Programación declarativa: lógica y restricciones

Programación Lógica con Restricciones

Constraint Logic Programming (*CLP*)

Mari Carmen Suárez de Figueroa Baonza

mcsuarez@fi.upm.es

Introducción (I)

- **CLP** es un lenguaje rico y potente para modelar problemas de optimización
  - Ofrece una forma declarativa para modelar problemas de satisfacción de restricciones
- El modelado se basa en variables, dominios y restricciones
  - Dominios: reales, enteros, booleanos, etc.
    - CLP(R)
  - Limitaciones particulares permitidas para cada dominio:
    - Por ejemplo, limitaciones aritméticas (+, *, =, ≤, ≥, <,>)
  - Algoritmos de resolución de restricciones: simplex, gauss, etc.

Introducción (II)

- Un **problema de satisfacción de restricciones** se puede representar como un triple formado por:
  - Un conjunto de variables V = {X1,...,Xn}
  - Para cada variable de V un conjunto de posibles valores Di, que llamaremos dominio de Xi
  - Un conjunto de restricciones, normalmente binarias, Cij(Xi,Xj) que determinan los valores que las variables pueden tomar simultáneamente
- El objetivo es encontrar un valor para cada variable de manera que se satisfagan todas las restricciones del problema
  - Cada restricción limita el conjunto de asignaciones para las variables implicadas

Introducción (III)

- Ventajas:
  - Mayor expresividad en el tratamiento de problemas
  - Diseño mas uniforme y mayor efectividad
    - Puede ahorrar mucha codificación
  - Aumento de la eficiencia
    - Gracias a la reducción del espacio de búsqueda

      LP: generate-and-test

      CLP: limitar-y-generar
- Desventajas:
  - Algoritmos de resolución (simplex, gauss, etc.) complejos que puede afectar al rendimiento
  - Necesidad de técnicas específicas para el tratamiento de los objetos

Restricciones en Ciao Prolog

- Planteadas como extensiones al sistema Prolog principal
  - Requieren la declaración inicial correspondiente
  - Definen operadores especiales para expresar las restricciones
    - **.=.** , **.>.** , **.>=.** etc.
  - Restricciones sobre el dominio de los racionales
    - :- use_package(clpq).
  - Restricciones sobre el dominio de los reales
    - :- use_package(clpr).

Restricciones en SWI Prolog

- Extensiones modulares al sistema Prolog principal
  - Requieren la declaración inicial correspondiente
  - Restricciones sobre el dominio de los racionales y los reales
    - :- use_module(library(clpq)) o bien :- use_module(library(clpr))
    - Las restricciones se expresan con los operadores aritméticos usuales, *pero* encerrados entre llaves: { }
    - Además, =/2 expresa una restricción de identidad (no de unificabilidad)
  - Restricciones sobre dominios finitos (discretos)
    - :- use_module(library(clpfd)). Similar a SICStus Prolog
    - Las restricciones se expresan con operadores específicos: **#>**, **#=**, etc.
    - Cuando las restricciones sólo acotan un rango de valores, label/1 genera (en *backtraking*) todos los valores posibles

CLP(X): Programas

- Un programa en CLP es una colección de reglas de la forma
  - a ← b1, . . . , bn (donde a es un átomo y los bi's son átomos o restricciones)
- Un hecho es una regla
  - a ← c (donde c es una restricción)
- Un objetivo (o consulta) G es una conjunción de restricciones y átomos
- Cada restricción es una fórmula de primer orden construida con restricciones primitivas
  - p(t1,t2,…,tn), con términos t1, t2,…,tn y p símbolo de predicado, es una restricción primitiva

CLP($\mathfrak{R}$)

- Prolog no puede resolver x-3 = y+5
- CLP ($\mathfrak{R}$) es un lenguaje basado en Prolog, que incluye capacidades para resolver restricciones sobre números reales
  - Expresiones aritméticas lineales: números, variables y operadores (negación, suma, resta, multiplicación y división)
  - Ejemplo: t1 R t2, donde R = { >, ≥, =, ≤, <, =}
- CLP ($\mathfrak{R}$) utiliza la misma estrategia de ejecución que Prolog
  - primero en profundidad, de izquierda a derecha
- CLP ($\mathfrak{R}$) es capaz de resolver directamente (in)-ecuaciones lineales sobre números reales

