Università di Genova
DIBRIS DIPARTIMENTO DI INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Towards Abstract and (hopefully) Compositional Operational Reasoning

Francesco Dagnino

T-LADIES kick-off

Who am I?

Postdoc at DIBRIS University of Genoa
Programming Languages research group
Genova Logic Group

Research Interests

- operational semantics and operational reasoning
- type systems (global types, session types, coeffect systems, ...)
- category theory for logics, type theories and programming languages

Reasoning about programs

formal guarantees on the behaviour of programs

- correctness of program transformations/approximations
  program equivalence and distance
- correctness of static/dynamic verification techniques
  type systems, program logics, ...

Reasoning about programs

formal guarantees on the behaviour of programs

- correctness of program transformations/approximations
  program equivalence and distance
- correctness of static/dynamic verification techniques
  type systems, program logics, ...

Bricks

- formal (mathematical) model of programs: syntax and semantics
- reasoning/proof principles and methods (induction and coinduction, logical relations and predicates, ...)

Operational vs Denotational

two approaches to formal semantics and reasoning

Operational vs Denotational

two approaches to formal semantics and reasoning

Denotational

- programs denote abstract mathematical objects (functions, relations, arrows in a category)
- abstract and quite modular theory
- heavy mathematical tools

Operational vs Denotational

two approaches to formal semantics and reasoning

Denotational

- programs denote abstract mathematical objects (functions, relations, arrows in a category)
- abstract and quite modular theory
- heavy mathematical tools

Operational

- describes how a program is executed/evaluated
- lightweight and versatile, wide applicability
- lack of abstract/general results, monolitic, case by case

Operational Reasoning

operational reasoning = (formal) reasoning based on an operational semantics

Operational Reasoning

operational reasoning = (formal) reasoning based on an operational semantics

several styles of operational semantics

- abstract machines
- small-step semantics
- big-step semantics
- evaluation semantics

Wishlist

all computer scientists are lazy!

reuse results/techniques already proved/introduced

Wishlist

all computer scientists are lazy!

reuse results/techniques already proved/introduced

Desiderata

- abstractness
  $\Rightarrow$ apply general results/techniques to specific instances

Wishlist

all computer scientists are lazy!

reuse results/techniques already proved/introduced

Desiderata

- abstractness
  ⇒ apply general results/techniques to specific instances
- modularity
  ⇒ compose results on smaller/simpler parts

Wishlist

all computer scientists are lazy!

reuse results/techniques already proved/introduced

Desiderata

- abstractness
  ⇒ apply general results/techniques to specific instances
- modularity
  ⇒ compose results on smaller/simpler parts

The harsh reality

- lack of abstract theories
- results tailored to specific languages
- monolitic development

What can we do?

Operational reasoning in-the-abstract

first steps...

- give a general/abstract definition of operational semantics
- develop general and modular techniques

What can we do?

Operational reasoning in-the-abstract

first steps...

- give a general/abstract definition of operational semantics
- develop general and modular techniques

In this talk

Part I Abstract Big-Step Semantics

Part II Abstract Evaluation Semantics

Part I

Abstract Big-Step Semantics

An example: call-by-value λ-calculus

An example: call-by-value λ-calculus

Syntax

$$\begin{array}{rcll} t, s & ::= & x \mid \lambda x.t \mid t\,s & \text{expressions} \\ v, w & ::= & \lambda x.t \mid n & \text{values} \end{array}$$

An example: call-by-value λ-calculus

Syntax

$$\begin{array}{rcll} t, s & ::= & x \mid \lambda x.t \mid t\,s & \text{expressions} \\ v, w & ::= & \lambda x.t \mid n & \text{values} \end{array}$$

Semantics

judgement: $t \Rightarrow v$
expression t evaluates to value v

An example: call-by-value λ-calculus

Syntax

$$\begin{array}{rcll} t, s & ::= & x \mid \lambda x.t \mid t\,s & \text{expressions} \\ v, w & ::= & \lambda x.t \mid n & \text{values} \end{array}$$

Semantics

judgement: $t \Rightarrow v$
expression t evaluates to value v

$$\frac{}{v \Rightarrow v} \qquad \frac{t_1 \Rightarrow \lambda x.s \quad t_2 \Rightarrow v \quad s[v/x] \Rightarrow w}{t_1\,t_2 \Rightarrow w}$$

Towards an abstract definition

guiding principles:

Towards an abstract definition

guiding principles:

- being language independent
  abstract from syntactic aspects
  similar to (abstract) rewriting systems

Towards an abstract definition

guiding principles:

