Operational Semantics in Languages and Logics

Robert Harper

Felleisen Halftime Festival
November 4, 2018

Es war einmal

Way back when I first met Matthias, he was saying,

> *I'm interested in all aspects of programming languages, except for types.*

Es war einmal

Way back when I first met Matthias, he was saying,

> *I'm interested in all aspects of programming languages, except for types.*

And I was saying,

> *This guy is nuts!*

Es war einmal

Way back when I first met Matthias, he was saying,

> *I'm interested in all aspects of programming languages, except for types.*

And I was saying,

> *This guy is nuts!*

And thinking,

> *(Or is it me?)*

Es war einmal

Way back when I first met Matthias, he was saying,

I'm interested in all aspects of programming languages, except for types.

And I was saying,

This guy is nuts!

And thinking,

(Or is it me?)

We seemed to be miles apart. Who is right?

Operational Methods

It was the era of operational semantics.

- Plotkin's *Structural Operational Semantics*.
- Milner, et al's *Definition of Standard ML*.

And a hiatus for denotational semantics.

- Successful treatment of higher-order languages.
- How to account for effects?

Types and Programming

It was also the era of types for programming languages:

- Milner's type inference for ML.
- Reynolds's and Girard's theory of polymorphism.

And the era of types for mathematics:

- Martin-Löf: Intuitionistic Type Theory.
- Constable, Bates: NuPRL type theory and system.

We were just beginning to understand the now-famous correspondences.

Formal Propositions as Types

Formal logic (defined by rules):

$$A_1\,\mathsf{true}, \ldots, A_n\,\mathsf{true} \vdash A\,\mathsf{true}$$

Formal type systems (defined by rules):

$$x_1 : A_1, \ldots, x_n : A_n \vdash M : A$$

Formal Propositions as Types

Formal logic (defined by rules):

$$A_1 \, \mathsf{true}, \ldots, A_n \, \mathsf{true} \vdash A \, \mathsf{true}$$

Formal type systems (defined by rules):

$$x_1 : A_1, \ldots, x_n : A_n \vdash M : A$$

By now this correspondence is true by definition!

- Not an isomorphism.
- Not solely due to Curry and Howard.
- No inherent computational meaning.

What Matthias Was Saying (Says Me)

What is the point of all these rules?

The subject matter is programming, done by people!

- Start with programs.
- Define types as specifications of behavior.

Emphasize meaning (content) over protocol (form).

What Matthias Was Saying (Says Me)

What is the point of all these rules?

The subject matter is programming, done by people!

- Start with programs.
- Define types as specifications of behavior.

Emphasize meaning (content) over protocol (form).

Where have we heard this before?

Brouwerian Mathematics

Contrary to Hilbert's formalism, Brouwer proposed his intuitionism.

- Start with constructions performed by people.
- Define propositions as specifications of constructions.

Brouwerian Mathematics

Contrary to Hilbert's formalism, Brouwer proposed his intuitionism.

- Start with constructions performed by people.
- Define propositions as specifications of constructions.

Mathematics is a human activity grounded in a shared understanding of construction.

Brouwerian Mathematics

Contrary to Hilbert's formalism, Brouwer proposed his intuitionism.

- Start with constructions performed by people.
- Define propositions as specifications of constructions.

Mathematics is a human activity grounded in a shared understanding of construction.

That is, programming is a human activity grounded in a shared understanding of computation.

Form follows function: formalisms are secondary matters of convenience.

Semantic Propositions as Types

Martin-Löf developed Brouwer's ideas into a unified theory of computation and proof:

Constructive Mathematics and Computer Programming, 1982.

Semantic Propositions as Types

Martin-Löf developed Brouwer's ideas into a unified theory of computation and proof:

Constructive Mathematics and Computer Programming, 1982.

Start with programs, define types as programs that classify other programs:

- A type means $A \Downarrow A_0$ and A_0 classifies other programs.
- $M \in A$ means $M \Downarrow M_0$ and M_0 is classified by A_0.

Semantic Propositions as Types

Martin-Löf developed Brouwer's ideas into a unified theory of computation and proof:

Constructive Mathematics and Computer Programming, 1982.

