The Time and Space of Interaction

Beniamino Accattoli[1] Ugo Dal Lago[2] Gabriele Vanoni[2]

[1]INRIA & LIX, École Polytechnique
[2]Università di Bologna & INRIA

CHoCoLa, 22 April, 2021

Work appeared or to appear in PPDP 2020, POPL 2021, LICS 2021

Complexity and the λ-calculus

The Simplest Setting: Closed Call-by-Name λ-calculus

We set our analysis in the simplest higher-order Turing-complete framework: pure untyped λ-calculus with weak head reduction. Abstractions are the *only* values.

The Simplest Setting: Closed Call-by-Name λ-calculus

We set our analysis in the simplest higher-order Turing-complete framework: pure untyped λ-calculus with weak head reduction. Abstractions are the *only* values.

Closed CbN λ-calculus

Terms	t, u	$::=$	$x \mid \lambda x.t \mid tu$
Context	C	$::=$	$\langle\cdot\rangle \mid Ct$
Reduction	$C\langle(\lambda x.t)u\rangle$	$\to$	$C\langle t\{x \leftarrow u\}\rangle$

The Simplest Setting: Closed Call-by-Name λ-calculus

We set our analysis in the simplest higher-order Turing-complete framework: pure untyped λ-calculus with weak head reduction. Abstractions are the *only* values.

Closed CbN λ-calculus

Terms	t, u	$::=$	$x \mid \lambda x.t \mid tu$
Context	C	$::=$	$\langle\cdot\rangle \mid Ct$
Reduction	$C\langle(\lambda x.t)u\rangle$	$\to$	$C\langle t\{x \leftarrow u\}\rangle$

Of course, λ-terms may **diverge**, *e.g.* $\Omega = (\lambda x.xx)(\lambda x.xx) \to \Omega$.

The Invariance Thesis

The Invariance Thesis

Reasonable machines simulate each other with polynomially bounded overhead in time and constant factor overhead in space.

Slot and van Emde Boas

The Invariance Thesis

Reasonable machines simulate each other with polynomially bounded overhead in time and constant factor overhead in space.

Slot and van Emde Boas

Turing machines and RAMs are reasonable with the usual cost models.

The Invariance Thesis

Reasonable machines simulate each other with polynomially bounded overhead in time and constant factor overhead in space.

Slot and van Emde Boas

Turing machines and RAMs are reasonable with the usual cost models.

Can we implement the λ-calculus in a **reasonable** way?

Polynomial Overhead Time Simulations

Polynomial Overhead Time Simulations

We consider the number of β-steps as the time cost model for the λ-calculus.

Polynomial Overhead Time Simulations

We consider the number of β-steps as the time cost model for the λ-calculus.

- The direction TM $\rightarrow$ λ-calculus is **easy**. [Dal Lago and Accattoli 2017]

Polynomial Overhead Time Simulations

We consider the number of β-steps as the time cost model for the λ-calculus.

- The direction TM $\rightarrow$ λ-calculus is **easy**. [Dal Lago and Accattoli 2017]
- The direction λ-calculus $\rightarrow$ TM is more delicate, because of **size explosion**.

Polynomial Overhead Time Simulations

We consider the number of β-steps as the time cost model for the λ-calculus.

- The direction TM $\rightarrow$ λ-calculus is **easy**. [Dal Lago and Accattoli 2017]
- The direction λ-calculus $\rightarrow$ TM is more delicate, because of **size explosion**.

Size Explosion

There exists a family of λ-terms t_n such that:

Polynomial Overhead Time Simulations

We consider the number of β-steps as the time cost model for the λ-calculus.

- The direction TM $\rightarrow$ λ-calculus is **easy**. [Dal Lago and Accattoli 2017]
- The direction λ-calculus $\rightarrow$ TM is more delicate, because of **size explosion**.

Size Explosion

There exists a family of λ-terms t_n such that:

$$|t_n| = \Theta(n)$$

Polynomial Overhead Time Simulations

We consider the number of β-steps as the time cost model for the λ-calculus.

- The direction TM $\rightarrow$ λ-calculus is **easy**. [Dal Lago and Accattoli 2017]
- The direction λ-calculus $\rightarrow$ TM is more delicate, because of **size explosion**.

