Введение в ИИ на примере языка Prolog

Расширенные возможности грамматик в Прологе

https://github.com/Inscriptor/IntroductionToAI/tree/master/pdf

Федеральное государственное автономное образовательное учреждение высшего образования
«Новосибирский национальный исследовательский государственный университет»

28 октября 2019 г.

Дополнительные аргументы

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

Вспомним грамматику, построенную в прошлый раз.

S → nounPhrase verbPhrase
nounPhrase → article noun
verbPhrase → verbExpr nounPhrase
verbExpr → modalVerb verb prep
article → a
article → the
noun → cat
noun → king
modalVerb → may
verb → look
prep → at

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

Она порождает фразы следующего вида.

- A cat may look at a king
- A cat may look at the king
- A king may look at a cat
- A king may look at the cat
- The cat may look at a king
- The king may look at a cat

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

OK, допустим, что мы хотим добавить сюда личные местоимения, которые хотим использовать в качестве подлежащего или дополнения.

- He may look at her
- A cat may look at him
- A king may look at her
- She may look at a cat

И тому подобное ...

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

Добавим немного правил для описания местоимений и правило, указывающее на то, что `nounPhrase` может состоять не только из артикля и имени существительного, но и из местоимения.

```
pro -->["he"].
pro -->["him"].
pro -->["she"].
pro -->["her"].
nounPhrase -->pro.
```

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

Такая грамматика действительно будет распознавать и генерировать фразы, которые мы хотели добавить в язык, но, вместе с тем, и другие, синтаксически некорректные с точки зрения английской грамматики.

- Him may look at a cat
- He may look at she
- The king may look at she
- The cat may look at she

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

- DCG — это модель, которую мы должны описать самостоятельно, и она не обладает никакими дополнительными знаниями о структуре языка.
- В частности, она не знает о правилах построения предложений в английском языке.
- В английском **he** и **she** — это субъектные личные местоимения (**subjective case**), и они не могут в предложении стоять на позиции объектов. Поэтому предложение *A cat may look at she* некорректно.
- Аналогично, **him** и **her** — это объектные местоимения (**objective case**), соответственно, они не могут стоять на суъектной позиции в предложении. Предложение *Him may look at a cat* также некорректное.

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

PERSONS	SINGULAR		PLURAL	
	Subjective Case	Objective Case	Subjective Case	Objective Case
1^{st} person	I	me	we	us
2^{nd} person	you	you	you	you
3^{rd} person	he,she,it	him,her,it	they	them

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

Чтобы исправить это, понадобятся дополнительные ограничения, а значит — и дополнительные правила в DCG.

```
sentense --> subjNP, verbPhrase.
subjNP --> article, noun.
subjNP --> spr.
objNP --> article, noun.
objNP --> opr.
verbPhrase --> verbExpr, objNP.
```

```
spr -->  ["he"].
spr --> ["she"].
opr --> ["him"].
opr --> ["her"].
```

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

- Данное решение работает, но не является эффективным.
- Относительно небольшие изменения в лексиконе повлекли значительные изменения в правилах грамматики.
- **He** и **him** (а также **she** и **her**) есть, по сути, одно и то же местоимение, только в разных формах.
- Данные формы обладают во многом схожими свойствами.
- Полученное решение может быть приемлемым только на очень небольших задачах, т.к. при расширении лексикона и языка мы можем столкнуться с необходимостью добавить десятки новых правил.

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

Рассмотрим следующую программу.

```
sentense -->  nounPhrase(subj),verbPhrase.
nounPhrase(_) -->  article, noun.
nounPhrase(P) -->  pro(P).
verbPhrase -->  verbExpr, nounPhrase(obj).
verbExpr -->  modalVerb, verb, prep.
modalVerb -->  ["may"].
verb -->  ["look"].
prep -->  ["at"].
```

```
article -->  ["a"].
article -->  ["the"].
noun -->  ["cat"].
noun -->  ["king"].
pro(subj) -->  ["he"].
pro(subj) -->  ["she"].
pro(obj) -->  ["him"].
pro(obj) -> ["her"].
```

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

- Данное решение **не содержит новых правил, кроме тех, что нужны для описания новых конструкций языка**.
- Можно сказать, что мы ввели новые признаки для описания различных типов предметных выражений (noun phrases).
- Дополнительные аргументы в правилах были использованы для того, чтобы показать, какие местоимения могут стоять на позиции субъекта, а какие — на позиции объекта.

