Automated Program Learning

MOSES

Nil Geisweiller

Novamente LLC

Xiamen University
AGI Summer School 2009

Outline

What is MOSES?

MOSES (*Meta-Optimizing Semantic Evolutionary Search*)

Evolutionary program learning, PhD Moshe Looks. 

1. Search programs that maximize the fitness function

What is MOSES?

MOSES (*Meta-Optimizing Semantic Evolutionary Search*)

Evolutionary program learning, PhD Moshe Looks. 

1. Search programs that maximize the fitness function
2. Take advantage of program semantics and program space topology

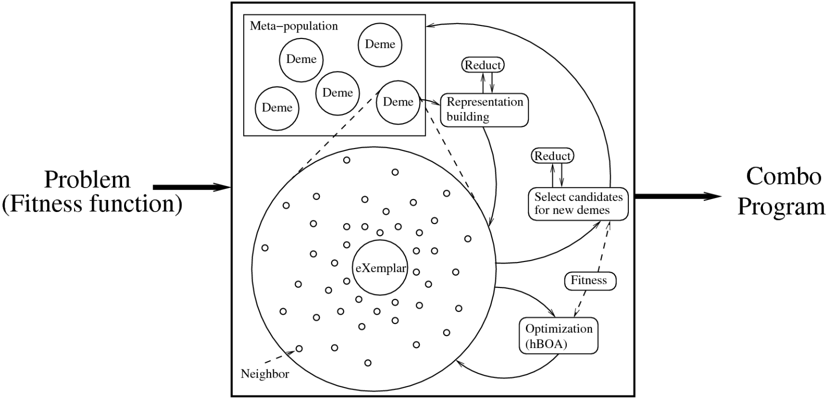
What is MOSES?

MOSES (*Meta-Optimizing Semantic Evolutionary Search*)

Evolutionary program learning, PhD Moshe Looks. 

1. Search programs that maximize the fitness function
2. Take advantage of program semantics and program space topology
3. Attempt to discover fitness landscape regularities to speed up the search

How it works?


Problem
(Fitness function)
Meta–population
Deme
Deme
Deme
Deme
Deme
Reduct
Representation building
Reduct
Select candidates for new demes
eXemplar
Fitness
Optimization (hBOA)
Neighbor
Combo
Program

How it works?

1. Reduction in normal form
   - minimize over-representation
   - improve syntactic vs semantic distance correlation
   - simplify or even improve structure

How it works?

1. Reduction in normal form
   - minimize over-representation
   - improve syntactic vs semantic distance correlation
   - simplify or even improve structure
2. Population building, representation-building defines the deme's neighborhood

How it works?

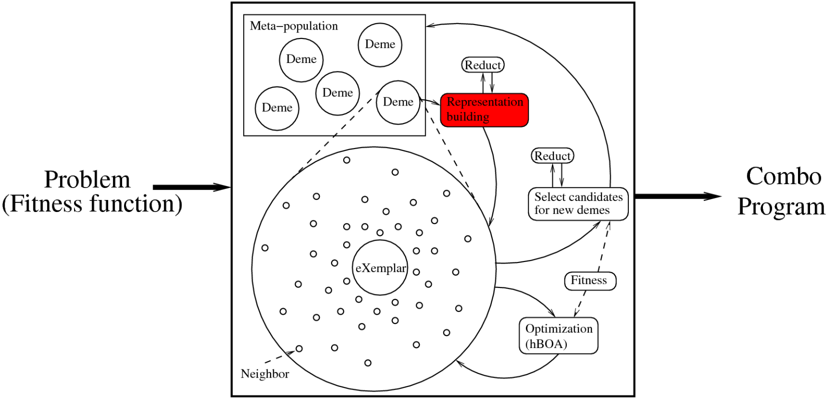
1. Reduction in normal form
   - minimize over-representation
   - improve syntactic vs semantic distance correlation
   - simplify or even improve structure
2. Population building, representation-building defines the deme's neighborhood
3. Optimization, find the best candidates in the deme's neighborhood. Learn how to differentiate good vs bad programs and bias the search accordingly
   - hBOA
   - Building-Block Hill Climbing (under development)

How it works?

