PyKE

Intelligent Systems 2
2023/2024

Beatriz Marques
Tiago Pereira

universidade
de aveiro

What is PyKE

PyKE was designed to create meta-programs, or writing programs that manipulate other programs.. So that you can assemble the Python functions that you've written and customize them for a specific situation ou use-case.

That said, what is the benedict of writing code in PyKE?

Why use PyKE?

- Adaptability
  - Using Pyke allows to combine our Pytho code into several other different configurations.
- Performance
  - We use PyKE to compile the Pyke code, into Python.
- Code Reuse
  - Makes our code to be more adaptable and a order of magnitude faster allows it to be reused in a correspondingly broader range of situations.
- Automatically generate code
  - We can write PyKE code to automatically generate python code to generate HTML templates, SQL or even Linux Configurations

Potencial Pyke Applications

- Decision making applications
- Back-end of compilers
- Automatic HTML generation and SQL statement generation
- Automatic program builder to reuse common set of functions for different specific situations.
- Control Units
- Diagnosis systems

Statements

A statement in PyKE is a fact. THat said we can think of statements as pieces of information that represent relations. It could be for instance:

- “Bruce is the son of Thomas and Norma” - statement in natural language
- family.son_of(Bruce, Thomas, Norma) - statement in Pyke language

THe statements are introduced to the program by the knowledge base.

Knowledge Bases

- Knowledge Bases are repositories that contain all the knowledge for the use case.
- We saw in the last slide, that the statement are composed by three parts, that define the fact.
- We can separate the statement in:

  family.son_of(Bruce, Thomas, Norma)

  family -> Knowledge Base

  son_of -> Knowledge entity

  (Bruce, Thomas, Norma) -> Arguments of the fact (this is python data)

  But there are several types of knowledge Bases

Fact Bases

Knowledge Base that contain the facts of the program.

There are two types of Facts:

- Case specific facts
  - They will be deleted when the engine reset is done to prepare for another run of the inference engine.
    - some_engine.assert_(kb_name, fact_name, arguments)
    - some_engine.add_case_specific_fact(kb_name, fact_name, arguments)
- Universal facts
  - Universal facts are never deleted. We can save universal facts in a .kbf file
  - We can also add universal facts by:
    - some_engine.add_universal_fact(kb_name, fact_name, arguments)

Rule Bases

- Knowledge Base that contains all the rules of the program.
- A rule base can contain forward-chaining and backward-chaining.
- They can be saved in a .krb file
- Rules have inheritance, if they are in the same category

Forward-Chaining

- Run automatically when the rule base is activated to assert new facts.
- Don’t determine if the statement is true

Backward-chaining

- When a rule base name is used as the knowledge base name in a statement, the system assumes that is backward-chaining
- Used to determine if a statement is true

Question Bases

- Knowledge Base that contains questions for end user, in various forms.
- Each question base in defined in a .kqb file
- This files contain all the information about the question (question, validation and output)
- The answers are remembered by multiple rules
- We can make the question, by the terminal (ask_tty) or by a dialog box (ask_wx)

Special Base

- Knowledge Base that contains a collection of helper knowledge entities, that contain special Python functions
- THe special functions are:
  - claim_goal
  - check_command
  - command
  - general_command

Pattern Matching

There are several types of pattern matching:

- Literal patterns
  - Data values that only match themselves
- Pattern variables
  - They will match everything

Literal Matching

Just ask:

- Is Bruce the son of Thomas and Norma?

