Quantitative Global Memory

Sandra Alves [1] Delia Kesner [2] Miguel Ramos [3]

SMS

22/06/2023

[1]CRACS/INESC-TEC, DCC, Faculdade de Ciências, Universidade do Porto
[2]IRIF, CNRS, Université Paris Cité & Institut Universitaire de France
[3]LIACC, DCC, Faculdade de Ciências, Universidade do Porto

Programming Languages

λ-calculus (Pure)

- Simple structure
- No side-effects
- Easy to reason about
- Useless for programmers(?)

Real (Impure)

- Complicated structure
- Side-effects
- Hard to reason about
- Interact with the real world

Programming Languages

Is the λ-calculus *useless* for programmers?

It seems possible that the correspondence might form the basis of a formal description of the semantics of ALGOL 60. As presented here it reduces the problem of specifying ALGOL 60 semantics to that of specifying the semantics of a structurally simpler language.

Peter Landin

in "Correspondence between ALGOL 60 and Church's Lambda-notation: part I"

Programming Languages

How can we add *effects* to pure languages?

(Without making them harder to reason about...)

[I]n order to interpret a programming language [...], we distinguish the object A of values (of type A) from the object TA of computations (of type A) [...]. We call T a notion of computation, since it abstracts away from the types of values computations may produce.

Eugenio Moggi

in "Notions of Computation and Monads"

Monadic Effects

(Moggi's CBV Encoding)

Global State

Let S be the type of states. Then $TA = S \gg (A \times S)$:

$$\begin{aligned} v &\rightsquigarrow \lambda s.(v, s) \\ t\ u &\rightsquigarrow \lambda s.\text{let } (u', s') = u\ s \\ &\qquad \text{in } (t\ u')\ s' \end{aligned}$$

Exceptions

Let E be the type of exceptions. Then $TA = (E + A)$:

$$\begin{aligned} v &\rightsquigarrow \text{in}_r(v) \\ t\ u &\rightsquigarrow \text{case } u \text{ of in}_l(e) \mapsto e \\ &\qquad\qquad \text{in}_r(v) \mapsto t\ v \end{aligned}$$

Effect Operations

What about the operations that *create* effects?

The computational λ-calculus is essentially the same as the simply typed λ-calculus except for making a careful systematic distinction between computations and values. [...] However, the calculus does not contain operations, the constructs that actually create the effects. [...]

Gordon Plotkin and John Power

in “Algebraic Operations and Generic Effects”

Effect Operations

Global State

Let ℓ be a state location:

- Retrieving a value:

$$\texttt{get}_\ell(\lambda x.t)$$

- Setting a value:

$$\texttt{set}_\ell(v, t)$$

Exceptions

Let e be an exception name:

- Raising an exception:

$$\texttt{raise}_e()$$

- Handling an exception:

$$\texttt{handle}_e(t, u)$$

Intersection Types

- Extension of simple types with type constructor $\cap$

if τ, σ are types, then $\boxed{\tau \cap \sigma}$ is a type

- Originally enjoy associativity, commutativity and idempotency

$$(\tau \cap \sigma) \cap \theta = \tau \cap (\sigma \cap \theta) \qquad (\tau \cap \sigma) = (\sigma \cap \tau) \qquad \boxed{(\tau \cap \tau) = \tau}$$

- Express models capturing qualitative computational properties

"*t* is terminating iff *t* is typable"

Non-Idempotent Intersection Types

- Intersection types that do not enjoy idempotency $(\tau \cap \tau) \neq \tau$
- Express models capturing upper bound quantitative computational properties

"t is terminating in at most X steps iff t is typable"

- Size of type derivations is an upper bound for

evaluation length + size of result

- Size explosion

$$t_0 := y \qquad t_n := (\lambda x.xx)t_{n-1} \quad \leadsto \quad \underbrace{t_n}_{\text{linear in } n} \to_\beta^n \overbrace{y^{2^n}}^{\text{exponential in } n}$$

Split and Exact Measures

- To obtain split measures

$$\underbrace{\text{counters in judgments + tight constants + persistent typing rules}}_{\boxed{\text{(evaluation length, size of result)}}}$$

- To obtain exact measures

$$\boxed{\text{tight derivations = minimal derivations}}$$

- Obtain models capturing exact quantitative computational properties

"t is terminating in exactly X steps with normal form of size Y
iff t is typable with counter (X, Y)"

Quantitative Global Memory

Goal

To build a quantitative model (expressed as a tight type system) that captures exact quantitative properties of a λ-calculus with operations that interact with a global state.

