Programação Declarativa

Paradigmas de Programação Avançados

Salvador Abreu, Universidade de Évora, 2019/20

Programação em Lógica com Restrições

Constraint Logic Programming

Problemas

Formulação

Integração com Prolog

Programação por Restrições

Formular problema como conjunto de relações entre variáveis.

(semelhante a sistemas de equações)

Variáveis em número finito.

Domínios das variáveis especificados:

Reais

Domínios finitos

Intervalos

Terminologia

CSP ≈ Constraint Satisfaction Problem

Um CSP **P = (V, D, C)**

Em que:

- **V** é um conjunto de variáveis { V1, V2, … }
- **D** é um conjunto de domínios das variáveis correspondentes, i.e. Di representa os valores admissíveis para Vi
- **C** é um conjunto de relações sobre V*, ditas *constraints* ou restrições

Abordagem - modelação

Encontrar **variáveis** para descrever o **estado** do problema.

Caracterizar o que constitui uma **solução**.

Eventualmente caracterizar a **qualidade** duma solução.

Variável

O que constitui uma variável num sistema de restrições?

> **Domínio**: especificação dos valores possíveis, *independentemente* das condições do problema estarem satisfeitas

Exemplos

- Inteiros,
- Reais,
- Herbrand (termos Prolog),
- **Domínios finitos,**
- Conjuntos,
- …

Restrições

São as relações (booleanas) que se devem observar entre variáveis.

Chamam-se **restrições** ou ***constraints***.

Exemplos de restrições (variáveis X, Y, …):

- $X \neq Y$
- $X = A*Y*Y + B*Y + C$
- $X > 10$
- $0 < X, X < 2*Y$
- ...

Restrições globais (global constraints)

Chamam-se restrições globais (global constraints) a relações mais elaboradas envolvendo várias variáveis.

Exemplo: **all_different**

Dizemos que a constraint **all_different(V1, V2, … Vn)** é satisfeita quando todos os Vi tomarem valores diferentes entre si.

É equivalente a termos uma série de constraints Vi ≠ Vj, só que é implementada de forma especial por cada sistema.

Ver o catálogo de constraints globais:

http://sofdem.github.io/gccat/index.html

Programação por restrições: sistemas

Há vários sistemas de constraint programming.

1. Extensão a linguagem (tipo DSL)

   Constraint Logic Programming ≈ Prolog + constraints (CLP)

   - **GNU Prolog (Prolog + CLP/FD)**
   - Eclipse CLP
   - Pacotes CLP para Prolog

2. Livraria (tipo conjunto de classes)

   Integra-se numa linguagem, sem introduzir novos conceitos. Exemplos:

   - Gecode (C++)
   - **Choco (Java)**

CLP(FD)

Constraint Logic Programming (Finite Domains)

Aumentamos o Prolog substituindo a **unificação** pela **satisfação de restrições**.

Novo tipo de variável: ***constraint variable***.

Tem um domínio. No caso FD são ***domínios finitos***. Tem um número finito (normalmente “pequeno”) de valores possíveis distintos.

As variáveis passam a ser declaradas:

- **`fd_domain(VARS, MIN, MAX)`**
- **`fd_domain(VARS, LISTA_VALORES)`**

Em que **`VARS`** pode ser uma variável ou uma lista de variáveis.

CLP(FD)

Constraints numéricas simples.

- **`A #= B`**
  - A e B são avaliados e é imposta a igualdade
- **`A #\= B`**
  - É imposta a desigualdade entre A e B
- **`A #< B, A #=< B, A #> B, A #>= B`**
  - Inequações entre A e B

Em todos os casos, se o goal suceder as variáveis ficam ***condicionadas***, por uma restrição que as liga.

A propagação é dita **bounds consistency**, ie. só assenta no mínimo e máximo dos valores admissíveis para as variáveis intervenientes.

Versões com propagação completa: **`#=#, #\=#, #<#, #=<#, #>#, #>=#`**

CLP(FD) - constraints globais

Algumas constraints globais:

- **`fd_all_different([V1, V2, …])`**

  Restringe todas as variáveis V1, V2, … a serem diferentes entre si (i.e. não podem ter um valor repetido)

- **`fd_element(I, [E1, E2, …], X)`**

  Restringe X a ser igual ao I-ésimo elemento da lista [E1, …] (a contar de 1)

CLP(FD) - constraints globais

Mais constraints globais:

- **`fd_atmost(N, LISTA, V)`**

  No máximo N variáveis de LISTA podem tomar o valor V.

- **`fd_atleast(N, LISTA, V)`**

  No mínimo N variáveis de LISTA devem tomar o valor V.

- **`fd_exactly(N, LISTA, V)`**

  Exatamente N variáveis de LISTA devem tomar o valor V.