The system will look in is facts and answer yes or no

Pattern Variables

We could ask to system, to tell us all the children of Thomas and Norma, so we use $

- family.son_of($son, Thomas, Norma)

We can also ask to the system, who are the Norma's children. For that we could use a anonymous pattern variable $_:

- family.son_of($son, $_father, Norma)

In the case, that we want to know if there is any family that the father as the same name as the son, we can:

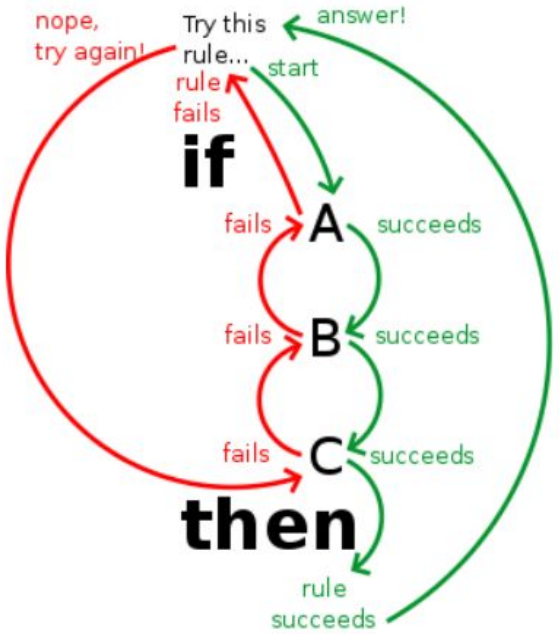
- family.son_of($father, $father, $_mother)

Rules

The concept of a rule is simple, being a simple if-the rule.

It uses the concept of backtracking, as we can see in the image in right

The concept of inference is put here, because the rules are not linked to each other. So PyKE uses forward-chaining and backward-chaining to process the rules



Plan and Automatic Program Generator

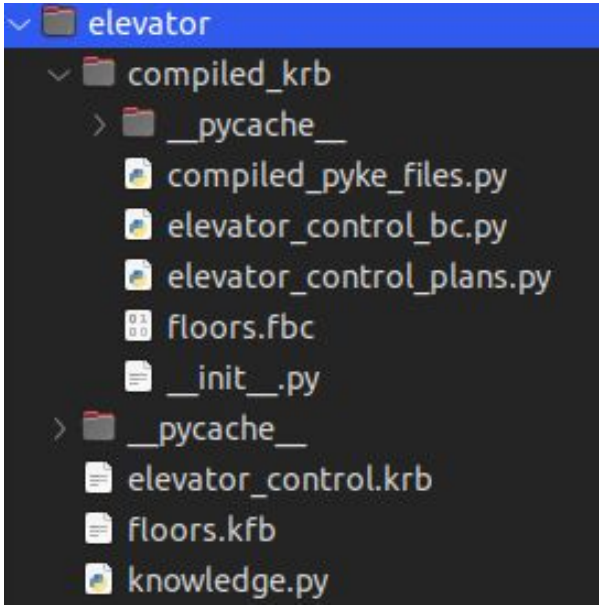
We can use PyKE to automatically combine Python functions into programs, attaching python functions to backward-chainig rules.

These function are written in the with clause at the end of the defined rule in the .krb file

We can also specifie Python functions to use a pattern variable, so that we can use PyKE to change the way that the functions are used

Demo
Elevator Exercise

Demo



The structure of our demo program is, as we can see on the left, is composed by the compiled_krb folder that is composed by the compiled python files, automatically created by PyKE.

The last three files are the actual code, made by us.

We have a .krb file that is our knowledge base of the type Rule Base.

We also have a .kfb file that is the our fact base.

THe knowledge.py is where we import the knowledge_engine from PyKE

.KFB File

Here we only need to represent all the facts from the elevator.

We decided to add in this files all the universal facts of the program, like the number of floors

So, we have:

```
1 # floors.kfb - factos universais
2
3 floor(1)
4 floor(2)
5 floor(3)
6 floor(4)
7
```

.KRB File

This file as said before is composed of all the rules of the program. THese rules are named:

- open_to_enter_action
- open_to_leave_action
- wait_to_enter_action
- wait_to leave_action
- wait_nobody_action
- up_action
- down_waiting_section
- down_action
- close_to_destiny_action
- close_action
- close_nobody_action