1. Reduction in normal form
   - minimize over-representation
   - improve syntactic vs semantic distance correlation
   - simplify or even improve structure
2. Population building, representation-building defines the deme's neighborhood
3. Optimization, find the best candidates in the deme's neighborhood. Learn how to differentiate good vs bad programs and bias the search accordingly
   - hBOA
   - Building-Block Hill Climbing (under development)
4. Deme management
   - Set of demes, meta-population
   - Diversity, preserving interesting demes

Outline

Representation-building: Build a deme's population



Building string of knobs

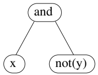
Population of a Deme

Centered around an exemplar, each neighbor is a variation of that exemplar according to the representation-building, a string of knobs.

`and(#1 not(#2))` $\Rightarrow$

Building string of knobs

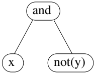
Domain specific rules to create knobs, example in the Boolean domain with `and(x not(y))`:



Building string of knobs

Domain specific rules to create knobs, example in the Boolean domain with `and(x not(y))`:

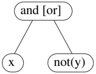
1. Under every junctor, $\forall$ v not already sibling, add [$\emptyset$, v, $not(v)$]



Building string of knobs

Domain specific rules to create knobs, example in the Boolean domain with `and(x not(y))`:

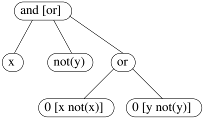
1. Under every junctor, $\forall$ v not already sibling, add [$\emptyset$, v, $not(v)$]
2. Any junctor can be flipped



Building string of knobs

Domain specific rules to create knobs, example in the Boolean domain with `and(x not(y))`:

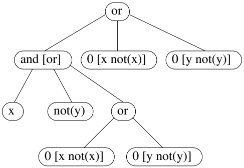
1. Under every junctor, $\forall$ v not already sibling, add [$\emptyset$, v, $not(v)$]
2. Any junctor can be flipped
3. Under every junctor add oposite junctor + children



Building string of knobs

Domain specific rules to create knobs, example in the Boolean domain with `and(x not(y))`:

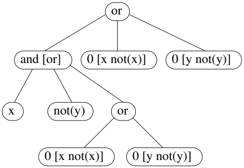
1. Under every junctor, $\forall\ v$ not already sibling, add $[\emptyset, v, not(v)]$
2. Any junctor can be flipped
3. Under every junctor add oposite junctor + children
4. Insert an oposite junctor above the root + children



Building string of knobs

Domain specific rules to create knobs, example in the Boolean domain with `and(x not(y))`:

1. Under every junctor, $\forall$ v not already sibling, add $[\emptyset, v, not(v)]$
2. Any junctor can be flipped
3. Under every junctor add oposite junctor + children
4. Insert an oposite junctor above the root + children
5. And a few more...



Building string of knobs

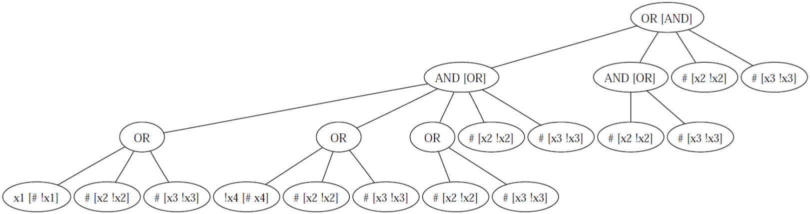


Figure: Built knobs for $\mathit{and}(x_1\,\mathit{not}(x_4))$, *extracted from Moshe's PhD*.

Optimization problem takes place on the knob string

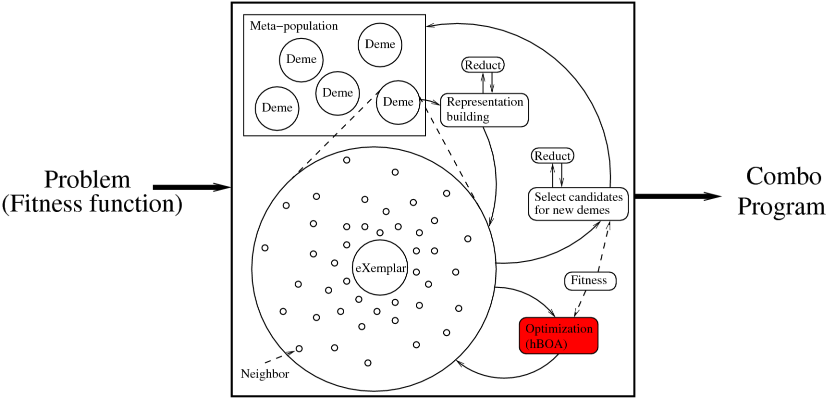
For example:

knob setting	combo (reduced)	distance
(and, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$)	and(x not(y))	0
(or, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$)	or(x not(y))	1
(and, $\emptyset$, $\emptyset$, not(x), $\emptyset$)	or(not(x) not(y))	1
(or, $\emptyset$, $\emptyset$, not(x), $\emptyset$)	true	2

