Тема 8. Основы языка ASP

Answer set programming (ASP) – форма декларативного программирования, ориентированная на сложные (в основном NP-сложные) задачи поиска. Используется для разработки автономных агентов, программирования роботов, формирования расписаний и пр. Поддерживает интеграцию с Python, C++, Prolog и др.

ASP анализирует проблему как набор фактов, описывает проблему с использованием правил и формирует решение в виде стабильной модели правил и фактов. ASP компилирует проблему в виде логической программы. ASP в процессе формирования стабильной модели непрерывно изменяет логическую программу.

Современные решатели наборов ответов (answer set solvers) работают с программами без переменных. Соответственно, осуществляется преобразование входной программы с переменными первого порядка в эквивалентную, но уже без переменных. Соответствующий преобразователь («grounder») – gringo. В качестве решателя могут использоваться clasp, claspfolio, или clingcon.

Программа объединяет в себе gringo и clasp (https://potassco.org/clingo/). Программа **clingo** находится в разделе Объявления курса «Логическое программирование» в системе Moodle.

В программе используются классические термы логического программирования: числа, константы, строки, переменные, токен «_», дополнительно #sup (для минимальных значений) и #inf (для максимальных значений). Переменные пишутся с большой буквы или знака «_», атомы – с маленькой буквы. Символ «\» означает переход на другую строку, допустимо экранирование вида «\\», «\n», «\"».

Функции – составные термы, например, `at(alex,time(11),X).`

Кортеж – функция без имени. Кортеж записывается в круглых скобках, например, `(at, alex,time(11),X).`

Число: `(14)`. Кортеж, состоящий из одного элемента: `(14,)`

Программа состоит из следующих конструкций:

Факт: `H0.`

Правило: `H0 :- T1, …, Tn.`

Ограничения целостности: `:- T1, …, Tn.`

Важно, что ограничения не могут выполняться одновременно. Ограничения используются для **удаления** «лишних» фактов (они ничего не «порождают»). В качестве «ИЛИ» используется «**;**», в качестве «И» – «**,**».

Пример программы:

```
a :- b.
b :- a.
```

Её результат – пустое множество.

```
Solving...
Answer: 1
SATISFIABLE
Models       : 1
Calls        : 1
Time         : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat:
0.00s)
CPU Time     : 0.000s
```

Основной подход: (1) сгенерировать потенциально стабильную модель, (2) удалить некорректные кандидаты. В целом логическая программа на ASP:

Программа = Данные + Генератор + Тестер (+ Оптимизатор)

Другая программа:

```
a :- not b.
b :- not a.
```

В данном случае в набор ответов входит либо a, либо b, пустой набор ответов некорректен, набор ответов, включающий a и b – некорректен.

Ответ clingo (**пример запуска: clingo.exe 0 ab_program.lp**):

```
Answer: 1 b
Answer: 2 a
SATISFIABLE
```

Если запустить **clingo.exe ab_program.lp**, то в ответе будет «a», но будет указано, что количество моделей «1+»

Рассмотрим программу с фактом, правилами и ограничением целостности.

```
a :- not b.
b :- not a.
c.
:- c, not b.
```

В данном случае «c» - факт, поэтому «b» должно быть истинно, соответственно, «a» - ложно.

```
Answer: 1
c b
SATISFIABLE
Models       : 1
```

Менее схематичный пример:

```
fly(X) :- bird(X), not neg_fly(X).
neg_fly(X) :- bird(X), not fly(X).
neg_fly(X) :- penguin(X).
% Факты
bird(tweety). chicken(tweety).
bird(tux). penguin(tux).
```

В данном примере появились переменные. Когда grounder будет «избавляться» от переменных, он получит следующие результаты:

```
fly(tweety) :- bird(tweety), not neg_fly(tweety).
fly(tux) :- bird(tux), not neg_fly(tux).
neg_fly(tweety) :- bird(tweety), not fly(tweety).
neg_fly(tux) :- bird(tux), not fly(tux).
neg_fly(tweety) :- penguin(tweety).
neg_fly(tux) :- penguin(tux).
```