Size Explosion

There exists a family of λ-terms t_n such that:

$$|t_n| = \Theta(n) \qquad t_n \rightarrow_\beta^n \mathsf{whnf}(t_n)$$

Polynomial Overhead Time Simulations

We consider the number of β-steps as the time cost model for the λ-calculus.

- The direction TM $\rightarrow$ λ-calculus is **easy**. [Dal Lago and Accattoli 2017]
- The direction λ-calculus $\rightarrow$ TM is more delicate, because of **size explosion**.

Size Explosion

There exists a family of λ-terms t_n such that:

$$|t_n| = \Theta(n) \qquad t_n \rightarrow_\beta^n \mathsf{whnf}(t_n) \qquad |\mathsf{whnf}(t_n)| = \Theta(2^n)$$

Polynomial Overhead Time Simulations

We consider the number of β-steps as the time cost model for the λ-calculus.

- The direction TM $\to$ λ-calculus is **easy**. [Dal Lago and Accattoli 2017]
- The direction λ-calculus $\to$ TM is more delicate, because of **size explosion**.

Size Explosion

There exists a family of λ-terms t_n such that:

$$|t_n| = \Theta(n) \qquad t_n \to_\beta^n \mathsf{whnf}(t_n) \qquad |\mathsf{whnf}(t_n)| = \Theta(2^n)$$

A nice proof can be carried out through the use of **abstract machines** (e.g. Krivine's).

Constant Overhead Space Simulations

Constant Overhead Space Simulations

It is not clear what the **space cost model** for the λ-calculus should be.

Constant Overhead Space Simulations

It is not clear what the **space cost model** for the λ-calculus should be.

Considering the maximum size of a term in a computation, one would have SPACE $>$ TIME, because of **size explosion**.

Constant Overhead Space Simulations

It is not clear what the **space cost model** for the λ-calculus should be.

Considering the maximum size of a term in a computation, one would have SPACE $>$ TIME, because of **size explosion**.

Abstract machines **share** redex arguments, thus consuming SPACE $\simeq$ TIME (unless some garbage collection mechanism is used).

An Unusual Machine: the IAM

We want to be **parsimonious** in **space**.

An Unusual Machine: the IAM

We want to be **parsimonious** in **space**.

Ideas

1. No tracing of β-redexes

An Unusual Machine: the IAM

We want to be **parsimonious** in **space**.

Ideas

1. No tracing of β-redexes
   $\Rightarrow$ space disentangled from time.

An Unusual Machine: the IAM

We want to be **parsimonious** in **space**.

Ideas

1. No tracing of β-redexes
   $\Rightarrow$ space disentangled from time.
2. **Backtracking** to retrieve β-redexes

An Unusual Machine: the IAM

We want to be **parsimonious** in **space**.

Ideas

1. No tracing of β-redexes
   $\Rightarrow$ space disentangled from time.
2. **Backtracking** to retrieve β-redexes
   $\Rightarrow$ wasting time to gain space.

An Unusual Machine: the IAM

We want to be **parsimonious** in **space**.

Ideas

1. No tracing of β-redexes
   $\Rightarrow$ space disentangled from time.
2. **Backtracking** to retrieve β-redexes
   $\Rightarrow$ wasting time to gain space.
3. Computation is **local**

An Unusual Machine: the IAM

We want to be **parsimonious** in **space**.

Ideas

1. No tracing of β-redexes
   $\Rightarrow$ space disentangled from time.
2. **Backtracking** to retrieve β-redexes
   $\Rightarrow$ wasting time to gain space.
3. Computation is **local**
   $\Rightarrow$ a **token** that travels inside the term.

An Unusual Machine: the IAM

We want to be **parsimonious** in **space**.

Ideas

1. No tracing of β-redexes
   $\Rightarrow$ space disentangled from time.
2. **Backtracking** to retrieve β-redexes
   $\Rightarrow$ wasting time to gain space.
3. Computation is **local**
   $\Rightarrow$ a **token** that travels inside the term.
4. The code never changes.

The Interaction Abstract and Space Complexity

The Interaction Abstract and Space Complexity

- **Mackie**: small runtime system for PCF.

The Interaction Abstract and Space Complexity

- **Mackie**: small runtime system for PCF.
- **Ghica**: compilation of higher-order functions to digital circuits.

The Interaction Abstract and Space Complexity

- **Mackie**: small runtime system for PCF.
- **Ghica**: compilation of higher-order functions to digital circuits.
- **Dal Lago** and **Schöpp**: functional programming in Logspace.

