Meta-MeTTa: an operational semantics for MeTTa

Lucius Gregory Meredith[1]
and Ben Goertzel[2]

[1] CEO, F1R3FLY.io 9336 California Ave SW, Seattle, WA 98103, USA
`f1r3fly.ceo.com`
[2] CEO, SingularityNet
`ben@singularitynet.io`

Abstract. We make the case for a reflective multi-agent formalism.

1 Towards a common language

Three of the most successful branches of scientific discourse all agree on the shape of a model adequate for expressing and effecting computation. Physics, computer science, and mathematics all use the same standard shape. A model adequate for computation comes with an algebra of states and "laws of motion."
One paradigmatic example from physics is Hilbert spaces and the Schroedinger equation. In computer science and mathematics the algebra of states is further broken down into a monad (the free algebra of states) and an algebra of the monad recorded as some equations on the free algebra.
Computer science represents laws of motion, aka state transitions, as rewrite rules exploiting the structure of states to determine transitions to new states. Mainstream mathematics is a more recognizable generalization of physics, coding state transitions, aka behavior, via morphisms (including automorphisms) between state spaces.
But all three agree to a high degree of specificity on what ingredients go into a formal presentation adequate for effecting computation.

1.1 Examples from computer science

λ-calculus TBD

π-calculus TBD

rho-calculus

The JVM TBD

2 A presentation of the semantics of MeTTa

A presentation of the semantics of MeTTa must therefore provide a monad describing the algebra of states, a structural equivalence quotienting the algebra of states, and some rewrite rules describing state transitions. Such a description is the minimal description that meets the standard for describing models of computation. Note that to present such a description requires at least that much expressive power in the system used to formalize the presentation. That is, the system used to present a model of computation is itself a model of computation admitting a presentation in terms of an algebra of states and some rewrites. This is why a meta-circular evaluator is a perfectly legitimate presentation. That is, a presentation of MeTTa's semantics in MeTTa is perfectly legitimate. Meta-circular presentations are more difficult to unpack, which is why such presentations are typically eschewed, but they are admissible. In fact, a meta-circular evaluator may be the most pure form of presentation.
But, this fact has an important consequence. No model that is at least Turing complete can be "lower level" than any other.

2.1 Rationale for such a presentation

The rationale for such a presentation is not simply that this is the way it's done. Instead, the benefits include

- an effective (if undecidable) notion of program equality;
- an independent specification allowing implementations;
- meta-level computation, including type checking, model checking, macros, computational reflection, etc.

2.2 MeTTa Operational Semantics

Algebra of States

Expressions

$$\begin{aligned} Expr ::=\ & (Expr\ [Expr]) \\ |\ & \{Expr\ [Expr]\} \\ |\ & (Expr\ |\ [Receipt]\ .\ [Expr]) \\ |\ & \{Expr\ |\ [Receipt]\ .\ [Expr]\} \\ |\ & Atom \end{aligned}$$

Expression sequences

$$\begin{aligned} [Expr] ::=\ & \epsilon \\ |\ & Expr \\ |\ & Expr``\text{``}[Expr] \end{aligned}$$

Bindings

$$\begin{aligned} Receipt ::=\ & ReceiptLinearImpl \\ |\ & ReceiptRepeatedImpl \\ |\ & ReceiptPeekImpl \end{aligned}$$

$$\begin{aligned} [Receipt] ::=\ & Receipt \\ |\ & Receipt;[Receipt] \end{aligned}$$

$$\begin{aligned} ReceiptLinearImpl ::=\ & [LinearBind] \\ LinearBind ::=\ & [Name]NameRemainder{\leftarrow}AtomSource \end{aligned}$$

$$\begin{aligned} [LinearBind] ::=\ & LinearBind \\ |\ & LinearBind\&[LinearBind] \end{aligned}$$

$$\begin{aligned} AtomSource ::=\ & Name \\ |\ & Name?! \\ |\ & Name!?([Expr]) \\ ReceiptRepeatedImpl ::=\ & [RepeatedBind] \\ RepeatedBind ::=\ & [Name]NameRemainder{\Leftarrow}Atom \end{aligned}$$

$$\begin{aligned} [RepeatedBind] ::=\ & RepeatedBind \\ |\ & RepeatedBind\&[RepeatedBind] \\ ReceiptPeekImpl ::=\ & [PeekBind] \\ PeekBind ::=\ & [Name]NameRemainder{\leftharpoondown}Atom \end{aligned}$$

$$\begin{aligned} [PeekBind] &::= PeekBind \\ &\mid\ PeekBind\&[PeekBind] \end{aligned}$$

$$\begin{aligned} ExprRemainder &::= ...ExprVar \\ &\mid\ \epsilon \\ NameRemainder &::= ...@ExprVar \\ &\mid\ \epsilon \end{aligned}$$