Outline

1. Introduction
2. Representation-Building
3. Optimization
4. Deme management
5. Demo...
6. Conclusion

Optimization: Find the best candidates inside a deme



MOSES' Optimization algorithms

1. hBOA, multivariate model-building (not yet ported to the OpenCog version, univariate model-building instead)

MOSES' Optimization algorithms

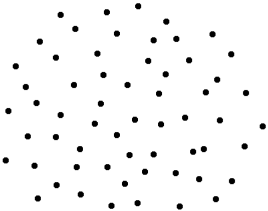
1. hBOA, multivariate model-building (not yet ported to the OpenCog version, univariate model-building instead)
2. Hill-Climbing

MOSES' Optimization algorithms

1. hBOA, multivariate model-building (not yet ported to the OpenCog version, univariate model-building instead)
2. Hill-Climbing
3. Building-Block Hill-Climbing (being ported to the OpenCog version)

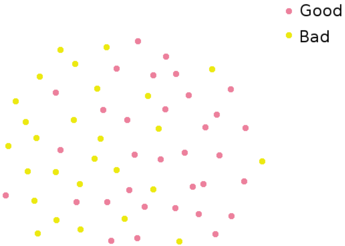
Hierarchical Bayesian Optimization Algorithm (hBOA)

Candidate	*score*
00001010	0.2
00011010	0.6
01000100	0.01
00110110	0.5
10010010	0.6
⋮	⋮



Hierarchical Bayesian Optimization Algorithm (hBOA)

Candidate	*score*
00001010	0.2
00011010	0.6
01000100	0.01
00110110	0.5
10010010	0.6
⋮	⋮



- Split the population in good vs bad candidates

Hierarchical Bayesian Optimization Algorithm (hBOA)

Candidate	*score*
00001010	0.2
00011010	0.6
01000100	0.01
00110110	0.5
10010010	0.6
⋮	⋮

- Split the population in good vs bad candidates
- Learn a classifier, but not too strict or incorrect

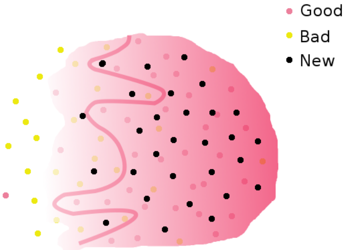
Hierarchical Bayesian Optimization Algorithm (hBOA)

Candidate	*score*
00001010	0.2
00011010	0.6
01000100	0.01
00110110	0.5
10010010	0.6
⋮	⋮

- Split the population in good vs bad candidates
- Learn a classifier, but not too strict or incorrect
- Probabilistic classifier, distribution of good candidates

Hierarchical Bayesian Optimization Algorithm (hBOA)

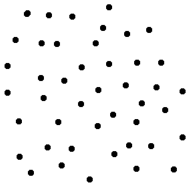
Candidate	*score*
00001010	0.2
00011010	0.6
01000100	0.01
00110110	0.5
10010010	0.6
⋮	⋮



- Split the population in good vs bad candidates
- Learn a classifier, but not too strict or incorrect
- Probabilistic classifier, distribution of good candidates
- Sample new candidates according to the distribution

Hierarchical Bayesian Optimization Algorithm (hBOA)

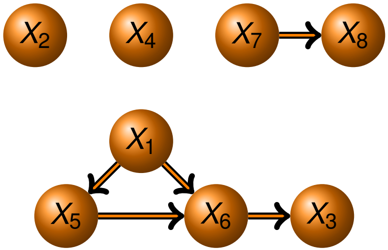
Candidate	*score*
10011010	0.5
00111011	0.6
11111001	0.7
00111010	0.4
10000001	0.1
⋮	⋮



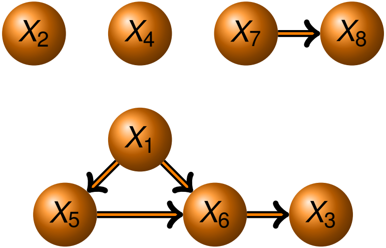
- Split the population in good vs bad candidates
- Learn a classifier, but not too strict or incorrect
- Probabilistic classifier, distribution of good candidates
- Sample new candidates according to the distribution
- Repeat on the new population