Programación Lógica vs. CLP($\Re$) (I)

- Ejemplo: (Prolog)
- q(X, Y, Z) :- Z = f(X, Y).
  - ?- q(3, 4, Z).
    - Z = f(3,4)
  - ?- q(X, Y, f(3,4)).
    - X = 3, Y = 4
  - ?- q(X, Y, Z).
    - Z = f(X,Y)
- Ejemplo: (Prolog)
- p(X, Y, Z) :- Z is X + Y.
  - ?- p(3, 4, Z). **% modo in-in-out**
    - Z = 7
  - ?- p(X, 4, 7). **% modo out-in-in**
    - Instantiation Error

Programación Lógica vs. CLP ($\Re$) (II)

- Ejemplo: (CLP($\Re$))
- p(X, Y, Z) :- Z .=. X + Y.
  - ?- p(3, 4, Z). **% modo in-in-out**
    - Z = 7
    - Yes
  - ?- p(X, 4, 7). **% modo out-in-in**
    - X = 3
    - Yes
  - ?- p(X, Y, 7). **% modo out-out-in**
    - X = 7 - Y
    - Yes

clp-ejemplo.pl

Programación Lógica vs. CLP(ℜ) (III)

- Ejemplo: Reducción del espacio de búsqueda
- Prolog (generar y testear):
  - solution(X, Y, Z) :- p(X), p(Y), p(Z), test(X, Y, Z).
  - p(14). p(15). p(16). p(7). p(3). p(11).
  - test(X, Y, Z) :- Y is X + 1, Z is Y + 1.
    - ?- solution(X, Y, Z).

      X = 14,

      Y = 15,

      Z = 16 ? ;

      no
  - 458 pasos (todas las soluciones: 465 pasos)

clp-ReduccionEspacioBusqueda.pl

Programación Lógica vs. CLP ($\Re$) (IV)

- Ejemplo: Reducción del espacio de búsqueda
- CLP($\Re$) (generar y testear):
  - solution(X, Y, Z) :- p(X), p(Y), p(Z), test(X, Y, Z).
  - p(14). p(15). p(16). p(7). p(3). p(11).
  - test(X, Y, Z) :- Y .=. X + 1, Z .=. Y + 1.
    - solution(X, Y, Z).

      Z = 16,

      Y = 15,

      X = 14 ?;

      no
  - 458 pasos (todas las soluciones: 465 pasos)

Programación Lógica vs. CLP (ℜ) (V)

- Ejemplo: Reducción del espacio de búsqueda
  - Cambiar '*test(X, Y, Z)*' al principio (restringir y generar):
  - solution(X, Y, Z) :- test(X, Y, Z), p(X), p(Y), p(Z).
  - p(14). p(15). p(16). p(7). p(3). p(11).
  - Prolog: test(X, Y, Z) :- Y is X + 1, Z is Y + 1.
    - ?- solution(X, Y, Z).

      Instantiation Error
  - CLP(ℜ): test(X, Y, Z) :- Y .=. X + 1, Z .=. Y + 1.
    - ?- solution(X, Y, Z).

      Z = 16,

      Y = 15,

      X = 14 ?;

      no
  - 6 pasos (todas las soluciones: 11 pasos)

$E = \frac{1}{2} mv^2 + 9.81\ mh$

- En Prolog, un procedimiento que calcule cualquiera de las cuatro variables dadas las otras tres:

```
energia(E,M,V,H):-
        number(M),
        number(V),
        number(H),
        E is 0.5*M*V*V + 9.81*M*H.

energia(E,M,V,H):-
        number(E),
        number(M),
        number(V),
        H is (E - 0.5*M*V*V)/(9.81*M).
```

```
energia(E,M,V,H):-
        number(E),
        number(M),
        number(H),
        V is sqrt((E - 9.81*M*H)/0.5*M).

energia(E,M,V,H):-
        number(E),
        number(V),
        number(H),
        M is E / (0.5*V*V + 9.81*H).
```

