Programación con restricciones - Tema 0 : Conceptos básicos

Rafael Caballero Roldán - Facultad de Informática - UCM

1. Introducción

Esta asignatura trata de la programación con restricciones, un paradigma de programación dedicado a problemas que impliquen una búsqueda compleja de soluciones. En lugar de implementar nosotros mismos los algoritmos, en programación con restricciones nos limitamos a especificar el problema, dejando a un programa externo, el resolutor, la tarea de encontrar la solución.

El paradigma apareció en 1987 en el contexto de la programación lógica por Jaffar y Lassez. Aunque al principio solo existia como parte de sistemas de programación lógico (en general en implementaciones de Prolog), pronto aparecieron librerías para lenguajes imperativos.

En nuestro curso vamos a utilizar el lenguaje MiniZinc, un lenguaje de modelado diseñado por Kim Marriot y Peter Stuckey. MiniZinc tiene la ventaja de ser un lenguaje de modelado puro, lo que nos permite concentrarnos en la programación con restricciones.

Antes de empezar a modelar en el lenguaje de programación con restricciones Minizinc, conviene que conozcamos un poco los conceptos en que se basa. Se trata de un tema un poco árido, pero ayudará a fijar ideas que estarán presentes durante todo el curso.

2. Variables y dominios

Una variable de *decisión* es una variable lógica que puede tomar valores dentro de un conjunto de valores llamado *dominio*. Normalmente se declara un dominio inicial al declarar la variable. Éste puede ser, por ejemplo, todos los números enteros, o solo los del rango -2..2. En general en PR solemos disponer, entro otro, de los siguientes dominios:

1. Dominios finitos: cuando se habla de dominios finitos nos referimos a los enteros. Alguien pensará: ¡¡¡¡¡Si los enteros no son finitos!!!! Vale, es verdad, pero en PR consideramos que sí. En MiniZinc por ejemplo se suelen tomar enteros entre -1 000 000 y +1 000 000.

2. SAT: Sólo se admiten variables booleanas y sus operaciones relacionadas. Pueden parecer muy restrictivos, pero son realmente potentes ya que

   a) Admite optimizaciones muy potentes → mejora de eficiencia.

b) Los problemas de dominios finitos se pueden reducir a SAT.

Por eso algunos resolutores de dominios finitos lo que hacen es trasladar el problema a SAT.

3. Reales: Restricciones sobre números reales. Son muy diferentes de los anteriores porque utilizan mecanismos de resolución basados en técnicas de análisis numérico.

Una limitación importante de la programación con restricciones es que los problemas no pueden mezclar diversos dominios. Debemos decidir si el problema tiene sentido en SAT, dominios finitos o reales, utilizar las primitivas del dominio correspondiente y usar su resolutor asociado.

3. Variables y dominios

1. Indicaremos mediante la notación $x_1 \in D_1, \ldots, x_k \in D_k$ que $x_1, \ldots, x_k$ es una secuencia de variables de decisión con dominio $D_1, \ldots, D_k$, respectivamente.

2. Una *restricción* C es una expresión que define un subconjunto del producto cartesiano $D_1 \times \ldots \times D_k$.
   Ejemplo: Consideramos las variables de decisión x, y, z con dominios

   $$D_x = \{3, 4, 5\},\ D_y = \{6, 8, 9\},\ D_z = \{4, 5\}$$

   la restricción $C \equiv x \neq z \wedge 2x \neq y$ define el subconjunto

   $$\begin{aligned}\{\quad & (3,8,4),\ (3,9,4),\ (3,8,5),\ (3,9,5),\\ & (4,6,5),\ (4,9,5),\\ & (5,6,4),\ (5,8,4),\ (5,9,4)\ \}\ \subseteq\ D_x \times D_y \times D_z\end{aligned}$$

3. Un *problema de satisfacción de restricciones* (*CSP* en la literatura en inglés) se representa mediante

   $$\prec\ x_1 \in D_1, \ldots, x_k \in D_k\ ;\ C_1, \ldots, C_p\ \succ$$

   con $C_1, \ldots, C_p$ restricciones definidas sobre subsecuencias de $x_1, \ldots, x_k$.
   Ejemplo: Sean x, y, z con D_x, D_y, D_z definidos como en el ejemplo del punto 2. Entonces:

   $$\mathcal{P} = \prec\ x \in D_x,\ y \in D_y,\ z \in D_z\ ;\ x \neq z,\ 2x \neq y\ \succ$$

   es un problema de satisfacción de restricciones.

4. Sean:

   - $\mathcal{X} = x_1, \ldots, x_k$ una secuencia de variables, con dominio $x_1 \in D_1, \ldots, x_k \in D_k$.
   - $\mathcal{Y} = x_{i_1}, \ldots, x_{i_l}$ una subsecuencia de variables de $\mathcal{X}$, con $i_1 < \ldots < i_l$ y $1 \leq i_j \leq k$ para $j = 1 \ldots l$.
   - $d = (d_1, \ldots, d_k)$ un elemento de $D_1 \times \ldots \times D_k$.

   Entonces definimos la *proyección* de d sobre $\mathcal{Y}$, denotada por $d[\mathcal{Y}]$ como $d[\mathcal{Y}] = d_{i_1}, \ldots, d_{i_l}$.