Distribution of Good Candidates: Bayesian Network

Candidate (X_i) i : 12345678	*score*
00001010	0.2
00011010	0.6
01000100	0.01
00110110	0.5
10010010	0.6
⋮	⋮

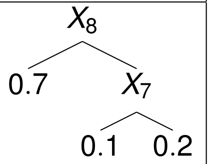


Decomposing the problem into sub-problems



Conditional Probability with Decision Tree

Marginal	*Prob*
$P(X_1 = 1)$	0.3
$P(X_2 = 1)$	0.05
$P(X_4 = 1)$	0.9
$P(X_7 = 1)$	0.88

Conditional	*Prob*
$P(X_8 \| X_7)$	
$P(X_5 \| X_1)$	...
$P(X_6 \| X_1, X_5)$	...
$P(X_3 \| X_6)$	...

In OpenCog for the moment only univariate

Univariate

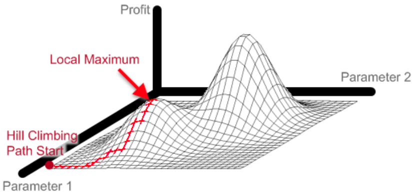
Only marginal probabilities

Marginal	*Prob*
$P(X_1 = 1)$	0.3
$P(X_2 = 1)$	0.05
$P(X_3 = 1)$	0.2
$P(X_4 = 1)$	0.5
$P(X_5 = 1)$	0.4
$P(X_6 = 1)$	0.7
$P(X_7 = 1)$	0.88
$P(X_8 = 1)$	0.1

Hill-Climbing, Building-Block Hill-Climbing

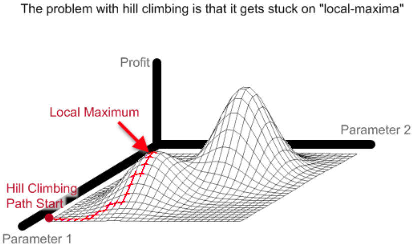
- Hill-Climbing

The problem with hill climbing is that it gets stuck on "local-maxima"



Hill-Climbing, Building-Block Hill-Climbing

- Hill-Climbing

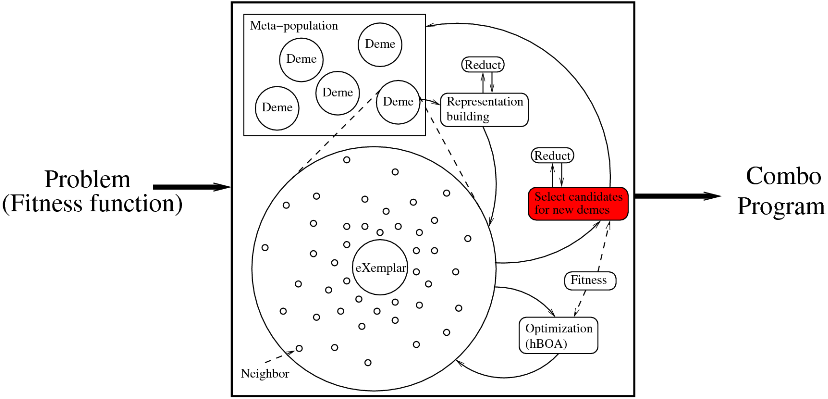


- Building-Block Hill-Climbing $\Rightarrow$ Redefine neighborhood to take short-cuts.

Outline

1. Introduction
2. Representation-Building
3. Optimization
4. **Deme management**
5. Demo...
6. Conclusion

Select Candidates for Future Demes



Preserving Diversity, candidates that behave differently and non-dominated

Preserving Diversity, candidates that behave differently and non-dominated

Preserving Diversity, candidates that behave differently and non-dominated

Preserving Diversity, candidates that behave differently and non-dominated

- Neither one dominates the other

Preserving Diversity, candidates that behave differently and non-dominated

- Neither one dominates the other
- But “Moshe Lewis” dominates then both

Behavioral score

Behavioral score

Partial order

- if c1 < c2 then c1 is dominated by c2

Behavioral score

Behavioral score

Partial order

- if c1 < c2 then c1 is dominated by c2
- if c1 > c2 then c1 dominates c2

Behavioral score

Behavioral score

Partial order

- if c1 < c2 then c1 is dominated by c2
- if c1 > c2 then c1 dominates c2
- otherwise, neither one dominates the other

