The MeTTa calculus

L.G. Meredith[1]

CEO, F1R3FLY.io 9336 California Ave SW, Seattle, WA 98103, USA,
ceo@f1r3fly.io

Abstract. We describe a core calculus that captures the operational semantics of the language MeTTa.

1 Introduction and motivation

In [1] we described a register machine based operational semantics for MeTTa. While this has some utility for proving the correctness of compilers, it is not conducive for reasoning about types and other more abstract aspects of MeTTa-based computation. Here we present a calculus, together with a efficient implementation and prove the correctness of the implementation.

Additionally, we use the OSLF algorithm developed by Meredith, Stay, and Williams to calculate a spatial-behavioral type system for MeTTa. Further, we adapt a well known procedure for proving the termination of rewrites to provide a token-based security model for the calculus and implementation. Beyond that we derive versions of fuzzy, stochastic, and quantum execution modes, automatically.

In general, presenting MeTTa as a graph structured lambda theory, otherwise known as a structured operational semantics, not only has the benefit that implementation follows the correct-by-construction methodology, but may be used to automatically derive and extend MeTTa with much needed features for programming real applications in a distributed and decentralized setting, as well as supporting well established programming paradigms, such as semi-colon delimited sequential assignment programs.

2 A symmetric reflective higher order concurrent calculus with backchaining

Note, in the following spec we use $[e]$ to denote a space-delimited finite sequence of e's; and we use $[e]_{seq}$ to denote a comma-delimited finite sequence of e's.

PROCESS
$$P, Q ::= 0 \mid G \mid \mathsf{for}(t \leftarrow x)P \mid x?P \mid *x \mid P|Q$$

NAME
$$x, y ::= @P$$

TERM
$$t, u ::= atom \mid ([t])$$

ATOM
$$atom ::= x \mid P$$

GROUND
$$G ::= \mathsf{Bool} \mid \mathsf{String} \mid \mathsf{Int} \mid C$$

COLLECTION
$$C ::= \mathtt{[}[t]_{seq}\mathtt{]} \mid \mathtt{(}t\ \mathtt{,}[t]_{seq}\mathtt{)} \mid \mathsf{Set}\mathtt{(}[t]_{seq}\mathtt{)} \mid \{[t\mathtt{:}t]\}$$

$$\text{EQUIV}\quad P|\mathsf{0} \equiv P \quad P|Q \equiv Q|P \quad P|(Q|R) \equiv (P|Q)\mathsf{R}$$

$$\text{ALPHA}\quad \frac{\mathsf{occurs}(t, y)}{\mathsf{for}(t\text{<->}x)P \equiv \mathsf{for}(t\{z/y\}\text{<->}x)(P\{z/y\})\ \mathsf{if} z \notin \mathsf{FN}(P)}$$

$$\text{COMM}\quad \frac{\sigma = \mathsf{unify}(t, u)}{\mathsf{for}(t\text{<->}x)P \mid \mathsf{for}(u\text{<->}x)Q \to P\dot{\sigma}|Q\dot{\sigma}}$$

$$\text{PAR}\quad \frac{P \to P'}{P|Q \to P'|Q} \qquad \text{EQUIV}\quad \frac{P \equiv P' \qquad P' \to Q' \qquad Q' \equiv Q}{P \to Q}$$

$$\text{REFL}\quad \frac{P \to P'}{x?P \to x!(P')}$$

where $\dot{\sigma}$ denotes the substitution that replaces all variable to process bindings with variable to name bindings. Thus, $\{P/x\} = \{@P/x\}$.

We denote the collection of process states generated (resp. recognized) by this grammar by Proc. Likewise, we denote the collection of spaces (aka channels or names) by @Proc.

2.1 Intuitive mapping to MeTTa

MeTTa language	MeTTa calculus
(addAtom *space term*)	for(*term* <-> *space*)0
(remAtom *space term*)	for(*term* <-> *space*)0
(? *space term*)	for(*term* <-> *space*)0

The key insight is the same one that Google and other Web2.0 companies made decades ago: $folders = tagging$. In other words, rather than actually building a container data structure constituting a "space" (as in AtomSpace), we merely tag atoms with the space(s) they occupy. This shift in perspective allows us to use the same construct (a kind of tagging) for adding atoms to a space; removing atoms from a space; and, querying for atoms in a space that match a given pattern.

Likewise a rewrite rule is a continuation $t\text{<->}\{\ P\ \}$. When it is tagged with a space, say x, i.e. $\mathsf{for}(t\text{<->}x)P$, may be thought of as added to the space.

Under this view, a space is equated with all the atoms (and rules) that have been tagged as in the space. That is, we may define a function space : @Proc → Proc by

$\mathsf{space}(x) = \Pi_i\ \mathsf{for}(t_i \leftrightarrow x)P_i$. Once we make this definition we note we have two interlated algebras of considerable additional interest. One is the algebra of spaces as tags, which is isomorphic to the algebra of process states, and another is the algebra of the spaces as collections of atoms (and rules).

The first affords the ability to programmatically define tags and filter on tags. The second affords the ability to reason about, filter, and traverse spaces-qua-collections. The fact that these two are interrelated means that spaces can be nested, or more generally composed in a variety of interesting ways. Indeed, some collections of spaces may be mutually recursively defined!

