Практическая работа №8.Логическое программирование

Концепция

Логическое программирование — это сочетание двух слов: логики и программирования. Логическое программирование — это парадигма программирования, в которой проблемы выражаются в виде фактов и правил в программных утверждениях, но в рамках системы формальной логики. Как и другие парадигмы программирования, такие как объектно-ориентированная, функциональная, декларативная, процедурная и т. Д., Это также особый подход к программированию.

Логическое программирование использует факты и правила для решения проблемы. Вот почему они называются строительными блоками логического программирования. Цель должна быть указана для каждой программы в логическом программировании. Чтобы понять, как можно решить проблему в логическом программировании, нам нужно знать о строительных блоках — факты и правила. На самом деле, каждая логическая программа нуждается в фактах для работы, чтобы она могла достичь поставленной цели. Факты в основном являются правдивыми утверждениями о программе и данных. Например, Дели является столицей Индии.На самом деле, правила — это ограничения, которые позволяют нам делать выводы о проблемной области. Правила в основном написаны в виде логических положений для выражения различных фактов. Например, если мы строим какую-либо игру, то должны быть определены все правила.

Установка полезных пакетов

Для запуска логического программирования на Python нам нужно установить следующие два пакета:

Kanren

Это дает нам возможность упростить способ создания кода для бизнес-логики. Это позволяет нам выразить логику в терминах правил и фактов. Следующая команда поможет вам установить kanren

```
pip install kanren
```

SymPy

SymPy — это библиотека Python для символической математики. Она нацелена на то, чтобы стать полнофункциональной системой компьютерной алгебры (CAS), сохраняя при этом код как можно более простым, чтобы быть понятным и легко расширяемым. Следующая команда поможет вам установить SymPy —

```
pip install sympy
```

Примеры логического программирования

Математические выражения

На самом деле мы можем найти неизвестные значения, используя логическое программирование очень эффективным способом. Следующий код Python поможет вам подобрать математическое выражение. Попробуйте сначала импортировать следующие пакеты:

```
fromkanrenimportrun, var, fact
from kanren.assoccomm import eq_assoccomm as eq
from kanren.assoccomm import commutative, associative
```

Нам нужно определить математические операции, которые мы собираемся использовать

```
add = 'add'
mul = 'mul'
```

И сложение, и умножение являются коммуникативными процессами. Следовательно, нам нужно указать это, и это можно сделать следующим образом:

```
fact(commutative, mul)
fact(commutative, add)
fact(associative, mul)
fact(associative, add)
```

Обязательно определить переменные; это можно сделать следующим образом

```
a, b = var('a'), var('b')
```

Нам нужно сопоставить выражение с исходным шаблоном. У нас есть следующий оригинальный шаблон, который в основном (5 + a) * b —

```
Original_pattern = (mul, (add, 5, a), b)
```

У нас есть два следующих выражения, которые соответствуют исходному шаблону:

```
exp1 = (mul, 2, (add, 3, 1))
exp2 = (add,5,(mul,8,1))
```

Вывод может быть напечатан с помощью следующей команды —

```
print(run(0, (a,b), eq(original_pattern, exp1)))
print(run(0, (a,b), eq(original_pattern, exp2)))
```

После запуска этого кода мы получим следующий вывод —

```
((3,2))
()
```

Первый вывод представляет значения для **a** и **b** . Первое выражение соответствовало исходному шаблону и возвращало значения для **a** и **b,** но второе выражение не соответствовало исходному шаблону, поэтому ничего не было возвращено.

Проверка на простые числа

С помощью логического программирования мы можем найти простые числа из списка чисел, а также можем генерировать простые числа. Код Python, приведенный ниже, найдет простое число из списка чисел, а также сгенерирует первые 10 простых чисел.
Давайте сначала импортируем необходимые пакеты

```
from kanren import isvar, run, membero
from kanren.core import success, fail, goaleval, condeseq, eq, var
from sympy.ntheory.generate import prime, isprime
import itertools as it
```

Теперь мы определим функцию с именем prime_check, которая будет проверять простые числа на основе заданных чисел в качестве данных.

```
def prime_check(x):
if isvar(x):
   return condeseq([(eq,x,p)] for p in map(prime, it.count(1)))
else:
   return success if isprime(x) else fail
```