Syntax

- We distinguish between values v and computations t (terms)
- Effect operations are used to interact with the global state
- The global state is defined through update operations
- Configurations are term-state pairs

$$
\begin{array}{rrcl}
\text{Values} & v, w & ::= & x \mid \lambda x.t \\
\text{Terms} & t, u & ::= & v \mid vt \mid \mathtt{get}_\ell(\lambda x.t) \mid \mathtt{set}_\ell(v, t) \\
\text{States} & s, q & ::= & \epsilon \mid \mathtt{upd}_\ell(v, s) \\
\text{Configurations} & c & ::= & (t, s)
\end{array}
$$

$$|v| := 0 \quad |vt| := 1 + |t| \quad |\mathtt{get}_\ell(\lambda x.t)| := |t| \quad |\mathtt{set}_\ell(v, t)| := |t|$$

$$|s| := 0 \quad |(t, s)| := |t|$$

Operational Semantics

(Configurations)

Let $\equiv$ be the equivalence relation generated by the following axiom

$$\mathtt{upd}_{\ell}(v, \mathtt{upd}_{\ell'}(w, s)) \quad \equiv_{\mathtt{c}} \quad \mathtt{upd}_{\ell'}(w, \mathtt{upd}_{\ell}(v, s)) \qquad \text{if } \ell \neq \ell'$$

$$\frac{}{((\lambda x.t)v, s) \to_{\beta_v} (t\{x\backslash v\}, s)} \qquad \frac{(t, s) \to_{\mathtt{r}} (u, q) \quad \mathtt{r} \in \{\beta_v, \mathtt{g}, \mathtt{s}\}}{(vt, s) \to_{\mathtt{r}} (vu, q)}$$

$$\frac{s \equiv \mathtt{upd}_{\ell}(v, q)}{(\mathtt{get}_{\ell}(\lambda x.t), s) \to_{\mathtt{g}} (t\{x\backslash v\}, s)} \qquad \frac{}{(\mathtt{set}_{\ell}(v, t), s) \to_{\mathtt{s}} (t, \mathtt{upd}_{\ell}(v, s))}$$

Weak reduction: we do not reduce inside abstractions

Operational Semantics ⚜ Example

$$
\begin{array}{ll}
 & ((\lambda x.\mathtt{get}_\ell(\lambda y.yx))(\mathtt{set}_\ell(\lambda x.x, z)), \epsilon) \\
\to_{\mathtt{s}} & ((\lambda x.\mathtt{get}_\ell(\lambda y.yx))z, \mathtt{upd}_\ell(\lambda x.x, \epsilon)) \\
\to_{\beta_v} & (\mathtt{get}_\ell(\lambda y.yz), \mathtt{upd}_\ell(\lambda x.x, \epsilon)) \\
\to_{\mathtt{g}} & ((\lambda x.x)z, \mathtt{upd}_\ell(\lambda x.x, \epsilon)) \\
\to_{\beta_v} & (z, \mathtt{upd}_\ell(\lambda x.x, \epsilon))
\end{array}
$$

$(0 \ \# \ \beta_v\text{-steps}, 0 \ \# \text{ memory accesses})$

Normal Forms & Blocked Configurations

(NF) Normal Forms

$$\overbrace{}^{\text{open terms}}$$

$$\begin{aligned} \texttt{no} &::= \boxed{x} \mid \lambda x.t \mid \texttt{ne} \\ \texttt{ne} &::= \boxed{x\ \texttt{no}} \mid (\lambda x.t)\ \texttt{ne} \end{aligned}$$

(BC) Blocked Configurations

$$\left.\begin{aligned} &(\texttt{get}_\ell(\lambda x.t), s) \\ &(v\ \texttt{get}_\ell(\lambda x.t), s) \end{aligned}\right. \quad \text{where } \ell \notin \texttt{dom}(s)$$

