SRA の技術情報誌
GS LetterNeo
Vol.195　2024 年 12 月

Python で始める 論理プログラミング -pyDatalog 編-

熊澤 努

技術本部 先端技術研究室

はじめに

今回は Python で論理プログラミングに挑戦しましょう。論理プログラミングは事実や規則を列挙して宣言的に記述するスタイルのプログラミング方式で、それらの記述に対して問い合わせを行うことで計算を実行します。論理プログラミング向けのプログラミング言語 Prolog[1]が有名です。この記事では Prolog の一部の機能をサポートする Datalog を扱います。Datalog を Python の実行環境で扱うためのパッケージである pyDatalog を使ってプログラムを書いて実行してみます。pyDatalog の詳しい説明やプログラム例は公式サイト

https://sites.google.com/site/pydatalog/home

にあります。

[1] SWI-Prolog という処理系（https://www.swi-prolog.org/）が公開されています。

pyDatalog を準備する

pyDatalog は pip でインストールすることができます。

```
> pip install pyDatalog
```

今回扱うプログラムは Python 3.12.3、pyDatalog 0.17.4 で動作させています。

階乗の計算

最初に公式サイトのチュートリアル[2]にある整数の階乗を計算するプログラを動かしてみます。 以下のプログラムで 10 の階乗を計算することができます。

```
from pyDatalog import pyDatalog

# 項の定義
pyDatalog.create_terms('factorial, N')

# Nの階乗
factorial[N] = N*factorial[N-1]
factorial[1] = 1

# 10!の値
print(factorial[10]==N)
```

pyDatalog.create_terms 関数は、引数に記述した項（論理プログラミングの用語で変数や関数のこと）を生成します。項はカンマ区切りの文字列で書きます。ここでは、プログラムで使う factorial と N という 2 つの変数を生成します。以降では、これらの変数を自由に使うことができます。

factorialはリストのような添え字付きの変数で、`factorial[N]`にNの階乗($N!$)の値を格納します。一般に2以上のNについて$N! = N \times (N-1)!$という関係が成り立つことを利用して、`factorial[N]=N*factorial[N-1]`としています。$N=1$のときは例外で、$1! = 1$なので`factorial[1] = 1`となります。ここで、2つの場合の計算を条件文で分岐させて書かないことが論理プログラミングの特徴です。これらは計算の手順ではなく、factorialの持つ関係や性質を書き下したパターンを表していると考えます。プログラムの実行時には、パターンを照合して適切な関係が自動的に選択されます。簡単にいうと、factorial[5]を計算するのであれば、N=5としてfactorial[5]を5*factorial[4]に書き換えていくことで計算が一歩進みます。factorial[1]とはかっこ内の数字が一致しないので、書き換えには使われません。以下同

[2] https://sites.google.com/site/pydatalog/Online-datalog-tutorial

様にNの値を4,3,2と1ずつ小さくしながら書き換えを行い、最後にN＝１、つまりfactorial[1]が登場した段階で例外的な場合である1への書き換えが適用すれば、全体が
$5 \times 4 \times 3 \times 2 \times 1$に書き換わり、階乗の計算が無事終わります。

一番下の行が、問い合わせあるいはクエリとよばれる質問文です。問い合わせの結果を Python の print 関数で出力します。ここでの問い合わせは「factorial[10]が N と等しいような N の値を求めよ」、つまり、「10!を求めよ」という意味です。==は Python の等号で、N は未知の変数です。この未知の値を見つけるのが論理プログラミングによる計算です。

プログラムは通常の python プログラムとして.py 拡張子を付けて保存します。その後実行すると、以下のような結果出力されて、10!が正しく計算されたことがわかります。

```
N
-------
3628800
```

ちょっとわかりにくいですが。結果が表形式になっています。Datalog はリレーショナルデータベースと関係が深く、表との相性が良いのも特徴です。

なお、N を具体的な数字、例えば 100 にして、

```
print(factorial[10]==100)
```

と質問することもできます。この場合は「10!の値は 10 と等しいか?」と質問したことになります。この場合は、 No を意味する以下の結果が質問の回答として出力されます。

