Lenguaje de Modelado GNU MathProg

Referencia del Lenguaje

para GLPK Versión 4.57

(BORRADOR, octubre del 2015)

Título original en inglés: Modeling Language GNU MathProg - Language Reference for GLPK Version 4.50

Traducción: Pablo Yapura, Facultad de Ciencias Agrarias y Forestales, Universidad Nacional de La Plata, La Plata, Argentina.

Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA.

Contenidos

Capítulo 1

Introducción

GNU MathProg es un lenguaje de modelado diseñado para describir modelos lineales de programación matemática.[1]

La descripción del modelo escrita en el lenguaje GNU MathProg consiste en un conjunto de sentencias y bloques de datos construidos por el usuario a partir de los elementos del lenguaje que se describen en este documento.

En un proceso que se denomina *traducción*, un programa denominado *traductor del modelo* analiza la descripción del modelo y la traduce en una estructura interna de datos, la que puede ser usada tanto para generar una instancia de un problema de programación matemática, como para obtener directamente una solución numérica del problema mediante un programa denominado *solver*.

1.1 El problema de la programación lineal

En MathProg el problema de la programación lineal (PL) se expresa como sigue:

minimizar (o maximizar)

$$z = c_1x_1 + c_2x_2 + \ldots + c_nx_n + c_0 \tag{1.1}$$

sujeto a las restricciones lineales

$$\begin{array}{l} L_1 \leq a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq U_1 \\ L_2 \leq a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq U_2 \\ \qquad\qquad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ L_m \leq a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq U_m \end{array} \tag{1.2}$$

[1] El lenguaje GNU MathProg es un subconjunto del lenguaje AMPL. Su implementación en GLPK está basada principalmente en el paper: *Robert Fourer, David M. Gay & Brian W. Kernighan*, "A Modeling Language for Mathematical Programming." *Management Science* 36 (1990), pp. 519-554.

y a las cotas de las variables

$$\begin{aligned} l_1 &\leq x_1 \leq u_1 \\ l_2 &\leq x_2 \leq u_2 \\ &\ldots\ldots \\ l_n &\leq x_n \leq u_n \end{aligned} \tag{1.3}$$

donde x_1, x_2, ..., x_n son variables; z es la función objetivo; c_1, c_2, ..., c_n son coeficientes de la función objetivo; c_0 es el término constante ("de traslación") de la función objetivo; a_{11}, a_{12}, ..., a_{mn} son coeficientes de las restricciones; L_1, L_2, ..., L_m son cotas inferiores de las restricciones; U_1, U_2, ..., U_m son cotas superiores de las restricciones; l_1, l_2, ..., l_n son cotas inferiores de las variables y u_1, u_2, ..., u_n son cotas superiores de las variables.

Las cotas de las variables y las cotas de las restricciones pueden ser tanto finitas como infinitas. Además, las cotas inferiores pueden ser iguales a las correspondientes cotas superiores. Entonces, están permitidos los siguientes tipos de variables y restricciones:

$-\infty < x < +\infty$	Variable libre (no acotada)
$l \leq x < +\infty$	Variable con cota inferior
$-\infty < x \leq u$	Variable con cota superior
$l \leq x \leq u$	Variable doblemente acotada
$l = x = u$	Variable fija
$-\infty < \sum a_j x_j < +\infty$	Forma lineal libre (no acotada)
$L \leq \sum a_j x_j < +\infty$	Restricción de inecuación "mayor o igual que"
$-\infty < \sum a_j x_j \leq U$	Restricción de inecuación "menor o igual que"
$L \leq \sum a_j x_j \leq U$	Restricción de inecuación doblemente acotada
$L = \sum a_j x_j = U$	Restricción de igualdad

Además de problemas puros de PL, MathProg también permite problemas de programación entera lineal mixta (PEM), en los que algunas o todas las variables se han restringido a ser enteras o binarias.

1.2 Objetos del modelo

En MathProg el modelo se describe mediante conjuntos, parámetros, variables, restricciones y objetivos, los que se denominan *objetos del modelo.*

El usuario introduce objetos particulares del modelo usando las sentencias del lenguaje. Cada objeto del modelo está provisto de un nombre simbólico que lo identifica de manera única y está pensado con propósitos de referencia.

Los objetos del modelo, incluyendo los conjuntos, pueden ser arreglos multidimensionales construidos sobre conjuntos indizantes. Formalmente, el arreglo n-dimensional A es el mapeo

$$A : \Delta \to \Xi, \tag{1.4}$$

donde $\Delta \subseteq C_1 \times \ldots \times C_n$ es el subconjunto del producto cartesiano de los conjuntos indizantes, Ξ es el conjunto de los miembros del arreglo. En MathProg, el conjunto Δ se denomina *dominio del subíndice*. Sus miembros son los n-tuplos $(i_1, \ldots, i_n)$, donde $i_1 \in C_1$, $\ldots$, $i_n \in C_n$.

Si $n = 0$, el producto cartesiano tiene exactamente un miembro (específicamente, un 0-tuplo), de forma tal que es conveniente pensar en los objetos escalares como arreglos 0-dimensionales que tienen un solo miembro.

El tipo de los miembros del arreglo se determina por el tipo del objeto del modelo correspondiente como sigue:

Objeto del modelo	Miembro del arreglo
Conjunto	Conjunto plano elemental
Parámetro	Número o símbolo
Variable	Variable elemental
Restricción	Restricción elemental
Objetivo	Objetivo elemental

Para referir al miembro particular de un objeto, el mismo debe estar provisto de *subíndices*. Por ejemplo, si a es un parámetro 2-dimensional definido sobre $I \times J$, una referencia a sus miembros particulares se puede escribir como $a[i, j]$, donde $i \in I$ y $j \in J$. Se sobreentiende que los objetos escalares no necesitan subíndices por ser 0-dimensionales.

1.3 Estructura de la descripción del modelo

A veces es deseable escribir un modelo que, en distintos momentos, puede requerir diferentes datos para solucionar cada instancia del problema usando el modelo. Por esta razón, en MathProg la descripción del modelo consta de dos partes: la *sección del modelo* y la *sección de los datos*.

La sección del modelo es la parte principal de la descripción del modelo que contiene las declaraciones de los objetos del modelo; es común a todos los problemas basados en el modelo correspondiente.

La sección de los datos es una parte opcional de la descripción del modelo que contiene los datos específicos para una instancia particular del problema.

Dependiendo de lo que sea más conveniente, las secciones del modelo y de los datos pueden disponerse en el mismo archivo o en dos archivos separados. Esta última característica permite tener un número arbitrario de secciones con datos diferentes para ser usadas con la misma sección del modelo.

Capítulo 2

Codificación de la descripción del modelo

La descripción del modelo se codifica en formato de texto plano usando el juego de caracteres ASCII. Los caracteres válidos en la descripción del modelo son los siguientes:

— caracteres alfabéticos:
`A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`
`a b c d e f g h i j k l m n o p q r s t u v w x y z _`

— caracteres numéricos:
`0 1 2 3 4 5 6 7 8 9`

— caracteres especiales:
`! " # & ' ( ) * + , - . / : ; < = > [ ] ^ { | } ~`

— caracteres no imprimibles:
`SP HT CR NL VT FF`

Dentro de los literales de cadena y de los comentarios, cualquier carácter ASCII (excepto los caracteres de control) son válidos.

Los caracteres no imprimibles no son significativos. Se pueden usar libremente entre las unidades léxicas para mejorar la legibilidad de la descripción del modelo. También se usan para separar unidades léxicas entre sí, en caso de no existir otra forma de hacerlo.

Sintácticamente, la descripción del modelo es una secuencia de unidades léxicas de las siguientes categorías:

— nombres simbólicos;

— literales numéricos;

— literales de cadena;

— palabras clave;

— delimitadores;

— comentarios.

Las unidades léxicas del lenguaje se discuten a continuación.

2.1 Nombres simbólicos

Un *nombre simbólico* consiste de caracteres alfabéticos y numéricos, el primero de los cuales debe ser alfabético. Todos los nombres simbólicos deben ser distintos (sensibilidad a las mayúsculas).

Ejemplos

```
alfa123
Esto_es_un_nombre
_P123_abc_321
```

Los nombres simbólicos se usan para identificar los objetos del modelo (conjuntos, parámetros, variables, restricciones y objetivos) y los índices.

Todos los nombres simbólicos (exceptuando los nombres de los índices) deben ser únicos, *i.e.* la descripción del modelo no debe tener objetos distintos con el mismo nombre. Los nombres simbólicos de los índices deben ser únicos dentro del alcance en el que son válidos.

2.2 Literales numéricos

Un *literal numérico* sigue la forma *xx*`E`*syy*, donde *xx* es un número con punto decimal optativo, *s* es el signo `+` o `-` e *yy* es un exponente decimal. La letra `E` es insensible a las mayúsculas y se puede codificar como `e`.

Ejemplos

```
123
3.14159
56.E+5
.78
123.456e-7
```

Los literales numéricos se usan para representar cantidades numéricas y tienen significado fijo obvio.

2.3 Literales de cadena

Un *literal de cadena* es una secuencia arbitraria de caracteres encerrados entre comillas, tanto simples como dobles. Ambas formas son equivalentes.

Si una comilla simple es parte de un literal de cadena encerrado entre comillas simples, se debe codificar dos veces. Análogamente, si una comilla doble es parte de un literal de cadena encerrado entre comillas dobles, se debe codificar dos veces.

Ejemplos

```
'Esta es una cadena'
"Esta es otra cadena"
'No debe usarse los 20''s'
"""Hola, che"" cantaba Favio."
```

Los literales de cadena se usan para representar cantidades simbólicas.