The Interaction Abstract and Space Complexity

- **Mackie**: small runtime system for PCF.
- **Ghica**: compilation of higher-order functions to digital circuits.
- **Dal Lago** and **Schöpp**: functional programming in Logspace.
- **Aubert et al.**: logic programming in Logspace.

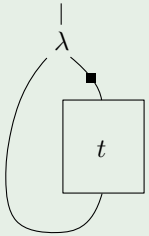
Positions

The position of the token inside the term t is represented via a pair (u, C) such that $C\langle u\rangle = t$.

Positions

The position of the token inside the term t is represented via a pair (u, C) such that $C\langle u\rangle = t$.

Example



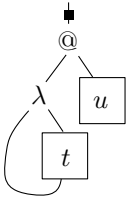
The position of the token ■ in the term $\lambda x.t$ is represented by the position $(t, \lambda x.\langle\cdot\rangle)$.

Traversing β-redexes

No information is saved, traversing a β-redex $(\lambda x.t)u$. The token remains untouched.

Traversing β-redexes

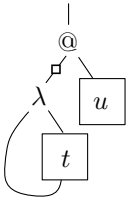
No information is saved, traversing a β-redex $(\lambda x.t)u$. The token remains untouched.



The token ■ is at the root of the redex.

Traversing β-redexes

No information is saved, traversing a β-redex $(\lambda x.t)u$. The token remains untouched.

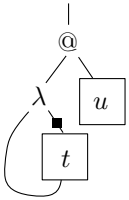


The token ■ moves to the left-hand side of the application, changing color.

$$ru \mid C \mid \blacksquare \quad \rightarrow_{\bullet 1} \quad r \mid C\langle\langle\cdot\rangle u\rangle \mid \square$$

Traversing β-redexes

No information is saved, traversing a β-redex $(\lambda x.t)u$. The token remains untouched.



The token $\square$ moves to the body of the abstraction, changing color again.

$$\lambda x.t \;\Big|\; C \;\Big|\; \square \quad \rightarrow_{\bullet 2} \quad t \;\Big|\; C\langle \lambda x.\langle\cdot\rangle\rangle \;\Big|\; \blacksquare$$

Quering Variables and their Arguments

Computation in the λ-calculus is done by substituting variables for arguments.

Quering Variables and their Arguments

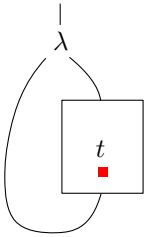
Computation in the λ-calculus is done by substituting variables for arguments.

Our machine first looks for variables, in red mode, going deep inside the term.

Quering Variables and their Arguments

Computation in the λ-calculus is done by substituting variables for arguments.

Our machine first looks for variables, in red mode, going deep inside the term.

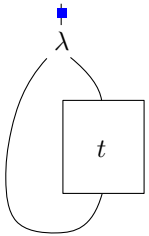


The token ■ is pointing a variable.

Quering Variables and their Arguments

Computation in the λ-calculus is done by substituting variables for arguments.

Our machine first looks for variables, in red mode, going deep inside the term.



The token ■ moves locally to the binder, changing its mode to blue, *i.e.* the machine is now looking for the argument of the variable.

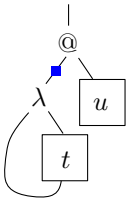
$$x \mid C\langle \lambda x.D \rangle \mid \blacksquare \quad \rightarrow_{\mathsf{var}} \quad \lambda x.D\langle x \rangle \mid C \mid \blacksquare$$

Evaluating Arguments

When an argument is found, it has to be evaluated.

Evaluating Arguments

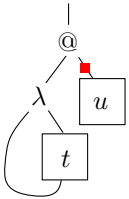
When an argument is found, it has to be evaluated.



The token ■, *i.e.* looking for an argument is on the left-hand side of an application.

Evaluating Arguments

When an argument is found, it has to be evaluated.



The token moves to the argument, changing its mode to red, *i.e.* looking for the head variable.

$$t \mid C\langle\langle\cdot\rangle u\rangle \mid \blacksquare \quad \rightarrow_{\mathsf{arg}} \quad u \mid C\langle t\langle\cdot\rangle\rangle \mid \blacksquare$$

Hiding Some Details

Hiding Some Details

There is also a **backtracking** mechanism (not explained here).