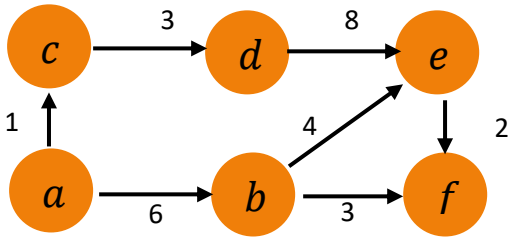
```
[]
```

有向グラフ上の探索

もう少し複雑な例として、有向非巡回グラフ（DAG）上の経路探索問題[3]を解いてみます。DAG とは、以下のような頂点（ノード）同士を矢印で接続したネットワーク構造で、かつ、矢印を繰り返し辿っても同じ頂点をぐるぐると何回も通ることがないようなものです。ここでは次の図に示した DAG を考えてみましょう。丸で表した頂点はある人のいる場所、矢印は場所から場所へ移動できるかどうかを、矢印のそばの数字は移動距離を表すものと考えま

[3] ここで扱う例は、pyDatalog の公式 GitHub にあるグラフ探索のプログラム例（https://github.com/pcarbonn/pyDatalog/blob/master/pyDatalog/examples/graph.py）を筆者が書き替えたものです。探索するグラフは、pyDatalog とは異なる論理プログラミングパッケージ pytholog の公式 GitHub (https://github.com/mnoorfawi/pytholog)の Graph Traversals with Pytholog に掲載されている問題を pyDatalog に書き換えて引用したものです。

す。今 a にいるとして、最短距離で f に移動したいとします。どんな経路（順路）をどれだけ移動すればよいでしょうか。答えは、a, b, f と辿った経路が最短で、距離は 9 です。これを pyDatalog で求めてみます。



(図は https://github.com/mnoorfawi/pytholog をもとに筆者が作成)

この問題は最短路問題といい、ダイクストラ法などの優れたアルゴリズムが知られています。ここではそういうアルゴリズムを使わず、頂点と矢印の関係性だけを使って解いてみます。

プログラムは次のページにまとめて掲載しました。Python の関数 dag_search 内に必要な記述をすべて書いています。dag_search に付けた@pyDatalog.program()は pyDatalog が提供するデコレータで、関数 dag_search 内に書いた変数を自動的に生成してくれます。では、dag_search の内容を、①から④の番号の順に簡単に説明していきます。問い合わせについては、最初の例のときと読み方は同じなので省略します。

① 上の DAG の矢印を書き下した DAG の定義です。例えば、edge(a,b,6)は「頂点 a から頂点 b は距離 6 である」ことを意味します。+記号は pyDatalog の記法で、定数の追加です。+edge(a,b,6)とすることで、a と b の間の矢印の存在を事実として記述しています。他の矢印の定義も同様です。

