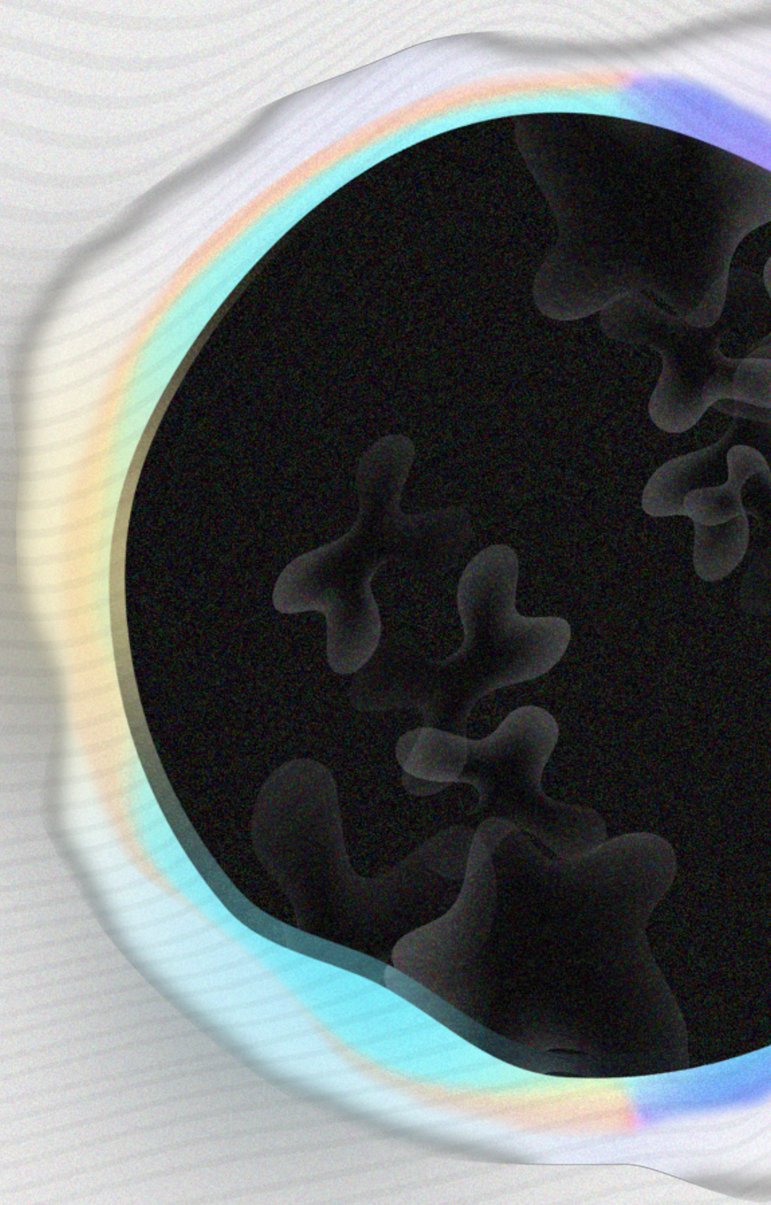
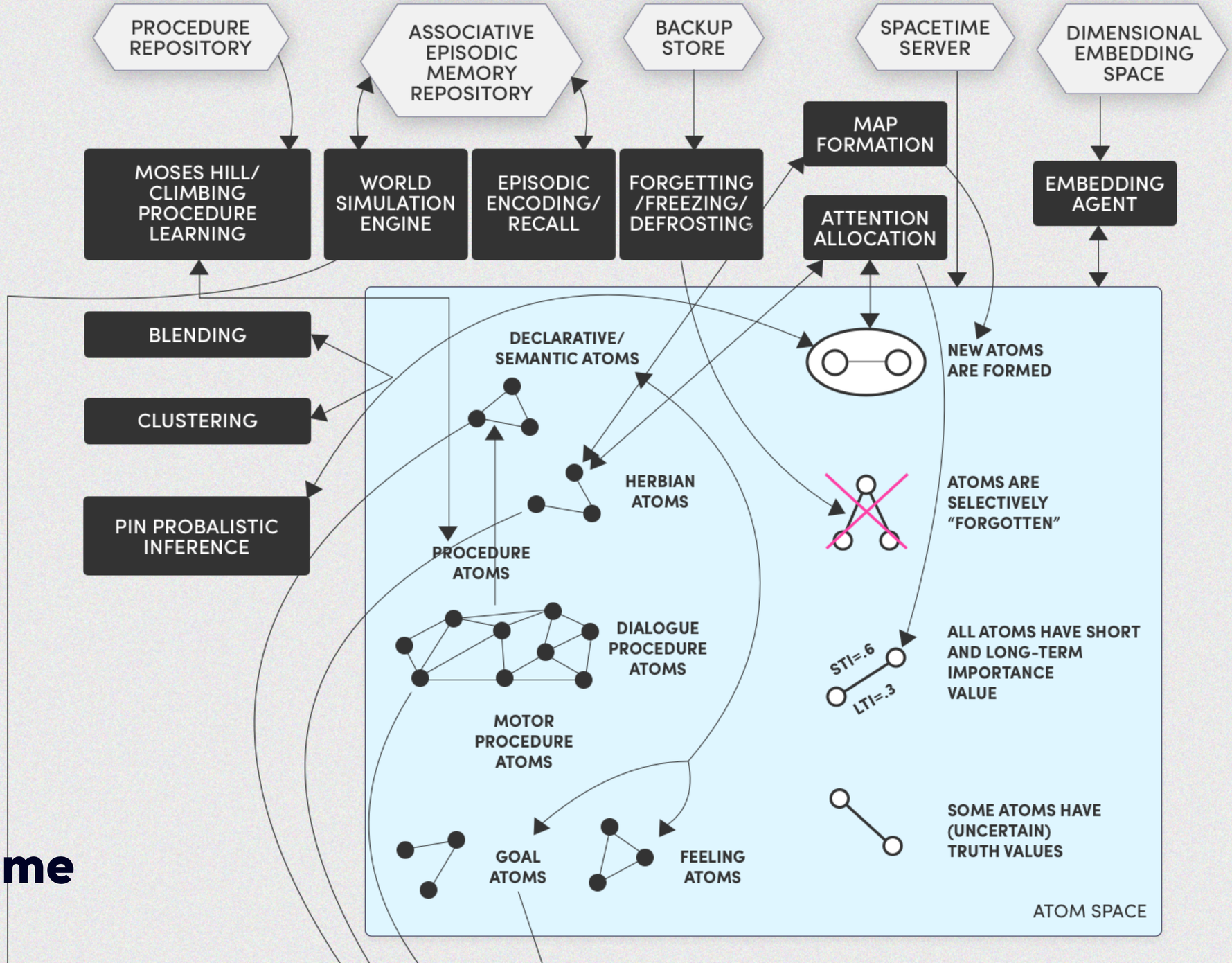
SingularityNET