Behavioral score

Behavioral score

Partial order

- if c1 < c2 then c1 is dominated by c2
- if c1 > c2 then c1 dominates c2
- otherwise, neither one dominates the other

Behavioral score

Behavioral score

Partial order

- if c1 < c2 then c1 is dominated by c2
- if c1 > c2 then c1 dominates c2
- otherwise, neither one dominates the other

For instance: vector of floats $(f_1, \ldots, f_n)$ where each f_i measure how well a candidate is doing for that particular feature.
If for all features i, $c1$ is doing better than $c2$, then $c1$ dominates $c2$.

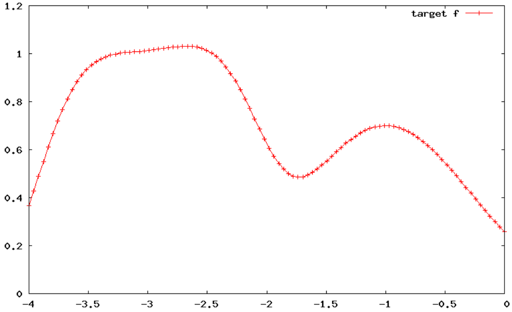
Deme selection

Keep non-dominated exemplars as potential deme.

Deme selection

Keep non-dominated exemplars as potential deme.

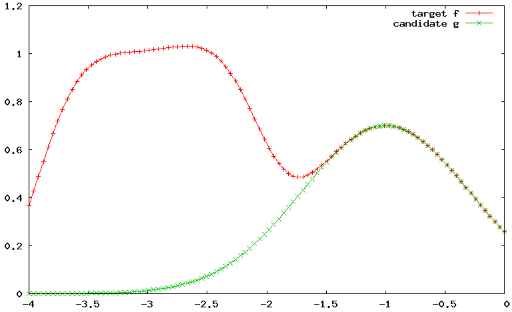
We want to learn $f(x)$.



Deme selection

Keep non-dominated exemplars as potential deme.

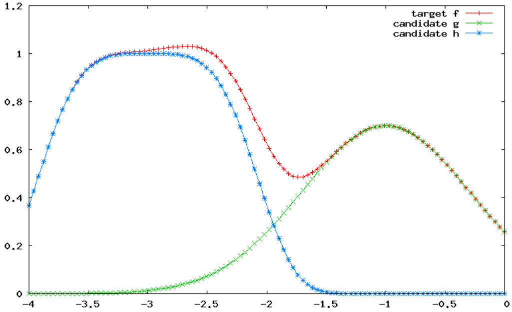
We want to learn $f(x)$.



Deme selection

Keep non-dominated exemplars as potential deme.

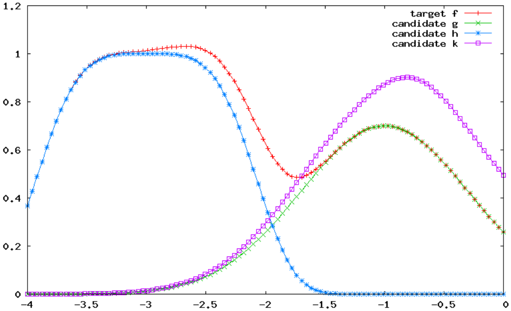
We want to learn $f(x)$.



Deme selection

Keep non-dominated exemplars as potential deme.

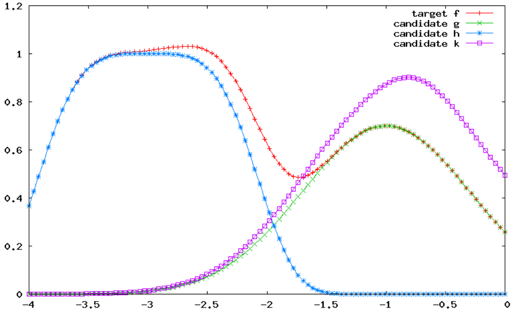
We want to learn $f(x)$.