Literals and builtins

$$\begin{aligned} Atom &::= Ground \\ &\mid\ Builtin \\ &\mid\ Var \\ Name &::= \_ \\ &\mid\ Var \\ &\mid\ @Expr \end{aligned}$$

$$\begin{aligned} [Name] &::= \epsilon \\ &\mid\ Name \\ &\mid\ Name,[Name] \\ BoolLiteral &::= \mathsf{true} \\ &\mid\ \mathsf{false} \\ Ground &::= BoolLiteral \\ &\mid\ LongLiteral \\ &\mid\ StringLiteral \\ &\mid\ UriLiteral \\ Builtin &::= ::= \\ &\mid\ = \\ &\mid\ : \\ ExprVar &::= \_ \\ &\mid\ Var \end{aligned}$$

Rewrite Rules

Query

$$\frac{\sigma_i = \mathsf{unify}(t_i, t')}{\langle !t', \ldots\rangle\langle \ldots, (= t_1 u_1), \ldots, (= t_k u_k), \ldots\rangle\langle \ldots\rangle \rightarrow \langle \ldots\rangle\langle \ldots, (= t_1 u_1), \ldots, (= t_k u_k), \ldots\rangle\langle \ldots, u_i\sigma_i\rangle}$$

References

1. Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. In *POPL*, pages 33–44. ACM, 2002.
2. Martín Abadi and Bruno Blanchet. Secrecy types for asymmetric communication. *Theor. Comput. Sci.*, 298(3):387–415, 2003.
3. Samson Abramsky. Algorithmic game semantics and static analysis. In *SAS*, volume 3672 of *Lecture Notes in Computer Science*, page 1. Springer, 2005.

4. Luís Caires. Behavioral and spatial observations in a logic for the pi-calculus. In *FoSSaCS*, pages 72–89, 2004.
5. Luis Caires. Spatial logic model checker, Nov 2004.
6. Luís Caires and Luca Cardelli. A spatial logic for concurrency (part I). *Inf. Comput.*, 186(2):194–235, 2003.
7. Luís Caires and Luca Cardelli. A spatial logic for concurrency - II. *Theor. Comput. Sci.*, 322(3):517–565, 2004.
8. Luca Cardelli. Brane calculi. In *CMSB*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–278. Springer, 2004.
9. Vincent Danos and Cosimo Laneve. Core formal molecular biology. In *ESOP*, volume 2618 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2003.
10. Cédric Fournet, Fabrice Le Fessant, Luc Maranget, and Alan Schmitt. Jocaml: A language for concurrent distributed and mobile programming. In *Advanced Functional Programming*, volume 2638 of *Lecture Notes in Computer Science*, pages 129–158. Springer, 2002.
11. Cédric Fournet, Georges Gontheir, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR 1996*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421. Springer-Verlag, 1996.
12. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
13. Allen L. Brown Jr., Cosimo Laneve, and L. Gregory Meredith. Piduce: A process calculus with native XML datatypes. In *EPEW/WS-FM*, volume 3670 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2005.
14. Cosimo Laneve and Gianluigi Zavattaro. Foundations of web transactions. In *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 282–298. Springer, 2005.
15. Greg Meredith. Documents as processes: A unification of the entire web service stack. In *WISE*, pages 17–20. IEEE Computer Society, 2003.
16. L. Gregory Meredith and Matthias Radestock. Namespace logic: A logic for a reflective higher-order calculus. In *TGC* [**?**], pages 353–369.
17. L. Gregory Meredith and Matthias Radestock. A reflective higher-order calculus. *Electr. Notes Theor. Comput. Sci.*, 141(5):49–67, 2005.
18. Lucius Meredith, Jan 2017.
19. Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.
20. Robin Milner. Elements of interaction - turing award lecture. *Commun. ACM*, 36(1):78–89, 1993.
21. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.
22. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, II. *Inf. Comput.*, 100(1):41–77, 1992.
23. Corrado Priami, Aviv Regev, Ehud Shapiro, and William Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Inf. Process. Lett.*, 80(1):25–31, 2001.
24. Aviv Regev, William Silverman, and Ehud Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Pacific Symposium on Biocomputing*, pages 459–470, 2001.
25. Davide Sangiorgi. Beyond bisimulation: The "up-to" techniques. In *FMCO*, volume 4111 of *Lecture Notes in Computer Science*, pages 161–171. Springer, 2005.
26. Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4):15:1–15:41, 2009.
27. Davide Sangiorgi and Robin Milner. The problem of "weak bisimulation up to". In *CONCUR*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1992.
28. Peter Sewell, Pawel T. Wojciechowski, and Asis Unyapoth. Nomadic pict: Programming languages, communication infrastructure overlays, and semantics for mobile computation. *ACM Trans. Program. Lang. Syst.*, 32(4):12:1–12:63, 2010.