- being language independent
  abstract from syntactic aspects
  similar to (abstract) rewriting systems
- describe the core structure of a big-step semantics

Towards an abstract definition

guiding principles:

- being language independent
  abstract from syntactic aspects
  similar to (abstract) rewriting systems
- describe the core structure of a big-step semantics
  ⇒ shape of rules
  describing the evaluation process

Defining big-step semantics

A big-step semantics is a triple $(C, R, \mathcal{R})$ where

Defining big-step semantics

A big-step semantics is a triple $(C, R, \mathcal{R})$ where

- C is a set of configurations
- R is a set of results

Defining big-step semantics

A big-step semantics is a triple $(C, R, \mathcal{R})$ where

- C is a set of configurations
- R is a set of results
- a judgement has shape $c \Rightarrow r$
  configuration c evaluates to result r

Defining big-step semantics

A big-step semantics is a triple $(C, R, \mathcal{R})$ where

- C is a set of configurations
- R is a set of results
- a judgement has shape $c \Rightarrow r$
  configuration c evaluates to result r
- $\mathcal{R}$ is a set of rules of shape

$$\frac{c_1 \Rightarrow r_1 \quad \ldots \quad c_n \Rightarrow r_n}{c \Rightarrow r}$$

where $n \geq 0$ and premises are totally ordered (left-to-right)

Example revisited

$$\frac{t_1 \Rightarrow \lambda x.s \quad t_2 \Rightarrow v \quad s[v/x] \Rightarrow w}{t_1\, t_2 \Rightarrow w}$$

Example revisited

$$\frac{t_1 \Rightarrow \lambda x.s \quad t_2 \Rightarrow v \quad s[v/x] \Rightarrow w}{t_1\, t_2 \Rightarrow w}$$

- evaluate t_1 and check that the result is an abstraction
- evaluate t_2
- evaluate the substitution and return the result

Example revisited

$$\frac{t_1 \Rightarrow \lambda x.s \quad t_2 \Rightarrow v \quad s[v/x] \Rightarrow w}{t_1\, t_2 \Rightarrow w}$$

- evaluate t_1 and check that the result is an abstraction
- evaluate t_2
- evaluate the substitution and return the result

other strategies

- right-to-left $\dfrac{t_2 \Rightarrow v \quad t_1 \Rightarrow \lambda x.s \quad s[v/x] \Rightarrow w}{t_1\, t_2 \Rightarrow w}$

Example revisited

$$\frac{t_1 \Rightarrow \lambda x.s \quad t_2 \Rightarrow v \quad s[v/x] \Rightarrow w}{t_1\, t_2 \Rightarrow w}$$

- evaluate t_1 and check that the result is an abstraction
- evaluate t_2
- evaluate the substitution and return the result

other strategies

- right-to-left $\dfrac{t_2 \Rightarrow v \quad t_1 \Rightarrow \lambda x.s \quad s[v/x] \Rightarrow w}{t_1\, t_2 \Rightarrow w}$
- late error detection $\dfrac{t_1 \Rightarrow v_1 \quad t_2 \Rightarrow v_2 \quad v_1 \Rightarrow \lambda x.s \quad s[v_2/x] \Rightarrow w}{t_1\, t_2 \Rightarrow w}$

Results I

An issue

in big-step semantics stuck and non-terminating computations are indistinguishable
$\Rightarrow$ in both cases no judgement is derivable

Results I

An issue

in big-step semantics stuck and non-terminating computations are indistinguishable
$\Rightarrow$ in both cases no judgement is derivable

we show that this distinction is hidden in any big-step semantics

- partial evaluation trees
- explicit wrong computations $c \Rightarrow \text{wrong}$
- explicit non-terminating computations $c \Rightarrow \infty$ (or via traces)

Results II

Proof technique for soundness

A predicate on configurations is sound if
the evaluation of a configuration satisfying the predicate cannot go wrong

we give a general proof technique for proving soundness w.r.t. any big-step semantics

Results II

Proof technique for soundness

A predicate on configurations is sound if
the evaluation of a configuration satisfying the predicate cannot go wrong

we give a general proof technique for proving soundness w.r.t. any big-step semantics

Semantics with observations

big-step semantics describing also the observable behaviour of a program
general extension to infinite behaviour

Part II

Abstract Evaluation Semantics

Reference language

simply-typed, fine grained, call-by-value λ-calculus with generic effects:

Reference language

simply-typed, fine grained, call-by-value λ-calculus with generic effects:

$$v, w ::= x \mid c \mid \langle\rangle \mid \lambda x.t \mid \langle v, w\rangle$$

$$t, s ::= \mathbf{val}\ v \mid vw \mid v.1 \mid v.2 \mid t\ \mathbf{to}\ x.s \mid \gamma(v_1, \ldots, v_n)$$

$$\sigma, \tau ::= \zeta \mid \sigma \to \underline{\tau} \mid \sigma \times \tau \mid \mathbf{1}$$

values and computations are kept separate

Reference language

simply-typed, fine grained, call-by-value λ-calculus with generic effects:

$$v, w ::= x \mid c \mid \langle\rangle \mid \lambda x.t \mid \langle v, w\rangle$$
$$t, s ::= \textbf{val}\ v \mid vw \mid v.1 \mid v.2 \mid t\ \textbf{to}\ x.s \mid \gamma(v_1, \ldots, v_n)$$

$$\sigma, \tau ::= \zeta \mid \sigma \to \underline{\tau} \mid \sigma \times \tau \mid \mathbf{1}$$

values and computations are kept separate

$\gamma : \sigma_1 \ldots \sigma_n \to \sigma$ is a (parametric) generic effect = atomic effectful operation (e.g., sempling from a distribution, storing a value in a location, ...)

Typing rules

$$\frac{}{\Gamma \vdash x : \sigma}\; x : \sigma \in \Gamma \qquad \frac{}{\Gamma \vdash c : \zeta_c}$$

$$\frac{\Gamma \vdash v_1 : \sigma_1 \quad \ldots \quad \Gamma \vdash v_n : \sigma_n}{\Gamma \vdash \gamma(v_1, \ldots, v_n) : \underline{\sigma}}\; \gamma : \sigma_1 \ldots \sigma_n \to \sigma \qquad \frac{}{\Gamma \vdash \langle\rangle : \mathbf{1}}$$

$$\frac{\Gamma \vdash v : \sigma}{\Gamma \vdash \mathbf{val}\, v : \underline{\sigma}} \qquad \frac{\Gamma \vdash t : \underline{\sigma} \quad \Gamma, x : \sigma \vdash s : \underline{\tau}}{\Gamma \vdash t\, \mathbf{to}\, x.s : \underline{\tau}}$$

$$\frac{\Gamma, x : \sigma \vdash t : \underline{\tau}}{\Gamma \vdash \lambda x.t : \sigma \to \underline{\tau}} \qquad \frac{\Gamma \vdash v : \sigma \to \underline{\tau} \quad \Gamma \vdash w : \sigma}{\Gamma \vdash vw : \underline{\tau}}$$

$$\frac{\Gamma \vdash v : \sigma \quad \Gamma \vdash w : \tau}{\Gamma \vdash \langle v, w\rangle : \sigma \times \tau} \qquad \frac{\Gamma \vdash v : \sigma \times \tau}{\Gamma \vdash v.1 : \underline{\sigma}} \qquad \frac{\Gamma \vdash v : \sigma \times \tau}{\Gamma \vdash v.2 : \underline{\tau}}$$

Monadic evaluation semantics

Λ_σ = set of closed computations of type σ
$\mathcal{V}_\sigma$ = set of closed values of type σ

Monadic evaluation semantics

Λ_σ = set of closed computations of type σ
$\mathcal{V}_\sigma$ = set of closed values of type σ
let $(T, \gg\!=, \eta)$ be a (strong) monad on $\mathcal{S}et$
a monadic evaluation semantics is a (family of) function

$$[\![-]\!] : \Lambda_\sigma \to T(\mathcal{V}_\sigma)$$

such that the following holds

Monadic evaluation semantics

Λ_σ = set of closed computations of type σ
$\mathcal{V}_\sigma$ = set of closed values of type σ
let $(T, \gg=, \eta)$ be a (strong) monad on $\mathcal{S}et$
a monadic evaluation semantics is a (family of) function

$$[\![-]\!] : \Lambda_\sigma \to T(\mathcal{V}_\sigma)$$

such that the following holds

$$[\![\mathbf{val}\ v]\!] = \eta(v) \qquad [\![(\lambda x.t)v]\!] = [\![t[v/x]]\!]$$
$$[\![t\ \mathbf{to}\ x.s]\!] = [\![t]\!] \gg= (v \mapsto [\![s[v/x]]\!]) \qquad [\![\langle v, w\rangle.1]\!] = \eta(v)$$
$$[\![\gamma(v_1, \ldots, v_n)]\!] = \widehat{\gamma}(v_1, \ldots, v_n) \qquad [\![\langle v, w\rangle.2]\!] = \eta(w)$$

where $\widehat{\gamma} : [\![\sigma_1]\!] \times \cdots \times [\![\sigma_n]\!] \to T([\![\sigma]\!])$ if $\gamma : \sigma_1 \ldots \sigma_n \to \sigma$