(FC) Final Configurations

$$\text{FC} = \text{BC} + (\text{NF}, s)$$

Encoding Arrow Types

$$\underbrace{A \Rightarrow B}_{\text{IL}} \overset{\text{Girard's CBV}}{\rightsquigarrow} \underbrace{!A \multimap !B}_{\text{ILL}} \overset{\text{Moggi's CBV}}{\rightsquigarrow} !A \multimap T(!B)$$

- $!A$ is an intersection of value types

$$!A = [A_1, \dots, A_n]$$

- T is the global state monad

$$TA = \mathcal{S} \gg (A \times \mathcal{S})$$

- $T(!A)$ is a computation wrapping an intersection of value types

$$T[A_1, \dots, A_n] = \mathcal{S} \gg ([A_1, \dots, A_n] \times \mathcal{S})$$

Types

- Values and Neutral Forms

$$
\begin{array}{rlll}
\text{Tight Constants} & \mathtt{tt} & ::= & \mathtt{v} \mid \mathtt{a} \mid \mathtt{n} \\
\text{Value Types} & \sigma & ::= & \mathtt{v} \mid \mathtt{a} \mid \mathcal{M} \mid \mathcal{M} \Rightarrow \delta \\
\text{Multi-types} & \mathcal{M} & ::= & [\sigma_i]_{i \in I} \text{ where } I \text{ is a finite set} \\
\text{Liftable Types} & \mu & ::= & \mathtt{v} \mid \mathtt{a} \mid \mathcal{M} \\
\text{Types} & \tau & ::= & \mathtt{n} \mid \sigma
\end{array}
$$

- States, Configurations, and Computations

$$
\begin{array}{rlll}
\text{State Types} & \mathcal{S} & ::= & \{\ell_i : \mathcal{M}_i\}_{i \in I} \text{ where all } \ell_i \text{ are distinct} \\
\text{Configuration Types} & \kappa & ::= & \tau \times \mathcal{S} \\
\text{Monadic Types} & \delta & ::= & \mathcal{S} \gg \kappa
\end{array}
$$

Typing

- Judgments are decorated with counters

$$\overbrace{(\;b\;,}^{\#\,\beta\text{-steps}}\;\underbrace{m\;,\;\overbrace{d\;)}^{|\text{normal form}|}}_{\#\text{ memory accesses}}$$

- We have three different kinds of typing judgments

$$\overbrace{\Gamma \vdash^{(b,m,d)} t : \delta}^{\textit{computations}} \qquad \overbrace{\Delta \vdash^{(b,m,d)} s : \mathcal{S}}^{\textit{states}} \qquad \overbrace{\Gamma \vdash^{(b,m,d)} (t, s) : \kappa}^{\textit{configurations}}$$

- Some typing rules have two (or more) different versions
  - *Consuming*: increase only b and m counters
  - *Persistent*: increase the d counter

Typing Rules ⚜ Values

$$\frac{}{x : [\sigma] \vdash^{(0,0,0)} x : \sigma}\ (\mathtt{ax})$$

$$\frac{\Gamma \vdash^{(b,m,d)} v : \mu}{\Gamma \vdash^{(b,m,d)} v : \mathcal{S} \gg (\mu \times \mathcal{S})}\ (\uparrow)$$

$$\frac{\Gamma; x : \mathcal{M} \vdash^{(b,m,d)} t : \mathcal{S} \gg \kappa}{\Gamma \vdash^{(b,m,d)} \lambda x.t : \mathcal{M} \Rightarrow (\mathcal{S} \gg \kappa)}\ (\lambda)$$

$$\frac{(\Gamma_i \vdash^{(b_i,m_i,d_i)} v : \sigma_i)_{i\in I}}{+_{i\in I}\Gamma_i \vdash^{(+_{i\in I}b_i,+_{i\in I}m_i,+_{i\in I}d_i)} v : [\sigma_i]_{i\in I}}\ (\mathtt{m})$$