Hiding Some Details

There is also a **backtracking** mechanism (not explained here).

It uses position **enriched** with history/log.

Hiding Some Details

There is also a **backtracking** mechanism (not explained here).

It uses position **enriched** with history/log.

The token is actually made of two data structures, tape and log.

Hiding Some Details

There is also a **backtracking** mechanism (not explained here).

It uses position **enriched** with history/log.

The token is actually made of two data structures, tape and log.

Implementation Theorem

The λ-term t has weak head normal form iff the IAM terminates on $(\underline{t}, \langle\cdot\rangle, \epsilon, \epsilon)$.

Types and Complexity

Non-Idempotent Intersection Types

Sequence Types for the CbN λ-Calculus

$$\begin{array}{lrcll} \text{Linear Types} & A & ::= & \star \mid S \to A & \\ \text{Sequence Types} & S & ::= & [A_1 \cdots A_n] & n \geq 0 \end{array}$$

Non-Idempotent Intersection Types

Sequence Types for the CbN λ-Calculus

$$\begin{array}{lrcll} \text{Linear Types} & A & ::= & \star \mid S \to A & \\ \text{Sequence Types} & S & ::= & [A_1 \cdots A_n] & n \geq 0 \end{array}$$

Correctness and Completeness

$\vdash t : \star$ if and only if $t \Downarrow$.

A Bridge Between Syntax and Semantics

Sequence types are nothing but another way of formulating operational/denotational **semantics** of the λ-calculus in a Linear Logic fashion. There are many other examples.

A Bridge Between Syntax and Semantics

Sequence types are nothing but another way of formulating operational/denotational **semantics** of the λ-calculus in a Linear Logic fashion. There are many other examples.

Operational Semantics

A Bridge Between Syntax and Semantics

Sequence types are nothing but another way of formulating operational/denotational **semantics** of the λ-calculus in a Linear Logic fashion. There are many other examples.

Operational Semantics

- Linear Substitution Calculus (*i.e. proof net reduction*)

A Bridge Between Syntax and Semantics

Sequence types are nothing but another way of formulating operational/denotational **semantics** of the λ-calculus in a Linear Logic fashion. There are many other examples.

Operational Semantics

- Linear Substitution Calculus (*i.e. proof net reduction*)
- Krivine's Abstract Machine

A Bridge Between Syntax and Semantics

Sequence types are nothing but another way of formulating operational/denotational **semantics** of the λ-calculus in a Linear Logic fashion. There are many other examples.

Operational Semantics

- Linear Substitution Calculus (*i.e. proof net reduction*)
- Krivine's Abstract Machine
- Interaction Abstract Machine

A Bridge Between Syntax and Semantics

Sequence types are nothing but another way of formulating operational/denotational **semantics** of the λ-calculus in a Linear Logic fashion. There are many other examples.

Operational Semantics

- Linear Substitution Calculus (*i.e. proof net reduction*)
- Krivine's Abstract Machine
- Interaction Abstract Machine
- Polyadic approximations

A Bridge Between Syntax and Semantics

Sequence types are nothing but another way of formulating operational/denotational **semantics** of the λ-calculus in a Linear Logic fashion. There are many other examples.

Operational Semantics

- Linear Substitution Calculus (*i.e. proof net reduction*)
- Krivine's Abstract Machine
- Interaction Abstract Machine
- Polyadic approximations

Denotational Semantics

A Bridge Between Syntax and Semantics

Sequence types are nothing but another way of formulating operational/denotational **semantics** of the λ-calculus in a Linear Logic fashion. There are many other examples.

Operational Semantics

- Linear Substitution Calculus (*i.e. proof net reduction*)
- Krivine's Abstract Machine
- Interaction Abstract Machine
- Polyadic approximations

Denotational Semantics

- **Relational model**

A Bridge Between Syntax and Semantics

Sequence types are nothing but another way of formulating operational/denotational **semantics** of the λ-calculus in a Linear Logic fashion. There are many other examples.

