Rulu Documentation

Release 0.1.0

Uri Barkai

February 27, 2016

Contents

Rulu provides a Pythonic, declarative interface for building rule-based expert systems.

Rulu is implemented over PyCLIPS, the Python wrapper of the CLIPS expert system library.

CHAPTER 1

Installation

```
pip install rulu
```

CHAPTER 2

User Documentation

2.1 Getting Started

In this demo we'll define a **Family Tree** expert system, that takes as input low-level information about family relations ("X is the father of Y", "Y is the father of Z") and **deduces** higher-level relations ("X is the grandfather of Z").

2.1.1 Step 1: Define the data model

The basic unit of information for the rule engine is the **fact**, which is simply a record containing one or more data fields. Facts are **structured**, so we need to start by defining the fields contained in each fact type. This translates to **template** definitions in CLIPS, and is analogous to **relation/table** definitions in an RDBMS.

```
from rulu import *

class IsFatherOf(Fact):
    father = StringField()
    son = StringField()
```

2.1.2 Step 2: Define rules

At the core of the expert system are the **rule definitions**. They provide the **business logic** for generating new facts based on known ones, according to various conditions.

We'll now define a rule that states the following: If *X* is the father of *Y*, and *Y* is the father of *Z*, then *X* is the grandfather of *Z*.

```
IsGrandfatherOf = RuleDef(
   match(IsFatherOf[1].son, IsFatherOf[2].father),
   action(Assert(grandfather=IsFatherOf[1].father, grandson=IsFatherOf[2].son, degree=0))
)
```

Let's break this definition to its different components:

1. This rule takes as input **two** facts of type `IsFatherOf`. They are referred to as '1' and '2' (but any name may be used).
2. The only condition is expressed in the **match** statement. It states that the `son` member of record #1 needs to equal the `father` member of record #2. Continuing the RDBMS analogy, a **match** statement is similar to a **JOIN** between tables.

3. Whenever the **match** condition holds, the rule is fired, and its actions are taken. In this case we use a single **Assert** action, which is the CLIPS terminology for creating new facts.
4. The above rule also contains an **implicit** definition for a fact type named `IsGrandfatherOf` with 3 fields (`grandfather`, `grandson`, `degree`).

Let's add another rule, to cover great-grandfathers of all degrees.

```
RuleDef(
    target(IsGrandfatherOf),
    match(IsFatherOf.son, IsGrandfatherOf.grandfather),
    action(Assert(grandfather=IsFatherOf.father,
                  grandson=IsGrandfatherOf.grandson,
                  degree=IsGrandfatherOf.degree+1))
)
```

Notes:

5. In the rule we used the **target** statement to specify explicitly the target fact type.
6. In the arithmetic expression `IsGrandfatherOf.degree+1`, it is important to note that the actual calculation will not run in Python. In fact, this example contains **pure CLIPS rules** in the sense that **no Python code** will run during rule engine execution. the Instead, Rulu will "compile" these rule definitions to CLIPS code, and CLIPS will take care of the entire run. This is great for large data sets because CLIPS (written in C) is **by far** more efficient than Python.

2.1.3 Step 3: Add input facts

```python
from rulu.engine import RuleEngine

engine = RuleEngine()
# Load the data model and rules (assuming they're defined in 'family.py')
engine.load_module('family')

engine.assert_('IsFatherOf', father='Adam', son='Seth')
engine.assert_('IsFatherOf', father='Seth', son='Enos')
engine.assert_('IsFatherOf', father='Enos', son='Kenan')
engine.assert_('IsFatherOf', father='Kenan', son='Mahalalel')
```

2.1.4 Step 4: Run the rule engine

```python
# Apply all rules while possible
engine.run()
```

2.1.5 Step 5: Read the output facts

```python
# Print output facts
for fact in engine.get_facts('IsGrandfatherOf'):
    print '{} is the {}grandfather of {}.'.format(
            fact.grandfather, 'great-'*fact.degree, fact.grandson)
```

2.1.6 Output

```
Enos is the grandfather of Mahalalel.
Seth is the grandfather of Kenan.
Adam is the grandfather of Enos.
Adam is the great-grandfather of Kenan.
Seth is the great-grandfather of Mahalalel.
Adam is the great-great-grandfather of Mahalalel.
```

2.2 Running Python Code

The rules in the previous example (see *Getting Started*) were **pure** CLIPS rules, that did not involve any Python code during the engine run. Rulu also provides easy ways to integrate with Python code, as explained here.

