AI-DSL for autonomous interoperability

Nil Geisweiller

SingularityNET & OpenCog Foundations

1. What?
   - Provide formal description of an AI
2. Why?
   - Inform why to use a certain AI
   - Enable interoperability between AIs
3. How?
   - Dependent Types (Idris, AGDA, Coq, Liquid Haskell)
   - Probabilistic Logic Networks (PLN), OpenCog Hyperion
   - Natural Language Processing (NLP)!? (SingularityNET!?)
   - . . .

* Braindump:

- OpenCog:
- MOSES:
- infer type signature of the candidates to evolve given the fitness function, and define program spaces fulfilling such type signature.
- Evolve candidates using search guided by reasoning, with a formalized local search and fitness function (SNET presentation)
- Learning via Reasoning: OpenCog Pattern Miner, mining patterns fulfilling a specification.
- Planning via Reasoning: discover plans (to control agent in env) provably probabilistically correct.
- Magic Haskell.
- Gluing mismatched programs. Need bridgers: bridgers need formal specification so they can be part of the network.
- Verified Stack-Based Genetic Programming via Dependent Types. (9 year old)
- NLP: because often the only available specification is the documentation.
- Why AI? Cause AI is hard. Either you think as hard as it is, or you go out there a get the

Code reuse, see https://vimeo.com/131194141 https://en.wikipedia.org/wiki/Code$_r eusehttps : //www.perforce.com/blog/qac/challenge - code - reuse - and - how - reuse - code - effectively$

- RepresentationalLanguage – Formal grammar to represent candidates
Fitness -> InitialPopulation -> OptimizedPopulation
Fitness -> ResourceManagement -> Termination -> InitialPopulation -> OptimizedPopulation
Fitness -> ResourceManagement -> BackgroundKnowledge -> Termination -> InitialPopulation -> OptimizedPopulation
- rp : RepresentationalLanguage
- Candidate = Instantiate rp - Fitness = Candidate -> Float - InitialPopulation = Set Candidate - OptimizedPopulation = Multimap Float Candidate - Termination – Termination criteria, may depend on the state of the learner! - ResourceManagement – How much time and space resources to allocate - BackgroundKnowledge – Any knowledge that could be useful (domain specific, meta-heuristics, etc). Requires a common language.

Split MOSES into 3 modules:
Vectorize – Turn syntax tree into vector space (program subspace) Optimize vector space – Find good vector candidate Meta-optimize – Discover regularities in the vector space
Example: MOSES, evolve syntax trees, call external AI for the optimization step in vector space.
* MOSES (very abstract) type signature:
moses : Fitness -> Population -> Population
data Candidate = ... – Syntax tree type Fitness = Candidate -> Float type Population = Map Candidate Float
* Optimize Vector Space (very abstract) type signature:
VecFitness -> (Vector Float) -> VecPopulation type VecFitness = Vector Float -> Float type VecPopulation = Map (Vector Float) Float
* Meta-optimize (very abstract) type signature:
type OptimizationRecord = Map (Vector Float) Float = VecPopulation data FitnessEstimate = ... – Probabilistic model moptimize : OptimizationRecord -> FitnessEstimate
Can run backward from the fitness to the candidates!

Bibtex:

https://arxiv.org/pdf/2003.09040.pdf TF-Coder: Program Synthesis for Tensor Manipulations
https://www.cs.purdue.edu/homes/lintan/publications/c2s-fse20.pdf C2S: Translating Natural Language Comments to FormalProgram Specifications
https://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-25.pdf Formal Specification for Deep Neural Networks
https://link.springer.com/article/10.1007/s10270-020-00825-2 An epistemic approach to the formal specification of statistical machine learning
https://arxiv.org/abs/1911.10735 CAMUS: A Framework to Build Formal Specifications for Deep Perception Systems Using Simulators
https://github.com/BerkeleyLearnVerify/VerifAI VerifAI is a software toolkit for the formal design and analysis of systems that include artificial intelligence (AI) and machine learning (ML) components
https://www.ijcai.org/Proceedings/2019/840 LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning
https://arxiv.org/abs/1706.08605 Developing Bug-Free Machine Learning Systems With