Тема 7. Анализ структуры термов. Constraint-программирование

Анализ структуры термов

Встроенные предикаты для анализа структуры термов:

- `functor(Term, Name, Arity)` – проверяет для Term, что Name – имя функтора Term, а Arity – количество параметров.
- `arg(N, Term, Argument)` – возвращает N-ый параметр функтора Term в переменной Argument.
- `Term =.. [Name| ArgumentList]` – создаёт функтор Term с именем Name и параметрами ArgumentList.
- `name(Atom, CharacterCodes)` – превращает атом Atom в список ASCII кодов в переменной CharacterCodes.

Примеры использования:

```
?- functor(test(e,r), Name, Arity).
Arity = 2
Name = test
yes
?- arg(3, test(a, b, c, d), Value).
Value = c
yes
?- test(2, f) =.. L.
L = [test,2,f]
yes
?- name(atom, List).
List = [97,116,111,109]
yes
```

Предикат «`=..`» называется univ. Он работает в обе стороны – как разбивает функтор на составные части, так и собирает его из списка.

Предикат `name` работает в обе стороны – как формирует список ASCII кодов по атому, так и формирует атом на основе ASCII кодов в списке.

Предположим задачу: необходимо написать программу, которая позволяет умножать все параметры функтора на 3 (считаем, что все параметры являются числовыми). Решение:

```
1. triple(F, F3):-
     F =.. [Name | P], triple(P, P3), F3 =.. [Name | P3].
2. triple([], []).
3. triple([X|T], [Y|T3]) :- triple(T, T3), Y is X * 3.
```

В первой строчке осуществляется разбиение функтора, вызов процедуры умножения для параметров и сборка нового функтора. Вторая строчка – «база индукции» – пустые списки параметров. В третьей строчке – «индукционный переход» – отделили голову ([X | T]), посчитали для хвоста (triple[T, T3]), голову утроили (Y is X * 3).

Constraint-программирование

Программирование в ограничениях (CLP программирование) позволяет задать математические ограничения задачи и не задумываться об алгоритме решения. Фактически, для решения будет использоваться не обычный полный перебор, а метод ветвей и границ. Различают CLP программирование в пространстве целых чисел, рациональных чисел, вещественных чисел и на ограниченных множествах. В настоящее время разработаны расширения для других языков программирования, таких как Java и C++.

GNU Prolog поддерживает два варианта CLP: целочисленный и на ограниченных множествах.

Математические равенства и неравенства, которые должны участвовать в CLP, снабжаются знаком #.

Например, равно «#=», больше «#>», меньше «#<».

При этом

(1) данные ограничения могут применяться к переменным, которым ещё не присвоены значения,

(2) в качестве результата можно получить либо конкретные значения, либо интервалы допустимых значений.

```
?- A #> 0, A #< 3.
A = _#2(1..2)
yes
?- A #> 0, A #< 2.
A = 1
yes
```

Обратите внимание: если указать обычные знаки «больше» и «меньше», то Пролог выдаст сообщение об ошибке, т.к. переменная не унифицированная.

На ограниченных множествах программирование называется CLP(FD) – finite domain.

Предикаты GNU Prolog:

`fd_all_different(L)` – в списке L перечисляются переменные, которые будут использоваться в CLP(FD), при этом переменным не должны быть присвоены значения. Предикат fd_all_different означает, что все переменные должны принимать разные значения.

`fd_domain(L, Min, Max)` – в списке L перечисляются переменные, Min и Max задают минимальное и максимальное значения, которые они могут принимать.

`fd_labeling(L)` – применяет метод ветвей и границ для подбора значений для переменных из списка L, используя ограничения из множеств.

Рассмотрим задачу Send more money.

```
    S E N D
+   M O R E
-----------
= M O N E Y
```

В данном примере сложения «в столбик» разные переменные должны принимать разные значения от 0 до 9, естественно S и M не равные нулю. Задачу можно решить и вручную, но это долго. Перебор «в лоб» всех возможных значений – тоже довольно длительный процесс. Решение же с использованием CLP занимает доли секунды.

```
go :-
     LD = [S, E, N, D, M, O, R, Y],
     fd_all_different(LD),
     fd_domain(LD, 0, 9),
     fd_domain([S, M], 1, 9),
     1000 * S + 100 * E + 10 * N + D + 1000 * M + 100 * O + 10
* R + E #= 10000 * M + 1000 * O + 100 * N + 10 * E + Y,
     fd_labeling(LD),
     write(LD).
```

В результате:

```
[S,E,N,D,M,O,R,Y] <==> [9,5,6,7,1,0,8,2]
```

И это единственное решение данной задачи.

Аналогично может быть решена задача:

```
  D O N A L D
+ G E R A L D
-------------
= R O B E R T
```

Данная задача тоже имеет одно решение:

```
[D,O,N,A,L,G,E,R,B,T] <==> [5,2,6,4,8,1,9,7,3,0]
```

Рассмотрим задачу решения Судоку. Правила: дана таблица 9x9, разделённая на 9 клеток размером 3x3. В неё записываются цифры от 1 до 9 таким образом, что

1. в каждой строке цифры не повторяются;
2. в каждом столбце цифры не повторяются;
3. в каждой клетке 3x3 цифры не повторяются.