Introduction to OpenCog

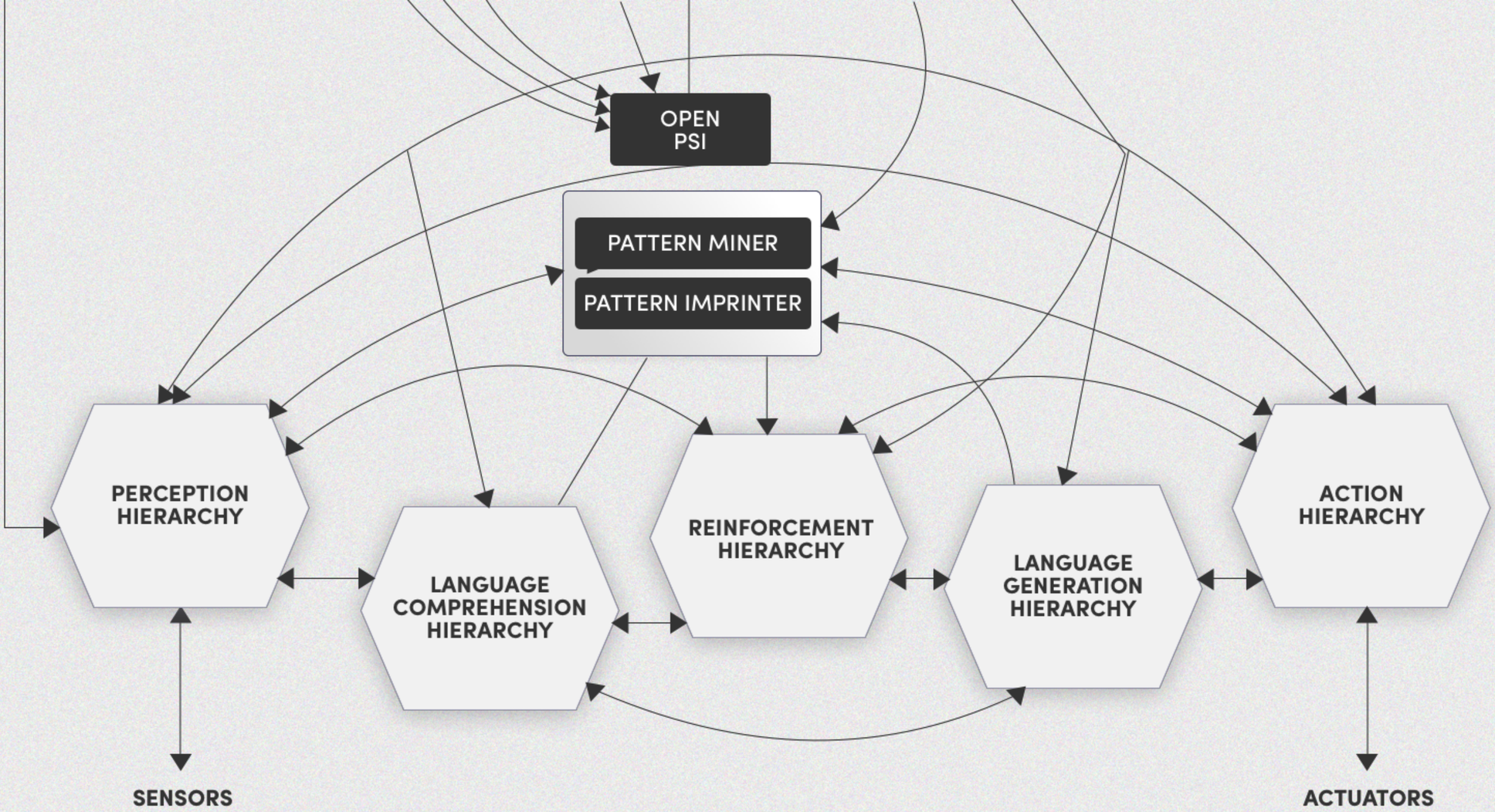


Anatoly Belikov
abelikov@singularitynet.io

```
$ https://github.com/singnet/
opencog-workshops


$
```


PROCEDURE REPOSITORY
ASSOCIATIVE EPISODIC MEMORY REPOSITORY
BACKUP STORE
SPACETIME SERVER
DIMENSIONAL EMBEDDING SPACE
MOSES HILL/ CLIMBING PROCEDURE LEARNING
WORLD SIMULATION ENGINE
EPISODIC ENCODING/ RECALL
FORGETTING /FREEZING/ DEFROSTING
MAP FORMATION
ATTENTION ALLOCATION
EMBEDDING AGENT
BLENDING
CLUSTERING
PIN PROBALISTIC INFERENCE
DECLARATIVE/ SEMANTIC ATOMS
HERBIAN ATOMS
PROCEDURE ATOMS
DIALOGUE PROCEDURE ATOMS
MOTOR PROCEDURE ATOMS
GOAL ATOMS
FEELING ATOMS
NEW ATOMS ARE FORMED
ATOMS ARE SELECTIVELY "FORGOTTEN"
STI=.6
LTI=.3
ALL ATOMS HAVE SHORT AND LONG-TERM IMPORTANCE VALUE
SOME ATOMS HAVE (UNCERTAIN) TRUTH VALUES
ATOM SPACE

CogPrime


OPEN
PSI
PATTERN MINER
PATTERN IMPRINTER
PERCEPTION
HIERARCHY
LANGUAGE
COMPREHENSION
HIERARCHY
REINFORCEMENT
HIERARCHY
LANGUAGE
GENERATION
HIERARCHY
ACTION
HIERARCHY
SENSORS
ACTUATORS

CogPrime

Use cases

- Chatbots
- Bioinformatics
- Unsupervised language learning
- Visual Question Answering VQA
- ...

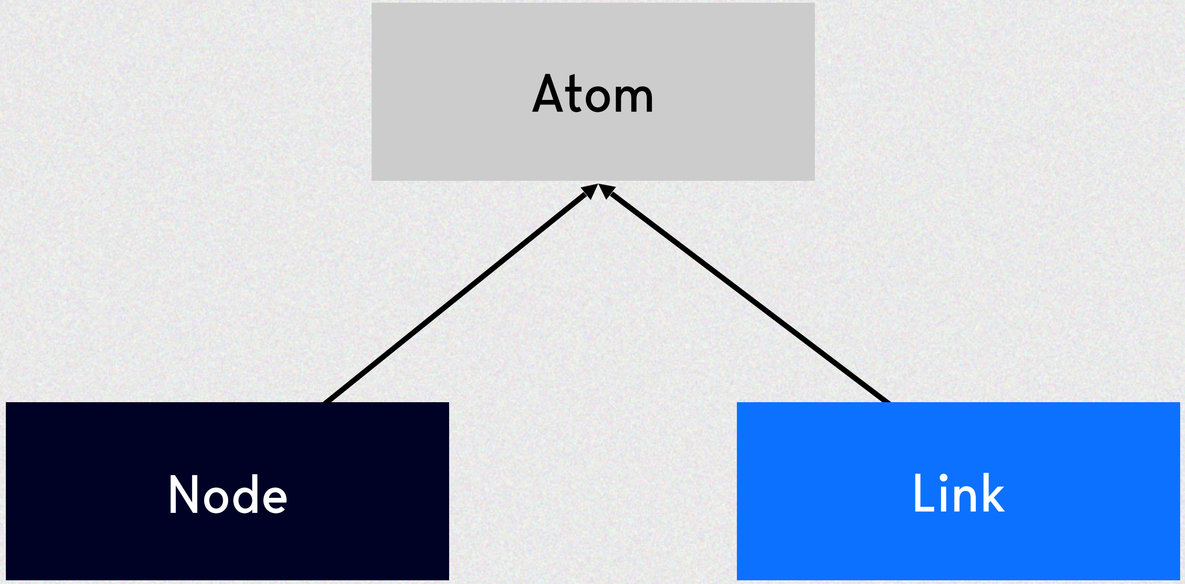
Atomspace - graph database

Hypergraph database designed to be primarily used as a storage of explicit, declarative knowledge