Start with programs, define types as programs that classify other programs:

- A type means $A \Downarrow A_0$ and A_0 classifies other programs.
- $M \in A$ means $M \Downarrow M_0$ and M_0 is classified by A_0.

More generally, define exact equality:

- $A \doteq B$ means $A \Downarrow A_0$, $B \Downarrow B_0$, and both classify the same programs.
- $M \doteq N \in A$ means $M \Downarrow M_0$, $N \Downarrow N_0$, and are interchangeable up to A_0.

The Univalence Axiom

Voevodsky proposed the univalence axiom for formal type theory:

$$\mathsf{ua}(E) : \mathsf{Equiv}(A, B) \simeq \mathsf{Id}_{\mathsf{U}}(A, B)$$

Equivalent types are identical (equivalence $\approx$ isomorphism).

HoTT: a formal account of univalence (and higher inductives):

- Add univalence as a new axiom.
- Justified by interpreting types as simplicial sets (spatial meaning).

The Univalence Axiom

Voevodsky proposed the univalence axiom for formal type theory:

$$\mathsf{ua}(E) : \mathsf{Equiv}(A, B) \simeq \mathsf{Id}_{\mathsf{U}}(A, B)$$

Equivalent types are identical (equivalence $\approx$ isomorphism).

HoTT: a formal account of univalence (and higher inductives):

- Add univalence as a new axiom.
- Justified by interpreting types as simplicial sets (spatial meaning).

But what is the computational meaning of univalence?

$$\mathsf{J}(x.N; \mathsf{refl}_M()) \equiv N[M/x]$$
$$\mathsf{J}(x.N; \mathsf{ua}(E)) \equiv \textbf{???}$$

Cannot just add axioms to type theory!

Identities and Paths

The difficulty is that $\mathsf{Id}_A(M, N)$ is inductively defined:

- It is the least reflexive relation, universality witnessed by J.
- Defined uniformly in A!

Identities and Paths

The difficulty is that $\mathsf{Id}_A(M, N)$ is inductively defined:

- It is the least reflexive relation, universality witnessed by J.
- Defined uniformly in A!

Already there was trouble with function extensionality:

$$\mathsf{Id}_{A \to B}(F, G) \;\not\simeq\; \Pi x : A.\mathsf{Id}_B(F\,x, G\,x).$$

Univalence is nothing but an extensionality principle.

Crucially, the evidence matters ... and what is evidence depends on the type!

$$\mathrm{ext}(P) \in \mathrm{Path}_{\_.A \to B}(F, G)$$
$$\mathrm{ua}(E) \in \mathrm{Path}_{\_.\mathrm{U}}(A, B)$$

Identities and Paths

Crucially, the evidence matters ... and what is evidence depends on the type!

$$\mathrm{ext}(P) \in \mathrm{Path}_{\_.A\to B}(F, G)$$
$$\mathrm{ua}(E) \in \mathrm{Path}_{\_.\mathrm{U}}(A, B)$$

Path types are equivalent, but not equal, to Id types.

- Not enough to provide computational meaning.
- No issue for the spatial meaning (modulo equivalence).

But what are path types? And how do we compute with them?

A path, or line, is a program parameterized by a dimension variable.

$$\begin{tikzcd} M\langle 0/x\rangle\langle 0/y\rangle \arrow[r, "M\langle 0/y\rangle"] \arrow[d, "M\langle 0/x\rangle"'] \arrow[dr, phantom, "M"] & M\langle 1/x\rangle\langle 0/y\rangle \arrow[d, "M\langle 1/x\rangle"] \\ M\langle 0/x\rangle\langle 1/y\rangle \arrow[r, "M\langle 1/y\rangle"'] & M\langle 1/x\rangle\langle 1/y\rangle \end{tikzcd}$$

More generally, squares, cubes, hypercubes . . . .

Think of x, y as continuously varying over $[0, 1]$ to draw the square.

Cubical Types

Types are n-cubes: $A \;\mathsf{type}\; [x_1, \ldots, x_n]$.

- An n-cube is a line between opposing faces in n ways.
- $A \Downarrow A_0$ and A_0 specifies n-cubes as elements.

