CLP

Programação Lógica com Restrições

- Resolver duas limitações do Prolog
  - Cada termo em Prolog deve ser explicitamente codificado e não é avaliado
    - $X + 1$ não é um termo avaliado em Prolog.
    - Uma variável só pode assumir um único valor definido sintática e semanticamente: átomo, inteiro ou estrutura
  - Computação uniforme, mas não tão poderosa: É efetuada busca em profundidade e busca de soluções *generate-and-test*

Programação Lógica com Restrições - Introdução

- CLP utiliza outras técnicas utilizadas em IA para tornar procedimentos de busca mais inteligentes: propagação, computação guiada pelos dados, *forward checking* e *lookahead*.

- Aplicações: planning, escalonamento, alocação de recursos, computação gráfica, projeto de circuitos, diagnóstico de falhas etc.

Programação Lógica com Restrições - Introdução

CLP vem de duas linhas principais de pesquisa:

- Introdução de estruturas de dados mais ricas e poderosas em PL (ex: substituir unificação por manipulação de restrições em um domínio).

- Técnicas de consistência: uso ativo de restrições para reduzir o espaço de busca.
  generate-and-test x *constrain-and-generate*

Programação Lógica com Restrições - Domínio Aritmético

- Prolog não consegue resolver $x - 3 = y + 5$.
- CLP(R): 1^a. linguagem a introduzir restrições aritméticas.
- Expressões aritméticas lineares compostas de: números, variáveis e operadores (negação, adição, subtração, multiplicação e divisão).
- Expressão no domínio aritmético: $t_1 R t_2$, com $R = \{>, \geq, <, \leq, =, \neq\}$

- Procedimentos de decisão mais utilizados:
  - Método de eliminação de Gauss.
  - Método simplexo (para desigualdades) é mais usado:
    - bom comportamento em média
    - popular
    - incremental

- Exemplo:

- Uma refeição é composta de entrada, prato principal e sobremesa.

- Problema: A partir do banco de dados de alimentos e seus valores calóricos, produzir um cardápio com refeições leves, ou seja, refeições cujo valor calórico não exceda 10 Kcal.

```
ref_leve(E, P, S, C) :-
    I + J + K #=< 10,
    I #> 0,
    J #> 0,
    K #> 0,
    C #= I + J + K,
    entrada(E, I),
    principal(P, J),
    sobremesa(S, K).
```

```
entrada(salada, 1).
entrada(sopa, 6).

principal(M, I) :-
    carne(M, I).
principal(M, I) :-
    peixe(M, I).

sobremesa(fruta, 2).
sobremesa(sorvete, 6).

carne(bife, 5).
carne(porco, 7).

peixe(linguado, 2).
peixe(atum, 4).
```

Programação Lógica com Restrições - Domínio Aritmético

Possíveis soluções para este problema:

Entrada	Principal	Sobremesa	Calorias
salada	bife	fruta	8
salada	porco	fruta	10
salada	linguado	fruta	5
salada	linguado	sorvete	9
salada	atum	fruta	7
sopa	linguado	fruta	10

Programação Lógica com Restrições - Domínio Aritmético

- Outro exemplo: Programa para multiplicar dois números complexos: $(R_1 + I_1 \times i) \times (R_2 + I_2 \times i)$

```
zmul(R1, I1, R2, I2, R3, I3) :-
    {R3 = R1 * R2 - I1 * I2},
    {I3 = R1 * I2 + R2 * I1}.
```

- Para a consulta `zmul(1, 2, 3, 4, R3, I3)`, as equações se tornam lineares, apresentando uma solução definida.

```
R3 = 5, I3 = 10
```

- Efetuando–se a consulta `zmul(1, 2, R2, I2, R3, I3)`, obtém–se uma solução indefinida:

```
R3 = R2 - 2.0 * I2
I3 = 2.0 * R2 + I2
```

Programação Lógica com Restrições - Domínio Aritmético

- Porém, CLP(R) não resolve equações não-lineares

- Na consulta `zmul(R1, 2, R2, 4, -5, 10), {R2 < 3}`, é obtido:

```
R1 > 1.0
3.0 - R1 * R2 = 0.0
```

- Técnicas para lidar com equações não–lineares (tentativa de linearização):
  - Axiomas de isolamento
  - Manipulação algébrica – *lazy treatment*

Programação Lógica com Restrições - Domínio Booleano

- Aplicação principal: projeto de circuitos (verificação de hardware) e prova de teoremas.

- Termos booleanos: valores verdade (F ou V), variáveis, conectivos lógicos, única restrição: igualdade.

- Vários algoritmos de unificação para restrições booleanas.

- Solução provê procedimento de decisão para cálculo proposicional (NP-completo).