Typing Rules ⚜ Computations

$$\frac{\Gamma \vdash^{(b,m,d)} v : \mathcal{M} \Rightarrow (\mathcal{S}_m \gg (\tau \times \mathcal{S}_f)) \qquad \Delta \vdash^{(b',m',d')} t : \mathcal{S}_i \gg (\mathcal{M} \times \mathcal{S}_m)}{\Gamma + \Delta \vdash^{(1+b+b',m+m',d+d')} vt : \mathcal{S}_i \gg (\tau \times \mathcal{S}_f)} \ (@)$$

$$\frac{\Gamma; x : \mathcal{M} \vdash^{(b,m,d)} t : \mathcal{S} \gg \kappa}{\Gamma \vdash^{(b,1+m,d)} \texttt{get}_\ell(\lambda x.t) : \{(\ell : \mathcal{M})\} \Cup \mathcal{S} \gg \kappa} \ (\texttt{get})$$

$$\frac{\Gamma \vdash^{(b,m,d)} v : \mathcal{M} \qquad \Delta \vdash^{(b',m',d')} t : \{(\ell : \mathcal{M})\}; \mathcal{S} \gg \kappa}{\Gamma + \Delta \vdash^{(b+b',1+m+m',d+d')} \texttt{set}_\ell(v, t) : \mathcal{S} \gg \kappa} \ (\texttt{set})$$

Typing Rules ⚜ States

$$\frac{}{\vdash^{(0,0,0)} \epsilon : \emptyset}\ (\texttt{emp})$$

$$\frac{\Gamma \vdash^{(b,m,d)} v : \mathcal{M} \qquad \Delta \vdash^{(b',m',d')} s : \mathcal{S}}{\Gamma + \Delta \vdash^{(b+b',m+m',d+d')} \texttt{upd}_{\ell}(v, s) : \{(\ell : \mathcal{M})\}; \mathcal{S}}\ (\texttt{upd})$$

Typing Rules ⚜ Configurations

$$\frac{\Gamma \vdash^{(b,m,d)} t : \mathcal{S} \gg \kappa \qquad \Delta \vdash^{(b',m',d')} s : \mathcal{S}}{\Gamma + \Delta \vdash^{(b+b',m+m',d+d')} (t,s) : \kappa} \ (\texttt{conf})$$

Exact Measures (Wrong)

Why do we need *tightness* and *persistent* typing rules?

Let $\sigma = [\mathtt{v}] \Rightarrow (\mathcal{S} \gg (\tau \times \mathcal{S}'))$.

$$
\dfrac{
\dfrac{}{x : [\sigma] \vdash^{(0,0,0)} x : [\mathtt{v}] \Rightarrow (\mathcal{S} \gg (\tau \times \mathcal{S}'))}\,(\mathtt{ax})
\qquad
\dfrac{\dfrac{\dfrac{}{y : [\mathtt{v}] \vdash^{(0,0,0)} y : \mathtt{v}}\,(\mathtt{ax})}{y : [\mathtt{v}] \vdash^{(0,0,0)} y : [\mathtt{v}]}\,(\mathtt{m})}{y : [\mathtt{v}] \vdash^{(0,0,0)} y : \mathcal{S} \gg ([\mathtt{v}] \times \mathcal{S})}\,(\uparrow)
}{
x : [\sigma], y : [\mathtt{v}] \vdash^{(1,0,0)} xy : \mathcal{S} \gg (\tau \times \mathcal{S}')
}\,(@)
$$

$$
(\underbrace{xy}_{|\mathrm{xy}| = 1}, s) \not\to \text{ for any } s
$$

Tightness Criteria

- τ is tight if it is a tight constant

$$\mathtt{v} \quad \mathtt{a} \quad \mathtt{n}$$

- $\mathcal{S}$ is tight if $\forall \ell \in \mathtt{dom}(\mathcal{S}).\mathtt{tight}(\mathcal{S}(\ell))$

$$\{(\ell_1 : [\mathtt{v}]), (\ell_2 : [\mathtt{a}, \mathtt{a}])\}$$

- $\mathcal{S} \gg (\tau \times \mathcal{S}')$ is tight if $\tau \times \mathcal{S}'$ is tight

$$\{(\ell_1 : [\mathcal{M} \Rightarrow \delta])\} \gg (\mathtt{a} \times \{(\ell_2 : [\,])\})$$