2.4 Palabras clave

Una *palabra clave* es una secuencia de caracteres alfabéticos y posiblemente algunos caracteres especiales.

Todas la palabras clave caen en alguna de dos categorías: las *palabras clave reservadas*, que no pueden usarse como nombres simbólicos, y las *palabras clave no reservadas*, que son reconocidas por el contexto y entonces pueden usarse como nombres simbólicos.

Las palabras clave reservadas son las siguientes:

```
and       else      mod       union
by        if        not       within
cross     in        or
diff      inter     symdiff
div       less      then
```

Las palabras clave no reservadas se describen en secciones posteriores.

Todas las palabras clave tienen un significado fijo, el que se explicará en las discusiones de las correspondientes construcciones sintácticas donde las palabras claves sean usadas.

2.5 Delimitadores

Un *delimitador* es tanto un carácter especial individual como una secuencia de dos caracteres especiales, como sigue:

```
+     **    <=    >     &&    :     |     [     >>
-     ^     =     <>    ||    ;     ~     ]     <-
*     &     ==    !=    .     :=    (     {
/     <     >=    !     ,     ..    )     }
```

Si el delimitador está compuesto por dos caracteres, no debe haber espacios entre ellos.

Todos los delimitadores tienen un significado fijo, el que se explicará en las discusiones de las correspondientes construcciones sintácticas donde los delimitadores sean usados.

2.6 Comentarios

Con propósitos de documentación, la descripción del modelo puede ser provista de *comentarios*, los que pueden ser de dos formas diferentes. La primera es un *comentario de una línea individual*, el que debe comenzar con el carácter `#` y se extiende hasta el final de la línea. La segunda forma es un *comentario en secuencia*, el que consiste en una secuencia de caracteres cualesquiera encerrados entre `/*` y `*/`.

Ejemplos

```
param n := 10; # Esto es un comentario
/* Esto es otro comentario */
```

Los comentarios son ignorados por el traductor del modelo y pueden aparecer en cualquier sitio de la descripción del modelo en la que que se permitan caracteres no imprimibles.

Capítulo 3

Expresiones

Una *expresión* es una regla para calcular un valor. En la descripción de un modelo, las expresiones se usan como constituyentes de ciertas sentencias.

En general, las expresiones están compuestas por operandos y operadores.

Dependiendo del tipo del valor resultante, todas las expresiones pertenecen a alguna de las siguientes categorías:

— expresiones numéricas;

— expresiones simbólicas;

— expresiones indizantes;

— expresiones de conjuntos;

— expresiones lógicas;

— expresiones lineales.

3.1 Expresiones numéricas

Una *expresión numérica* es una regla para calcular un valor numérico individual representado como un número de punto flotante.

La expresión numérica primaria puede ser un literal numérico, un índice, un parámetro no-indizado, un parámetro indizado, una función interna de referencia, una expresión numérica iterada, una expresión numérica condicional u otra expresión numérica encerrada entre paréntesis.

Ejemplos

`1.23`	(literal numérico)
`j`	(índice)
`tiempo`	(parámetro no-indizado)
`a['Mayo de 2003',j+1]`	(parámetro indizado)
`abs(b[i,j])`	(función de referencia)
`sum{i in S diff T} alfa[i] * b[i,j]`	(expresión iterada)
`if i in I then 2 * p else q[i+1]`	(expresión condicional)
`(b[i,j] + .5 * c)`	(expresión parentética)

Empleando ciertos operadores aritméticos se pueden construir expresiones numéricas más generales conteniendo dos o más expresiones numéricas primarias.

Ejemplos

```
j+1
2 * a[i-1,j+1] - b[i,j]
sum{j in J} a[i,j] * x[j] + sum{k in K} b[i,k] * x[k]
(if i in I and p >= 1 then 2 * p else q[i+1]) / (a[i,j] + 1.5)
```

3.1.1 Literales numéricos

Si la expresión numérica primaria es un literal numérico, el valor resultante es obvio.

3.1.2 Índices

Si la expresión numérica primaria es un índice, el valor resultante es el valor corriente asignado al índice.

3.1.3 Parámetros no-indizados

Si la expresión numérica primaria es un parámetro no-indizado (el que debe ser 0-dimensional), el valor resultante es el valor del parámetro.

3.1.4 Parámetros indizados

La expresión numérica primaria que se refiere a parámetros indizados tiene la siguiente forma sintáctica:

$$nombre[i_1, i_2, \ldots, i_n]$$

donde *nombre* es el nombre simbólico del parámetro e i_1, i_2, ..., i_n son subíndices.

Cada subíndice debe ser una expresión numérica o simbólica. El número de subíndices en la lista de subíndices debe ser igual a la dimensión del parámetro con el cual está asociada la lista de subíndices.

Los valores reales de las expresiones de subíndices se usan para identificar al miembro particular del parámetro que determina el valor resultante de la expresión primaria.

3.1.5 Funciones de referencia

En MathProg existen las siguientes funciones internas, las que se pueden usar en expresiones numéricas:

`abs(`x`)`	$\lvert x\rvert$, valor absoluto de x
`atan(`x`)`	$\arctan x$, valor principal del arcotangente de x (en radianes)
`atan(`y`, `x`)`	$\arctan y/x$, valor principal del arcotangente de y/x (en radianes). En este caso, los signos de ambos argumentos, y y x, se usan para determinar el cuadrante del valor resultante
`card(`X`)`	$\lvert X\rvert$, el cardinal (número de elementos) del conjunto X
`ceil(`x`)`	$\lceil x\rceil$, el menor entero no menor que x ("techo de x")
`cos(`x`)`	$\cos x$, coseno de x (en radianes)
`exp(`x`)`	e^x, exponencial en base e de x
`floor(`x`)`	$\lfloor x\rfloor$, el mayor entero no mayor que x ("piso de x")
`gmtime()`	el número de segundos transcurridos desde las 00:00:00 del 1 de enero de 1970, Tiempo Universal Coordinado (para los detalles ver la Sección B.1, página 59)
`length(`c`)`	$\lvert c\rvert$, longitud de la cadena de caracteres c
`log(`x`)`	$\log x$, logaritmo natural de x
`log10(`x`)`	$\log_{10} x$, logaritmo común (decimal) de x
`max(`x_1`, `x_2`, ..., `x_n`)`	el mayor de los valores x_1, x_2, ..., x_n
`min(`x_1`, `x_2`, ..., `x_n`)`	el menor de los valores x_1, x_2, ..., x_n
`round(`x`)`	redondeo de x al entero más próximo
`round(`x`, `n`)`	redondeo de x a n dígitos decimales
`sin(`x`)`	$\sin x$, seno de x (en radianes)
`sqrt(`x`)`	$\sqrt{x}$, raíz cuadrada no-negativa de x
`str2time(`c`, `f`)`	conversión de la cadena de caracteres c a tiempo calendario (para los detalles ver la Sección B.2, página 60)
`trunc(`x`)`	truncado de x al entero más próximo
`trunc(`x`, `n`)`	truncado de x a n dígitos decimales
`Irand224()`	generación de un entero pseudo-aleatorio uniformemente distribuido en $[0, 2^{24})$
`Uniform01()`	generación de un número pseudo-aleatorio uniformemente distribuido en $[0, 1)$
`Uniform(`a`, `b`)`	generación de un número pseudo-aleatorio uniformemente distribuido en $[a, b)$
`Normal01()`	generación de una variable gaussiana pseudo-aleatoria con $\mu = 0$ y $\sigma = 1$
`Normal(`μ`, `σ`)`	generación de una variable gaussiana pseudo-aleatoria con μ y σ dadas

Los argumentos de todas la funciones internas, excepto `card`, `length` y `str2time`, deben ser expresiones numéricas. El argumento de `card` debe ser una expresión de conjunto. El argumento

de `length` y ambos argumentos de `str2time` deben ser expresiones simbólicas.

El valor resultante de una expresión numérica que es una función de referencia es el resultado de aplicar la función a sus argumentos.

Se debe notar que cada función generadora pseudo-aleatoria tiene un argumento latente (*i.e.* algún estado interno) que cambia cada vez que se aplica la función. Así, si la función se aplica repetidamente, aún con argumentos idénticos, debido al efecto colateral siempre se producirán valores resultantes diferentes.

3.1.6 Expresiones iteradas

Una *expresión numérica iterada* es una expresión numérica primaria que tiene la siguiente forma sintáctica:

operador-iterado expresión-indizante integrando

donde *operador-iterado* es el nombre simbólico del operador iterado que se ejecutará (ver más adelante), *expresión-indizante* es una expresión indizante que introduce índices y controla la iteración e *integrando* es una expresión numérica que participa en la operación.

En MathProg existen cuatro operadores iterados que se pueden usar en expresiones numéricas:

`sum`	sumatoria	$\sum_{(i_1,\ldots,i_n)\in\Delta} f(i_1,\ldots,i_n)$
`prod`	multiplicatoria	$\prod_{(i_1,\ldots,i_n)\in\Delta} f(i_1,\ldots,i_n)$
`min`	mínimo	$\min_{(i_1,\ldots,i_n)\in\Delta} f(i_1,\ldots,i_n)$
`max`	máximo	$\max_{(i_1,\ldots,i_n)\in\Delta} f(i_1,\ldots,i_n)$

donde $i_1, \ldots, i_n$ son índices introducidos en la expresión indizante, Δ es el dominio, el conjunto de los n-tuplos especificados en la expresión indizante que define valores particulares asignados a los índices para ejecutar la operación iterada y $f(i_1, \ldots, i_n)$ es el integrando, una expresión numérica cuyo valor resultante depende de los índices.