Programação Lógica com Restrições - Domínio Booleano

- Exemplo:
  Somador completo
  (operadores # (XOR),
  * (AND) e + (OR)):

```
add(I1, I2, Ce, O, Cs) :-
    X1 = I1 # I2,
    A1 = I1 * I2,
    O  = X1 # Ce,
    A2 = Ce * X1,
    Cs = A1 + A2.
```

- Consulta
  `add(a, b, c, O, Cs)`
  retorna:

```
O  = a # b # c
Cs = (a * b) # (a * c) # (b * c)
```

Programação Lógica com Restrições - Técnicas de consistência

- Eliminam *labelings* inconsistentes em estágios anteriores da computação propagando informação sobre as variáveis.

- Exemplos: *arc-consistency*, *forward checking*, propagação generalizada.

- Exemplo: escalonamento de tarefas.

Programação Lógica com Restrições - Técnicas de consistência

```
tarefas(L) :-
    L = [T1, T2, T3, T4, T5, T6],
    fd_domain(L, 1, 5),
    T1 #< T2,
    T1 #< T3,
    T2 #< T6,
    T3 #< T5,
    T4 #< T5,
    T5 #< T6,
    T2 #\= T3,
    fd_labelingff(L).
```

Programação Lógica com Restrições - Técnicas de consistência

- Funcionam propagando informações sobre as variáveis.

- Exemplo: $T1 \in \{1, 2, 3, 4, 5\}$, $T2 \in \{1, 2, 3, 4, 5\}$

- before(T1,T2) – técnica de consistência:
  - $T1 \in \{1, 2, 3, 4\}$ – Valor 5 removido de $T1$ pois não existe nenhum outro valor em $T2$ que atenda $T1 < T2$.
  - $T2 \in \{2, 3, 4, 5\}$ – Valor 1 removido de $T2$, pela mesma razão.

Programação Lógica com Restrições - Consistência de arco

- $T1 \in \{1,2\}; T2 \in \{2,3,4\}; T3 \in \{2,3\}; T4 \in \{1,2,3\}; T5 \in \{3,4\}; T6 \in \{4,5\}$

- Valor 2 de T1 é selecionado (T1 é *labeled* com valor 2)

- Por propagação: $T2 \in \{3,4\}$ e $T3 \in \{3\}$

- $T2 \in \{4\}$, por $T2 \neq T3$

- No final: $T1 \in \{2\}, T2 \in \{4\}, T3 \in \{3\}, T4 \in \{1,2,3\}, T5 \in \{4\}, T6 \in \{5\}$.

Outro exemplo: N-rainhas

```
queens(N):-
    statistics(runtime,[Start|_]),
    top(N),
    statistics(runtime,[End|_]),
    T is End-Start,
    write('%execution time ='), write(T), write(' milliseconds'), nl.
```

```
top(N):-
    make_list(N,List),
    fd_domain(List, 1, N),
    constrain_queens(List),
    fd_labelingff(List),
    write(List),nl.

constrain_queens([]).
constrain_queens([X|Y]):-
    safe(X,Y,1),
    constrain_queens(Y).
```

```
safe(_,[],_).
safe(X,[Y|T],K):-
    noattack(X,Y,K),
    K1 is K+1,
    safe(X,T,K1).

noattack(X,Y,K):-
    X #\= Y,
    X+K #\= Y,
    X-K #\= Y.

make_list(0,[]):-!.
make_list(N,[_|Rest]):-
    N1 is N-1,
    make_list(N1,Rest).
```

Programação Lógica com Restrições - Domínios finitos

- Utilização de máximos e mínimos para resolver restrições em domínios numéricos lineares.

- Ex: X, Y, Z têm domínios [1..10] com restrição $2x + 3y + 2 < z$

- Removendo valores inconsistentes:
  - 10 é o maior valor para Z, logo: $2x + 3y < 8$
  - 1 é o menor valor possível para y, logo: $2x < 5$
  - X só pode assumir valores $\{1, 2\}$
  - $3y < 6, y < 2, y \in \{1\}$
  - $z > 7, z \in \{8, 9, 10\}$

Programação Lógica com Restrições - Técnica básica de programação

- Declarar variáveis do problema e seus domínios.
- Estabelecer as restrições.
- Procurar solução.
- Exemplo

```
?- [X,Y,Z]::1..10,
   2 * X + 3 * Y + 2 #< Z,
   indomain(X), indomain(Y), indomain(Z).
```

Referências

(1) GNU Prolog - Contém biblioteca CLP para Domínios Finitos (CLP(FD))
http://gnu-prolog.inria.fr/

(2) B-Prolog - Sistema que também contém biblioteca CLP(FD)
http://www.probp.com/download1002.htm

(3) YAP Prolog - Contém biblioteca CLP para Números Racionais (CLP(Q)) e Reais (CLP(R))
http://www.ncc.up.pt/~vsc/Yap/

Referências

(4) P. Codognet and D. Diaz *A simple and Efficient Boolean Solver for Constraint Logic Programming* http://www.cs.cmu.edu/afs/cs.cmu.edu/user/fp/courses/95-lp/misc/papers/long_clp_b.ps

(5) http://www.cs.mu.oz.au/671/Slides/robdd-clpb-4.ps