Exemplos CLP(FD)

```
| ?- fd_domain(X, 0, 20).

X = _#0(0..20)

yes
| ?- fd_domain(X, 0, 20), Y #< X.

X = _#0(1..20)
Y = _#19(0..19)

yes
| ?-
```

Exemplos CLP(FD)

```
| ?- fd_domain(X, 0, 20), Y #< X, X #\= 10.

X = _#0(1..9:11..20)
Y = _#19(0..19)

yes
| ?- fd_domain(X, 0, 20), Y #< X, X #= 2*Z.

X = _#0(2..20)
Y = _#19(0..19)
Z = _#49(1..10)

yes
```

Modelos de consistência

Consistência de Arco (arc-consistency)

- Os domínios das variáveis podem ser não convexos (i.e. ter “buracos”)
- Implementação mais complexa
- Mais propagação
- Operadores da forma **#op#**
  - Ex. `2*X #=# Y+3, Z #<# 2*X*X + 5*Y*Y`

Consistência de Limites (bounds-consistency)

- Os domínios são representados por um intervalo { min .. max }
- Menos complexo de implementar
- Menos propagação (restringe menos)
- Operadores da forma **#op**
  - Ex. `2*X #= Y+3, Z #< 2*X*X + 5*Y*Y`

Exemplos CLP(FD)

```
| ?- fd_domain(X, 0, 20), Y #< X, X #=# 2*Z.

X = _#0(2:4:6:8:10:12:14:16:18:20)
Y = _#19(0..19)
Z = _#49(1..10)

yes
```

CLP(FD) - exemplos de uso

Exemplo: criptaritmética

```
  SEND
+ MORE
-------
= MONEY
```

As letras representam algarismos, que não se podem repetir.

Ideia: exprimir tudo o que sabemos como constraints.

1) Enumerar as variáveis: S, E, N, D, M, O, R, Y, que vão representar algarismos distintos.
2) Colocar a “soma” como uma constraint.
3) S e M não podem ser zero.

CLP(FD) - SEND+MORE=MONEY

Ficamos com:

```
fd_domain([S,E,N,D,M,O,R,Y], 0, 9),
S #> 0, M #> 0,
fd_all_different([S, E, N, D, M, O, R, Y])
```

Mas também:

```
D+10*(N+10*(E+10*S)) + E+10*(R+10*(O+10*M)) #=
  Y+10*(E+10*(N+10*(O+10*M)))
```

Ao lançar este goal, ficamos com várias variáveis com domínios “largos”, i.e. com mais dum valor possível, e algumas já só com um valor.

CLP(FD) - SEND+MORE=MONEY

```
| ?- fd_domain([S,E,N,D,M,O,R,Y], 0, 9), S #> 0, M #> 0,
     fd_all_different([S, E, N, D, M, O, R, Y]),
     D+10*(N+10*(E+10*S)) + E+10*(R+10*(O+10*M)) #=
       Y+10*(E+10*(N+10*(O+10*M))).

D = _#51(2..8)
E = _#17(4..7)
M = 1
N = _#34(5..8)
O = 0
R = _#102(2..8)
S = 9
Y = _#119(2..8)

yes
| ?-
```

Iterar sobre as soluções implícitas - labeling

Quando o resultado incluir variáveis de constraints que não foram reduzidas (i.e. que ainda não são ground), podemos forçar o CLP(FD) a ***enumerar*** todas as soluções individuais.

É o predicado **`fd_labeling/1`**

O seu argumento são as variáveis de constraint que devem ser sucessivamente reduzidas a um só valor, dentro dos possíveis dado o estado atual.

O que acontece é o CLP(FD) produzir todas as soluções, disponibilizando-as em backtracking.

Iterar sobre as soluções implícitas - labeling

```
| ?- fd_domain([S,E,N,D,M,O,R,Y], 0, 9), S #> 0, M #> 0,
fd_all_different([S, E, N, D, M, O, R, Y]),
     D+10*(N+10*(E+10*S)) + E+10*(R+10*(O+10*M)) #=
       Y+10*(E+10*(N+10*(O+10*M))),
     fd_labeling([S,E,N,D,M,O,R,Y]).

D = 7
E = 5
M = 1
N = 6
O = 0
R = 8
S = 9
Y = 2 ?
yes
| ?-
```

Ajeitemos o Prolog/CLP(FD)

```
| ?- X=[S,E,N,D,M,O,R,Y],
     fd_domain(X, 0, 9), S #> 0, M #> 0,
     fd_all_different(X),
     D+10*(N+10*(E+10*S)) + E+10*(R+10*(O+10*M)) #=
       Y+10*(E+10*(N+10*(O+10*M))),
     fd_labeling(X).

D = 7
E = 5
M = 1
N = 6
O = 0
R = 8
S = 9
Y = 2
```

Exemplo: rainhas

Tabuleiro de xadrez de NxN, com N rainhas, que não se ataquem.