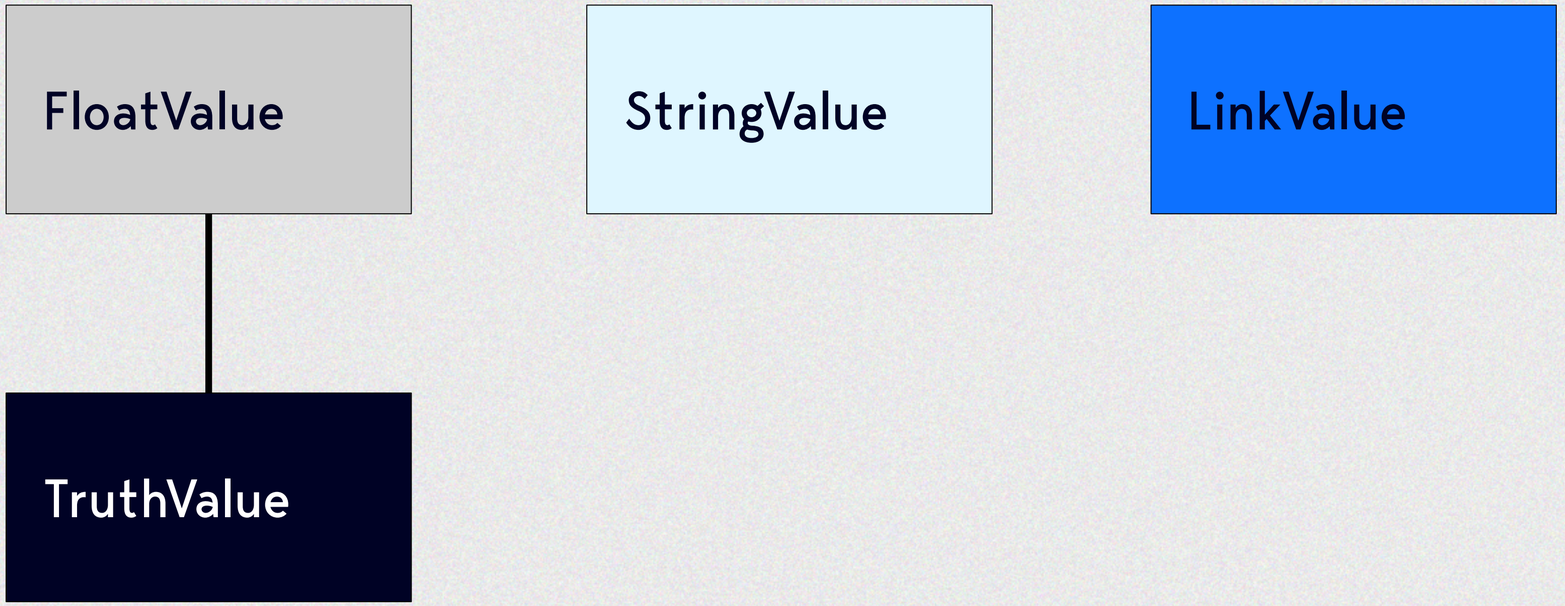
Hypergraph defines programming language - Atomese

Has a number of associated mechanisms:
pattern matching, unified rule engine, moses

Types of data in atomspace: Atoms



Types of data in atomspace: Values



Atomspace - API introduction

Python:

```
>> from opencog.atomspace import AtomSpace, types

>> atomspace = AtomSpace()
```

Scheme:

```
scheme> (use-modules (opencog))

scheme> (define atomspace (cog-new-atomspace))
```

Pattern matcher examples

```
$ docker run -p8888:8888  -it demo-opencog
/home/relex/opencog-intro-master/
notebook.sh
```

```
$ http://localhost:8888/notebooks/and-
link-example.ipynb
```

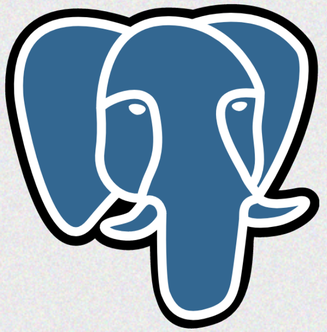
Loading and storing

Load scheme file from scheme or python:

```
>> scheme_eval(atomspace, '(load-from-path “atomspace.scm")')
```

Indexed by atom type and name

Use PostgreSQL as backend storage



jupyter

Files | Running | Clusters

Select items to perform actions on them.

0 / Name Last Modified

- docker — 2 hours ag
- and-link-example.ipynb — Running 2 hours ag
- atomese-examples.ipynb — 2 hours ag
- socrates-explained.ipynb — 2 hours ag
- socrates.ipynb — 2 hours ag
- common.py — 2 hours ag

[3]:
```
InheritanceLink(ConceptNode("size"), ConceptNode("attribute"))
InheritanceLink(ConceptNode("color"), ConceptNode("attribute"))
```

[3]:
```
(InheritanceLink
  (ConceptNode "color") ; [4437725137865661335][1]
  (ConceptNode "attribute") ; [82861190687766677766][1]
) ; [15737919214045963885][1]
```

different types of color and size

[ ]:
```
InheritanceLink(ConceptNode("small"), ConceptNode("size"))
InheritanceLink(ConceptNode("big"), ConceptNode("size"))
InheritanceLink(ConceptNode("green"), ConceptNode("color"))
```

Nodes examples

```
scheme@(guile-user)> (ConceptNode "Test")
scheme@(guile-user)> (VariableNode "X")
scheme@(guile-user)> (PredicateNode "is-red")
```

Links examples: ListLink

```
>>> lst1 = ListLink(ConceptNode("red"),
                    VariableNode("$X"),
                    ConceptNode("blue"))
>>> print(str(lst1))
(ListLink
  (ConceptNode "red")
  (VariableNode "$X")
  (ConceptNode "blue")
)
```

TruthValue

```
> from opencog.atomspace import TruthValue

> tv = TruthValue(0.5, 0.92)

P(Heads) = 0.5, .3

> ConceptNode(“Heads”).tv = tv
```