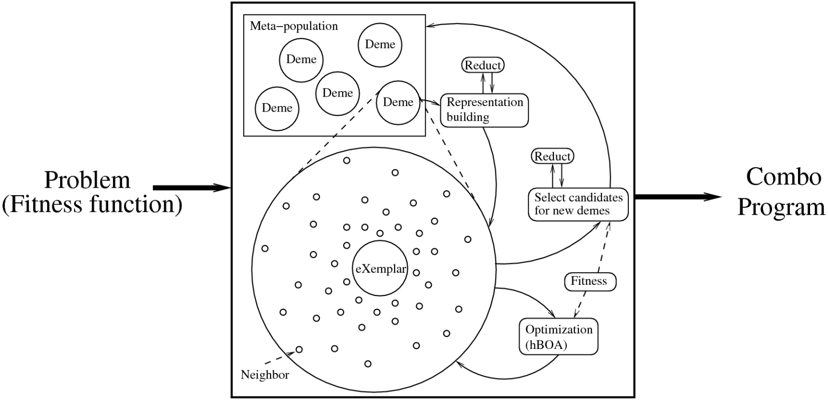
Deme selection

Keep non-dominated exemplars as potential deme.

We want to learn $f(x)$.



$k(x)$ is dominates by $g(x)$ and will not be included in the meta-population


Problem
(Fitness function)
Meta–population
Deme
Deme
Deme
Deme
Deme
Reduct
Representation
building
Reduct
Select candidates
for new demes
eXemplar
Fitness
Optimization
(hBOA)
Neighbor
Combo
Program

Some benchmark

Technique	Computational Effort			
	3-parity	4-parity	5-parity	6-parity
Univariate MOSES	6,151	73,977	2,402,523	342,280,092
Evolutionary programming [15]	28,500	181,500	2,100,000	no solutions
Genetic programming [49]	96,000	384,000	6,528,000	no solutions
MOSES	5,112	72,384	1,581,212	100,490,013

Figure: Computational effort to find solution of n-parity 99% of the time *(extracted from Moshe's PhD thesis)*

$$n\text{-}parity(b_1, \ldots, b_n) = even\left(\sum_{i=1}^{n} int(b_i)\right)$$

Outline

1. Introduction
2. Representation-Building
3. Optimization
4. Deme management
5. Demo...
6. Conclusion

Outline

1. Introduction
2. Representation-Building
3. Optimization
4. Deme management
5. Demo...
6. Conclusion

Conclusion

MOSES outperforms Genetic Algorithms because:

1. minimize over-representation (Reduction in normal form)

Conclusion

MOSES outperforms Genetic Algorithms because:

1. minimize over-representation (Reduction in normal form)
2. build expressive deme population by taking into account operator semantics (representation-building)

Conclusion

MOSES outperforms Genetic Algorithms because:

1. minimize over-representation (Reduction in normal form)
2. build expressive deme population by taking into account operator semantics (representation-building)
3. Maintain diversity in the meta-population (deme managment)

Conclusion

MOSES outperforms Genetic Algorithms because:

1. minimize over-representation (Reduction in normal form)
2. build expressive deme population by taking into account operator semantics (representation-building)
3. Maintain diversity in the meta-population (deme managment)
4. Attempt to find regularities in one deme's population to speed up optimization (model-building)

What remains to be done?

1. Only handles Boolean, continuous, numerico-boolean and (partially) action-perception expressions
   $\Rightarrow$ more programmatic construct

What remains to be done?

1. Only handles Boolean, continuous, numerico-boolean and (partially) action-perception expressions
   $\Rightarrow$ more programmatic construct
2. Representation-building is hard-coded per operator set
   $\Rightarrow$ Generalized for operator properties (especially useful if combined with PLEASURE)

What remains to be done?

1. Only handles Boolean, continuous, numerico-boolean and (partially) action-perception expressions
   $\Rightarrow$ more programmatic construct
2. Representation-building is hard-coded per operator set
   $\Rightarrow$ Generalized for operator properties (especially useful if combined with PLEASURE)
3. Model-building is slow
   $\Rightarrow$ improve model-building (better Bayesian learning, transfer learning across populations), or use other optimization method

What remains to be done?

1. Only handles Boolean, continuous, numerico-boolean and (partially) action-perception expressions
   $\Rightarrow$ more programmatic construct
2. Representation-building is hard-coded per operator set
   $\Rightarrow$ Generalized for operator properties (especially useful if combined with PLEASURE)
3. Model-building is slow
   $\Rightarrow$ improve model-building (better Bayesian learning, transfer learning across populations), or use other optimization method
4. No transfer learning across problem instances
   $\Rightarrow$ Integrative AGI, Attention Allocation, PLN, etc.