Open_to_enter action

Exemplo de uma ação com utilização de plans

```
#ficheiro com regras

# quando uma pessoa esta a espera e o elevador esta vazio nesse piso com a porta fechada
open_to_enter_action
    use actions($CurrentFloor,$PassengersInside,$PassengersWaiting,$WhereTo,$WhereWaiting,$Door,open)
taking (current_floor,where_to,where_waiting)
    when
        floors.floor($CurrentFloor)
        floors.floor($WhereWaiting)
        check $Door == 0
        check $CurrentFloor == $WhereWaiting
        check $PassengersInside == 0
        check $PassengersWaiting == 1
    with
        print(f"Pessoa a espera no piso {current_floor} com o elevador no piso {current_floor} com a
porta fechada")
```

But where is Python?

Now we are going to show you how to work on a python file

First we need to import the PyKE library

```
1 from pyke import knowledge_engine
```

The main

We use a simple main, composed by a loop, that uses a function to control the elevator and selects if the user wants to test the demo again

```
40      #reinicia o motor de inferencia para limpar os factos definidos
41      engine.reset()
42
43
44
45  while True:
46      control_elevator()
47      user_input = input("Do you want to test again? (yes/no): ").lower()
48      if user_input != 'yes':
49          break
```

control_elevator( ) function

We start by create a new knowledge_engine from the PyKE library and associate the archive with the knowledge engine.

```
3 def control_elevator():
4     #cria uma instância do motor de inferência da biblioteca Pyke e associa-o ao arquivo atua
5     engine = knowledge_engine.engine(__file__)
```

Load data

We update the data from the actual state

```
7     #dados do estado atual
8     CurrentFloor = int(input("Enter current floor: "))
9     PassengersInside = int(input("Enter number of passengers inside: "))
10    PassengersWaiting = int(input("Enter number of passengers waiting: "))
11    WhereTo = int(input("Enter destination floor: "))
12    WhereWaiting = int(input("Enter floor where passengers are waiting: "))
13    Door = int(input("Is the door open? (S-1/N-0)?"))
14    print()
```

Activate the knowledge base

Activate the knowledge base in the PyKE engine. It uses the rules in the .krb file

```
16    # ativa a base de conhecimento chamada 'elevator_control' no motor de inferência Pyke, utiliza as
  regras definidas no elevator_control.krb
17    engine.activate('elevator_control')
```

Add facts

Add the facts into the knowledge base

```
19      #adiciona factos a base de conhecimento
20      engine.assert_('current_floor', 'current_floor', (CurrentFloor,))
21      engine.assert_('passengers_inside', 'passengers_inside', (PassengersInside,))
22      engine.assert_('passengers_waiting', 'passengers_waiting', (PassengersWaiting,))
23      engine.assert_('where_to', 'where_to', (WhereTo,))
24      engine.assert_('where_waiting','where_waiting',(WhereWaiting,))
25      engine.assert_('door','door',(Door,))
```

Try

Try to prove and infer the rules added to the engine

```
28      try:
29          #tentar provar um objetivo no sistema
30          vals,plans = engine.prove_1_goal('elevator_control.actions($CurrentFloor,$PassengersInside,
    $PassengersWaiting,$WhereTo,$WhereWaiting,$Door,$Action)', CurrentFloor = CurrentFloor,
    PassengersInside = PassengersInside, PassengersWaiting = PassengersWaiting, WhereTo = WhereTo,
    WhereWaiting = WhereWaiting,Door= Door)
31          print('The elevator should: ' + vals['Action'])
32          print()
33          print("Plan: ")
34          plans(CurrentFloor,WhereTo,WhereWaiting)
35          print()
36      except knowledge_engine.CanNotProve:
37          print('No action possible')
38          print()
```

Reset

We need to reset the PyKE engine and clean all the specific cases from the actual program

```
40      #reinicia o motor de inferencia para limpar os factos definidos
41      engine.reset()
```

Thank you

This slides use the information from:
https://pyke.sourceforge.net/index.html