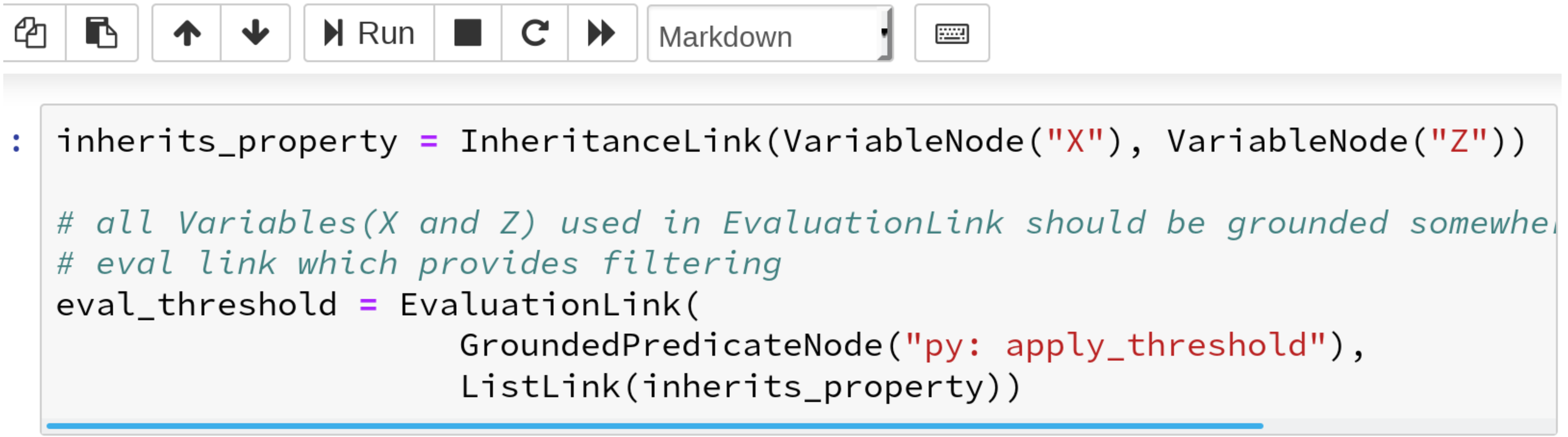
Query example

let's select all animals and attributes, filtering by some threshold

first define function and EvaluationLink

[ ]:
```
threshold = 0.6

def apply_threshold(atom):
    if atom.tv.mean < threshold:
        return TruthValue(0, 1)
    return TruthValue(1, 1)
```

EvaluationLink will accept link X <- Z



now build the BindLink

Constrain that Variable "X" inherits from "item"

BindLink and pattern matcher

BindLink

```
<variables> - optional
<pattern to match>
<rewrite term>
```

Pattern example:

```
ListLink(ConceptNode("red"),
         VariableNode("$X"),
         ConceptNode("blue"))
```

Constrain that Variable "X" inherits from "item"

[9]:
```
select_items = InheritanceLink(VariableNode("X"), ConceptNode("animal"))
```

[10]:
```
# using NotLink
Z_is_not_item = NotLink(IdenticalLink(VariableNode("Z"), ConceptNode("animal")))
```

[11]:
```
and_link = AndLink(eval_threshold, inherits_property, Z_is_not_item, select_items)
bindlink_filter = BindLink(and_link,
                           inherits_property)


print(bindlink(atomspace, bindlink_filter))
```

```
(SetLink
  (InheritanceLink (stv 0.800000 0.900000)
    (ConceptNode "frog")
    (ConceptNode "small")
```

```
print(bindlink(atomspace, bindlink_filter))
```

```
(SetLink
  (InheritanceLink (stv 0.800000 0.900000)
    (ConceptNode "frog")
    (ConceptNode "small")
  )
  (InheritanceLink (stv 0.600000 0.900000)
    (ConceptNode "tiger")
    (ConceptNode "red")
  )
  (InheritanceLink (stv 0.700000 0.900000)
    (ConceptNode "frog")
    (ConceptNode "green")
  )
  (InheritanceLink (stv 0.900000 0.900000)
    (ConceptNode "tiger")
    (ConceptNode "big")
  )
  (InheritanceLink (stv 0.800000 0.900000)
    (ConceptNode "sparrow")
    (ConceptNode "small")
  )
)
```

- docker
- and-link-example.ipynb
- atomese-examples.ipynb Running
- socrates-explained.ipynb
- socrates.ipynb
- common.py
- LICENSE

Another usefull type of link is ExecutionOutputLink
which used to run callback function on matched pattern
ExecutionOutputLink should return Atom

ExecutionOutputLink structure

1. GroundedSchemaNode
2. ListLink of arguments

---

let's define a callback function which returns a ConceptNode

[17]:
```
def add_node():
    return ConceptNode("Test")
```

Now GroundedSchemaNode which accepts string of following format
"<language>:<function name>"

where language can be any of "lib", "py", "scm"

In case of lib atomspace will try to open library with dlopen
so any shared library may be used

[18]:
```
add_node_schema = GroundedSchemaNode("py: add_node")
```

Now make Execution output link

[19]:
```
exec1 = ExecutionOutputLink(
         add_node_schema,
         ListLink())
```

ExecutionOutputLink may be used inside of bindlink or
it may be executed by execute_atom function

[20]:
```
print("running ExecutionOutputLink")
print(execute_atom(atomspace, exec1))
```

```
running ExecutionOutputLink
(ConceptNode "Test")
```

ExecutionOutputLinks may be nested(actually any links may)

```
def add_link(node):
    ev = EvaluationLink(
            PredicateNode("ok"),
            node)
    ev.tv = TruthValue(0.1, 0.8)
    return ev

exec2 = ExecutionOutputLink(
            GroundedSchemaNode("py: add_link"),
            ListLink(
              ExecutionOutputLink(
                  GroundedSchemaNode("py: add_node"),
                  ListLink()))
        )

print(execute_atom(atomspace, exec2))
```

```
nested ExectutionOutputLinks
(EvaluationLink (stv 0.100000 0.800000)
  (PredicateNode "ok")
  (ConceptNode "Test")
)
```

Unified Rule Engine

Inference engine based on graph rewriting rules.

Allows to perform declarative inference.

Rules and data defined in the same language - atomese

Can do backward and forward chaining

Unified Rule Engine

Inference in first-order logic

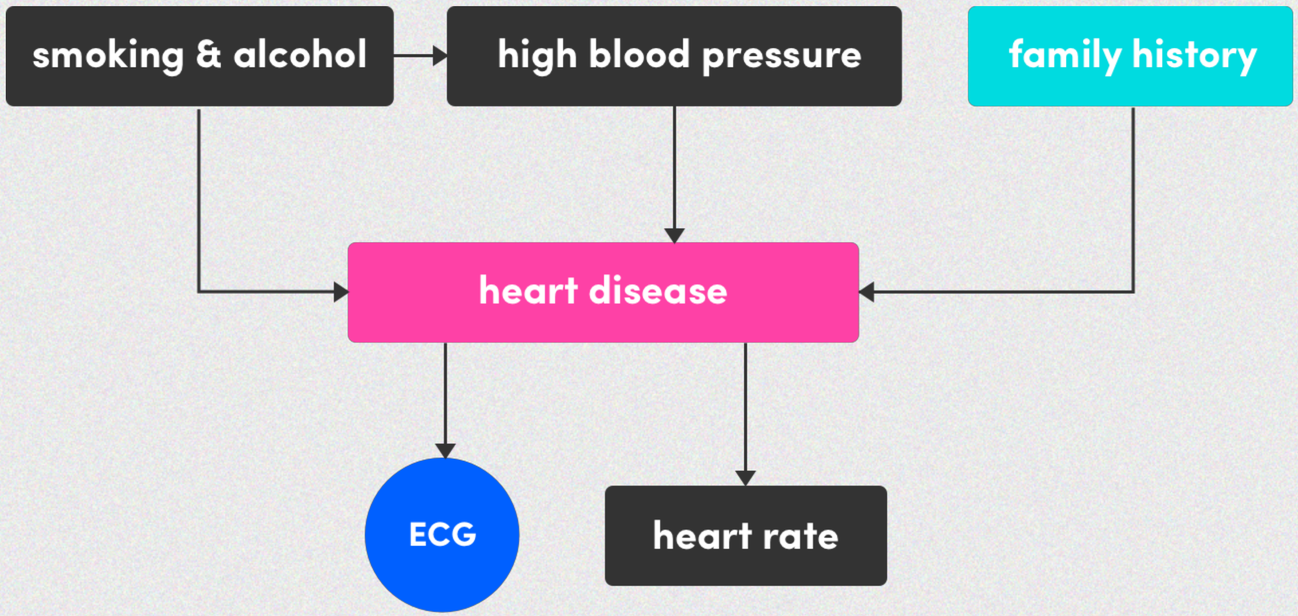
Automatic differentiation

Graphical models

Probabilistic logic inference

Fuzzy logic inference

...