Operational Semantics

- Linear Substitution Calculus (*i.e. proof net reduction*)
- Krivine's Abstract Machine
- Interaction Abstract Machine
- Polyadic approximations

Denotational Semantics

- **Relational model**
- Game semantics

Typing Rules

$$\frac{}{x : [A] \vdash x : A}\ \textsf{T-Var} \qquad \frac{\Gamma, x : S \vdash t : A}{\Gamma \vdash \lambda x.t : S \to A}\ \textsf{T-}\lambda \qquad \frac{}{\Gamma \vdash \lambda x.t : \star}\ \textsf{T-}\lambda_\star$$

$$\frac{\Gamma \vdash t : [A'_1, \ldots, A'_n] \to A \quad [\Delta_i \vdash u : A'_i]_{i \in [1,\ldots,n]}}{\Gamma \uplus \sum_{i \in [1,\ldots,n]} \Delta_i \vdash tu : A}\ \textsf{T-@}$$

A Machine Built on Type Derivations

A Machine Built on Type Derivations

Given a type derivation ending in $\vdash t : \star$, we can consider a **token machine** that starts from the root occurrence of $\star$.

A Machine Built on Type Derivations

Given a type derivation ending in $\vdash t : \star$, we can consider a **token machine** that starts from the root occurrence of $\star$.

One can move in the *natural* way inside the derivation, following that occurrence of $\star$.

A Machine Built on Type Derivations

Given a type derivation ending in $\vdash t : \star$, we can consider a **token machine** that starts from the root occurrence of $\star$.

One can move in the *natural* way inside the derivation, following that occurrence of $\star$.

The obtained machine is strongly bisimilar to the **Interaction Abstract Machine**.

A Machine Built on Type Derivations

Given a type derivation ending in $\vdash t : \star$, we can consider a **token machine** that starts from the root occurrence of $\star$.

One can move in the *natural* way inside the derivation, following that occurrence of $\star$.

The obtained machine is strongly bisimilar to the **Interaction Abstract Machine**.

The **multiplicative** part is handled by **types**.

A Machine Built on Type Derivations

Given a type derivation ending in $\vdash t : \star$, we can consider a **token machine** that starts from the root occurrence of $\star$.

One can move in the *natural* way inside the derivation, following that occurrence of $\star$.

The obtained machine is strongly bisimilar to the **Interaction Abstract Machine**.

The **multiplicative** part is handled by **types**.

The **exponential** part is **no more needed**: in type derivations each subterm has already been copied/discarded. This way, each copy is used only once, in a **linear** way.

The Machine in Action

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [[\star] \to \star] \vdash x : [\star_{\downarrow 16}] \to \star_{\uparrow 6}} \quad \dfrac{}{y : [\star] \vdash y : \star_{\uparrow 17}}}{y : [\star], x : [[\star] \to \star] \vdash xy : \star_{\uparrow 5}}}{y : [\star] \vdash \lambda x.xy : [[\star_{\uparrow 15}] \to \star_{\downarrow 7}] \to \star_{\uparrow 4}}}{\vdash \lambda y.\lambda x.xy : [\star_{\downarrow 18}] \to [[\star_{\uparrow 14}] \to \star_{\downarrow 8}] \to \star_{\uparrow 3}} \quad \dfrac{}{\vdash \mathsf{I} : \star_{\uparrow 19}} \quad \dfrac{\dfrac{}{z : [\star] \vdash z : \star_{\uparrow 11}}}{\vdash \lambda z.z : [\star_{\downarrow 12}] \to \star_{\uparrow 10}}}{\vdash (\lambda y.\lambda x.xy)\mathsf{I} : [[\star_{\uparrow 13}] \to \star_{\downarrow 9}] \to \star_{\uparrow 2}}}{\vdash (\lambda y.\lambda x.xy)\mathsf{I}(\lambda z.z) : \star_{\uparrow 1}}$$

Computing the Run Time

Computing the Run Time

We observe the following facts:

Computing the Run Time

We observe the following facts:

- there is only **one** initial state,

Computing the Run Time

We observe the following facts:

- there is only **one** initial state,
- there are **no cycles**.

Computing the Run Time

We observe the following facts:

- there is only **one** initial state,
- there are **no cycles**.

As a consequence:

Computing the Run Time

We observe the following facts:

- there is only **one** initial state,
- there are **no cycles**.

As a consequence:

1. all states are reached exactly **once**, and thus

Computing the Run Time

We observe the following facts:

- there is only **one** initial state,
- there are **no cycles**.

As a consequence:

1. all states are reached exactly **once**, and thus
2. run time = number of states = number of occurrences of $\star$ in the derivation.