- $\mathtt{tight}(\mathcal{M})$ holds if all $\sigma \in \mathcal{M}$ are tight

$$[\,] \quad [\mathtt{a}, \mathtt{a}, \mathtt{v}, \mathtt{n}]$$

- $\tau \times \mathcal{S}$ is tight if τ and $\mathcal{S}$ are tight

$$\mathtt{n} \times \{(\ell_1 : [\mathtt{a}, \mathtt{v}]), (\ell_2 : [\,])\}$$

- Φ is tight if has a tight conclusion

$$\Phi \triangleright x : [\mathtt{a}], y : [\mathtt{v}] \vdash^{(0,0,0)} xy : \mathtt{n}$$

Typing Rules ⚜ Persistent

$$\frac{}{\vdash^{(0,0,0)} \lambda x.t : \mathtt{a}} \; (\lambda_{\mathtt{p}})$$

$$\frac{\Gamma \vdash^{(b,m,d)} t : \mathcal{S} \gg (\mathtt{tt} \times \mathcal{S}')}{(x : [\mathtt{v}]) + \Gamma \vdash^{(b,m,1+d)} xt : \mathcal{S} \gg (\mathtt{n} \times \mathcal{S}')} \; (@_{\mathtt{p1}})$$

$$\frac{\Gamma \vdash^{(b,m,d)} u : \mathcal{S} \gg (\mathtt{n} \times \mathcal{S}')}{\Gamma \vdash^{(b,m,1+d)} (\lambda x.t)u : \mathcal{S} \gg (\mathtt{n} \times \mathcal{S}')} \; (@_{\mathtt{p2}})$$

Exact Measures (Correct)

$$\dfrac{\dfrac{\dfrac{}{y : [\mathtt{a}] \vdash^{(0,0,0)} y : \mathtt{a}}\ (\mathtt{ax})}{y : [\mathtt{a}] \vdash^{(0,0,0)} y : \emptyset \gg (\mathtt{a} \times \emptyset)}\ (\uparrow)}{x : [\mathtt{v}], y : [\mathtt{a}] \vdash^{(\boxed{0},0,\boxed{1})} xy : \emptyset \gg (\mathtt{n} \times \emptyset)}\ (@_{\mathtt{p1}})$$

$$(\underbrace{xy}_{|\mathrm{xy}| = \boxed{1}}, s) \not\to \text{ for any } s$$

Validity of the Model

Soundness

If $\Phi \triangleright \Gamma \vdash^{(b,m,d)} (t,s) : \kappa$ tight,
then $\exists (u,q)$ s.t. $u \in \mathtt{no}$ and $(t,s) \twoheadrightarrow^{(b,m)} (u,q)$,
with b β-steps, m $\mathtt{g}/\mathtt{s}$-steps, and $|(u,q)| = d$.

Completeness

If $(t,s) \twoheadrightarrow^{(b,m)} (u,q)$ s.t. $u \in \mathtt{no}$,
then $\exists \Phi \triangleright \Gamma \vdash^{(b,m,|(u,q)|)} (t,s) : \kappa$ tight.

Typing Example

Let us consider the term exemplifying the operational semantics:

$$((\lambda x.\mathtt{get}_\ell(\lambda y.yx))(\mathtt{set}_\ell(\lambda x.x, z)), \epsilon) \twoheadrightarrow^{(2,2)} (\overbrace{z}^{|z| = 0}, \mathtt{upd}_\ell(\lambda x.x, \epsilon))$$

Typing Example

Let Φ be the following derivation for $\lambda x.\mathtt{get}_l(\lambda y.yx)$:

$$\dfrac{\dfrac{}{y : \mathcal{M} \vdash^{(0,0,0)} y : [\mathtt{v}] \Rightarrow \emptyset \gg (\mathtt{v} \times \emptyset)}\ (\mathtt{ax}) \quad \dfrac{\dfrac{\dfrac{}{x : [\mathtt{v}] \vdash^{(0,0,0)} x : \mathtt{v}}\ (\mathtt{ax})}{x : [\mathtt{v}] \vdash^{(0,0,0)} x : [\mathtt{v}]}\ (\mathtt{m})}{x : [\mathtt{v}] \vdash^{(0,0,0)} x : \emptyset \gg ([\mathtt{v}] \times \emptyset)}\ (\uparrow)}{\dfrac{y : \mathcal{M}, x : [\mathtt{v}] \vdash^{(1,0,0)} yx : \emptyset \gg (\mathtt{v} \times \emptyset)}{\dfrac{x : [\mathtt{v}] \vdash^{(1,1,0)} \mathtt{get}_l(\lambda y.yx) : \{(l : \mathcal{M})\} \gg (\mathtt{v} \times \emptyset)}{\vdash^{(1,1,0)} \lambda x.\mathtt{get}_l(\lambda y.yx) : [\mathtt{v}] \Rightarrow (\{(l : \mathcal{M})\} \gg (\mathtt{n} \times \emptyset))}\ (\lambda)}\ (\mathtt{get})}\ (\mathtt{@})$$

Typing Example

Let Ψ be the following derivation for $\mathtt{set}_l(\lambda x.x, z)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\mathtt{v}] \vdash^{(0,0,0)} x : \mathtt{v}}\,(\mathtt{ax})}{x : [\mathtt{v}] \vdash^{(0,0,0)} x : \emptyset \gg (\mathtt{v} \times \emptyset)}\,(\uparrow)}{\vdash^{(0,0,0)} \lambda x.x : [\mathtt{v}] \Rightarrow \emptyset \gg (\mathtt{v} \times \emptyset)}\,(\lambda)}{\vdash^{(0,0,0)} \lambda x.x : \mathcal{M}}\,(\mathtt{m}) \quad \dfrac{\dfrac{\dfrac{}{z : [\mathtt{v}] \vdash^{(0,0,0)} z : \mathtt{v}}\,(\mathtt{ax})}{z : [\mathtt{v}] \vdash^{(0,0,0)} z : [\mathtt{v}]}\,(\mathtt{m})}{z : [\mathtt{v}] \vdash^{(0,0,0)} z : \{(l : \mathcal{M})\} \gg ([\mathtt{v}] \times \{(l : \mathcal{M})\})}\,(\uparrow)}{z : [\mathtt{v}] \vdash^{(0,1,0)} \mathtt{set}_l(\lambda x.x, z) : \emptyset \gg ([\mathtt{v}] \times \{(l : \mathcal{M})\})}\,(\mathtt{set})$$

Typing Example

Using Φ and Ψ, we can build the following tight derivation:

$$\dfrac{\dfrac{\Phi \quad \Psi}{z : [\mathtt{v}] \vdash^{(2,2,0)} (\lambda x.\mathtt{get}_l(\lambda y.yx))(\mathtt{set}_l(\mathtt{I}, z)) : \emptyset \gg (\mathtt{v} \times \emptyset)}\,(@) \qquad \dfrac{}{\vdash^{(0,0,0)} \epsilon : \emptyset}\,(\mathtt{emp})}{z : [\mathtt{v}] \vdash^{(2,2,0)} ((\lambda x.\mathtt{get}_l(\lambda y.yx))(\mathtt{set}_l(\mathtt{I}, z)), \epsilon) : \mathtt{v} \times \emptyset}\,(\mathtt{conf})$$

$$((\lambda x.\mathtt{get}_\ell(\lambda y.yx))(\mathtt{set}_\ell(\lambda x.x, z)), \epsilon) \twoheadrightarrow^{(2,2)} (\underbrace{z}_{|z| = 0}, \mathtt{upd}_\ell(\lambda x.x, \epsilon))$$

Conclusion

We have provided a foundational step into the development of quantitative models for programming languages with effects:

- Presented a simple language with global memory access capabilities
- Fixed a particular evaluation strategy following a weak CBV approach
- Provided a quantitative model cable of extracting and discriminate between exact measures for:
  - Length of evaluation
  - Number of memory accesses
  - Size of normal forms

Future Work

Different Effects

- Exceptions
- Non-determinism
- I/O
- ...

Different Strategies

- CBV (full)
- CBN
- CBNeed
- ...

Unifying Frameworks

- CBPV
- E.Eff.-Calculus
- Bang-Calculus
- ...

The End