- En CLP(R):

```
% Ciao Prolog
:- use_package(clpr).

energia(E,M,V,H):-
        E .=. 0.5*M*V*V + 9.81*M*H.
```

```
% SWI Prolog
:- use_module(library(clpq)).

energia(E,M,V,H):-
        { E = 0.5*M*V*V + 9.81*M*H }.
```

Ecuaciones Lineales (CLP($\mathbb{R}$))

- Multiplicación de vectores (de números reales):
  - (x1, x2, . . . , xn) · (y1, y2, . . . , yn) = x1·y1 + … + xn·yn
- Representamos los vectores como listas de números
  - prod([], [], 0).
  - prod([X|Xs], [Y|Ys],P) :- P .=. X * Y + Rest,
prod(Xs, Ys, Rest).
- La unificación se convierte en resolución de restricciones
  - ?- prod([2, 3], [4, 5], K).
    - K = 23
  - ?- prod([2, 3], [5, X2], 22).
    - X2 = 4
  - ?- prod([2, 7, 3], [Vx, Vy, Vz], 0).
    - Vx = -1.5*Vz - 3.5*Vy
- Cualquier respuesta calculada es, en general, una ecuación sobre las variables de la consulta

Sistemas de Ecuaciones Lineales (CLP($\mathfrak{R}$))

- ¿Podemos resolver sistemas de ecuaciones?
  - 3x + y = 5
  - x + 8y = 3
  - ?- prod([3, 1], [X, Y], 5), prod([1, 8], [X, Y], 3).
    - X = 1.6087, Y = 0.173913
- Se puede construir un predicado más general imitando la notación vectorial matemática A·x = b:
  - system(_Vars, [], []).
  - system(Vars, [Co|Coefs], [Ind|Indeps]) :-
    prod(Vars, Co, Ind),
    system(Vars, Coefs, Indeps).
  - ?- system([X, Y], [[3, 1],[1, 8]],[5, 3]).
    - X = 1.6087, Y = 0.173913

% podemos expresar y resolver sistemas de ecuaciones

clp-EcuacionesLineales.pl

CLP($\mathfrak{R}$): Ejemplo

- Una comida consiste en un entrante, un plato principal y un postre

  Suponemos que existe una base de datos con distintos tipos de comida y sus valores calóricos

  Se debe producir un menú con comida *light* (valor calórico menor de 10Kcal)

clp-lightMeal.pl

Ejemplo: Pasatiempo

- Este pasatiempo consta de las siguientes afirmaciones:
  - Un alemán y un británico viven cada uno en una casa de diferente color y tienen diferentes mascotas
  - El alemán vive en la casa verde
  - Hay un perro en la casa blanca
- Se plantea la siguiente pregunta:
  - ¿Quién tiene un gato?

clp_pasatiempo.pl

Ejercicio: Pasatiempo

- Este pasatiempo consta de las siguientes afirmaciones:
  - Un alemán, un británico y un sueco viven cada uno en una casa de color diferente, tienen diferentes mascotas y gustan diferentes bebidas
  - El alemán vive en la casa verde
  - El sueco bebe café
  - Al británico no le gustan los gatos
  - Hay un perro en la casa blanca
  - El sueco vive junto a la casa azul
  - Alguien bebe agua y tiene un pez
- Se plantea la siguiente pregunta:
  - ¿Quién bebe té?

clp_pasatiempo_ejercicio.pl

Ejercicio: SEND + MORE = MONEY

- … que quiere decir, en inglés, “Envía más dinero“
- Sustituye, en la suma siguiente, las letras por cifras (de 0 a 9) teniendo en cuenta que a cada letra distinta le corresponde una cifra diferente
  - Las variables *S*, *E*, *N*, *D*, *M*, *O*, *R*, *Y* representan dígitos entre 0 and 9
  - La tarea consiste en encontrar valores para estas variables de manera que la operación SEND+MORE=MONEY es correcta
  - Todas las variables deben tomar valores únicos
  - Los números deben estar bien formados (lo que implica que *M*>0 y *S*>0

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

clp-puzzle-swi.pl

Programación declarativa: lógica y restricciones

Programación Lógica con Restricciones

Constraint Logic Programming (*CLP*)

Mari Carmen Suárez de Figueroa Baonza

mcsuarez@fi.upm.es