Counting the $\star$: a Semantic Measure of the IAM run time

$$\|\star\| = 1 \qquad \|S \to A\| = \|S\| + \|A\| \qquad \big\|[A_1, \ldots, A_n]\big\| = \textstyle\sum_{1 \le i \le n} \|A_i\|$$

$$\frac{}{\vdash^{0} \lambda x.t : \star} \qquad \frac{\vdash^{w}}{\vdash^{w+\|S \to A\|} \lambda x.t : S \to A}$$

$$\frac{}{\vdash^{\|A\|} x : A} \qquad \frac{\vdash^{w} \quad \left[\vdash^{v_i}\right]_{i \in [1,\ldots,n]}}{\vdash^{w+\sum v_i+\|A\|} tu : A}$$

Higher-Order Types need Time

Higher-Order Types need Time

- **IAM (unreasonable?):** weights, thus run time, depend on types,

Higher-Order Types need Time

- **IAM (unreasonable?):** weights, thus run time, depend on types,
- **KAM (time reasonable):** weights, thus run time, do *not* depend on types.

Higher-Order Types need Time

- **IAM (unreasonable?):** weights, thus run time, depend on types,
- **KAM (time reasonable):** weights, thus run time, do *not* depend on types.

Indeed, give a term t such that $t \to^n \lambda x.u$:

Higher-Order Types need Time

- **IAM (unreasonable?):** weights, thus run time, depend on types,
- **KAM (time reasonable):** weights, thus run time, do *not* depend on types.

Indeed, give a term t such that $t \to^n \lambda x.u$:

- **types** of subterms of t can be **exponential** in $|t|$ and n,

Higher-Order Types need Time

- **IAM (unreasonable?):** weights, thus run time, depend on types,
- **KAM (time reasonable):** weights, thus run time, do *not* depend on types.

Indeed, give a term t such that $t \to^n \lambda x.u$:

- **types** of subterms of t can be **exponential** in $|t|$ and n,
- the **size** of the type derivation for t is **polynomial** in $|t|$ and n.

Higher-Order Types need Time

- **IAM (unreasonable?):** weights, thus run time, depend on types,
- **KAM (time reasonable):** weights, thus run time, do *not* depend on types.

Indeed, give a term t such that $t \to^n \lambda x.u$:

- **types** of subterms of t can be **exponential** in $|t|$ and n,
- the **size** of the type derivation for t is **polynomial** in $|t|$ and n.

Ultimately, IAM inefficiency comes from the use of **higher-order** types.

Recap

- IAM run time can be characterized by intersection type derivations.

How Can Space Be Captured?

How Can Space Be Captured?

We need to refine the type system:

How Can Space Be Captured?

We need to refine the type system:

Sequence Types for the CbN λ-Calculus

$$
\begin{array}{rrcll}
\text{Linear types} & A, A' & ::= & \star \mid T \to A & \\
\text{Tree types} & T, T' & ::= & [G_1, \ldots, G_n] & n \geq 0 \\
\text{(Generic) Types} & G, G' & ::= & A \mid T &
\end{array}
$$

How Can Space Be Captured?

We need to refine the type system:

Sequence Types for the CbN λ-Calculus

$$\begin{array}{rrcll} \text{Linear types} & A, A' & ::= & \star \mid T \to A & \\ \text{Tree types} & T, T' & ::= & [G_1, \ldots, G_n] & n \geq 0 \\ \text{(Generic) Types} & G, G' & ::= & A \mid T & \end{array}$$

Space Type Measure

$$\|\star\| := 0 \qquad \|T \to A\| := \max\{\|T\|, \|A\| + 1\}$$

$$\left\|[G_1, .., G_n]\right\| := \mathsf{X} + \max_i\{\|G_i\|\}$$

The Inference Rules

$$\frac{}{x : [A] \vdash x : A}\ \text{T-Var} \qquad \frac{\Gamma, x : T \vdash t : A}{\Gamma \vdash \lambda x.t : T \to A}\ \text{T-}\lambda$$

$$\frac{\Gamma \vdash t : T \to A \quad \Delta \vdash u : T}{\Gamma \uplus \Delta \vdash tu : A}\ \text{T-@} \qquad \frac{}{\Gamma \vdash \lambda x.t : \star}\ \text{T-}\lambda_\star$$

$$\frac{\Gamma_i \vdash t : G_i \quad 1 \leq i \leq n}{[\uplus_{i=1}^{n} \Gamma_i] \vdash t : [G_1, .., G_n]}\ \text{T-many} \qquad \frac{}{\vdash t : [\cdot]}\ \text{T-none}$$