O que significa “atacar-se”?

1. Na mesma linha
2. Na mesma coluna
3. Na mesma diagonal

R1	1	2	3	4
R2	1	2	3	4
R3	1	2	3	4
R4	1	2	3	4

Como exprimir?

1. Uma variável por rainha, Ri, sendo que a rainha i está na linha i.
   É impossível (por construção) haver duas rainhas na mesma linha.
2. $Rj \neq Ri, \ \forall \ j>i$
3. $Rj-Ri \neq k$, com $k \in \{j-i, i-j\} \ \forall \ j>i$

Exemplo CLP(FD): rainhas

Como fazer N variáveis com valores de 1 a N?

```
length(R, N),
fd_domain(R, 1, N)
```

Como dizer que não há conflitos?

```
ok([]).
ok([R|Rs]) :- ok(Rs, R, 1), ok(Rs).
```

Cada uma:

```
ok([], _, _).
ok([Rj|Rs], Ri, I) :-
  I1 is I+1,
  ok(Rs, Ri, I1),
  Ri #\= Rj, Ri #\= Rj+I, Ri+I #\= Rj.
```

Exemplo CLP(FD)

```
rainhas(N, R) :-
  length(R, N),
  fd_domain(R, 1, N),
  ok(R),
  fd_labeling(R).
```

Experimentando:

```
| ?- rainhas(4, X).
X = [2,4,1,3] ? ;
X = [3,1,4,2] ? ;
no
| ?- rainhas(10, X).
X = [1,3,6,8,10,5,9,2,4,7] ? ;
X = [1,3,6,9,7,10,4,2,5,8] ? ;
X = [1,3,6,9,7,10,4,2,8,5] ?
(1 ms) yes
| ?-
```

Terminologia

COP = Constraint Optimisation Problem

Um COP P=(P', G)

Em que:

- P' é um CSP, e P'=(V, D, C)
- G é uma função de V* para R

Pretendemos optimizar P para G, i.e. encontrar uma solução S de P' tal que G(S) seja máximo (ou mínimo)

Exemplo: TSP

Caixeiro Viajante (*Travelling Salesman Problem*, ou *TSP*)

Existem N cidades $C=\{C_1, \ldots C_N\}$

A distância entre cada par de cidades C_i e C_j é dada por D_{ij}

A ideia é encontrar um caminho P=(P1, P2, …, PN) tal que:

- Os Pi sejam cidades (i.e. $Pi \in C$)
- O ponto de chegada coincida com o ponto de partida (PN=P1)
- Todas as cidades figuram exatamente uma vez no percurso
  - Nenhuma cidade figura 2 vezes
  - Todas as cidades são incluídas
- O custo total (K=D12+D23+...+D(N-1)N) seja mínimo

Constraints de otimização

Duas constraints globais:

```
fd_minimize(GOAL, VAR)
fd_maximize(GOAL, VAR)
```

Ambas vão chamar GOAL, repetindo (não via backtracking), até encontrar um valor optimo para VAR.

Por exemplo, **`fd_minimize/2`** funciona assim:

- Da primeira vez tenta resolver GOAL e observa o valor V0 de VAR
- Da vez seguinte, acrescenta uma constraint **`VAR #< V0`** e repete GOAL, resultando no valor V1
- Vai iterando, acrescentando sempre **`VAR #< Vi`** até que não seja possível melhorar

Programação em Lógica com Restrições

Constraint Logic Programming

Constraints Booleanas Reificadas

Constraints Disjuntivas

Restrições Booleanas

Domínio Booleano

Valores possíveis: **true** e **false**

Mapeados para 1 e 0 (inteiros)

Operações sobre Booleanos

Seja E uma expressão de restrições sobre Booleanos, E pode tomar a forma:

Variável	domínio 0..1
0 (integer)	0 (false)
1 (integer)	1 (true)
#\ E	não E
E1 #<=> E2	E1 equivalente a E2
E1 #\<=> E2	E1 **não equivalente a** E2 (i.e. E1 **diferente de** E2)
E1 ## E2	E1 **XOR** E2 (i.e. E1 não equivalente a E2)
E1 #==> E2	E1 **implica** E2
E1 #\==> E2	E1 **não implica** E2
E1 #/\ E2	E1 **E** E2
E1 #\/\ E2	E1 **NAND** E2
E1 #\/ E2	E1 **OU** E2
E1 #\\/ E2	E1 **NOR** E2