pyknow Documentation

Release 0.0.8

Roberto Abdelkadder Martinez, David Francos Cuartero

Apr 05, 2018

Contents

CHAPTER 1

User Guide

1.1 Introduction

1.1.1 Philosophy

We aim to implement a Python alternative to CLIPS, as compatible as possible. With the goal of making it easy for the CLIPS programmer to transfer all of his/her knowledge to this platform.

1.1.2 Features

- Python 3 compatible.
- Pure Python implementation.
- Matcher based on the RETE algorithm.

1.1.3 Difference between CLIPS and PyKnow

1. CLIPS is a programming language, PyKnow is a Python library. This imposes some limitations on the constructions we can do (specially on the LHS of a rule).
2. CLIPS is written in C, PyKnow in Python. A noticeable impact in performance is to be expected.
3. In CLIPS you add facts using *assert*, in Python *assert* is a keyword, so we use *declare* instead.

1.2 Installation

1.2.1 From PyPI

To install PyKnow, run this command in your terminal:

```
$ pip install pyknow
```

1.2.2 Getting the source code

PyKnow is developed on Github.

You can clone the repository using the git command:

```
$ git clone https://github.com/buguroo/pyknow.git
```

Or you can download the releases in .zip or .tar.gz format.

Once you have a copy of the source, you can install it running this command:

```
$ python setup.py install
```

1.3 The Basics

An expert system is a program capable of pairing up a set of **facts** with a set of **rules** to those facts, and execute some actions based on the matching rules.

1.3.1 Facts

Facts are the basic unit of information of PyKnow. They are used by the system to reason about the problem.

Let's enumerate some facts about *Facts*, so... metafacts ;)

1. The class *Fact* is a subclass of *dict*.

```
>>> f = Fact(a=1, b=2)
>>> f['a']
1
```

2. Therefore a *Fact* does not mantain an internal order of items.

```
>>> Fact(a=1, b=2)  # Order is arbirary :O
Fact(b=2, a=1)
```

3. In contrast to *dict*, you can create a *Fact* without keys (only values), and *Fact* will create a numeric index for your values.

```
>>> f = Fact('x', 'y', 'z')
>>> f[0]
'x'
```

4. You can mix autonumeric values with key-values, but autonumeric must be declared first:

```
>>> f = Fact('x', 'y', 'z', a=1, b=2)
>>> f[1]
'y'
>>> f['b']
2
```

5. You can subclass *Fact* to express different kinds of data or extend it with your custom functionality.

```
class Alert(Fact):
    """The alert level."""
    pass

class Status(Fact):
    """The system status."""
    pass

f1 = Alert('red')
f2 = Status('critical')
```

```
from pyknow import Fact
from django.contrib.auth.models import User as DjangoUser

class User(Fact):
    @classmethod
    def from_django_model(cls, obj):
        return cls(pk=obj.pk,
                   name=obj.name,
                   email=obj.email)

    def save_to_db(self):
        return DjangoUser.create(**self)
```

1.3.2 Rules

In PyKnow a **rule** is a callable, decorated with *Rule*.

Rules have two components, LHS (left-hand-side) and RHS (right-hand-side).

- The *LHS* describes (using **patterns**) the conditions on which the rule * should be executed (or fired).
- The *RHS* is the set of actions to perform when the rule is fired.

For a *Fact* to match a *Pattern*, all pattern restrictions must be **True** when the *Fact* is evaluated against it.

```
class MyFact(Fact):
    pass

@Rule(MyFact())  # This is the LHS
def match_with_every_myfact():
    """This rule will match with every instance of `MyFact`."""
    # This is the RHS
    pass

@Rule(Fact('animal', family='felinae'))
def match_with_cats():
    """
    Match with every `Fact` which:

      * f[0] == 'animal'
      * f['family'] == 'felinae'

    """
    print("Meow!")
```

You can use logic operators to express complex *LHS* conditions.