El valor resultante de una expresión numérica iterada es el resultado de aplicar el operador iterado a sus integrandos a través de todos los n-tuplos contenidos en el dominio.

3.1.7 Expresiones condicionales

Una *expresión numérica condicional* es una expresión numérica primaria que tiene una de las dos formas sintácticas siguientes:

`if` b `then` x `else` y

`if` b `then` x

donde b es una expresión lógica, mientras que x e y son expresiones numéricas.

El valor resultante de un expresión condicional depende del valor de la expresión lógica que sigue a la palabra clave `if`. Si toma el valor *verdadero*, el valor de la expresión condicional es el valor de la expresión que sigue a la palabra clave `then`. De otro modo, si la expresión lógica toma el valor *falso*, el valor de la expresión condicional es el valor de la expresión que sigue a la palabra clave `else`. Si se usa la segunda forma sintáctica, la reducida, y la expresión lógica toma el valor *falso*, el valor resultante de la expresión condicional será cero.

3.1.8 Expresiones parentéticas

Cualquier expresión numérica puede ser encerrada entre paréntesis, lo que las torna sintácticamente en una expresión numérica primaria.

Los paréntesis pueden usarse en expresiones numéricas, como en el álgebra, para especificar el orden deseado en el cual se ejecutarán las operaciones. Cuando se usan paréntesis, la expresión entre paréntesis se evalúa antes que el valor resultante sea usado.

El valor resultante de la expresión parentética es idéntico al valor de la expresión encerrada entre paréntesis.

3.1.9 Operadores aritméticos

En MathProg existen los siguientes operadores aritméticos, los que se pueden usar en expresiones numéricas:

`+` x	más unario
`-` x	menos unario
x `+` y	adición
x `-` y	sustracción
x `less` y	diferencia positiva (si $x < y$ entonces 0, de otro modo $x - y$)
x `*` y	multiplicación
x `/` y	división
x `div` y	cociente de la división exacta
x `mod` y	resto de la división exacta
x `**` y, x `^` y	exponenciación (elevación a una potencia)

donde x e y son expresiones numéricas.

Si la expresión incluye más de un operador aritmético, todos los operadores se ejecutan de izquierda a derecha, de acuerdo con la jerarquía de las operaciones (ver más adelante), con la única excepción del operador de exponenciación que se ejecuta de derecha a izquierda.

El valor resultante de la expresión que contiene operadores aritméticos es el resultado de aplicar los operadores a sus operandos.

3.1.10 Jerarquía de las operaciones

La siguiente lista muestra la jerarquía de las operaciones en expresiones numéricas:

Operación	Jerarquía
Evaluación de funciones (`abs`, `ceil`, etc.)	1.ª
Exponenciación (`**`, `^`)	2.ª
Más y menos unario (`+`, `-`)	3.ª
Multiplicación y división (`*`, `/`, `div`, `mod`)	4.ª
Operaciones iteradas (`sum`, `prod`, `min`, `max`)	5.ª
Adición y sustracción (`+`, `-`, `less`)	6.ª
Evaluación condicional (`if ... then ... else`)	7.ª

Esta jerarquía se usa para determinar cual de dos operaciones consecutivas se realizará primero. Si el primer operador tiene jerarquía mayor o igual que el segundo, la primera operación se realiza. En caso contrario, el segundo operador es comparado con el tercero y así sucesivamente. Cuando se alcanza el final de la expresión, todas las operaciones restantes se realizan en el orden inverso.

3.2 Expresiones simbólicas

Una *expresión simbólica* es una regla para calcular un valor simbólico individual representado como una cadena de caracteres.

La expresión simbólica primaria puede ser un literal de cadena, un índice, un parámetro no-indizado, un parámetro indizado, una función interna de referencia, una expresión simbólica condicional u otra expresión simbólica encerrada entre paréntesis.

También está permitido usar una expresión numérica como la expresión simbólica primaria, en cuyo caso el valor resultante de la expresión numérica se convierte automáticamente al tipo simbólico.

Ejemplos

`'Mayo de 2003'`	(literal de cadena)
`j`	(índice)
`p`	(parámetro no-indizado)
`s['abc',j+1]`	(parámetro indizado)
`substr(nombre[i],k+1,3)`	(función de referencia)
`if i in I then s[i,j] & "..." else t[i+1]`	(expresión condicional)
`((10 * b[i,j]) & '.bis')`	(expresión parentética)

Empleando el operador de concatenación se pueden construir expresiones simbólicas más generales conteniendo dos o más expresiones simbólicas primarias.

Ejemplos

```
'abc[' & i & ',' & j & ']'
"desde " & ciudad[i] " hasta " & ciudad[j]
```

Los principios de evaluación de las expresiones simbólicas son enteramente análogos a los dados para las expresiones numéricas (ver más atrás).

3.2.1 Funciones de referencia

En MathProg existen las siguientes funciones internas que pueden ser usadas en expresiones simbólicas:

`substr(`c`,` x`)`	subcadena de c empezando en la posición x
`substr(`c`,` x`,` y`)`	subcadena de c empezando en la posición x con longitud y
`time2str(`t`,` f`)`	conversión de tiempo calendario a una cadena de caracteres (para los detalles, ver Sección B.3, página 61)

El primer argumento de `substr` debe ser una expresión simbólica, mientras que el segundo y el tercero (opcional) deben ser expresiones numéricas.

El primer argumento de `time2str` debe ser una expresión numérica y su segundo argumento debe ser una expresión simbólica.

El valor resultante de una expresión simbólica que es una función de referencia es el resultado de aplicar la función a sus argumentos.

3.2.2 Operadores simbólicos

Actualmente, en MathProg existe un único operador simbólico:

`c & t`

donde c y t son expresiones simbólicas. Este operador implica la concatenación de sus dos operandos simbólicos, los que son cadenas de caracteres.

3.2.3 Jerarquía de las operaciones

La siguiente lista muestra la jerarquía de las operaciones en expresiones simbólicas:

Operación	Jerarquía
Evaluación de operaciones numéricas	1.ª-7.ª
Concatenación (`&`)	8.ª
Evaluación condicional (`if` ... `then` ... `else`)	9.ª

Esta jerarquía tiene el mismo significado que se explicó anteriormente para expresiones numéricas (ver Subsección 3.1.10, página 17).

3.3 Expresiones de indización e índices

Una *expresión indizante* es una construcción auxiliar que especifica un conjunto plano de n-tuplos e introduce índices. Tiene dos formas sintácticas:

`{` $entrada_1$`,` $entrada_2$`, ...,` $entrada_m$ `}`

`{` $entrada_1$`,` $entrada_2$`, ...,` $entrada_m$ `:` $predicado$ `}`

donde *entrada*$_1$, *entrada*$_2$, ..., *entrada*$_m$ son entradas indizantes y *predicado* es una expresión lógica que especifica un predicado opcional (condición lógica).

Cada *entrada indizante* en la expresión indizante puede tomar una de las tres formas siguientes:

$$i \texttt{ in } C$$
$$\texttt{(}i_1\texttt{, } i_2\texttt{, ...,}i_n\texttt{) in } C$$
$$C$$

donde i_1, i_2, ..., i_n son índices y C es una expresión de conjunto (discutida en la próxima sección) que especifica el conjunto básico.

El número de índices de la entrada indizante debe coincidir con la dimensión del conjunto básico C, *i.e.* si C consiste de 1-tuplos, se debe usar la primera forma y, si C consiste de n-tuplos, siendo $n > 1$, la segunda forma es la que debe usarse.

Si se usa la primera forma de la entrada indizante, el índice i sólo puede ser un índice (ver más adelante). Si se usa la segunda forma, los índices i_1, i_2, ..., i_n pueden ser indistintamente índices o alguna expresión numérica o simbólica, siempre que al menos uno de los índices sea un índice. La tercera forma, reducida, de la entrada indizante tiene el mismo efecto que si i (si C es 1-dimensional) o i_1, i_2, ..., i_n (si C es n-dimensional) se hubieran especificado todos como índices.

Un *índice* es un objeto auxiliar del modelo que actúa como una variable individual. Los valores asignados a los índices son componentes de los n-tuplos de conjuntos básicos, *i.e.* algunas cantidades numéricas y simbólicas.

Para referenciarlos, los índices pueden ser provistos con nombres simbólicos. Sin embargo, a diferencia de otros objetos del modelo (conjuntos, parámetros, etc.), los índices no necesitan ser declarados explícitamente. Cada nombre simbólico *no-declarado* que se usa en una posición indizante de alguna entrada indizante es reconocido como el nombre simbólico correspondiente al índice.

Los nombre simbólicos de los índices son válidos solamente dentro del alcance de la expresión indizante en la que se introdujo el índice. Más allá del alcance, estos índices son completamente inaccesibles, de modo que los mismos nombres simbólicos se pueden usar para diferentes propósitos, en particular para representar índices en otras expresiones indizantes.

El alcance de la expresión indizante, en el que las declaraciones implícitas de los índices son válidas, depende del contexto en que se usa la expresión indizante:

— Si la expresión indizante se usa en un operador-iterado, su alcance se extiende hasta el final del integrando;

— Si la expresión indizante se usa como una expresión de conjunto primaria, su alcance se extiende hasta el final de esta expresión indizante;

— Si la expresión indizante se usa para definir el dominio del subíndice en la declaración de algún objeto del modelo, su alcance se extiende hasta el final de la correspondiente sentencia.