Теперь нам нужно объявить переменную, которая будет использоваться —

```
x = var()
print((set(run(0,x,(membero,x,(12,14,15,19,20,21,22,23,29,30,41,44,52,62,65,85)),
(prime_check,x)))))
print((run(10,x,prime_check(x))))
```

Вывод вышеприведенного кода будет следующим:

```
{19, 23, 29, 41}

(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
```

Решая головоломки

Логическое программирование может быть использовано для решения многих задач, таких как 8-головоломка, головоломка Зебра, Судоку, N-королева и т. Д. Здесь мы рассмотрим пример варианта головоломки Зебра, который выглядит следующим образом:

```
There are five houses.
The English man lives in the red house.
The Swede has a dog.
The Dane drinks tea.
The green house is immediately to the left of the white house.
They drink coffee in the green house.
The man who smokes Pall Mall has birds.
In the yellow house they smoke Dunhill.
In the middle house they drink milk.
The Norwegian lives in the first house.
The man who smokes Blend lives in the house next to the house with cats.
In a house next to the house where they have a horse, they smoke Dunhill.
The man who smokes Blue Master drinks beer.
The German smokes Prince.
The Norwegian lives next to the blue house.
They drink water in a house next to the house where they smoke Blend.
```

Мы решаем вопрос о том, кто владеет зеброй с помощью Python.

Импортируем необходимые пакеты —

```
from kanren import *
from kanren.core import lall
importtime
```

Теперь нам нужно определить две функции — **left ()** и **next (),** чтобы проверить, чей дом оставлен или рядом с чьим домом —

```
def left(q, p, list):
   return membero((q,p), zip(list, list[1:]))
def next(q, p, list):
   return conde([left(q, p, list)], [left(p, q, list)])
```

Теперь мы объявим переменную house следующим образом:

```
houses = var()
```

Нам нужно определить правила с помощью пакета lall следующим образом.Есть 5 домов

```
rules_zebraproblem = lall(
   (eq, (var(), var(), var(), var(), var()), houses),

   (membero,('Englishman', var(), var(), var(), 'red'), houses),
   (membero,('Swede', var(), var(), 'dog', var()), houses),
   (membero,('Dane', var(), 'tea', var(), var()), houses),
   (left,(var(), var(), var(), var(), 'green'),
   (var(), var(), var(), var(), 'white'), houses),
   (membero,(var(), var(), 'coffee', var(), 'green'), houses),
   (membero,(var(), 'Pall Mall', var(), 'birds', var()), houses),
   (membero,(var(), 'Dunhill', var(), var(), 'yellow'), houses),
   (eq,(var(), var(), (var(), var(), 'milk', var(), var()), var(), var()), houses),
   (eq,(('Norwegian', var(), var(), var(), var()), var(), var(), var(), var()),
houses),
   (next,(var(), 'Blend', var(), var(), var()),
```

```
   (var(), var(), var(), 'cats', var()), houses),
   (next,(var(), 'Dunhill', var(), var(), var()),
   (var(), var(), var(), 'horse', var()), houses),
   (membero,(var(), 'Blue Master', 'beer', var(), var()), houses),
   (membero,('German', 'Prince', var(), var(), var()), houses),
   (next,('Norwegian', var(), var(), var(), var()),
   (var(), var(), var(), var(), 'blue'), houses),
   (next,(var(), 'Blend', var(), var(), var()),
   (var(), var(), 'water', var(), var()), houses),
   (membero,(var(), var(), var(), 'zebra', var()), houses)
)
```

Теперь запустите решатель с предыдущими ограничениями

```
solutions = run(0, houses, rules_zebraproblem)
```

С помощью следующего кода мы можем извлечь вывод из решателя —

```
output_zebra = [house for house in solutions[0] if 'zebra' in house][0][0]
```

Следующий код поможет распечатать решение —

```
print ('\n'+ output_zebra + 'owns zebra.')
```

Вывод вышеуказанного кода будет следующим:

```
German owns zebra.
```