docker


and-link-example.ipynb


atomese-examples.ipynb


socrates-explained.ipynb


socrates.ipynb Running


common.py


LICENSE


notebook.sh


opencog.log


README.md

[5]:
```
pattern = InheritanceLink(ConceptNode("Socrates"), ConceptNode("mortal"))
```

Specify rule base.
All rule inheriting from ConceptNode("PLN") will be considered by the rule engine

[6]:
```
rule_base = ConceptNode("PLN")
```

Construct BackwardChainer object
It's arguments are atomspace, rule_base, and pattern to infer

[7]:
```
chainer = BackwardChainer(atomspace, rule_base, pattern)
chainer.do_chain()
print(chainer.get_results())
```

```
(SetLink
```

Running backward chainer

```
> request =
InheritanceLink(ConceptNode("Socrates"),
                ConceptNode("mortal"))
> tr = AtomSpace() # trace atomspace
> rule_base = ConceptNode("PLN") # rules to use
> chainer = BackwardChainer(atomspace, rule_base,
      request,
      trace_as=tr)
> chainer.do_chain()
> print(chainer.get_results())
```

Defining problem

```
> InheritanceLink(ConceptNode("Socrates"),
                  ConceptNode("man")).tv =
                TruthValue(0.97, 0.92)
> InheritanceLink(ConceptNode("man"),
                  ConceptNode("mortal")).tv =
                TruthValue(0.98, 0.94)
InheritanceLink(ConceptNode("Socrates"),
                  ConceptNode("mortal")).tv = ???
```

Result

```
> (SetLink
      (InheritanceLink (stv 0.9508 0.828)
      (ConceptNode(“Socrates”),
      (ConceptNode("mortal"))
```

Rule structure:

```
BindLink
    <variables>
    AndLink
        <premise-1>
        ...
        <premise-n>
    <conclusion-pattern>
```

Calculate conclusion with ExecutionOutputLink

ExecutionOutputLink

GroundedSchemaNode <formula-name>

List

<conclusion-pattern>

<other arguments>

1 / Name Last M

- docker 5 h
- and-link-example.ipynb 3 h
- atomese-examples.ipynb an
- socrates-explained.ipynb Running 5 h
- socrates.ipynb 4 mi
- common.py 5 h
- LICENSE 5 h

Deduction rule example

premise

A -> B

B -> C

```
condition = AndLink(BA, CB)

BA = InheritanceLink(var_b, var_a)

CB = InheritanceLink(var_c, var_b)
```

Ergo: A -> C

```
rewrite = ExecutionOutputLink(
                GroundedSchemaNode("scm: deduction-formula"),
                ListLink(CA, CB, BA))

deduction_link = BindLink(condition, rewrite)
```

[4]:

```
def get_deduction_rule(var_a=VariableNode("$A"),
                       var_b=VariableNode("$B"),
                       var_c=VariableNode("$C")):
    BA = InheritanceLink(var_b, var_a)
    CB = InheritanceLink(var_c, var_b)
    CA = InheritanceLink(var_c, var_a)
    condition = AndLink(BA, CB, NotLink(IdenticalLink(var_a, var_c)))
    rewrite = ExecutionOutputLink(GroundedSchemaNode("scm: deduction-
                                  ListLink(CA, CB, BA))

    deduction_link = BindLink(condition, rewrite)
    return deduction_link
```

```
InheritanceLink(ConceptNode("Socrates"), ConceptNode("man")).tv = Tru
InheritanceLink(ConceptNode("man"), ConceptNode("mortal")).tv = Truth
```

[10]:
```
deduction_link = get_deduction_rule()
```

[11]:
```
print("running bindlink: \n{0}".format(deduction_link))
print(bindlink(atomspace, deduction_link))
```

```
(BindLink
  (AndLink
    (InheritanceLink
      (VariableNode "$B")
      (VariableNode "$A")
    )
    (NotLink
      (IdenticalLink
        (VariableNode "$A")
        (VariableNode "$C")
      )
    )
    (InheritanceLink
      (VariableNode "$C")
      (VariableNode "$B")
    )
  )
```

```
(ExecutionOutputLink
  (GroundedSchemaNode "scm: deduction-formula")
  (ListLink
    (InheritanceLink
      (VariableNode "$C")
      (VariableNode "$A")
    )
    (InheritanceLink
      (VariableNode "$C")
      (VariableNode "$B")
    )
    (InheritanceLink
      (VariableNode "$B")
      (VariableNode "$A")
    )
  )
)
```

```
(SetLink
  (InheritanceLink (stv 0.950800 0.828000)
    (ConceptNode "Socrates")
    (ConceptNode "mortal")
  )
)
```

Add new link Philosopher <- man

[9]:
```
InheritanceLink(ConceptNode("Philosopher"), ConceptNode("man")).tv
```

[ ]:
```
print("running with the same link and Philosopher")

print(bindlink(atomspace, deduction_link))
```

Replace variables with corresponding ConceptNodes

[10]:
```
deduction_link = get_deduction_rule(var_a=ConceptNode("mortal"),
                                    var_c=ConceptNode("Socrates"))
```

[11]:
```
print("running bindlink: \n{0}".format(deduction_link))
print(bindlink(atomspace, deduction_link))
```

```
running bindlink:
(BindLink
  (AndLink
    (InheritanceLink
      (VariableNode "$B")
      (ConceptNode "mortal")
    )
```

BindLink the query was answered with

```
(AndLink
   (InheritanceLink
     (VariableNode "$B")
     (ConceptNode "mortal")
   )
   (NotLink
     (IdenticalLink
       (ConceptNode "Socrates")
       (ConceptNode "mortal")
     )
   )
   (InheritanceLink
     (ConceptNode "Socrates")
     (VariableNode "$B")
   ))
```

```
(ExecutionOutputLink
 (GroundedSchemaNode "scm: deduction-formula")
 (ListLink
   (InheritanceLink (stv 0.9508 0.828)
     (ConceptNode "Socrates")
     (ConceptNode "mortal")
   )
   (InheritanceLink
     (ConceptNode "Socrates")
     (VariableNode "$B")
   )
   (InheritanceLink
     (VariableNode "$B")
     (ConceptNode "mortal")
   )))
```

Running VQA demo from the docker image

```
$ docker pull opencog/vqa
$ wget https://s3-us-west-2.amazonaws.com/abelikov/data-small.tar.gz && tar -xvf data-small.tar.gz
$ docker run  -v `pwd`/data:/home/relex/projects/data -p 8889:8888 -it demo-opencog
```