② `reachable(X,Y,W)`は「頂点 X から頂点 Y に矢印を繰り返し辿ることで、距離 W で到達できる」ことを意味します。`reachable` は二つのパターンから成りますが、a と b のように X と Y が隣り合っているときが第一のパターンです。隣り合っている場合は `edge` にある情報をそのまま使うだけです。ここで、<=は不等号ではなく、右辺が成り立つならば左辺も成り立つという関係性（含意）を表します。つまり、「`edge(X,Y,W)`ならば `reachable(X,Y,W)`である」、言い換えると「X から Y への距離 W の矢印があれば、X から Y まで距離 W で到達できる」という関係性が成り立つということです。わかりにくい場合は「`reachable(X,Y,W)`とは `edge(X,Y,W)`のことである」と思っても構いません。第二の少し複雑なパターンは、a と e のように X と Y が隣り合っていない場合です。a から e へは、b を経由するか、c と d を経由する必要があります。`reachable` は再帰的に記述しており、X から経由する頂点 Z まで到達可能で、かつ(記号&)、Z から Y に直接移れれば到達可能というわけです。距離 W は W1（X から Z までの距離）と W2(Z から Y までの距離)の合計と等しいです。

```
from pyDatalog import pyDatalog

@pyDatalog.program()
def dag_search():
    # ①DAGの定義
    +edge(a,b,6)
    +edge(a,c,1)
    +edge(b,e,4)
    +edge(b,f,3)
    +edge(c,d,3)
    +edge(d,e,8)
    +edge(e,f,2)

    print('################################################')
    print('# ② XからYに距離Wで到達できるか?')
    print('################################################')

    reachable(X,Y,W) <= edge(X,Y,W)
    reachable(X,Y,W) <= edge(X,Z,W1)&reachable(Z, Y, W2)&(W==W1+W2)&(X!=Y)

    print('aからfへの距離はいくつか?')
    print(reachable(a,f,W))

    print('aからfへの距離3で到達できるか?')
    print(reachable(a,f,3))

    print('################################################')
    print('# ③ XからYへは頂点Pを通って到達でき、その距離はWか?')
    print('################################################')

    paths(X,Y,P,W) <= edge(X,Y,W) & (P==[X,Y])
    paths(X,Y,P,W) <= paths(X,Z,P1,W2)&edge(Z,Y,W1)&(X!=Y)&(P==P1+[Y])&
(W==W1+W2)

    print('aからfへの途中の路と距離を列挙せよ')
    print(paths(a,f,P,W))

    print('aからfへの路で距離12のものを列挙せよ')
    print(paths(a,f,P,12))

    print('################################################')
    print('# ④ XからYへの最短路')
    print('################################################')

    (shortest_path[X,Y]==min_(P, order_by=W)) <= paths(X,Y,P,W)

    print('aからfまでの最短路はどれか')
    print(shortest_path[a, f]==P)
```

③ paths は頂点 X から頂点 Y まで矢印を繰り返し辿っていったときの経路です。経路は X から Y までに経由する頂点の列で表します（X と Y も含みます）。上の a から e までのように 1 つとは限りません。X と Y が隣り合っている場合が第一のパターンで、経路は(X, Y)というタプルです。プログラムでは経路を Python のリストの記法で[X, Y]と書いていますが、pyDatalog はタプルとして扱うので注意が必要です。第二のパターンが X と Y が隣り合ってない場合で、経由する頂点 Z を考える点は reachable と同じです。ただし、経路 P は、X から Z までの経路 P1 に Y を付け加えたものになります。なお、paths は経路すべてを列挙する点に注意してください。

④ 頂点 X から頂点 Y への最短路を表す shortest_path の性質を記述しています。「paths(X,Y,P,W)、つまり X から Y に距離 W の経路 P があるならば、そのような経路 P の内の最も W が小さいものが最短路である」ということを書いています。min_は pyDatalog の組込み関数で、order_by で指定した W が最小になるような P を返します。X から Y への経路をすべて列挙して、その中で距離が最小のものを最短路としているということです。

このプログラムを実行した結果を下に掲載します。

```
################################################
# ② XからYに距離Wで到達できるか?
################################################
aからfへの距離はいくつか?
W
--
9
12
14
aからfへの距離3で到達できるか?
[]
################################################
# ③ XからYへは頂点Pを通って到達でき、その距離はWか?
################################################
aからfへの途中の路と距離を列挙せよ
P                         | W
--------------------------|---
('a', 'b', 'f')           | 9
('a', 'b', 'e', 'f')      | 12
('a', 'c', 'd', 'e', 'f') | 14
aからfへの路で距離12のものを列挙せよ
P
-------------------
('a', 'b', 'e', 'f')
################################################
# ④ XからYへの最小コストの路
################################################
aからfまでの最短路はどれか
P
--------------
('a', 'b', 'f')
```

出力結果と先の図の DAG を比較すると、経路、距離、最短路が正しく求められていることがわかります。

おわりに

今回は pyDatalog を使って、論理プログラミングを Python で実践してみました。pyDatalog はオブジェクト指向の機能もサポートしていて、実体同士の関係性を記述することが可能です。機会があればこちらの機能も紹介してみたいと思います。

GSLetterNeo Vol.195
2024 年 12 月 20 日発行
発行者　株式会社 SRA 技術本部 先端技術研究室

編集者　熊澤努　方 学芬
バックナンバー https://www.sra.co.jp/public/sra/gsletter/
お問い合わせ　gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋 2-32-8

夢を。Yawaraka Innovation
やわらかいのべーしょん