Elements of n-cubes are n-cubes of elements: $M \in A \; [x_1, \ldots, x_n]$.

- $M \Downarrow M_0$ and M_0 satisfies A_0 in all aspects.
- Coherently ... (never mind) ...

Cubical operational semantics: cubes compute their aspects.

There must be enough lines/squares/cubes of types and elements:

$$
\begin{tikzcd}
{} \arrow[r, "x"] \arrow[d, "y"'] & {} & \cdot \arrow[rr, "M"] \arrow[dd, "P"'] & & \cdot \arrow[dd, "Q"] \\
{} & & & & \\
& & \cdot \arrow[rr, dashed, "\mathsf{hcom}_A^{0 \rightsquigarrow 1}(M; y.P, y.Q)"'] & & \cdot
\end{tikzcd}
$$

(Defines $P^{-1} \cdot M \cdot Q$, combining inversion and concatenation.)

Kan Conditions

Type lines induce coercions between their end points:

$$\mathsf{coe}_{x.A}^{0 \rightsquigarrow 1}(-) \in A\langle 0/x\rangle \to A\langle 1/x\rangle \ [\ldots]$$

whenever A is a line of types:

$$A \in \mathsf{U}\ [\ldots, x]$$

By univalence, all type lines are equivalences.

Coercion uses the equivalence to transport elements from one side to the other.

That is, the end points are interchangeable in all contexts!

What Next?

Cubical proof assistants: RedPRL, RedTT.

Sterling, Favonia, Angiuli, Cavallo, Niu.

Higher-dimensional parametricity: relevalence.

Cavallo (cf, Vezossi)

At Cornell, free choice sequences, the quintessential Brouwerian concept.

Bickford, Cohen, Constable, Rahli.

(Some of) What I Learned From Matthias

There is a science of computer programming grounded in mathematics!

Operational methods are fundamental, both conceptually and pedagogically.

(Some of) What I Learned From Matthias

There is a science of computer programming grounded in mathematics!

Operational methods are fundamental, both conceptually and pedagogically.

Programming is a human activity: act as if people mattered!

Computation is about people first and foremost.

(Some of) What I Learned From Matthias

There is a science of computer programming grounded in mathematics!

Operational methods are fundamental, both conceptually and pedagogically.

Programming is a human activity: act as if people mattered!

Computation is about people first and foremost.

Stand up for what you believe in, you just may be right!

(And don't be afraid to be wrong.)

Und wenn er nicht gestorben ist, dann lebt er noch heute!

Matthias is, of course, right!

(Except for the part about the types.)

And we were not that far apart after all:

Take computation seriously!

Und wenn er nicht gestorben ist, dann lebt er noch heute!

Matthias is, of course, right!

(Except for the part about the types.)

And we were not that far apart after all:

Take computation seriously!

Thank you, Matthias, for your influence, inspiration, irritation, and friendship.

And best wishes for the next half!

Acknowledgements

Thanks to many, including

- Students: Carlo Angiuli, Evan Cavallo, Favonia, Jon Sterling.
- Colleagues: Guillaume Brunerie, Thierry Coquand, Simon Huber, Dan Licata, Anders Mörtberg.
- Inspiration: Robert Constable, Per Martin-Löf, Vladimir Voevodsky†.

Supported by AFOSR MURI FA9550-15-1-0053, Tristan Nguyen, PM.

References

Carlo Angiuli and Robert Harper. Meaning explanations at higher dimension. *Indagationes Mathematicae*, 29:135–149, 2018. Virtual Special Issue – L.E.J. Brouwer after 50 years.

Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. Cartesian cubical type theory. (Unpublished manuscript), December 2017.

Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. Cartesian cubical computational type theory: Constructive reasoning with paths and equalities. In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, Leibniz International Proceedings in Informatics, Schloss Dagstuhl - Leibniz-Zentrum in Informatics, September 2018.

Evan Cavallo and Robert Harper. Higher inductive types in cubical computational type theory. In *ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2019)*, January 2019. (To appear).

Jon Sterling and et al. RedPRL – the People's Refinement Logic. `http://www.redprl.org/`, 2018.