В результате упрощения получаем:

```
fly(tweety) :- not neg_fly(tweety).
neg_fly(tweety) :- not fly(tweety).
neg_fly(tux).
```

Тогда наш результат:

```
Answer: 1
bird(tweety) chicken(tweety) bird(tux) penguin(tux)
```

```
neg_fly(tux) fly(tweety)
      Answer: 2
      bird(tweety) chicken(tweety) bird(tux) penguin(tux)
neg_fly(tux) neg_fly(tweety)
      SATISFIABLE
      Models       : 2
```

В ASP есть константы, истина #true и ложь #false.

```
      #true.
      not #false.
      not not #true.
      % Ограничения
      :- #false.
      :- not #true.
      :- not not #false.
```

Написанная выше программа – истинна.

Что будет результатом программы, приведённой ниже (файл «arithf.lp»)?

```
      left(7).
      right(2).
      plus(L + R) :- left(L), right(R).
      minus(L - R) :- left(L), right(R).
```

Предлагаемый вариант запуска: **clingo.exe - -text arithf.lp**

Правильный ответ:

```
      left(7).
      right(2).
      plus(9).
      minus(5).
```

При выполнении унификации могут проверяться дополнительные условия в теле правил, в качестве значений – использоваться любые термы.

Для обеспечения компактной записи поддерживаются интервалы чисел, записываемые как i..j (интервал чисел от i до j включительно).

```
      size(2).
      grid(1..S,1..S) :- size(S).
```

Выдаст следующий результат:

```
      size(2).
      grid(1,1).
      grid(2,1).
      grid(1,2).
      grid(2,2).
```

Знак «;» обеспечивает задание альтернатив. Соответственно, того же результата можно добиться следующей программой:

```
      grid((1;2),(1;2)).
```

В качестве значений альтернатив могут быть любые логические термы.

Знак «:» позволяет задавать условия. Формат: L0: L1, …, Ln.

При задании в «теле» – все варианты значений должны выполняться одновременно, при задании в «голове» задаёт альтернативы. Пример:

```
      person(jane).  person(john).
      day(mon).  day(tue).  day(wed).  day(thu).  day(fri).
```

```
available(jane) :- not on(fri).
available(john) :- not on(mon), not on(wed).
meet :- available(X) : person(X).
on(X) : day(X) :- meet.
```

При этом 5 и 6 строка будут преобразованы в

```
meet :- available(jane), available(john).
on(mon); on(tue); on(wed); on(thu); on(fri) :- meet.
```

И результатом будут:

```
Answer: 1
person(jane) person(john) day(mon) day(tue) day(wed) day(thu)
day(fri) available(jane) available(john) meet on(thu)
Answer: 2
person(jane) person(john) day(mon) day(tue) day(wed) day(thu)
day(fri) available(jane) available(john) meet on(tue)
SATISFIABLE
```

При задании условий для окончания перечисления правой части после «:» используется знак «;». Пример:

```
set(1..4).
next(X,Z) :- set(X), #false : X < Y, set(Y), Y < Z; set(Z), X
< Z.
```

В данном случае условие не верно (`#false : X < Y, set(Y), Y < Z`), если между X и Z может поместиться Y.

```
Answer: 1
set(1) set(2) set(3) set(4) next(1,2) next(2,3) next(3,4)
SATISFIABLE
```

Важно! Переменные, унифицированные внутри условия, не имеют значений за пределами условия.

Функции-агрегаторы позволяют генерировать факты с учётом заданных условий. В качестве функций выступают: #count, #sum, #sum+, #min, #max.

```
10 <= #sum { 4 : course(db); 6 : course(ai); 8 :
course(project); 3 : course(xml) } < 15.
```

Результат:

```
Answer: 1 course(db) course(ai)
Answer: 2 course(db) course(ai) course(xml)
Answer: 3 course(db) course(project)
Answer: 4 course(project) course(xml)
Answer: 5 course(ai) course(project)
SATISFIABLE
```

Вариант применения с #count:

```
#count{ 4 : course(db); 6 : course(ai); 8 : course(project); 3
: course(xml) } = 1.
```

Другой вариант записи с тем же результатом:

```
{ course(db); course(ai); course(project); course(xml) } = 1.
```

Или:

```
1 { course(db); course(ai); course(project); course(xml) } 1.
```

В данном случае число слева указывает нижнюю границу количества, правое число – верхнюю границу количества.