3 Some useful features

3.1 Replication and freshness

PROCESS

$$P, Q ::= \ldots \mid\ !P \mid\ \mathsf{new}\ x\ \mathsf{in}\ \{\ P\ \}$$

In the core calculus, when two terms rendezvous at a space $(\mathsf{for}(t \leftrightarrow x)P \mid \mathsf{for}(u \leftrightarrow x)Q)$ they are *consumed* and replaced by their continuations $(P\dot{\sigma}|Q\dot{\sigma})$. It is frequently useful in programming applications to leave one or the other in place. Thus, when $!\mathsf{for}(t \leftrightarrow x)P$ rendezvous with $\mathsf{for}(u \leftrightarrow x)Q$ it reduces to $!\mathsf{for}(t \leftrightarrow x)P|P\dot{\sigma}|Q\dot{\sigma}$.

Likewise, in programming applications it is often useful to guarantee that computations rendezvous in a private space. The state denoted by $\mathsf{new}\ x\ \mathsf{in}\ \{\ P\ \}$ guarantees that x is private in the scope P. Therefore, $\mathsf{new}\ x\ \mathsf{in}\ \{\ \mathsf{for}(t \leftrightarrow x)P \mid \mathsf{for}(u \leftrightarrow x)Q\}$ guarantees that the rendezvous happens in a private space.

3.2 Fork-join concurrency

This next bit of syntactic sugar illustrates the value of the for-comprehension. Specifically, it facilitates the introduction of fork-join concurrency, which is predominant in human decision-making processes. The following syntax should be read as an expansion of the core calculus, *replacing* the much simpler for-comprehension with a more articulated one.

PROCESS

$$P, Q ::= \ldots \mid\ \mathsf{for}([\mathsf{Join}])P$$

JOINS

$$[\mathsf{Join}] ::= \mathsf{Join} \mid\ \mathsf{Join};[\mathsf{Join}]$$

JOIN

$$\mathsf{Join} ::= [\mathsf{Query}]$$

QUERIES

$$[\mathsf{Query}] ::= \mathsf{Query} \mid\ \mathsf{Query}\&[\mathsf{Query}]$$

QUERY

$$\mathsf{Query} ::= t \leftrightarrow x$$

In case the BNF is a little opaque, here is the template.

```
for(
  y11 <-> x11 & ... & ym1 <-> xm1 ; // received in any order,
     ... ; // but all received before the next row
  y1n <-> x1n & ... & ymn <-> xmn
){ P }
```

As mentioned previously, the predominant pattern of human decision making processes (such as loan approval processes, or academic paper reviews) involve fork-join concurrency. The syntactic sugar provided here certainly supports that kind of coordination amongst processes. For example,

```
for(
    // received in any order
    true <-> reviewer1 & true <-> reviewer2 & true <-> reviewer3
    ){
      // acceptance notification and publication process
      P
    }
```

Yet, it affords much more sophisticated control than this, while also providing a programming paradigm that is familiar to modern programmers, namely semi-colon delimited sequential assignment-based programming.

4 From calculus to efficient implementation

TBD

5 Tokenized security

SECURITY-TOKENS
$$T ::= () \mid s \mid T : T$$

SECURED-PROCESSES
$$S ::= \{P\}_s \mid T \mid S|S$$

MULTI-PARTY-SIGS
$$s ::= () \mid \mathsf{hash}(<signature>) \mid s\&s$$

where $\{P\}_s$ is a process signed by a digital signature.

COMM-COSIGNED-PAR-EXTERNAL-SEQUENTIAL

$$\frac{\sigma = \mathsf{unify}(t, u)}{\{\mathsf{for}(t \leftrightarrow x)P\}_{s_1} \mid \{\mathsf{for}(u \leftrightarrow x)P\}_{s_2} \mid s_1 \& s_2 : T \to \{P\dot{\sigma} | Q\dot{\sigma}\}_{s_1 \& s_2} \mid T}$$

COMM-COSIGNED-PAR-EXTERNAL-CONCURRENT

$$\frac{\sigma = \mathsf{unify}(t, u)}{\{\mathsf{for}(t \leftrightarrow x)P\}_{s_1} \mid \{\mathsf{for}(u \leftrightarrow x)P\}_{s_2} \mid s_1 : T_1 \mid s_2 : T_1 \to \{P\dot{\sigma} | Q\dot{\sigma}\}_{s_1 \& s_2} \mid T_1 \mid T_2}$$

COMM-SIGNED

$$\frac{P \to P'}{\{P\}_s \mid s : T \to \{P'\}_s \mid T}$$

COMM-COSIGNED-PAR-INTERNAL

$$\frac{P \to P'}{\{P\}_{s_1 \& s_2} \mid s_1 : T_1 \mid s_2 : T_2 \to \{P'\}_s \mid T_1 \mid T_2}$$

6 Spatial-behavioral types

TBD

7 Stochastic and quantum execution

7.1 Stochastic execution

TBD

7.2 Quantum execution

TBD

8 Conclusions and future research

TBD

Acknowledgments. The author wishes to thank SingularityNet.io for their support of this work.

References

1. Adam Vandervorst Lucius Gregory Meredith, Ben Goertzel. Meta-metta: An operational semantics for metta, 2023.