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

Где располагаются дополнительные аргументы в полной версии правила? Как мы помним, операция `-->` — это всего лишь краткая запись обычных правил Пролога, где опущены два аргумента, задающие разностные списки.

```
sentense -->  nounPhrase,verbPhrase.
sentense(A,B) :- nounPhrase(A,C),verbPhrase(C,B).

sentense -->  nounPhrase(subj),verbPhrase.
sentense(A,B) :- nounPhrase(subj,A,C),verbPhrase(C,B).
```

Дополнительные аргументы

Контекстно-свободные грамматики с параметрами

Несмотря на простоту, грамматики с дополнительными аргументами являются мощным иснтрументом для описания сложных синтаксических конструкций. Имея представление о том, как задавать DCG с дополнительными параметрами, можно моделировать достаточно сложные языки и решать множество задач. Хотя DCG более не используются так широко, как до появления алгоритмов машинного обучения, не стоит недооценивать их эффективность и выразительность. В частности, DCG (и регулярные модели) до сих пор используются при разработке компиляторов, морфологических парсеров и лемматизаторов.

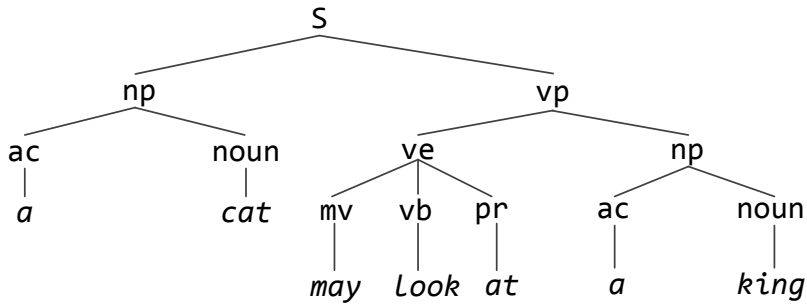
Дополнительные аргументы

Построение деревьев грамматического разбора

- До сих пор наши программы, использующие DCG, могли ответить **«да»** или **«нет»** на вопрос, является ли заданное предложение фразой языка грамматики и сгенерировать все фразы этого языка.
- Как правило, такой информации недостаточно, и мы хотим узнать, как именно происходит разбор предложений, т.е. построить дерево разбора (parse tree).

Дополнительные аргументы

Построение деревьев грамматического разбора



Дополнительные аргументы

Построение деревьев грамматического разбора

```
sentense(s(NP,VP)) -->  nounPhrase(NP), verbPhrase(VP).
nounPhrase(np(A,N)) -->  article(A), noun(N).
verbPhrase(vp(VE,NP)) -->  verbExpr(VE), nounPhrase(NP).
verbExpr(ve(MV,V,P)) -->  modalVerb(MV), verb(V), prep(P).
article(art("a")) -->  ["a"].
article(art("the")) -->  ["the"].
noun(n("cat")) -->  ["cat"].
noun(n("king")) -->  ["king"].
modalVerb(mv("may")) -->  ["may"].
verb(v("look")) -->  ["look"].
prep(p("at")) -->  ["at"].
```

Дополнительные аргументы

Построение деревьев грамматического разбора

Задав в качестве входных данных нашу обычную фразу, получим ответ `true` и полную структуру разбора.

a cat may look at a king

s(np(art("a"), n("cat")), vp(ve(mv("may"), v("look"), p("at")), np(art("a"), n("king")))).

Очевидно, что такой вид дерева разбора не является обязательным и единственно верным. Для решения задач можно описывать дерево так, как это наиболее удобно для дальнейшей его обработки.

Дополнительные цели в правилах

Дополнительные цели в правилах

Реализация контекстно-зависимой грамматики

- Рассмотрим формальный язык $a^n b^n c^n - \{\varepsilon\}$. Можно доказать, что данный язык **не является контекстно-свободным**.
- Невозможно описать грамматику, порождающую данный язык, не используя дополнительные параметры. В качестве такого параметра возьмем счетчик букв a, b и c.
- `s --> ablock(Count),bblock(Count),cblock(Count).`
- Блоки из нуля элементов выражаются пустыми строками. Остается определить переход от блока длины $n+1$ к блоку длины n. Но для этого необходимо задать дополнительные условия.