El mecanismo de indización implementado mediante las expresiones indizantes se explica mejor con algunos ejemplos que se discuten a continuación.

Sean tres conjuntos:

$$A = \{4, 7, 9\},$$
$$B = \{(1, Ene), (1, Feb), (2, Mar), (2, Abr), (3, May), (3, Jun)\},$$
$$C = \{a, b, c\},$$

donde A y C consisten de 1-tuplos (singletones) y B consiste de 2-tuplos (duplos). Considérese la siguiente expresión indizante:

```
{i in A, (j,k) in B, l in C}
```

donde `i`, `j`, `k` y `l` son índices.

Aunque MathProg no es un lenguaje de programación por procedimientos, para cualquier expresión indizante se puede dar una descripción algorítmica equivalente. En particular, la descripción algorítmica de la expresión indizante anterior se vería como sigue:

para todo $i \in A$ **ejecutar**
  para todo $(j, k) \in B$ **ejecutar**
    para todo $l \in C$ **ejecutar**
      acción;

donde los índices i, j, k y l son asignados consecutivamente a los correspondientes componentes de los n-tuplos a partir de los conjuntos básicos A, B y C y *acción* es alguna acción que depende del contexto en el que se esté usando la expresión indizante. Por ejemplo, si la acción fuese imprimir los valores corrientes de los índices, la impresión se vería como sigue:

$$\begin{array}{llll}
i = 4 & j = 1 & k = Ene & l = a \\
i = 4 & j = 1 & k = Ene & l = b \\
i = 4 & j = 1 & k = Ene & l = c \\
i = 4 & j = 1 & k = Feb & l = a \\
i = 4 & j = 1 & k = Feb & l = b \\
 & & \cdots & \\
i = 9 & j = 3 & k = Jun & l = b \\
i = 9 & j = 3 & k = Jun & l = c
\end{array}$$

Sea la expresión indizante del ejemplo usada en la siguiente operación iterada:

```
sum{i in A, (j,k) in B, l in C} p[i,j,k,l]
```

donde `p` es un parámetro numérico 4-dimensional o alguna expresión numérica cuyos valores resultantes dependan de `i`, `j`, `k` y `l`. En este caso la acción es la sumatoria, de modo que el valor resultante de la expresión numérica primaria es:

$$\sum_{i \in A, (j,k) \in B, l \in C} (p_{ijkl}).$$

Ahora, sea la expresión indizante del ejemplo usada como una expresión de conjunto primaria. En este caso, la acción es reunir a todos los 4-tuplos (cuádruplos) de la forma (i, j, k, l) en un

conjunto, de modo que el valor resultante de tal operación es simplemente el producto cartesiano de los conjuntos básicos:

$$A \times B \times C = \{(i, j, k, l) : i \in A, (j, k) \in B, l \in C\}.$$

Se debe notar que, en este caso, la misma expresión indizante podría escribirse en la forma reducida:

```
{A, B, C}
```

ya que los índices i, j, k y l no son referenciados y, consecuentemente, sus nombres simbólicos no necesitan ser especificados.

Finalmente, sea la expresión indizante del ejemplo usada como el dominio del subíndice en la declaración de algún objeto 4-dimensional del modelo, por ejemplo un parámetro numérico:

```
param p{i in A, (j,k) in B, l in C} ...;
```

En este caso la acción es generar los miembros del parámetro, donde cada uno de los cuales tiene la forma $p[i, j, k, l]$.

Como se dijo anteriormente, algunos índices en la segunda forma de las entradas indizantes pueden ser expresiones numéricas o simbólicas, no solamente índices. En este caso, los valores resultantes de tales expresiones desempeñan el papel de algunas condiciones lógicas para seleccionar, solamente, aquellos n-tuplos del producto cartesiano de los conjuntos básicos que satisfacen estas condiciones.

Considérese, por ejemplo, la siguiente expresión indizante:

```
{i in A, (i-1,k) in B, l in C}
```

donde `i`, `k` y `l` son índices e `i-1` es una expresión numérica. La descripción algorítmica de esta expresión indizante sería la siguiente:

para todo $i \in A$ **ejecutar**
  para todo $(j, k) \in B$ **y** $j = i - 1$ **ejecutar**
    para todo $l \in C$ **ejecutar**
      acción;

Así, si la expresión indizante se usara como una expresión de conjunto primaria, el conjunto resultante sería el siguiente:

$$\{(4, May, a), (4, May, b), (4, May, c), (4, Jun, a), (4, Jun, b), (4, Jun, c)\}.$$

Debe notarse que, en este caso, el conjunto resultante consistirá de 3-tuplos y no de 4-tuplos, puesto que en la expresión indizante no hay un índice que corresponda al primer componente de los 2-tuplos del conjunto B.

La regla general es: el número de componentes de los n-tuplos definido por una expresión indizante es igual al número de índices de tal expresión, en la que la correspondencia entre los índices y los componentes de los n-tuplos en el conjunto resultante es posicional, *i.e.* el primer índice se corresponde con el primer componente, el segundo índice se corresponde con el segundo componente, etc.

En algunos casos es necesario seleccionar un subconjunto del producto cartesiano de algunos conjuntos. Esto se puede lograr mediante el empleo de un predicado lógico opcional, el que se especifica en la expresión indizante.

Considérese, por ejemplo, la siguiente expresión indizante:

```
{i in A, (j,k) in B, l in C: i <= 5 and k <> 'Mar'}
```

donde la expresión lógica que sigue a los dos puntos es un predicado. La descripción algorítmica de tal expresión indizante sería la siguiente:

para todo $i \in A$ **ejecutar**
  para todo $(j,k) \in B$ **ejecutar**
    para todo $l \in C$ **ejecutar**
      si $i \leq 5$ **y** $k \neq$ 'Mar' **entonces**
        acción;

Así, si esta expresión indizante se usara como una expresión de conjunto primaria, el conjunto resultante sería el siguiente:

$$\{(4,1,Ene,a),(4,1,Feb,a),(4,2,Abr,a),\ldots,(4,3,Jun,c)\}.$$

Si no se especifica un predicado en la expresión indizante, se asume uno que toma el valor *verdadero*.

3.4 Expresiones de conjunto

Una *expresión de conjunto* es una regla para calcular un conjunto elemental, *i.e.* una colección de n-tuplos cuyos componentes son cantidades numéricas y simbólicas.

La expresión de conjunto primaria puede ser un conjunto de literales, un conjunto no-indizado, un conjunto indizado, un conjunto "aritmético", una expresión indizante, una expresión de conjunto iterada, una expresión de conjunto condicional u otra expresión de conjunto encerrada entre paréntesis.

Ejemplos

`{(123,'aaa'), (i+1,'bbb'), (j-1,'ccc')}`	(conjunto de literales)
`I`	(conjunto no-indizado)
`S[i-1,j+1]`	(conjunto indizado)
`1..t-1 by 2`	(conjunto "aritmético")
`{t in 1..T, (t+1,j) in S: (t,j) in F}`	(expresión indizante)
`setof{i in I, j in J}(i+1,j-1)`	(expresión de conjunto iterada)
`if i < j then S[i,j] else F diff S[i,j]`	(expresión de conjunto condicional)
`(1..10 union 21..30)`	(expresión de conjunto parentética)

Empleando ciertos operadores de conjunto se pueden construir expresiones de conjunto más generales conteniendo dos o más expresiones de conjunto primarias.

Ejemplos

```
(A union B) inter (I cross J)
1..10 cross (if i < j then {'a', 'b', 'c'} else {'d', 'e', 'f'})
```

3.4.1 Conjuntos de literales

Un *conjunto de literales* es una expresión de conjunto primaria que tiene las dos formas sintácticas siguientes:

$$\texttt{\{}e_1\texttt{, }e_2\texttt{, ..., }e_m\texttt{\}}$$

$$\texttt{\{(}e_{11}\texttt{, ..., }e_{1n}\texttt{), (}e_{21}\texttt{, ..., }e_{2n}\texttt{), ..., (}e_{m1}\texttt{, ..., }e_{mn}\texttt{)\}}$$

donde e_1, ..., e_m, e_{11}, ..., e_{mn} son expresiones numéricas o simbólicas.

Si se usa la primera forma, el conjunto resultante consiste de 1-tuplos (singletones), enumerados entre las llaves. Se permite especificar un conjunto vacío como `{ }`, el que no tiene 1-tuplos. Si se usa la segunda forma, el conjunto resultante consiste de n-tuplos enumerados entre las llaves, donde cada n-tuplo particular está compuesto por los correspondientes componentes enumerados entre los paréntesis. Todos los n-tuplos deben tener el mismo número de componentes.

3.4.2 Conjuntos no-indizados

Si la expresión de conjunto primaria es un conjunto no-indizado (el que debe ser 0-dimensional), el conjunto resultante es un conjunto elemental asociado con el objeto conjunto correspondiente.

3.4.3 Conjuntos indizados

La expresión de conjunto primaria que se refiere a un conjunto indizado tiene la siguiente forma sintáctica:

$$\textit{nombre}\texttt{[}i_1\texttt{, }i_2\texttt{, ..., }i_n\texttt{]}$$

donde *nombre* es el nombre simbólico del objeto conjunto e i_1, i_2, ..., i_n son subíndices.

Cada subíndice debe ser una expresión numérica o simbólica. El número de subíndices en la lista de subíndices debe coincidir con la dimensión del objeto conjunto al cual está asociada la lista de subíndices.

Los valores corrientes de las expresiones de los subíndices se usan para identificar un miembro particular del objeto conjunto que determina el conjunto resultante.