Monadic evaluation semantics

Λ_σ = set of closed computations of type σ
$\mathcal{V}_\sigma$ = set of closed values of type σ
let $(T, \gg=, \eta)$ be a (strong) monad on $\mathcal{S}et$
a monadic evaluation semantics is a (family of) function

$$[\![-]\!] : \Lambda_\sigma \to T(\mathcal{V}_\sigma)$$

such that the following holds

$$[\![\textbf{val } v]\!] = \eta(v) \qquad [\![(\lambda x.t)v]\!] = [\![t[v/x]]\!]$$
$$[\![t \textbf{ to } x.s]\!] = [\![t]\!] \gg= (v \mapsto [\![s[v/x]]\!]) \qquad [\![\langle v, w\rangle.1]\!] = \eta(v)$$
$$[\![\gamma(v_1, \ldots, v_n)]\!] = \widehat{\gamma}(v_1, \ldots, v_n) \qquad [\![\langle v, w\rangle.2]\!] = \eta(w)$$

where $\widehat{\gamma} : [\![\sigma_1]\!] \times \cdots \times [\![\sigma_n]\!] \to T([\![\sigma]\!])$ if $\gamma : \sigma_1 \ldots \sigma_n \to \sigma$

it is usually defined as a fixpoint

Syntactic graph

values and computation form a graph $\mathcal{Syn}$ where

- nodes are typing environments Γ, value type σ and computation types $\underline{\sigma}$
- edges from Γ to σ are values s.t. $\Gamma \vdash v : \sigma$
  edges from Γ to $\underline{\sigma}$ are computations s.t. $\Gamma \vdash t : \underline{\sigma}$

Abstract monadic evaluation semantics

Abstract monadic evaluation semantics

let $\mathcal{B}$ be a category with finite products
$(T, \gg\!=, \eta)$ a (strong) monad on $\mathcal{B}$

Abstract monadic evaluation semantics

let $\mathcal{B}$ be a category with finite products
$(T, \gg=, \eta)$ a (strong) monad on $\mathcal{B}$

Operational Structure

a $\mathcal{Syn}$-operational struture on $\mathcal{B}$ consists of

- a diagram $S \colon \mathcal{Syn} \to \mathcal{B}$ such that
  $S(x_1 : \sigma_1, \ldots, x_n : \sigma_n) = S(\sigma_1) \times \cdots \times S(\sigma_n)$

Abstract monadic evaluation semantics

let $\mathcal{B}$ be a category with finite products
$(T, \gg=, \eta)$ a (strong) monad on $\mathcal{B}$

Operational Structure

a $\mathcal{Syn}$-operational struture on $\mathcal{B}$ consists of

- a diagram $S \colon \mathcal{Syn} \to \mathcal{B}$ such that $S(x_1 : \sigma_1, \ldots, x_n : \sigma_n) = S(\sigma_1) \times \cdots \times S(\sigma_n)$
- families of arrows

$$\widehat{\iota} \colon 1 \to S(\mathbf{1}) \qquad \widehat{c} \colon 1 \to S(\zeta)$$

$$\widehat{p_1}_{\sigma,\tau} \colon S(\sigma \times \tau) \to S(\sigma) \qquad \widehat{p_2}_{\sigma,\tau} \colon S(\sigma \times \tau) \to S(\tau)$$

$$\widehat{\beta}_{\sigma,\tau} \colon S(\sigma \to \underline{\tau}) \times S(\sigma) \to S(\tau) \qquad \widehat{\gamma} \colon S(\sigma_1) \times \cdots \times S(\sigma_n) \to T(S(\sigma))$$

$$\widehat{e}_{\sigma} \colon S(\underline{\sigma}) \to T(S(\sigma))$$

satisfying some commutative diagrams

Example: $\mathcal{S}et$-based semantics

- $S(\sigma) = \mathcal{V}_\sigma$ and $S(\underline{\sigma}) = \Lambda_\sigma$
  $S(x_1 : \sigma_1, \ldots, x_n : \sigma_n) = \mathcal{V}_{\sigma_1} \times \cdots \times \mathcal{V}_{\sigma_n}$

- if $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash v : \sigma$ then
  $S(v) = (v_1, \ldots, v_n) \mapsto v[v_1/x_1, \ldots, v_n/x_n]$
- if $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash t : \underline{\sigma}$ then
  $S(t) = (v_1, \ldots, v_n) \mapsto t[v_1/x_1, \ldots, v_n/x_n]$