2.2.1 Python Actions

We can write the entire action portion of a rule in python. For example, compare the **declarative** (CLIPS) rule from the previous section:

```
IsGrandfatherOf = RuleDef(
   match(IsFatherOf[1].son, IsFatherOf[2].father),
   action(Assert(grandfather=IsFatherOf[1].father, grandson=IsFatherOf[2].son, degree=0))
)
```

With the following **procedural** (Python) form:

```
IsGrandfatherOf = RuleDef(
    fields(grandfather=String, grandson=String, degree=Integer),
    match(IsFatherOf[0].son, IsFatherOf[1].father)
)

@IsGrandfatherOf._python_action
def action(assert_, IsFatherOf):
    assert_(grandfather=IsFatherOf[0].father, grandson=IsFatherOf[1].son, degree=0)
```

Notes:

1. Python actions are defined using the `_python_action` decorator.
2. The function's parameters include an `assert_` function to create new facts, as well as all the input facts contained in the rule (in this case `IsFatherOf` accessible as a dictionary).
3. Note the explicit **fields** declaration. In the previous example, Rulu could automatically deduce the target fields from the `Assert` statement. Here the fact assertion is done in run-time, so the fields cannot be deduced up front and need to be declared.

2.2.2 Python Functions

We can use Python code even when writing rules using the declarative (CLIPS) form. The functions need to be **registered** in advance, as follows:

```
@RuleFunc
def increment(x):
    return x+1
```

```
@RuleFunc
def take_father(x):
    return x.father

RuleDef(
    target(IsGrandfatherOf),
    match(IsFatherOf.son, IsGrandfatherOf.grandfather),
    action(Assert(grandfather=take_father(IsFatherOf), grandson=IsGrandfatherOf.grandson, greatness=
)
```

Note: in this case the function return types can be deduced from context. In other cases they have to be declared explicitly using `@IntegerRuleFunc`, `StringRuleFunc` etc.

2.3 CLIPS Functions

CLIPS provides many useful functions (see chapter 12 of the CLIPS Reference Manual) for use in rule definitions. All these functions may be used in Rulu as follows:

greeting.py

```
from rulu import *

# Declare a built-in CLIPS function by name and return value.
strcat = clips_func('str-cat', String)

class Entity(Fact):
    name = StringField()

Greeting = RuleDef(
    action(Assert(text=strcat("Hello, ", Entity.name, "!")))
)
```

Input Facts

Entity	name
0	World

Output Facts

Greeting	text
0	Hello, World!

2.4 Fact I/O

In the basic example we used Python code to insert input facts into the expert system's working memory. We also used Python code to process the output facts directly. It is often useful to import/export facts from/to other formats for integration with other data processing tools (or simply for storage on disk).

2.4.1 CLIPS format

The native CLIPS format can be used for storing facts in files:

```
from rulu.engine import RuleEngine

engine = RuleEngine()
engine.load_module('family')
engine.load('family-in.clp')
engine.run()
engine.save('family-out.clp')
```

family-in.clp

```
(IsFatherOf (father "Adam") (son "Seth"))
(IsFatherOf (father "Seth") (son "Enos"))
(IsFatherOf (father "Enos") (son "Kenan"))
(IsFatherOf (father "Kenan") (son "Mahalalel"))
```

family-out.clp

```
(initial-fact)
(IsFatherOf (father "Adam") (son "Seth"))
(IsFatherOf (father "Seth") (son "Enos"))
(IsFatherOf (father "Enos") (son "Kenan"))
(IsFatherOf (father "Kenan") (son "Mahalalel"))
(IsGrandfatherOf (greatness 0) (grandson "Mahalalel") (grandfather "Enos"))
(IsGrandfatherOf (greatness 0) (grandson "Kenan") (grandfather "Seth"))
(IsGrandfatherOf (greatness 0) (grandson "Enos") (grandfather "Adam"))
(IsGrandfatherOf (greatness 1) (grandson "Kenan") (grandfather "Adam"))
(IsGrandfatherOf (greatness 1) (grandson "Mahalalel") (grandfather "Seth"))
(IsGrandfatherOf (greatness 2) (grandson "Mahalalel") (grandfather "Adam"))
```

2.4.2 Pandas dataframes

The dataframe is a very convenient structure for holding fact data:

```
>>> from rulu.rulu_io import facts_to_df
>>> facts_to_df(engine, 'IsFatherOf')

  father        son
0   Adam       Seth
1   Seth       Enos
2   Enos      Kenan
3  Kenan  Mahalalel

[4 rows x 2 columns]
```

2.4.3 Export to Excel

2.4.4 Database integration

TBD :)