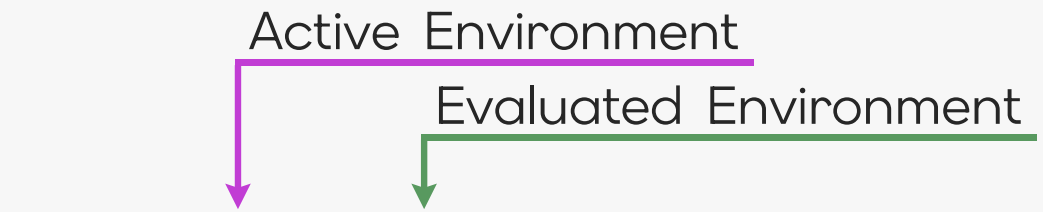
Where $E_v(x) = \lambda x.[* \leftarrow b]E, \quad [* \leftarrow b]E\{x \leftarrow y\} = [* \leftarrow b']E'$

$$(E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H}) \rightsquigarrow_{m_1^b} (E_1[z \leftarrow xy], E_v, \mathcal{H})$$

xy

$E_v(x)$ already holds this information

Reverting a step: the idea



$$(E_1[z \leftarrow xy], E_v, \mathcal{H}) \rightsquigarrow_{m_1} (E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H})$$

Where $E_v(x) = \lambda x.[* \leftarrow b]E, \quad [* \leftarrow b]E\{x \leftarrow y\} = [* \leftarrow b']E'$

$$(E_1[z \leftarrow b']E', E_v, \langle x, y\rangle : \mathcal{H}) \rightsquigarrow_{m_1^b} (E_1[z \leftarrow xy], E_v, \mathcal{H})$$

xy

$E_v(x)$ already holds this information

We just need to save the pair x, y with cost $O(1)$

Main Results

Theorem (Implementation)

The RCAM implements $\rightarrow_\beta$-evaluation.

Theorem (The RCAM is bi-linear)

For any closed term t, and any RCAM execution $\rho : \iota(\underline{t}) \rightsquigarrow_f^* S$, the cost of implementing ρ on a RAM is $O((|\rho|_p + 1) \cdot |t|)$.

Theorem (Space overhead)

For any closed term t, and any RCAM execution $\rho : \iota(\underline{t}) \rightsquigarrow_f^* S$, the maximum size of the history during the execution ρ on a RAM is $O((|\rho|_p + 1) \cdot |t|)$

Main Results

Theorem (Backward Determinism)

The transitions $\rightsquigarrow_b$ are deterministic.

Theorem (Loop)

Let S' b any state of the RCAM, such that $S \rightsquigarrow_f S'$. Then the evaluation step can always be reverted, i.e. $S' \rightsquigarrow_b S$ and vice-versa.

Theorem (Well-foundedness)

There is no infinite reverse computation, i.e. we do not have sequences $(S_i)_{i \in \mathbb{N}}$ such that $S_{i+1} \rightsquigarrow_f S_i$ for all $i \in N$.

Conclusions

Summary

- Explored Finest Crumbling
- Efficient Reversible Crumbling Abstract Machine
- Compact Landauer's embedding

Future Works

- Open/Strong Call-by-Value
  - Interesting applications to ITPs
- Scalability to Call-by-Name and Call-by-Need

Thank you!