Автомат (граф) представляется в формате Graphviz/DOT

Грамматика представляется в формате Graphviz/DOT с дополнительными условиями.

* Меткой ребра может быть либо терминал, либо нетерминал
* Стартовые состояния имеют метку, соответствующую названию нетерминала (label = “S”) и зелёный цвет (color = “green”)
* Финальные состояния имеют метку, соответствующую названию нетерминала (label = “S”) и фигуру doublecircle shape="doublecircle")

Пример автомата для грамматики S -> a S b | eps

digraph g

{

rankdir = LR

1[label="S", shape="doublecircle", color="green"]

2[label="S",shape="doublecircle"]

1 -> 3[label = "a"]

3 -> 4[label = "S"]

4 -> 2[label = "b"]

}

Состояние 1 является и стартовым и финальным.

Грамматику в НФХ можно представлять в произвольном формате. Думаю, я предоставлю несколько больших графов и стандартных грамматик в виде автоматов и будем на них запускать. А преобразование в НФХ можно либо реализовать, либо проделать руками над каждым из примеров.

Формат грамматики для 1 задачи. Предположим, что она сразу в “почти НФХ”.

* На одной строке одно правило.
* Разделитель левой и правой части — двоеточие
* eps — обозначение для эпсилон
* Символы разделяются произвольным кол-вом пробелов
* Пример:

N : S D

N : eps

N : t

**Оформление решения**

* Решение представляет из себя консольное приложение, принимающее на вход файл с грамматикой, файл графом и опционально путь к файлу, в который будет напечатан результат. Если файл под результат не указан, то вывод печатается в консоль. Формат вывода: на одной строке одна тройка, без скобок, разделитель — запятая. Результатом является множество троек (i,N,j) для всех нетерминалов и всех вершин, между которыми существует путь, выводимый из данного нетерминала.
* Выкладывается на GitHub. Лишние файлы (бинарники, локальные конфиги, трассы отладки и т.д.) не выкладывются.
* Снабжается скриптом сборки, который должен работать “из коробки”: прикладные зависимости докачиваются автоматически, ны вещи типа отсутствия компиляторов нужно выдавать диагностику, говорящую чего именно не хватает.
* Снабжается юнит-тестами, исполнимыми любым достаточно известным фреймворком. Должны запускаться на запуск скрипта сборки и выдавать какой-нибудь адекватный результат.

**Простые примеры для отладки.**

1. S -> a S b | a b

Digraph g {

0 -> 0 [label=”a”]

0 -> 0 [label=”b”]

}

2) E -> E + E | E \* E | n

Digraph g {

0 -> 1 [label=”n”]

1 -> 2 [label=”+”]

2 -> 3 [label=”n”]

3 -> 0 [label=”\*”]

}

3) S -> a S b | S S | a b

Digraph g {

0 -> 0 [label=”a”]

1 -> 1 [label=”b”]

0 -> 1 [label=”a”]

1 -> 0 [label=”b”]

}

**Референсные данные для тестирования на больших входах (да, цифры это просто токены)**

Грамматика Q1

S -> 4 S 3

S -> 2 S 1

S -> 4 3

S -> 2 1

Грамматика Q2

S -> B 3

S -> 3

B -> 4 B 3

B -> 4 3

Грамматика Q3

S -> 3S 4

S -> 1 S 2

S -> 3 B 4

S -> 1 B 2

B -> 5 B | 5

Графы: <https://drive.google.com/file/d/1rKNY2sHs-15Zoe7kPBTF5X9G5mAhHzmD/view?usp=sharing>

Обратите внимание, что набор токенов в разных грамматиках разный, а набор графов будет один. В нём набор меток на рёбрах — это некое надмножество объединения всех терминальных алфавитов.

В колонке “Результат ...” указано кол-во троек вида (i, S, j) (т.е. Троек, сопутствующих путям, выводимым из нетерминала S). Нужно воспринимать их как контрольные числа.

|  |  |  |
| --- | --- | --- |
| Граф | Результат для Q1 | Результат для Q2 |
| skos | 810 | 1 |
| generations | 2164 | 0 |
| travel | 2499 | 63 |
| univ-bench | 2540 | 81 |
| atom-primitive | 15454 | 122 |
| biomedical-measure-primitive | 15156 | 2871 |
| foaf | 4118 | 10 |
| people-pets | 9472 | 37 |
| funding | 17634 | 1158 |
| wine | 66572 | 133 |
| pizza | 56195 | 1262 |