Дополнительные цели в правилах

Реализация контекстно-зависимой грамматики

Дополнительные условия записываются в фигурных скобках.

```
ablock(0) -->  [].
ablock(NewCount) -->  [a],ablock(Count), {NewCount is Count + 1}.
bblock(0) -->  [].
bblock(NewCount) -->  [b],bblock(Count), {NewCount is Count + 1}.
cblock(0) -->  [].
cblock(NewCount) -->  [c],cblock(Count), {NewCount is Count + 1}.
```

Дополнительные цели в правилах

Реализация контекстно-зависимой грамматики

Раскрыв «синтаксический сахар», увидим, как выглядят правила Пролога, соответствующие правилам грамматики.

```
ablock(NewCount,A,B) :- 'C'(A, a, C),
                        ablock(Count, C, B),
                        NewCount is Count + 1.
```

Правило `'C'(A, a, C)` означает, что мы получим список `C` из списка `A`, убрав `a`.

Дополнительные цели в правилах

Разделение правил и лексикона

- Под разделением правил и лексикона мы будем подразумевать удаление из правил грамматики всех упоминаний конкретных слов. Вместо этого, вся информация о доступных словах записывается в отдельную структуру.
- Во-первых, такой подход позволяет более гибко использовать грамматику, так как правила описывают лишь *общие синтаксические шаблоны*, что вполне логично и теоретически позволяет переиспользовать грамматику на разных лексиконах.
- Во-вторых, реальные лексиконы могут включать десятки и сотни тысяч слов, для каждого из которых доступно большое количество информации, включая фонологические, морфологические и семантические свойства. Таким образом, мы описываем грамматику, как самостоятельную сущность, и затем используем хранимый отдельно лексикон для эффективного доступа к информации, в частности используя **First Argument Indexing**.

Выводы

- Как мы увидели, DCG имеют множество достоинств и позволяют решать задачи, касающиеся анализа как формальных, так и естественных языков.
- Решение реальных задач, как правило, требует введения большого числа дополнительных параметров, что делает правила очень громоздкими.
- Кроме того, на недостатки самих DCG накладываются и недостатки Пролога, например зацикливание леворекурсивных правил.
- При этом, DCG достаточно легко освоить и использовать для описания языков и правил. Язык Prolog предоставляет простые и удобные инструменты для описания DCG. Реализация простой грамматики, подобной той, что была рассмотрена выше, на императивном языке (например C++), потребует гораздо больше времени и строк кода. Это будет проще на функциональном языке (LISP или Haskell), однако их освоение с нуля требует больше времени, чем освоение языка Prolog.

Задачи для самостоятельной работы

Задачи для самостоятельной работы

Усовершенствуйте распознаватель S-выражений таким образом, чтобы превратить его в парсер. В числе прочего, внесите следующие изменения:

1. Добавьте в список доступных спецсимволов символ * (звездочка).
2. Конструкции вида `()` и `[]` должны быть синтаксически некорректными, в отличие от `{}`.
3. Реализовать распознавание **программ** — последовательностей вида `{Context}_Body`, где в фигурных скобках указан контекст Г, символом `_` (underscore) обозначен любой из доступных разделителей, а `Body` — S-выражение.
4. Пользуясь построенным при помощи парсера деревом синтаксического разбора, реализовать вывод типов S-выражений, являющихся телом программы, правилами расширенного алгоритма Хиндли-Милнера.

Задачи для самостоятельной работы

Проверка

Откройте файл `tasks.clj`. В нем содержатся проверочные задания. Для заданий с 1 по 13 приведены типы соответствующих выражений. Задания 14–16 это «задачи на 5».

1. Предопределенными типами считаются `:Int`, `:Str`, `:Bool` и `:List`.
2. Тестовые программы должны быть прочитаны из файла.
3. Для каждой тестовой программы в файл с результатами тестов должны быть выведены ее номер, полученный алгоритмом тип, правильный ответ и результат сравнения полученного ответа с правильным.

Задачи для самостоятельной работы

Допущения

Допускается разделение тестовых программ по отдельным файлам.

Допускается замена символов переноса строки в тестовых программах на другие разделители (т.е. можно записать программу в одну строчку).

Допускается вывод результатов на экран.