The Inference Rules - Weighted

$$\frac{}{x : [A] \overset{\|A\|}{\vdash} x : A}\ \text{T-Var}$$

$$\frac{\Gamma, x : T \overset{w}{\vdash} t : A}{\Gamma \overset{\max\{w, \|T \to A\|\}}{\vdash} \lambda x.t : T \to A}\ \text{T-}\lambda$$

$$\frac{\Gamma \overset{w}{\vdash} t : T \to A \quad \Delta \overset{v}{\vdash} u : T}{\Gamma \uplus \Delta \overset{\max\{w,v\}}{\vdash} tu : A}\ \text{T-@}$$

$$\frac{}{\Gamma \overset{0}{\vdash} \lambda x.t : \star}\ \text{T-}\lambda_\star$$

$$\frac{\Gamma_i \overset{w_i}{\vdash} t : G_i \quad 1 \le i \le n}{[\uplus_{i=1}^{n} \Gamma_i] \overset{X + \max_i\{w_i\}}{\vdash} t : [G_1, .., G_n]}\ \text{T-many}$$

$$\frac{}{\overset{0}{\vdash} t : [\cdot]}\ \text{T-none}$$

Recap

- IAM run time can be characterized by sequence type derivations.
- IAM space consumption can be characterized by tree type derivations.

Typing Turing Machines

We would like to prove that the IAM is invariant.

Typing Turing Machines

We would like to prove that the IAM is invariant.
About time, the difficult direction is TM $\to$ IAM. We need to reason about the IAM execution of terms which are the image of *an* encoding of TM in the λ-calculus.

Typing Turing Machines

We would like to prove that the IAM is invariant.
About time, the difficult direction is TM $\to$ IAM. We need to reason about the IAM execution of terms which are the image of *an* encoding of TM in the λ-calculus.

The *while loop* of Turing machines is typically implemented through the fixed point combinator $\Theta := \theta\theta$.

Typing Turing Machines

We would like to prove that the IAM is invariant.
About time, the difficult direction is TM $\to$ IAM. We need to reason about the IAM execution of terms which are the image of *an* encoding of TM in the λ-calculus.

The *while loop* of Turing machines is typically implemented through the fixed point combinator $\Theta := \theta\theta$.

$$\begin{array}{rcl} \mathbb{F}^{\overline{A}}_{n+1} & := & [\mathbb{Y}^{\overline{A}}_{n+1}, \ldots, \mathbb{Y}^{\overline{A}}_{0}] \to A_{n+1} \\ \mathbb{Y}^{\overline{A}}_{n+1} & := & [A_n] \to A_{n+1} \\ \mathbb{X}^{\overline{A}}_{n+1} & := & [\mathbb{X}^{\overline{A}}_{n}, \ldots, \mathbb{X}^{\overline{A}}_{0}] \to \mathbb{F}^{\overline{A}}_{n+1} \end{array}$$

Typing Turing Machines

We would like to prove that the IAM is invariant.
About time, the difficult direction is TM $\to$ IAM. We need to reason about the IAM execution of terms which are the image of *an* encoding of TM in the λ-calculus.

The *while loop* of Turing machines is typically implemented through the fixed point combinator $\Theta := \theta\theta$.

$$\begin{array}{rcl} \mathbb{F}^{\overline{A}}_{n+1} & := & [\mathbb{Y}^{\overline{A}}_{n+1}, \ldots, \mathbb{Y}^{\overline{A}}_{0}] \to A_{n+1} \\ \mathbb{Y}^{\overline{A}}_{n+1} & := & [A_n] \to A_{n+1} \\ \mathbb{X}^{\overline{A}}_{n+1} & := & [\mathbb{X}^{\overline{A}}_{n}, \ldots, \mathbb{X}^{\overline{A}}_{0}] \to \mathbb{F}^{\overline{A}}_{n+1} \end{array}$$

Lemma

For each $n \geq 0$ and type list $\overline{A}$, $\vdash \theta : \mathbb{X}^{\overline{A}}_{n}$ and $\vdash \Theta : \mathbb{F}^{\overline{A}}_{n}$.