Аналогично:

```
{ a; b; c }.
#sum { 1 : a; 2 : b; 3 : c } > 5.
```

Даёт:

```
Answer: 1
a b c
SATISFIABLE
```

ASP предлагает инструменты оптимизации, т.н. мягкие ограничения (#minimize, #maximize). Формат записи: `#minimize{w@p,t : L, ... , w@p,t : L}`. В данном случае минимизация будет выполняться по весам `w` с учётом приоритетов `@p`.

Пример:

Дано пять отелей, их «звёздность», стоимость и шумность.

```
{ hotel(1..5) } = 1.
star(1,5). cost(1,170).
star(2,4). cost(2,140).
star(3,3). cost(3,90).
star(4,3). cost(4,75).  main_street(4).
star(5,2). cost(5,60).
noisy :- hotel(X), main_street(X).
#maximize { Y@1,X : hotel(X), star(X,Y) }.
#minimize { Y/Z@2,X : hotel(X), cost(X,Y), star(X,Z) }.
:~ noisy. [ 1@3 ]
```

В последней строке – «мягкое ограничение», оно применяется с учётом приоритета и может быть заменено следующей минимизацией:

```
#minimize { 1@3 : noisy }.
```

Последнее ограничение говорит, что не нужен шумный отель. В первую очередь требуется максимизировать звёздность, во вторую – минимизировать отношение стоимости на звездность.

```
Answer: 1
... hotel(1)
Optimization: 0 34 -5
Answer: 2
... hotel(3)
Optimization: 0 30 -3
OPTIMUM FOUND
```

Из программы видно, что под требование минимизации подходят два отеля №3 и №5, но у отеля №3 выше «звёздность». В данном случае многоточием заменены результаты star(1,5) и т.д.

Комментарии пишутся с использованием символа «`%`», для многострочных комментариев используется символ открытия «`%*`» и символ закрытия «`*%`» комментариев.

Директива `#show` ограничивает результаты вывода. Для примера с отелями:

```
#show hotel/1.
```

При использовании данной директивы в качестве результата будут выведены только hotel(1) и hotel(3).

Вычисление чисел Фибоначчи.

```
#const n=10.
```

```
number(1..n).
fib(0, 1).
fib(1, 1).
fib(N, X1 + X2) :- number(N), N > 1, fib(N - 1, X1), fib(N - 2, X2).
#show fib/2.
```

Результат:

```
fib(0,1) fib(1,1) fib(2,2) fib(3,3) fib(4,5) fib(5,8) fib(6,13) fib(7,21) fib(8,34) fib(9,55) fib(10,89)
```

Директива `#const` позволяет задавать константы, которые могут изменяться в командной строке.

```
#const x = 42.
#const y = f(x,z).
p(x,y).
```

Результат:

```
p(42,f(42,z))
```

Замена константы: `clingo.exe --text -c x="2+2*2" -c z=7 1.lp`

```
p(6,f(6,7)).
```

Сравнение с Прологом.

```
on(a,b).
on(b,c).
above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).
```

Пример вопроса:

```
|?- above(a,c).
true ?
yes
|?-
```

Нерабочая программа для Пролога:

```
on(a,b).
on(b,c).
above(X,Y) :- above(X,Z), on(Z,Y).
above(X,Y) :- on(X,Y).
```

В случае выполнения программы в ASP будет получен результат:
on(a,b) on(b,c) above(a,b) above(b,c) above(a,c)

Пролог	ASP
Ориентация на вопросы	Создание моделей
Сверху вниз	Снизу вверх
Язык программирования	Язык написания моделей
Унификация, наследуемые термы	Создание экземпляров, «плоские» термы