- hypernetimpl
- splitnet
- tbd_cog
- interface-images-demo.ipynb Running
- __init__.py
- answer_by_image.py
- download_data.sh
- gui_helpers.py

jupyter interface-images (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

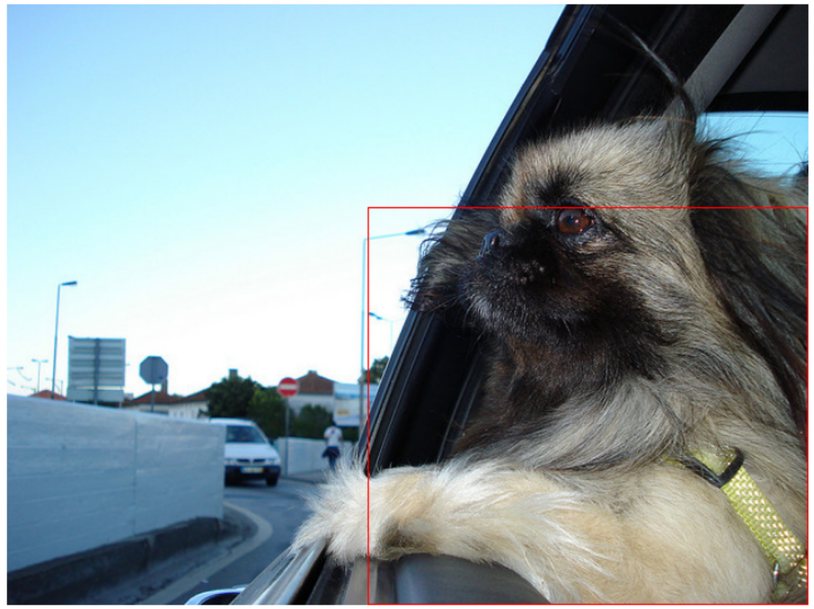
Trusted | Python 3

Run Code

In [16]:
```
window = MainWindow(images, vqa, use_camera=False)
window.display()
```

idx 880

Is the dog black?



yes

```
(conj-bc (AndLink
    (InheritanceLink (VariableNode "$X") (ConceptNode "BoundingBox"))
    (EvaluationLink (GroundedPredicateNode "py:runNeuralNetwork") (Lis
tLink (VariableNode "$X") (ConceptNode "dog")) )
    (EvaluationLink (GroundedPredicateNode "py:runNeuralNetwork") (Lis
tLink (VariableNode "$X") (ConceptNode "black")) )
  )
 )

(AndLink (stv 0.503759 1.000000)
  (EvaluationLink (stv 0.503759 1.000000)
    (GroundedPredicateNode "py:runNeuralNetwork")
    (ListLink
      (ConceptNode "BoundingBox-1")
      (ConceptNode "dog")
    )
  )
  (InheritanceLink (stv 1.000000 1.000000)
    (ConceptNode "BoundingBox-1")
    (ConceptNode "BoundingBox")
  )
  (EvaluationLink (stv 0.607480 1.000000)
    (GroundedPredicateNode "py:runNeuralNetwork")
    (ListLink
      (ConceptNode "BoundingBox-1")
```

list of models

pole metal trunk sand truck counter floor player plane person hand letters leaf jacket ball wooden bush blue phone vehicle pants window ground girl child bus shirt picture man logo zebra car wood standing flag seat wheel beach tire green sidewalk mirror skateboard cat surfboard clock gray hat red pizza large pillow shoe banana tall orange roof train white bird cloud sitting bench windows people plastic shadow tree airplane part old round young grass box clear water square chair light house tail building big sink bed hair gold small field cow desk laptop black bottle bike dog shelf background woman bear writing helmet empty snow board door concrete sign bag giraffe road motorcycle kite book food tile bowl plant short dark yellow boat top street fence bright rock long plate wave pink umbrella open striped vase flower cup foot tan shoes boy sky elephant toilet glasses purple glass reflection arm wall brown silver sheep table horse cake

Building VQA with OpenCog

Building blocks:

Atomspace for storing facts about world

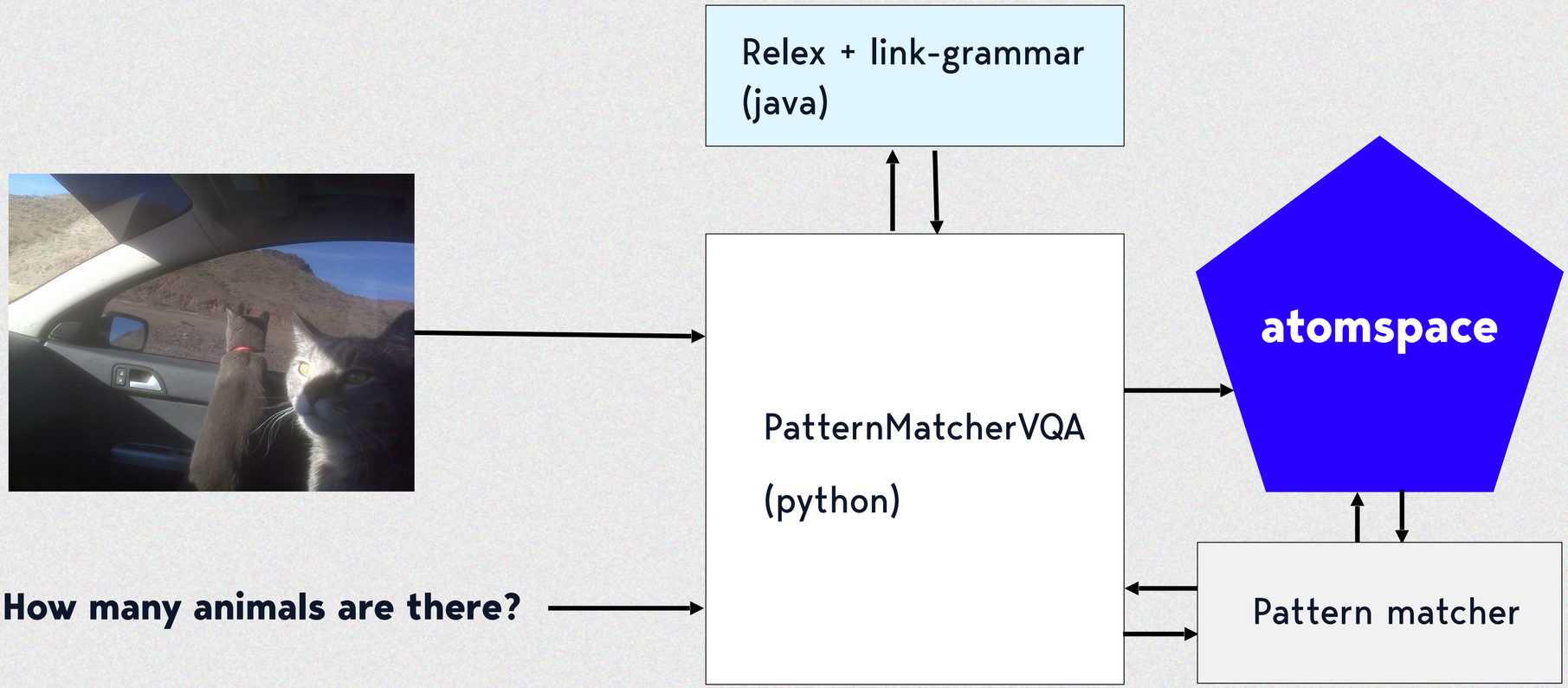
Link-Grammar + RelEx for text processing