Typing Turing Machines

We would like to prove that the IAM is invariant.
About time, the difficult direction is TM $\to$ IAM. We need to reason about the IAM execution of terms which are the image of *an* encoding of TM in the λ-calculus.

The *while loop* of Turing machines is typically implemented through the fixed point combinator $\Theta := \theta\theta$.

$$\begin{array}{rcl} \mathbb{F}^{\overline{A}}_{n+1} & := & [\mathbb{Y}^{\overline{A}}_{n+1}, \ldots, \mathbb{Y}^{\overline{A}}_{0}] \to A_{n+1} \\ \mathbb{Y}^{\overline{A}}_{n+1} & := & [A_n] \to A_{n+1} \\ \mathbb{X}^{\overline{A}}_{n+1} & := & [\mathbb{X}^{\overline{A}}_{n}, \ldots, \mathbb{X}^{\overline{A}}_{0}] \to \mathbb{F}^{\overline{A}}_{n+1} \end{array}$$

Lemma

For each $n \geq 0$ and type list $\overline{A}$, $\vdash \theta : \mathbb{X}^{\overline{A}}_{n}$ and $\vdash \Theta : \mathbb{F}^{\overline{A}}_{n}$.

Observation

$\|\mathbb{X}_n\| = \Omega(2^n)$.

Recap

- IAM run time can be characterized by sequence type derivations.
- IAM space consumption can be characterized by tree type derivations.
- Given a TM that halts in n steps, its encoding in the λ-calculus takes $\Omega(2^n)$ steps when evaluated on the IAM.

Typing Turing Machines - Tree Types

The fixed point operator can be analogously typed with tree types.

Typing Turing Machines - Tree Types

The fixed point operator can be analogously typed with tree types.

$$\begin{array}{rrcl} \Theta : \mathbb{F}^{\overline{A}}_{n+1} & := & \mathbb{T}^{\overline{A}}_{n+1} \to A_{n+1} \\ \text{where} \quad \mathbb{T}^{\overline{A}}_{n+1} & := & [\mathbb{Y}^{\overline{A}}_{n+1}, [\mathbb{T}^{\overline{A}}_{n}]] \\ \text{and} \quad \mathbb{Y}^{\overline{A}}_{n+1} & := & [A_n] \to A_{n+1} \end{array}$$

Typing Turing Machines - Tree Types

The fixed point operator can be analogously typed with tree types.

$$\begin{array}{rrcl} & \Theta : \mathbb{F}^{\overline{A}}_{n+1} & := & \mathbb{T}^{\overline{A}}_{n+1} \to A_{n+1} \\ \text{where} & \mathbb{T}^{\overline{A}}_{n+1} & := & [\mathbb{Y}^{\overline{A}}_{n+1}, [\mathbb{T}^{\overline{A}}_{n}]] \\ \text{and} & \mathbb{Y}^{\overline{A}}_{n+1} & := & [A_n] \to A_{n+1} \end{array}$$

Lemma

For each $n \geq 0$ and type list $\overline{A}$, $\vdash \theta : \mathbb{F}^{\overline{A}}_{n}$.

Typing Turing Machines - Tree Types

The fixed point operator can be analogously typed with tree types.

$$\begin{array}{rrcl} & \Theta : \mathbb{F}^{\overline{A}}_{n+1} & := & \mathbb{T}^{\overline{A}}_{n+1} \to A_{n+1} \\ \text{where} & \mathbb{T}^{\overline{A}}_{n+1} & := & [\mathbb{Y}^{\overline{A}}_{n+1}, [\mathbb{T}^{\overline{A}}_{n}]] \\ \text{and} & \mathbb{Y}^{\overline{A}}_{n+1} & := & [A_n] \to A_{n+1} \end{array}$$

Lemma

For each $n \geq 0$ and type list $\overline{A}$, $\vdash \theta : \mathbb{F}^{\overline{A}}_{n}$.

Observation

$\|\mathbb{F}_n\| = \Omega(n)$.

Recap

- IAM run time can be characterized by sequence type derivations.
- IAM space consumption can be characterized by tree type derivations.
- Given a TM that halts in n steps, its encoding in the λ-calculus takes $\Omega(2^n)$ steps when evaluated on the IAM.
- Given a TM that halts in n steps, its encoding in the λ-calculus consumes $\Omega(n)$ space when evaluated on the IAM.

Ongoing and Future Work

Analyzing the space/time behaviour of environment machines *with* garbage collection.