В исходных условиях задачи задаются некоторые из начальных цифр и человек решает Судоку, пытаясь посчитать или угадать остальные числа. В общем случае каждая задача Судоку имеет только одно правильное решение.

Пронумеруем ячейки Судоку, как если бы они задавались в Excel: A1, A2 и т.д., тогда для решения Судоку может использоваться следующая программа.

```
go :-
LD =
[A1,B1,C1,D1,E1,F1,G1,H1,I1,
 A2,B2,C2,D2,E2,F2,G2,H2,I2,
 A3,B3,C3,D3,E3,F3,G3,H3,I3,
 A4,B4,C4,D4,E4,F4,G4,H4,I4,
 A5,B5,C5,D5,E5,F5,G5,H5,I5,
 A6,B6,C6,D6,E6,F6,G6,H6,I6,
 A7,B7,C7,D7,E7,F7,G7,H7,I7,
 A8,B8,C8,D8,E8,F8,G8,H8,I8,
 A9,B9,C9,D9,E9,F9,G9,H9,I9],
fd_domain(LD, 1, 9),
% Ограничения на строки
fd_all_different([A1,B1,C1,D1,E1,F1,G1,H1,I1]),
fd_all_different([A2,B2,C2,D2,E2,F2,G2,H2,I2]),
fd_all_different([A3,B3,C3,D3,E3,F3,G3,H3,I3]),
```

```
    fd_all_different([A4,B4,C4,D4,E4,F4,G4,H4,I4]),
    fd_all_different([A5,B5,C5,D5,E5,F5,G5,H5,I5]),
    fd_all_different([A6,B6,C6,D6,E6,F6,G6,H6,I6]),
    fd_all_different([A7,B7,C7,D7,E7,F7,G7,H7,I7]),
    fd_all_different([A8,B8,C8,D8,E8,F8,G8,H8,I8]),
    fd_all_different([A9,B9,C9,D9,E9,F9,G9,H9,I9]),
    % Ограничения на столбцы
    fd_all_different([A1,A2,A3,A4,A5,A6,A7,A8,A9]),
    fd_all_different([B1,B2,B3,B4,B5,B6,B7,B8,B9]),
    fd_all_different([C1,C2,C3,C4,C5,C6,C7,C8,C9]),
    fd_all_different([D1,D2,D3,D4,D5,D6,D7,D8,D9]),
    fd_all_different([E1,E2,E3,E4,E5,E6,E7,E8,E9]),
    fd_all_different([F1,F2,F3,F4,F5,F6,F7,F8,F9]),
    fd_all_different([G1,G2,G3,G4,G5,G6,G7,G8,G9]),
    fd_all_different([H1,H2,H3,H4,H5,H6,H7,H8,H9]),
    fd_all_different([I1,I2,I3,I4,I5,I6,I7,I8,I9]),
    % Ограничения на клетки 3 на 3
    fd_all_different([A1,B1,C1,A2,B2,C2,A3,B3,C3]),
    fd_all_different([A4,B4,C4,A5,B5,C5,A6,B6,C6]),
    fd_all_different([A7,B7,C7,A8,B8,C8,A9,B9,C9]),
    fd_all_different([D1,E1,F1,D2,E2,F2,D3,E3,F3]),
    fd_all_different([D4,E4,F4,D5,E5,F5,D6,E6,F6]),
    fd_all_different([D7,E7,F7,D8,E8,F8,D9,E9,F9]),
    fd_all_different([G1,H1,I1,G2,H2,I2,G3,H3,I3]),
    fd_all_different([G4,H4,I4,G5,H5,I5,G6,H6,I6]),
    fd_all_different([G7,H7,I7,G8,H8,I8,G9,H9,I9]),
    % Решение задачи
    fd_labeling(LD),
    % Вывод результата на экран
    write([A1,B1,C1,D1,E1,F1,G1,H1,I1]), nl,
    write([A2,B2,C2,D2,E2,F2,G2,H2,I2]), nl,
    write([A3,B3,C3,D3,E3,F3,G3,H3,I3]), nl,
    write([A4,B4,C4,D4,E4,F4,G4,H4,I4]), nl,
    write([A5,B5,C5,D5,E5,F5,G5,H5,I5]), nl,
    write([A6,B6,C6,D6,E6,F6,G6,H6,I6]), nl,
    write([A7,B7,C7,D7,E7,F7,G7,H7,I7]), nl,
    write([A8,B8,C8,D8,E8,F8,G8,H8,I8]), nl,
    write([A9,B9,C9,D9,E9,F9,G9,H9,I9]).
```

Для решения конкретной задачи Судоку перед вызовом fd_labeling достаточно указать значения известных переменных. Например, так: A1 #= 7, B1 #= 4 и т.д. А в приведённом варианте программа переберёт все возможные Судоку.

При решении данной задачи с использованием полного перебора всех вариантов времени потребуется существенно больше.