Faster-RCNN for bounding box and feature extraction

Our neural network models for classification

Unified rule engine and pattern matcher for computing answers

VQA pipeline overview



Knowledge base

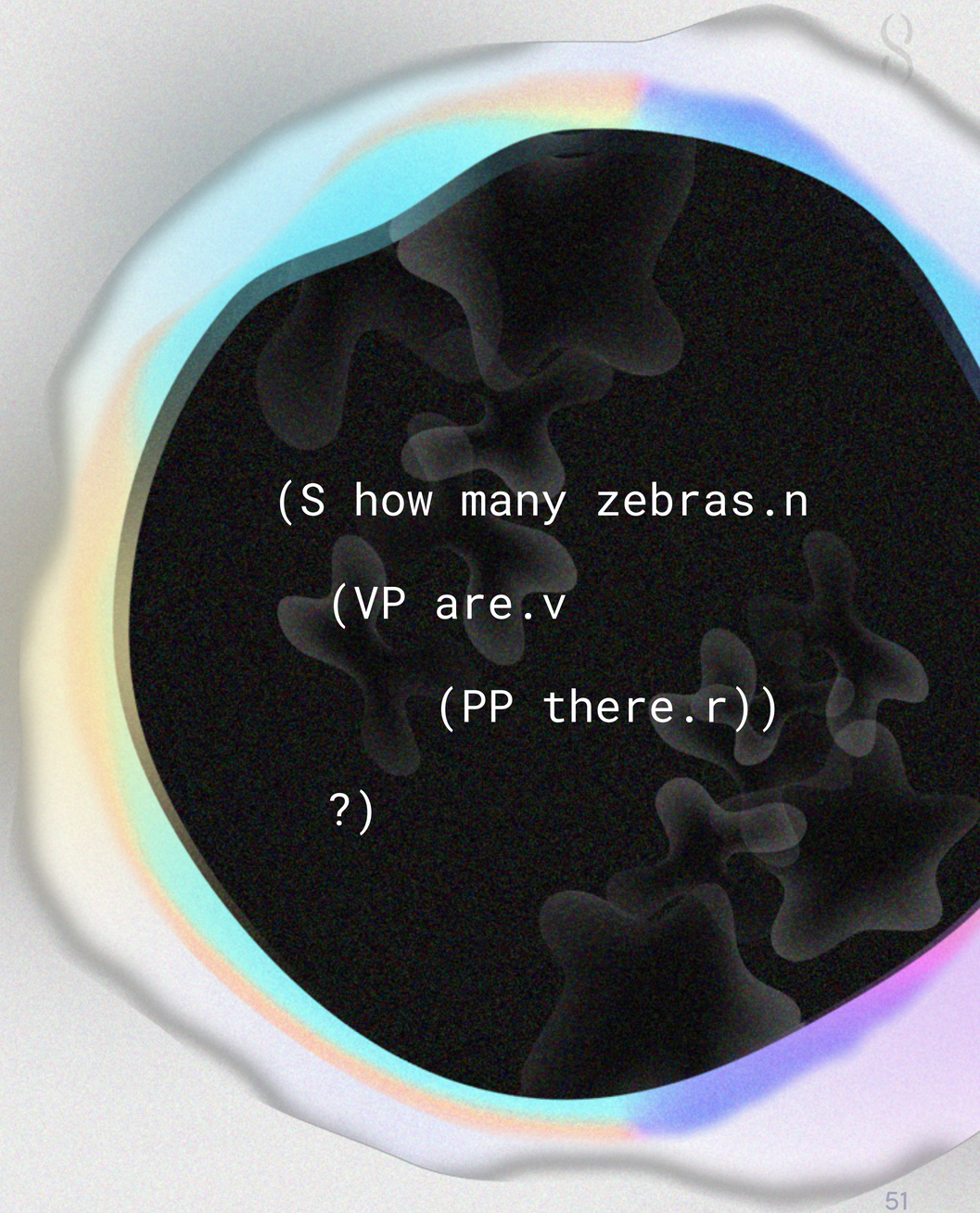
```
(InheritanceLink (stv 0.795 0.71)
        (ConceptNode "van") (ConceptNode "vehicle"))
(InheritanceLink (stv 0.576 0.525)
        (ConceptNode "van") (ConceptNode "auto"))
```

VQA pipeline: text processing

```
sentence => link-grammar => relex => template => query
```

Taking a look at link-grammar

```
linkparser> How many zebras are there?

Found 22 linkages (6 had no P.P. violations)

       Linkage 1, cost vector = (UNUSED=0 DIS=
2.00 LEN=8)

     +------------------Xp-----------------+
   +-----------WV----------+               |
   +->Ws--+--H-+--Dmc-+--Spx-+--Pp--+     |
   |      |    |      |      |      |     |
LEFT-WALL how many zebras.n are.v there.r ?
```



How link-grammar works?

Link-grammar has predefined dictionary or words associated with allowed link types.

(word: link types) pairs define what type of links can be attached to which word and from which side.

Also number of global rules for example that links cannot cross each other.

Check the rules:

github.com/opencog/link-grammar/blob/master/data/en/4.0.dict

RelEx

Dependency parser that built on graph rewriting rules

Operates on graph built from link-grammar parse

Adds new semantic links between words

Relex parse for "How many zebras are there?":

```
_predadj(zebra, there)

_quantity(zebra, _$qVar)
```

Vertices attributes of relex sentence graph


```
pos(how, adv)

penn-POS(how, RB)

pos(?, punctuation)

noun_number(zebra, plural)

pos(zebra, noun)

penn-POS(zebra, NNS)

pos(_$qVar, adj)
```

```
pos(be, verb)

penn-POS(be, VBP)

pos(there, adj)

tense(there, present)

penn-POS(there, JJ)

HYP(there, T)

QUERY-TYPE(_$qVar,
                    how_much)

penn-POS(_$qVar, CD)
```

Query example

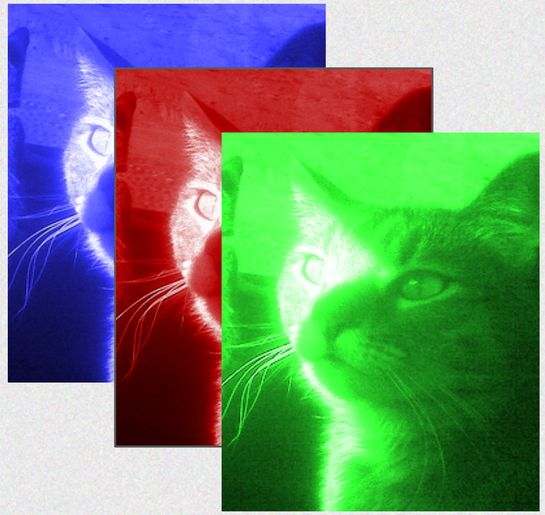
Is the dog black?

```
_predadj(A, B)::_predadj(dog, black)

(conj-bc (AndLink
    (InheritanceLink (VariableNode "$X") (ConceptNode "BoundingBox"))
    (EvaluationLink (GroundedPredicateNode "py:runNeuralNetwork")

          (ListLink (VariableNode "$X") (ConceptNode "dog")) )
    (EvaluationLink (GroundedPredicateNode "py:runNeuralNetwork")

          (ListLink (VariableNode "$X") (ConceptNode "black")) ))
 )
```