3.4.4 Conjuntos "aritméticos"

La expresión de conjunto primaria que constituye un conjunto "aritmético" tiene las dos formas sintácticas siguientes:

$$t_0\ \texttt{..}\ t_1\ \texttt{by}\ \delta t$$

$$t_0\ \texttt{..}\ t_1$$

donde t_0, t_1 y δt son expresiones numéricas (el valor de δt no debe ser cero). La segunda forma es equivalente a la primera con $\delta t = 1$.

Si $\delta t > 0$, el conjunto resultante se determina como sigue:

$$\{t : \exists k \in \mathcal{Z}(t = t_0 + k\delta t,\ t_0 \leq t \leq t_1)\}.$$

De otro modo, si $\delta t < 0$, el conjunto resultante se determina como sigue:

$$\{t : \exists k \in \mathcal{Z}(t = t_0 + k\delta t,\ t_1 \leq t \leq t_0)\}.$$

3.4.5 Expresiones de indización

Si la expresión de conjunto primaria es una expresión indizante, el conjunto resultante se determina como se ha descripto anteriormente en la Sección 3.3, página 19.

3.4.6 Expresiones iteradas

Una *expresión de conjunto iterada* es una expresión de conjunto primaria que tiene la siguiente forma sintáctica:

`setof` *expresión-indizante integrando*

donde *expresión-indizante* es una expresión indizante que introduce índices y controla la iteración e *integrando* es tanto una expresión numérica o simbólica individual como una lista de expresiones numéricas o simbólicas separadas por coma y encerradas entre paréntesis.

Si el integrando es una expresión numérica o simbólica individual, el conjunto resultante está compuesto por 1-tuplos y se determina como sigue:

$$\{x : (i_1, \ldots, i_n) \in \Delta\},$$

donde x es un valor del integrando, i_1, ..., i_n son índices introducidos en la expresión indizante y Δ es el dominio, un conjunto de n-tuplos especificados por la expresión indizante que define los valores particulares asignados a los índices para realizar la operación iterada.

Si el integrando es una lista conteniendo m expresiones numéricas y simbólicas, el conjunto resultante está compuesto por m-tuplos y se determina como sigue:

$$\{(x_1, \ldots, x_m) : (i_1, \ldots, i_n) \in \Delta\},$$

donde x_1, ..., x_m son valores de las expresiones en la lista de integrandos e i_1, ..., i_n y Δ tienen el mismo significado anterior.

3.4.7 Expresiones condicionales

Una *expresión de conjunto condicional* es una expresión de conjunto primaria que tiene la siguiente forma sintáctica:

`if` b `then` X `else` Y

donde b es una expresión lógica y X e Y son expresiones de conjunto que deben definir conjuntos de igual dimensión.

El valor resultante de la expresión condicional depende del valor de la expresión lógica que sigue a la palabra clave `if`. Si toma el valor *verdadero*, el conjunto resultante es el valor de la expresión que sigue a la palabra clave `then`. De otro modo, si la expresión lógica toma el valor *falso*, el conjunto resultante es el valor de la expresión que sigue a la palabra clave `else`.

3.4.8 Expresiones parentéticas

Cualquier expresión de conjunto puede encerrarse entre paréntesis, lo que la convierte sintácticamente en una expresión de conjunto primaria.

Los paréntesis pueden usarse en expresiones de conjunto, como en el álgebra, para especificar el orden deseado en el cual se ejecutarán las operaciones. Cuando se usan paréntesis, la expresión entre paréntesis se evalúa antes que el valor resultante sea usado.

El valor resultante de la expresión parentética es idéntico al valor de la expresión encerrada entre paréntesis.

3.4.9 Operadores de conjunto

En MathProg existen los siguientes operadores de conjunto, los que se pueden usar en expresiones de conjunto:

X `union` Y	union $X \cup Y$
X `diff` Y	diferencia $X \backslash Y$
X `symdiff` Y	diferencia simétrica $X \oplus Y = (X \backslash Y) \cup (Y \backslash X)$
X `inter` Y	intersección $X \cap Y$
X `cross` Y	producto cartesiano ("cruzado") $X \times Y$

donde X e Y son expresiones de conjunto que deben definir conjuntos de la misma dimensión (excepto para el producto cartesiano).

Si la expresión incluye más de un operador de conjunto, todos los operadores se ejecutan de izquierda a derecha, de acuerdo con la jerarquía de las operaciones (ver más adelante).

El valor resultante de la expresión que contiene operadores de conjunto es el resultado de aplicar los operadores a sus operandos.

La dimensión del conjunto resultante, *i.e.* la dimensión de los n-tuplos de los que consiste el conjunto resultante, es la dimensión de los operandos, excepto en el producto cartesiano, en la que la dimensión del conjunto resultante es la suma de las dimensiones de sus operandos.

3.4.10 Jerarquía de las operaciones

La siguiente lista muestra la jerarquía de las operaciones en expresiones de conjunto:

Operación	Jerarquía
Evaluación de operaciones numéricas	1.ª-7.ª
Evaluación de operaciones simbólicas	8.ª-9.ª
Evaluación de conjuntos iterados o "aritméticos" (`setof`, `..`)	10.ª
Producto cartesiano (`cross`)	11.ª
Intersección (`inter`)	12.ª
Unión y diferencia (`union`, `diff`, `symdiff`)	13.ª
Evaluación condicional (`if ... then ... else`)	14.ª

Esta jerarquía tiene el mismo significado que se explicó anteriormente para expresiones numéricas (ver Subsección 3.1.10, página 17).

3.5 Expresiones lógicas

Una *expresión lógica* es una regla para calcular un valor lógico individual, el que puede ser tanto *verdadero* como *falso*.

La expresión lógica primaria puede ser una expresión numérica, una expresión relacional, una expresión lógica iterada u otra expresión lógica encerrada entre paréntesis.

Ejemplos

`i+1`	(expresión numérica)
`a[i,j] < 1.5`	(expresión relacional)
`s[i+1,j-1] <> 'Mar' & anho`	(expresión relacional)
`(i+1,'Ene') not in I cross J`	(expresión relacional)
`S union T within A[i] inter B[j]`	(expresión relacional)
`forall{i in I, j in J} a[i,j] < .5 * b[i]`	(expresión lógica iterada)
`(a[i,j] < 1.5 or b[i] >= a[i,j])`	(expresión lógica parentética)

Empleando ciertos operadores lógicos se pueden construir expresiones lógicas más generales conteniendo dos o más expresiones lógicas primarias.

Ejemplos

```
not (a[i,j] < 1.5 or b[i] >= a[i,j]) and (i,j) in S
(i,j) in S or (i,j) not in T diff U
```

3.5.1 Expresiones numéricas

El valor resultante de una expresión lógica primaria, cuando es una expresión numérica, es *verdadero* si el valor resultante de la expresión numérica es distinto de cero. De otro modo, el valor resultante de la expresión lógica es *falso*.

3.5.2 Operadores relacionales

En MathProg existen los siguientes operadores relacionales, los que se pueden usar en expresiones lógicas:

x `<` y	comprueba si $x < y$
x `<=` y	comprueba si $x \leq y$
x `=` y, x `==` y	comprueba si $x = y$
x `>=` y	comprueba si $x \geq y$
x `>` y	comprueba si $x > y$
x `<>` y, x `!=` y	comprueba si $x \neq y$
x `in` Y	comprueba si $x \in Y$
(`(`x_1`,...,`x_n`)` `in` Y	comprueba si $(x_1, \ldots, x_n) \in Y$
x `not in` Y, x `!in` Y	comprueba si $x \notin Y$
(`(`x_1`,...,`x_n`)` `not in` Y, (`(`x_1`,...,`x_n`)` `!in` Y	comprueba si $(x_1, \ldots, x_n) \notin Y$
X `within` Y	comprueba si $X \subseteq Y$
X `not within` Y, X `!within` Y	comprueba si $X \not\subseteq Y$

donde x, x_1, ..., x_n e y son expresiones numéricas o simbólicas, mientras que X e Y son expresiones de conjunto.

Notas:

1. En las operaciones `in`, `not in` y `!in` el número de componentes del primer operando debe ser igual a la dimensión del segundo operando.

2. En las operaciones `within`, `not within` y `!within` ambos operandos deben tener la misma dimensión.

Todos los operadores relacionales listados anteriormente tienen su significado matemático convencional. El valor resultante es *verdadero* si los operandos satisfacen la correspondiente relación, o es *falso* en caso contrario. (Debe notarse que los valores simbólicos se ordenan lexicográficamente y que cualquier valor numérico precede a cualquier valor simbólico.)

3.5.3 Expresiones iteradas

Una *expresión lógica iterada* es una expresión lógica primaria que tiene la siguiente forma sintáctica:

operador-iterado expresión-indizante integrando

donde *operador-iterado* es el nombre simbólico del operador iterado que se ejecutará (ver más adelante), *expresión-indizante* es una expresión indizante que introduce índices y controla la iteración e *integrando* es una expresión lógica que participa en la operación.

En MathProg existen dos operadores iterados que se pueden usar en expresiones lógicas:

`forall`	cuantificador-$\forall$	$\forall(i_1, \ldots, i_n) \in \Delta[f(i_1, \ldots, i_n)]$,
`exists`	cuantificador-$\exists$	$\exists(i_1, \ldots, i_n) \in \Delta[f(i_1, \ldots, i_n)]$,

donde i_1, ..., i_n son índices introducidos en la expresión indizante, Δ es el dominio, un conjunto de n-tuplos especificados por la expresión indizante que define los valores particulares asignados a los índices para ejecutar la operación iterada y $f(i_1, \ldots, i_n)$ es el integrando, una expresión lógica cuyo valor resultante depende de los índices.