Example: $\mathcal{S}et$-based semantics

- $S(\sigma) = \mathcal{V}_\sigma$ and $S(\underline{\sigma}) = \Lambda_\sigma$
  $S(x_1 : \sigma_1, \ldots, x_n : \sigma_n) = \mathcal{V}_{\sigma_1} \times \cdots \times \mathcal{V}_{\sigma_n}$

- if $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash v : \sigma$ then
  $S(v) = (v_1, \ldots, v_n) \mapsto v[v_1/x_1, \ldots, v_n/x_n]$
- if $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash t : \underline{\sigma}$ then
  $S(t) = (v_1, \ldots, v_n) \mapsto t[v_1/x_1, \ldots, v_n/x_n]$

- $\widehat{\beta}(\lambda x.t, v) = t[v/x]$
  $\widehat{p_i}(\langle v_1, v_2 \rangle) = v_i$
  $\widehat{e}(t) = [\![t]\!]$

Results

- operational semantics beyond $\mathcal{S}et$ (e.g., stochastic λ calculus in measurable spaces)
- general definition of operational logical relations in terms of fibrations
- proved once and for all the fundamental lemma of operational logical relations
- mathematical foundations of differential logical relations for effectful higher-order distances between programs

References

- Francesco Dagnino, Viviana Bono, Elena Zucca and Mariangiola Dezani-Ciancaglini (2020). ”Soundness conditions for big-step semantics”. ESOP 2020
- Davide Ancona, Francesco Dagnino, Jurriaan Rot and Elena Zucca (2020). ”A big-step from finite to infinite computations. ECOOP 2020, special issue in Science of Computer Programming
- Francesco Dagnino (2021). ”Flexible Coinduction”. PhD Thesis
- Francesco Dagnino (2022). ”A meta-theory for big-step semantics”. ACM Transactions on Computational Logic
- Francesco Dagnino and Francesco Gavazzo (2022). ”A Fibrational Tale of Operational Logical Relations”. FSCD 2022

A quick comparison

Big-Step Semantics

- more common, based on inference rules, easily understandable
- too weak structure (just sets of rules)

Evaluation Semantics

- rich structure, syntax directed
- easy to implement, formalisation in proof-assistant
- non-termination is difficult
- more sophisticated tools

We are just at the beginning!

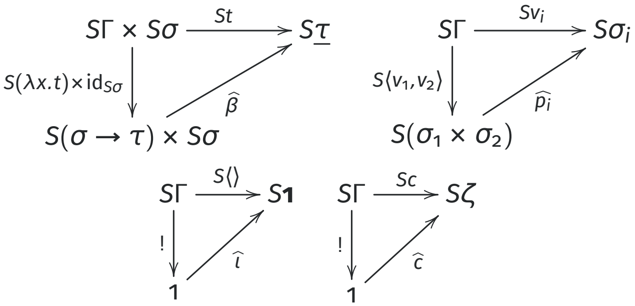
- abstract evaluation semantics for arbitrary language
- infinite behaviour in abstract evaluation semantics (delay monad?)
- modularised versions of the two approaches
- composition operators
- language translations, morphisms of operational semantics
- ... suggestions?

Questions?

Thank you!

Università
di Genova

Diagrams for operational structures



Diagrams for operational structures

$$
\begin{array}{ccc}
S\Gamma & \xrightarrow{S(\textbf{val}\ v)} & S\underline{\sigma} \\
{\scriptstyle S(v)}\downarrow & & \downarrow{\scriptstyle \widehat{e}} \\
S\sigma & \xrightarrow[\eta]{} & T(S\sigma)
\end{array}
\qquad
\begin{array}{ccc}
S\Gamma & \xrightarrow{S(\gamma(v_1,\ldots,v_n))} & S(\underline{\sigma}) \\
{\scriptstyle (S(v_1),\ldots,S(v_n))}\downarrow & & \downarrow{\scriptstyle \widehat{e}} \\
S\sigma_1 \times \cdots \times S\sigma_n & \xrightarrow[\widehat{\gamma}]{} & T(S\sigma)
\end{array}
$$

$$
\begin{array}{ccccc}
S\Gamma & & \xrightarrow{S(t\ \textbf{to}\ x.s)} & & S\underline{\tau} \\
{\scriptstyle \langle \mathrm{id}, S(t)\rangle}\downarrow & & & & \downarrow{\scriptstyle \widehat{e}} \\
S\Gamma \times S\underline{\sigma} & \xrightarrow[\mathrm{id}\times\widehat{e}]{} & S\Gamma \times T(S\sigma) & \xrightarrow[\gg\!=(\widehat{e}\circ S(s))]{} & T(S\tau)
\end{array}
$$

Diagrams for operational structures