Image processing

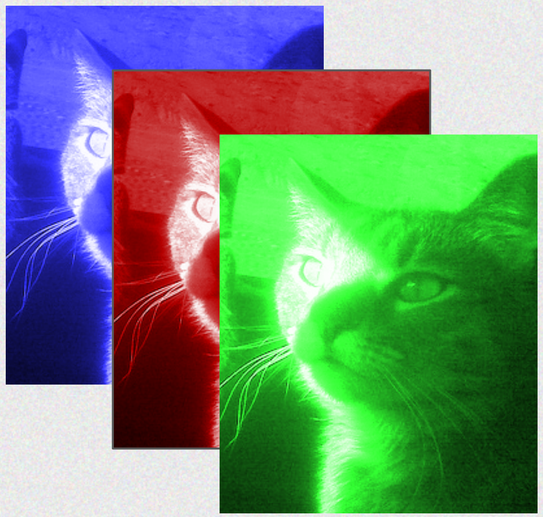
Faster-RCNN

Bounding Boxes & features



Bounding boxes => atomspace

Bounding Boxes & features



```
ConceptNode("BoundingBox1")
ConceptNode("BoundingBox2")
ConceptNode("BoundingBox3")
```

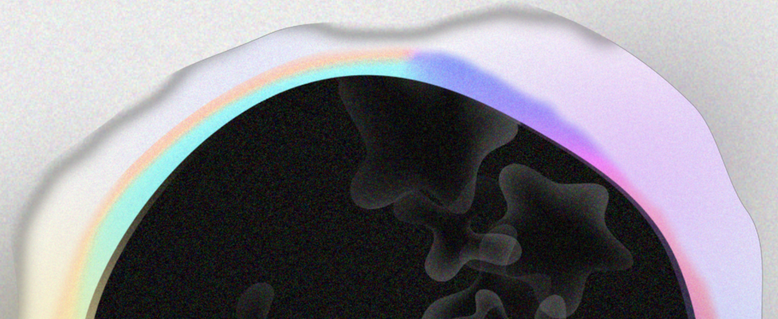
atomspace

Bounding boxes => atomspace

```
> (InheritanceLink
                (ConceptNode("BoundingBox1")
                (ConceptNode("BoundingBox"))
```

Use of URE and pattern matcher for queries

- URE allows to use declarative approach for computing answers
- We use fuzzy logic rules to compute truth value of queries



idx 908

Is the cat gray?

yes

```
(cog-execute! (BindLink
  (TypedVariableLink (VariableNode "$X") (TypeNode "C
onceptNode"))
  (AndLink
    (InheritanceLink (VariableNode "$X") (ConceptNode
"BoundingBox"))
    (EvaluationLink (GroundedPredicateNode "py:runNeu
ralNetwork") (ListLink (VariableNode "$X") (ConceptNo
de "cat")) )
    (EvaluationLink (GroundedPredicateNode "py:runNeu
ralNetwork") (ListLink (VariableNode "$X") (ConceptNo
de "gray")) )
  )
  (AndLink
    (InheritanceLink (VariableNode "$X") (ConceptNode
```

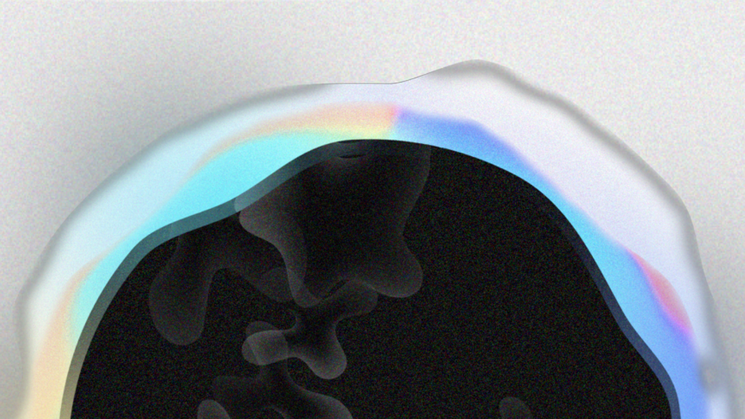
Integrating opencog with neural networks

Stay tuned!

Passing arbitrary python objects between ExecutionOutputLinks

Allows to express computation graph as pytorch expression

Allows to integrate and update ontologies



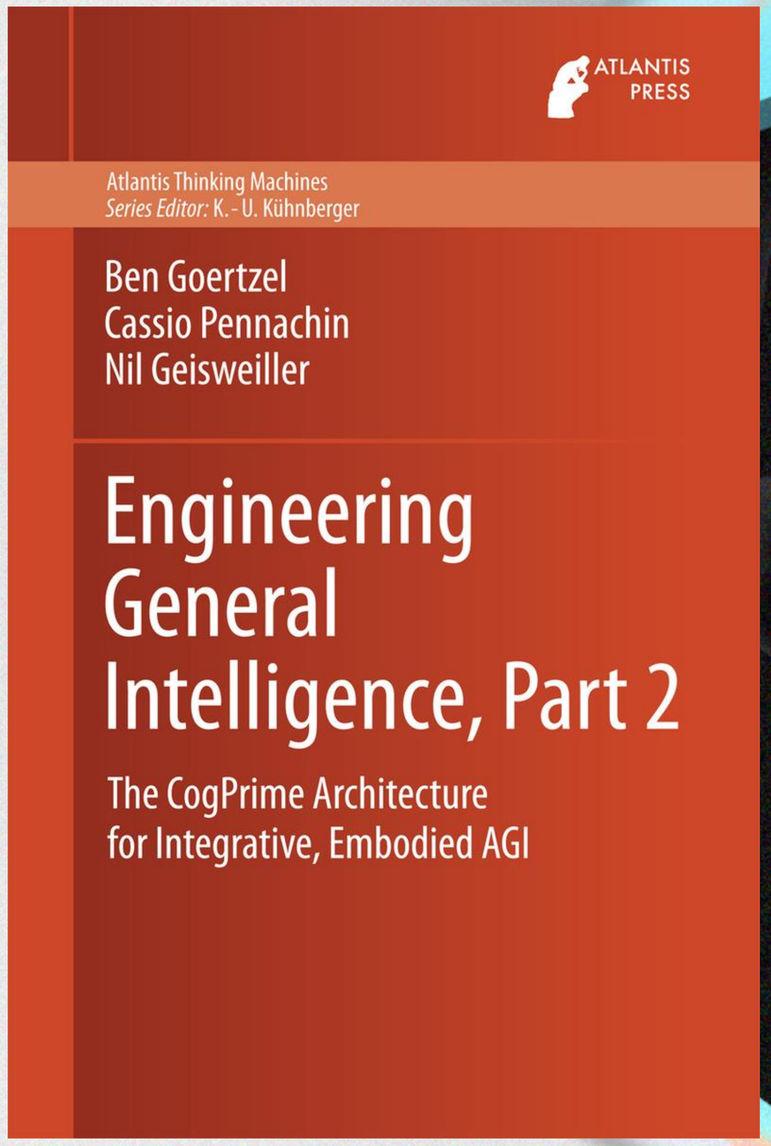
Resources

wiki.opencog.org/

github.com/opencog/

github.com/singnet/semantic-vision

blog.singularitynet.io



SingularityNET