# AS-MOSES

Nil Geisweiller

SingularityNetathon 2018

OpenCog Foundation

# MOSES: Status

MOSES is a program learner

- MOSES: Meta-Optimization Semantic Evolutionary Search (Moshe Looks)
- C++ version
- Learn Combo programs
- To be ported for the AtomSpace

# MOSES: Recall

What makes MOSES special

- Reduction in normal form

$$f(x) = 1 * x + 0 \quad \Rightarrow \quad f(x) = x$$

  - Avoid over-representation
  - Increase syntax vs semantics correlation
  - Simplify subsequent evolution

# MOSES: Recall

What makes MOSES special

- Reduction in normal form

$$f(x) = 1 * x + 0 \quad \Rightarrow \quad f(x) = x$$

    - Avoid over-representation
    - Increase syntax vs semantics correlation
    - Simplify subsequent evolution
- Deme management
    - Islands of diverse program subspaces
    - “Clever” representation building

# MOSES: Recall

What makes MOSES special

- Reduction in normal form

$$f(x) = 1 * x + 0 \quad \Rightarrow \quad f(x) = x$$

    - Avoid over-representation
    - Increase syntax vs semantics correlation
    - Simplify subsequent evolution
- Deme management
    - Islands of diverse program subspaces
    - “Clever” representation building
- Optimization
    - Attempt to learn the fitness landscape
    - In practice Stochastic Hillclimbing + Crossover

# AS-MOSES: Port to the AtomSpace

Why porting MOSES to the AtomSpace?

Synergies between MOSES and the rest of OpenCog

- Atomese fitness function
- Atomese candidate programs
- Search in the AtomSpace
- Integrate background knowledge
- Meta-learning

# AS-MOSES: Atomese

Program example:

$$f(x, y) = x + 2 * y$$

Atomese:

```
(Lambda
  (VariableList
    (Variable "x")
    (Variable "y")
  (Plus
    (Variable "x")
    (Times
      (Number 2)
      (Variable "y"))))
```

# AS-MOSES: Deme Representation

Representation:

$$f(x) = [-1, 1] * x + [0, 0.5, 1]$$

Atomese:

```
(Quote
  (Lambda
    (Variable "$x")
    (Plus
      (Times
        (Unquote (Variable "$k0"))
        (Variable "x"))
      (Unquote (Variable "$k1")))))
```

$k0$ and $k1$ are the knob variables

# AS-MOSES: Deme Representation

Generate all candidates with the following Atomese program:

```
(Put
  (VariableList
    (Variable "$k0")
    (Variable "$k1"))
  (Quote
    (Lambda
      (Variable "$x")
      (Plus
        (Times
          (Unquote (Variable "$k0"))
          (Variable "$x"))
        (Unquote (Variable "$k1")))))
  (Set
    (List (Number -1) (Number 0))
    (List (Number -1) (Number 0.5))
    (List (Number -1) (Number 1))
    (List (Number 1) (Number 0))
    (List (Number 1) (Number 0.5))
    (List (Number 1) (Number 2))))
```

⇒

```
(Lambda
  (Variable "$x")
  (Plus
    (Times
      (Number -1)
      (Variable "$x"))
    (Number 0)))

(Lambda
  (Variable "$x")
  (Plus
    (Times
      (Number -1)
      (Variable "$x"))
    (Number 0.5)))

(Lambda
  (Variable "$x")
  (Plus
    (Times
      (Number -1)
      (Variable "$x"))
    (Number 1)))
```

```
(Lambda
  (Variable "$x")
  (Plus
    (Times
      (Number 1)
      (Variable "$x"))
    (Number 0)))

(Lambda
  (Variable "$x")
  (Plus
    (Times
      (Number 1)
      (Variable "$x"))
    (Number 0.5)))

(Lambda
  (Variable "$x")
  (Plus
    (Times
      (Number 1)
      (Variable "$x"))
    (Number 1)))
```

# AS-MOSES: Reduction

## Axiomatize Atomese Reduction

For instance

```
(Evaluation (stv 1 1)
  (Predicate "reduce-to")
  (List
    (Lambda
      (Variable "$x")
      (Plus
        (Times
          (Number 1)
          (Variable "$x"))
        (Number 0)))
    (Lambda
      (Variable "$x")
      (Variable "$x"))
```

means

$$f(x) = 1 * x + 0$$

reduces to

$$f(x) = x$$

# AS-MOSES: Optimization

- Goal: Incorporate background knowledge (including meta-learning) in the optimization process

# AS-MOSES: Optimization

- Goal: Incorporate background knowledge (including meta-learning) in the optimization process
- Suggested solution: See optimization as reasoning process. Possibly dedicated policies for efficiency.
    - EDAs: Probably trivial
    - Hillclimbing: Less trivial but still

# AS-MOSES: Optimization, Hillclimbing Example

Hillclimbing Axioms:

1. Candidates with similar knob settings tend to be syntactically similar

   ```
   Implication (stv 0.8 0.2)
     Predicate "similar-knob-settings"
     Predicate "similar-syntax"
   ```

2. Syntactically similar candidates tend to be loosely semantically similar

   ```
   Implication (stv 0.4 0.01)
     Predicate "similar-syntax"
     Predicate "similar-semantics"
   ```

3. Semantically similar candidates tend to have similar fitnesses

   ```
   Implication (stv 0.6 0.1)
     Predicate "similar-semantics"
     Predicate "similar-fitness"
   ```

4. If candidate P1 and P2 have similar fitnesses, P2's fitness is close to P1's fitness

   ```
   ImplicationScope (stv 1 1)
     $P1, $P2
     And
       Evaluation
         Predicate "similar-fitness"
         List
           $P1
           $P2
       Evaluation
         Predicate "fitness"
         $P1
     Evaluation
       Predicate "fitness"
       $P2
   ```

# AS-MOSES: Optimization

URE query:

```
Evaluation
  Predicate "MOSES:fitness"
  $X
```

- URE fitness: maximize strength and confidence of the query
- Chain axioms 1 to 4, using deduction, fuzzy conjunction and conditional instantiation to explore neighborhood
- Once a better candidate is found, the URE fitness will be incentivized to re-chain axioms 1 to 4 with that better candidate.
- We can help the URE!

# AS-MOSES: Architecture

Open to discussion...

# AS-MOSES: Architecture

Open to discussion...

Personal suggestion:

- Wrap existing MOSES C++ components in Scheme/Atomese so it appears as if AS-MOSES is an Atomese program
- Progressively infuse reasoning