Para el cuantificador-$\forall$, el valor resultante de la expresión lógica iterada es *verdadero* si el valor

del integrando es *verdadero* para todos los n-tuplos contenidos en el dominio, de otro modo es *falso*.

Para el cuantificador-$\exists$, el valor resultante de la expresión lógica iterada es *falso* si el valor del integrando es *falso* para todos los n-tuplos contenidos en el dominio, de otro modo es *verdadero*.

3.5.4 Expresiones parentéticas

Cualquier expresión lógica puede encerrarse entre paréntesis, lo que la convierte sintácticamente en una expresión lógica primaria.

Los paréntesis pueden usarse en expresiones lógicas, como en el álgebra, para especificar el orden deseado en el cual se ejecutarán las operaciones. Cuando se usan paréntesis, la expresión entre paréntesis se evalúa antes que el valor resultante sea usado.

El valor resultante de la expresión parentética es idéntico al valor de la expresión encerrada entre paréntesis.

3.5.5 Operadores lógicos

En MathProg existen los siguientes operadores lógicos, los que se pueden usar en expresiones lógicas:

`not` x, `!`x	negación $\neg\ x$
x `and` y, x `&&` y	conjunción ("y" lógico) $x\ \&\ y$
x `or` y, x `||` y	disyunción ("o" lógico) $x \vee y$

donde x e y son expresiones lógicas.

Si la expresión incluye más de un operador lógico, todos los operadores se ejecutan de izquierda a derecha, de acuerdo con la jerarquía de las operaciones (ver más adelante). El valor resultante de la expresión que contiene operadores lógicos es el resultado de aplicar los operadores a sus operandos.

3.5.6 Jerarquía de las operaciones

La siguiente lista muestra la jerarquía de las operaciones en expresiones lógicas:

Operación	Jerarquía
Evaluación de operaciones numéricas	1.ª-7.ª
Evaluación de operaciones simbólicas	8.ª-9.ª
Evaluación de operaciones de conjuntos	10.ª-14.ª
Operaciones relacionales (`<`, `<=`, etc.)	15.ª
Negación (`not`, `!`)	16.ª
Conjunción (`and`, `&&`)	17.ª
Cuantificación-$\forall$ y -$\exists$ (`forall`, `exists`)	18.ª
Disyunción (`or`, `||`)	19.ª

Esta jerarquía tiene el mismo significado que se explicó anteriormente para expresiones numéricas (ver Subsección 3.1.10, página 17).

3.6 Expresiones lineales

Una *expresión lineal* es una regla para calcular la denominada *forma lineal*, que es una función lineal (o afín) de variables elementales.

La expresión lineal primaria puede ser una variable no-indizada, una variable indizada, una expresión lineal iterada, una expresión lineal condicional u otra expresión lineal encerrada entre paréntesis.

También está permitido usar una expresión numérica como una expresión lineal primaria, en cuyo caso el valor resultante de la expresión numérica se convierte automáticamente a una forma lineal que incluye el término constante solamente.

Ejemplos

`z`	(variable no-indizada)
`x[i,j]`	(variable indizada)
`sum{j in J} (a[i,j] * x[i,j] + 3 * y[i-1])`	(expresión lineal iterada)
`if i in I then x[i,j] else 1.5 * z + 3.25`	(expresión lineal condicional)
`(a[i,j] * x[i,j] + y[i-1] + .1)`	(expresión lineal parentética)

Empleando ciertos operadores aritméticos se pueden construir expresiones lineales más generales conteniendo dos o más expresiones lineales primarias.

Ejemplos

```
2 * x[i-1,j+1] + 3.5 * y[k] + .5 * z
(- x[i,j] + 3.5 * y[k]) / sum{t in T} abs(d[i,j,t])
```

3.6.1 Variables no-indizadas

Si la expresión lineal primaria es una variable no-indizada (que debe ser 0-dimensional), la forma lineal resultante es aquella variable no-indizada.

3.6.2 Variables indizadas

La expresión lineal primaria que se refiere a una variable indizada tiene la siguiente forma sintáctica:

$$nombre[i_1, i_2, \ldots, i_n]$$

donde *nombre* es el nombre simbólico de la variable del modelo e i_1, i_2, ..., i_n son subíndices.

Cada subíndice debe ser una expresión numérica o simbólica. El número de subíndices en la lista de subíndices debe ser igual a la dimensión de la variable del modelo con la cual está asociada la lista de subíndices.

Los valores corrientes de las expresiones de los subíndices se usan para identificar un miembro particular de la variable del modelo que determina la forma lineal resultante, la cual es una variable elemental asociada con el miembro correspondiente.

3.6.3 Expresiones iteradas

Una *expresión lineal iterada* es una expresión lineal primaria que tiene la siguiente forma sintáctica:

`sum` *expresión-indizante integrando*

donde *expresión-indizante* es una expresión indizante que introduce índices y controla la iteración e *integrando* es una expresión lineal que participa en la operación.

La expresión lineal iterada se evalúa exactamente de la misma manera que la expresión numérica iterada (ver Subsección 3.1.6, página 16), excepto que el integrando participante en la sumatoria es una forma lineal y no un valor numérico.

3.6.4 Expresiones condicionales

Una *expresión lineal condicional* es una expresión lineal primaria que tiene alguna de las dos formas sintácticas siguientes:

`if` b `then` f `else` g

`if` b `then` f

donde b es una expresión lógica, mientras que f y g son expresiones lineales.

La expresión lineal condicional se evalúa exactamente de la misma manera que la expresión numérica condicional (ver Subsección 3.1.7, página 16), excepto que los operandos participantes en la operación son formas lineales y no valores numéricos.

3.6.5 Expresiones parentéticas

Cualquier expresión lineal puede encerrarse entre paréntesis, lo que la convierte sintácticamente en una expresión lineal primaria.

Los paréntesis pueden usarse en expresiones lineales, como en el álgebra, para especificar el orden deseado en el cual se ejecutarán las operaciones. Cuando se usan paréntesis, la expresión entre paréntesis se evalúa antes que el valor resultante sea usado.

El valor resultante de la expresión parentética es idéntico al valor de la expresión encerrada entre paréntesis.

3.6.6 Operadores aritméticos

En MathProg existen los siguientes operadores aritméticos, los que se pueden usar en expresiones lineales:

`+` f	más unario
`-` f	menos unario
f `+` g	adición
f `-` g	sustracción
x `*` f, f `*` x	multiplicación
f `/` x	división

donde f y g son expresiones lineales, mientras que x es una expresión numérica (más precisamente, una expresión lineal conteniendo únicamente el término constante).

Si la expresión incluye más de un operador aritmético, todos los operadores se ejecutan de izquierda a derecha, de acuerdo con la jerarquía de las operaciones (ver más adelante). El valor resultante de la expresión que contiene operadores aritméticos es el resultado de aplicar los operadores a sus operandos.

3.6.7 Jerarquía de las operaciones

La jerarquía de las operaciones aritméticas usadas en las expresiones lineales es la misma que en las expresiones numéricas (ver Subsección 3.1.10, página 17).

Capítulo 4

Sentencias

Las *sentencias* son unidades básicas de la descripción del modelo. En MathProg todas las sentencias se clasifican en dos categorías: sentencias declarativas y sentencias funcionales.

Las *sentencias declarativas* (sentencia *set*, sentencia *parameter*, sentencia *variable*, sentencia *constraint* y sentencia *objective*) se usan para declarar objetos de cierto tipo del modelo y definir ciertas propiedades de tales objetos.

Las *sentencias funcionales* (sentencia *solve*, sentencia *check*, sentencia *display*, sentencia *printf*, sentencia *loop* y sentencia *table*) están ideadas para ejecutar ciertas acciones específicas.

Debe notarse que las sentencias declarativas pueden seguir cualquier orden arbitrario, lo cual no afecta el resultado de la traducción. Sin embargo, todos los objetos del modelo deben ser declarados antes de ser referenciados en otras sentencias.

4.1 Sentencia set

`set` *nombre alias dominio* `,` *atributo* `,` ... `,` *atributo* `;`

nombre es el nombre simbólico del conjunto;

alias es un literal de cadena opcional que especifica un alias para el conjunto;

dominio es una expresión-indizante opcional que especifica el dominio del subíndice del conjunto;

atributo, ..., *atributo* son atributos opcionales del conjunto (las comas que preceden a los atributos se pueden omitir).

Atributos opcionales

`dimen` n
: especifica la dimensión de los n-tuplos de los que consiste el conjunto;

`within` *expresión*
: especifica un superconjunto que restringe al conjunto o a todos sus miembros (conjuntos elementales) a estar incluido en aquel superconjunto;

`:=` *expresión*
especifica un conjunto elemental asignado al conjunto o sus miembros;

`default` *expresión*
especifica un conjunto elemental asignado al conjunto o sus miembros cuando no haya datos apropiados disponibles en la sección de datos.

Ejemplos

```
set nodos;
set arcos within nodos cross nodos;
set paso{p in 1..maxiter} dimen 2 := if p = 1 then arcos else paso[p-1]
   union setof{k in nodos, (i,k) in paso[p-1], (k,j) in paso[p-1]}(i,j);
set A{i in I, j in J}, within B[i+1] cross C[j-1], within D diff E,
   default {('abc',123), (321,'cba')};
```

La sentencia set declara un conjunto. Si no se especifica el dominio del subíndice, el conjunto será simple, de otro modo será un arreglo de conjuntos elementales.