4. Soluciones de un problema de satisfacción de restricciones

1. Sea

$$\mathcal{P} =\prec x_1 \in D_1, \ldots, x_k \in D_k \ ; C_1, \ldots, C_p \succ$$

un problema de satisfacción de restricciones. Se dice que una tupla $(d_1, \ldots, d_k)$ es una *solución* $\mathcal{P}$ si para toda C_i, con $1 \leq i \leq p$ definida sobre variables $\mathcal{Y}_i$ subsecuencia $x_1, \ldots, x_k$, se tiene que $d[\mathcal{Y}] \in C_i$.

2. Denotamos por $Sol(\mathcal{P})$ el conjunto de soluciones para el CSP $\mathcal{P}$.

3. Dado un CSP $\mathcal{P}$ y una función $f : Sol(\mathcal{P}) \to \mathbb{R}$ (es decir una función que asigna a cada solución un número real), se llama *problema de optimización del CSP* $\mathcal{P}$ al problema consistente en encontrar un máximo (o un mínimo) para la función f.

4. Un CSP que tiene al menos una solución se dice *consistente* (es decir $\mathcal{P}$ es consistente sii $Sol(\mathcal{P}) \neq \emptyset$).

5. Un CSP que no tiene soluciones se dice *inconsistente* (es decir $\mathcal{P}$ es inconsistente sii $Sol(\mathcal{P}) = \emptyset$). MiniZinc a veces detecta inconsistencias sin necesidad de realizar búsqueda -encuentra contradicciones sintácticas-. En este caso el sistema nos dirá que es insatisfactible, pero en realidad se trata de un sistema incosistente.

6. Sean $\mathcal{P}_1$ y $\mathcal{P}_2$ dos CSPs. Se dice que ambos son equivalentes con respecto a $\mathcal{X}$ si y solo si:

$$\{d[\mathcal{X}] \mid d \in Sol(\mathcal{P}_1)\} = \{d[\mathcal{X}] \mid d \in Sol(\mathcal{P}_2)\}$$

5. Consistencia local

La idea es diseñar técnicas capaces de anticipar el fallo, durante la búsqueda de soluciones (es decir en el diseño del resolutor).

5.1. Nodo-consistencia

1. Decimos que un CSP es nodo-consistente si para toda variable de decisión x toda restricción unaria sobre x coincide con el dominio de x.

   Ejemplo: Consideramos el problema

$$\mathcal{P} =\prec x_1 \in \mathbb{N}, \ldots, x_k \in \mathbb{N} \ ; C, x_1 \geq 0, \ldots, x_k \geq 0 \succ$$

   donde C no contiene restricciones unarias. ¿Es nodo consistente?

Ejemplo: Consideramos el problema

$$\mathcal{P} = \prec x_1 \in \mathbb{N}, \ldots, x_{k-1} \in \mathbb{N}, x_k \in \mathbb{Z} \ ; \ C, x_1 \geq 0, \ldots, x_n \geq 0 \succ$$

donde C no contiene restricciones unarias. ¿Es nodo consistente?

2. Cualquier CSP se puede transformar en otro nodo-consistente aplicando la inferencia:

$$\frac{\prec x \in D \ ; \ C \succ}{\prec x \in C \cap D \ ; \ C \succ}$$

Resultado: Un CSP es nodo-consistente si y solo si es cerrado bajo la inferencia anterior.

5.2. Arco-consistencia

1. Sea C una restricción binaria sobre dos variables x, y con dominios D_x, D_y, es decir $C \subset D_x \times D_y$. Decimos que C es arco-consistente si:
   - $\forall a \in D_x, \exists b \in D_y : (a, b) \in C$
   - $\forall b \in D_y, \exists a \in D_x : (a, b) \in C$
2. Decimos que un CSP es arco-consistente si todas sus restricciones binarias son arco-consistentes.
3. **Ejemplo:** Consideramos el problema

$$\mathcal{P} = \prec x \in [5.,10], \ y \in [3.,7] \ ; \ x < y \succ$$

¿Es arco consistente?

4. **Ejemplo:** El modelado del problema de las n-reinas visto en clase es arco-consistente para todo $n \geq 3$.
5. ¿Todo sistema basado en restricciones binarias arco-consistente es consistente?
6. Consideramos el problema

$$\mathcal{P} = \prec x \in \{a, b\}, \ y \in \{a\} \ ; \ x = y \succ$$

¿Es arco consistente? ¿Es consistente?

7. Cualquier CSP se puede transformar en otro arco-consistente aplicando las inferencias:

$$\frac{\prec x \in D_x, y \in D_y \ ; \ C \succ}{\prec x \in D'_x, y \in D_y \ ; \ C \succ}$$

donde $D'_x = \{a \in D_x \mid \exists b \in D_y : (a, b) \in C\}$

$$\frac{\prec x \in D_x, y \in D_y \ ; \ C \succ}{\prec x \in D_x, y \in D'_y \ ; \ C \succ}$$

donde $D'_y = \{b \in D_y \mid \exists a \in D_x : (a, b) \in C\}$

8. Resultado: Un CSP es arco-consistente si y solo si es cerrado bajo las inferencias anteriores.