El atributo `dimen` especifica la dimensión de los n-tuplos de los que consiste el conjunto (si es un conjunto simple) o sus miembros (si el conjunto es un arreglo de conjuntos elementales), en la que n debe ser un entero sin signo desde 1 hasta 20. Como mucho se puede especificar un atributo `dimen`. Si no se especifica el atributo `dimen`, la dimensión de los n-tuplos se determina implícitamente por otros atributos (por ejemplo, si hay una expresión de conjunto que sigue a `:=` o a la palabra clave `default`, se usará la dimensión de los n-tuplos del correspondiente conjunto elemental). Si no hay información disponible sobre la dimensión, se asume `dimen 1`.

El atributo `within` especifica una expresión de conjunto cuyo valor resultante es un superconjunto usado para restringir al conjunto (si es un conjunto simple) o a sus miembros (si el conjunto es un arreglo de conjuntos elementales) a estar incluido en aquel superconjunto. Se puede especificar un número arbitrario de atributos `within` en la misma sentencia set.

El atributo de asignación (`:=`) especifica una expresión de conjunto usada para evaluar conjuntos elementales asignados al conjunto (si es un conjunto simple) o sus miembros (si el conjunto es un arreglo de conjuntos elementales). Si se especifica el atributo de asignación, el conjunto es *calculable* y consecuentemente no es necesario proveerle datos en la sección de datos. Si no se especifica el atributo de asignación, entonces se deben proveer datos para el conjunto en la sección de datos. Como mucho, se puede especificar una asignación o un atributo `default` para el mismo conjunto.

El atributo `default` especifica una expresión de conjunto usada para evaluar conjuntos elementales asignados al conjunto (si es un conjunto simple) o sus miembros (si el conjunto es un arreglo de conjuntos elementales) toda vez que no haya datos apropiados disponibles en la sección de datos. Si no se especifica una asignación o un atributo `default`, la carencia de datos causará un error.

4.2 Sentencia parameter

`param` *nombre alias dominio* `,` *atributo* `,` ... `,` *atributo* `;`

nombre es el nombre simbólico del parámetro;

alias es un literal de cadena opcional que especifica un alias para el parámetro;

dominio es una expresión indizante opcional que especifica el dominio del subíndice del parámetro;

atributo, ..., *atributo* son atributos opcionales del parámetro (las comas que preceden a los atributos se pueden omitir).

Atributos opcionales

`integer`
: especifica que el parámetro es entero;

`binary`
: especifica que el parámetro es binario;

`symbolic`
: especifica que el parámetro es simbólico;

relación expresión
: (donde *relación* es alguno de: `<`, `<=`, `=`, `==`, `>=`, `>`, `<>`, `!=`)
especifica una condición que restringe al parámetro o a sus miembros a satisfacer aquella condición;

`in` *expresión*
: especifica un superconjunto que restringe al parámetro o a sus miembros a estar incluidos en aquel superconjunto;

`:=` *expresión*
: especifica un valor asignado al parámetro o a sus miembros;

`default` *expresión*
: especifica un valor asignado al parámetro o a sus miembros cuando no haya datos apropiados disponibles en la sección de datos.

Ejemplos

```
param unidades{insumo,producto} >= 0;
param ganancia{producto, 1..T+1};
param N := 20 integer >= 0 <= 100;
param combinacion 'n elige k' {n in 0..N, k in 0..n} :=
   if k = 0 or k = n then 1 else combinacion[n-1,k-1] + combinacion[n-1,k];
param p{i in I, j in J}, integer, >= 0, <= i+j, in A[i] symdiff B[j],
   in C[i,j], default 0.5 * (i + j);
param mes symbolic default 'May' in {'Mar', 'Abr', 'May'};
```

La sentencia parameter declara un parámetro. Si el dominio del subíndice no se especifica, el parámetro es un parámetro simple (escalar); de otro modo es un arreglo n-dimensional.

Los atributos de tipo `integer`, `binary` y `symbolic` califican el tipo de valores que se le puede asignar al parámetro como se muestra a continuación:

Atributo de tipo	Valores asignado
(no especificado)	Cualquier valor numérico
`integer`	Solamente valores numéricos enteros
`binary`	Tanto 0 como 1
`symbolic`	Cualquier valor numérico y simbólico

El atributo `symbolic` no se puede especificar junto con otros atributos de tipo. Cuando se especifica debe preceder a todos los demás atributos.

El atributo de condición especifica una condición opcional que restringe los valores asignados al parámetro a satisfacer esta condición. Este atributo tiene las siguientes formas sintácticas:

`<` v	comprueba si $x < v$
`<=` v	comprueba si $x \leq v$
`=` v, `==` v	comprueba si $x = v$
`>=` v	comprueba si $x \geq v$
`>` v	comprueba si $x \geq v$
`<>` v, `!=` v	comprueba si $x \neq v$

donde x es un valor asignado al parámetro y v es el valor resultante de una expresión numérica o simbólica especificada en el atributo de condición. Se puede especificar un número arbitrario de atributos de condición para el mismo parámetro. Si el valor que se está asignando al parámetro durante la evaluación del modelo viola al menos una de las condiciones especificadas, se producirá un error. (Debe notarse que los valores simbólicos se ordenan lexicográficamente y que cualquier valor numérico precede a cualquier valor simbólico.)

El atributo `in` es semejante al atributo de condición y especifica una expresión de conjunto cuyo valor resultante es un superconjunto usado para restringir los valores numéricos o simbólicos asignados al parámetro a estar incluidos en aquel superconjunto. Se puede especificar un número arbitrario de atributos `in` para el mismo parámetro. Si el valor que se está asignando al parámetro durante la evaluación del modelo no pertenece al menos a uno de los superconjuntos especificados, se producirá un error.

El atributo de asignación (`:=`) especifica una expresión numérica o simbólica usada para calcular un valor asignado al parámetro (si es un parámetro simple) o a sus miembros (si el parámetro es un arreglo). Si se especifica el atributo de asignación, el parámetro es *calculable* y consecuentemente no es necesario proveer datos en la sección de datos. Si no se especifica el atributo de asignación, entonces se deben proveer datos para el parámetro en la sección de datos. Como mucho, se puede especificar una asignación o un atributo `default` para el mismo conjunto.

El atributo `default` especifica una expresión numérica o simbólica que se usa para calcular un valor asignado al parámetro o a sus miembros, toda vez que no haya datos apropiados disponibles en la sección de datos. Si no se especifica una asignación ni un atributo `default`, la carencia de datos causará un error.

4.3 Sentencia variable

`var` *nombre alias dominio* , *atributo* , ... , *atributo* ;

nombre es el nombre simbólico de la variable;

alias es un literal de cadena opcional que especifica un alias para la variable;

dominio es una expresión indizante opcional que especifica el dominio del subíndice de la variable;

atributo, ..., *atributo* son atributos opcionales de la variable (las comas que preceden a los atributos se pueden omitir).

Atributos opcionales

`integer`
restringe la variable a ser entera;

`binary`
restringe la variable a ser binaria;

`>=` *expresión*
especifica una cota inferior para la variable;

`<=` *expresión*
especifica una cota superior para la variable;

`=` *expresión*
especifica un valor fijo para la variable.

Ejemplos

```
var x >= 0;
var y{I,J};
var elaborar{p in producto}, integer, >= compromiso[p], <= mercado[p];
var almacenar{insumo, 1..T+1} >= 0;
var z{i in I, j in J} >= i+j;
```

La sentencia variable declara una variable. Si no se especifica el dominio del subíndice, la variable es una variable simple (escalar); de otro modo es un arreglo n-dimensional de variables elementales.

Las variables elementales asociadas con una variable del modelo (si es una variable simple) o sus miembros (si es un arreglo) corresponde a las variables en la formulación del problema de PL/PEM (ver Sección 1.1, página 6). Debe notarse que sólo variables elementales realmente referenciadas en algunas restricciones y/u objetivos se incluirán en la instancia del problema de PL/PEM que se generará.

Los atributos de tipo `integer` y `binary` restringen la variable a ser entera o binaria, respectivamente. Si no se especifica un atributo de tipo, la variable será continua. Si todas las variables en el modelo son continuas, el problema correspondiente será de la clase PL. Si hay al menos una variable entera o binaria, el problema será de la clase PEM.

El atributo de cota inferior (`>=`) especifica una expresión numérica para calcular la cota inferior de la variable. Se puede especificar una cota inferior como máximo. Por defecto, todas las variables no tienen cota inferior (excepto las binarias), de modo que si se requiere que sea no-negativa, su cota inferior cero debe especificarse explícitamente.

El atributo de cota superior (`<=`) especifica una expresión numérica para calcular la cota superior de la variable. Se puede especificar una cota superior como máximo.

El atributo de valor fijo (`=`) especifica una expresión numérica para calcular el valor en el cual se fijará la variable. Este atributo no puede especificarse junto con los atributos de cota.

4.4 Sentencia constraint

`s.t.` *nombre alias dominio* `:` *expresión* `,` `=` *expresión* `;`

`s.t.` *nombre alias dominio* `:` *expresión* `,` `<=` *expresión* `;`

`s.t.` *nombre alias dominio* `:` *expresión* `,` `>=` *expresión* `;`

`s.t.` *nombre alias dominio* `:` *expresión* `,` `<=` *expresión* `,` `<=` *expresión* `;`

`s.t.` *nombre alias dominio* `:` *expresión* `,` `>=` *expresión* `,` `>=` *expresión* `;`

nombre es el nombre simbólico de la restricción;

alias es un literal de cadena opcional que especifica un alias para la restricción;

dominio es una expresión indizante opcional que especifica el dominio del subíndice de la restricción;

expresión es una expresión lineal usada para calcular un componente de la restricción (las comas que siguen a las expresiones pueden omitirse).

(La palabra clave `s.t.` se puede escribir como `subject to`, o como `subj to` o ser completamente omitida.)

Ejemplos

```
s.t. r: x + y + z, >= 0, <= 1;
limite{t in 1..T}: sum{j in producto} elaborar[j,t] <= max_producto;
subject to balance{i in insumo, t in 1..T}:
   almacenar[i,t+1] = almacenar[i,t] -
   sum{j in producto} unidades[i,j] * elaborar[j,t];
subject to ltn 'limite tiempo normal' {t in tiempo}:
   sum{p in producto} pt[p] * rprd[p,t] <= 1.3 * dpp[t] * brigadas[t];
```

La sentencia constraint declara una restricción. Si no se especifica el dominio del subíndice, la restricción es una restricción simple (escalar); de otro modo, es un arreglo n-dimensional de restricciones elementales.

Las restricciones elementales asociadas con la restricción del modelo (si es una restricción simple) o sus miembros (si es un arreglo) corresponde a las restricciones lineales en la formulación del problema de PL/PEM (ver Sección 1.1, página 6).

Si la restricción tiene la forma de una igualdad o desigualdad simple, *i.e.* incluye dos expresiones, una de las cuales está a continuación de los dos puntos y la otra está a continuación del signo relacional `=`, `<=` o `>=`, ambas expresiones de la sentencia pueden ser expresiones lineales. Si la restricción tiene la forma de una doble desigualdad, *i.e.* incluye tres expresiones, la expresión del medio puede ser una expresión lineal mientras que la de la izquierda y la de la derecha sólo pueden ser expresiones numéricas.

Generar el modelo es, groseramente hablando, generar sus restricciones, las que son siempre evaluadas para todo el dominio del subíndice. A su vez, la evaluación de las restricciones lleva a la evaluación de otros objetos del modelo tales como los conjuntos, los parámetros y las variables.

La construcción de una restricción lineal real incluida en la instancia del problema, la cual corresponde a una restricción elemental particular, se realiza como sigue.

Si la restricción tiene la forma de una igualdad o desigualdad simple, la evaluación de ambas expresiones lineales resulta en dos formas lineales:

$$\begin{aligned} f &= a_1x_1 + a_2x_2 + \ldots + a_nx_n + a_0, \\ g &= b_1x_1 + b_2x_2 + \ldots + b_nx_n + b_0, \end{aligned}$$

donde x_1, x_2, ..., x_n son variables elementales; a_1, a_2, ..., a_n, b_1, b_2, ..., b_n son coeficientes numéricos y a_0 y b_0 son términos constantes. Luego, todos los términos lineales de f y g se llevan al lado izquierdo y los términos constantes se llevan al lado derecho, para dar la restricción elemental final en forma estándar:

$$(a_1 - b_1)x_1 + (a_2 - b_2)x_2 + \ldots + (a_n - b_n)x_n \left\{ \begin{matrix} = \\ \leq \\ \geq \end{matrix} \right\} b_0 - a_0.$$

Si la restricción tiene la forma de una doble desigualdad, la evaluación de la expresión lineal del medio resulta en una forma lineal:

$$f = a_1x_1 + a_2x_2 + \ldots + a_nx_n + a_0,$$

y la evaluación de las expresiones numéricas de la izquierda y de la derecha dará dos valores numéricos l y u, respectivamente. Luego, el término constante de la forma lineal se lleva tanto a la izquierda como a la derecha para dar la restricción elemental final en forma estándar:

$$l - a_0 \leq a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq u - a_0.$$

4.5 Sentencia objective

`minimize` *nombre alias dominio* `:` *expresión* `;`

`maximize` *nombre alias dominio* `:` *expresión* `;`

nombre es el nombre simbólico del objetivo;

alias es un literal de cadena opcional que especifica un alias para el objetivo;

dominio es una expresión indizante opcional que especifica el dominio del subíndice del objetivo;

expresión es una expresión lineal usada para calcular la forma lineal del objetivo.

Ejemplos

```
minimize objetivo: x + 1.5 * (y + z);
maximize ganancia_total: sum{p in producto} ganancia[p] * elaborar[p];
```

La sentencia objective declara un objetivo. Si no se especifica el dominio del subíndice, el objetivo es un objetivo simple (escalar); de otro modo, es un arreglo n-dimensional de objetivos elementales.

Los objetivos elementales asociados con el objetivo del modelo (si es un objetivo simple) o sus miembros (si es un arreglo) corresponden a restricciones lineales generales en la formulación del problema de PL/PEM (ver Sección 1.1, página 6). Sin embargo, a diferencia de las restricciones, las correspondientes formas lineales son libres (no tienen cota).

La construcción de una restricción lineal real incluida en la instancia del problema, la cual corresponde a una restricción elemental particular, se realiza como sigue. La expresión lineal especificada en la sentencia objective se evalúa para resultar en una forma lineal:

$$f = a_1x_1 + a_2x_2 + \ldots + a_nx_n + a_0,$$

donde x_1, x_2, ..., x_n son variables elementales; a_1, a_2, ..., a_n son coeficientes numéricos y a_0 es el término constante. Luego se usa la forma lineal para construir la restricción elemental final en forma estándar:

$$-\infty < a_1x_1 + a_2x_2 + \ldots + a_nx_n + a_0 < +\infty.$$

Como norma, la descripción del modelo contiene solamente una sentencia objective para definir la función objetivo usada en la instancia del problema. Sin embargo, se permite declarar un número arbitrario de objetivos, en cuyo caso la función objetivo real es el primer objetivo que se encuentra en la descripción del modelo. Otros objetivos también se incluyen en la instancia del problema pero ellos no afectan la función objetivo.

4.6 Sentencia solve

```
solve ;
```

La sentencia solve es opcional y puede usarse solamente una vez. Si no se usa la sentencia solve, se asume una al final de la sección del modelo.

La sentencia solve provoca que el modelo sea resuelto, lo que significa calcular los valores numéricos de todas las variables del modelo. Esto permite usar variables en sentencias luego de la sentencia solve, de la misma forma que si fuesen parámetros numéricos.

Debe notarse que las sentencias variable, constraint y objective no pueden usarse luego de la sentencia solve, *i.e.* todos los componentes principales del modelo deben ser declarados antes de la sentencia solve.

4.7 Sentencia check

```
check dominio : expresión ;
```

dominio es una expresión indizante opcional que especifica el dominio del subíndice de la sentencia check;

expresión es una expresión lógica que especifica una condición lógica para ser comprobada (los dos puntos que preceden a *expresión* pueden omitirse).

Ejemplos

```
check: x + y <= 1 and x >= 0 and y >= 0;
check sum{i in ORIGEN} oferta[i] = sum{j in DESTINO} demanda[j];
check{i in I, j in 1..10}: S[i,j] in U[i] union V[j];
```

El sentencia check permite comprobar el valor resultante de una expresión lógica especificada en la sentencia. Si el valor es *falso* se reporta un error.

Si el dominio del subíndice no se especifica, la comprobación se ejecuta solamente una vez. Especificar el dominio del subíndice permite ejecutar múltiples comprobaciones para cada n-tuplo en el conjunto dominio. En este último caso, la expresión lógica puede incluir índices introducidos en la correspondiente expresión indizante.

4.8 Sentencia display

```
display dominio : ítem , ... , ítem ;
```

dominio es una expresión indizante opcional que especifica el dominio del subíndice de la sentencia display;

ítem, ..., *ítem* son ítems que se mostrarán (los dos puntos que preceden al primer ítem pueden omitirse).

Ejemplos

```
display: 'x =', x, 'y =', y, 'z =', z;
display sqrt(x ** 2 + y ** 2 + z ** 2);
display{i in I, j in J}: i, j, a[i,j], b[i,j];
```

La sentencia display evalúa todos los ítems especificados en la sentencia y escribe sus valores en la salida estándar (terminal) en formato de texto plano.

Si el dominio del subíndice no se especifica, los ítems son evaluados y mostrados solamente una vez. Especificar el dominio del subíndice produce que los ítems sean evaluados y mostrados para cada n-tuplo en el conjunto dominio. En este último caso, los ítems pueden incluir índices introducidos en la correspondiente expresión indizante.

Un ítem a ser mostrado puede ser un objeto del modelo (conjunto, parámetro, variable, restric-

ción u objetivo) o una expresión.

Si el ítem es un objeto calculable (*i.e.* un conjunto o parámetro provisto con el atributo de asignación), el mismo es evaluado a través de todo su dominio y luego se muestra su contenido (*i.e.* el contenido del arreglo de objetos). De otro modo, si el ítem no es un objeto calculable, solamente se muestra su contenido corriente (*i.e.* los miembros realmente generados durante la evaluación del modelo).

Si el ítem es una expresión, la misma se evalúa y se muestra su valor resultante.

4.9 Sentencia printf

`printf` *dominio* `:` *formato* `,` *expresión* `,` ... `,` *expresión* `;`

`printf` *dominio* `:` *formato* `,` *expresión* `,` ... `,` *expresión* `>` *archivo* `;`

`printf` *dominio* `:` *formato* `,` *expresión* `,` ... `,` *expresión* `>>` *archivo* `;`

dominio es una expresión indizante opcional que especifica el dominio del subíndice de la sentencia printf;

formato es una expresión simbólica cuyo valor especifica una cadena de control de formato (los dos puntos que preceden a la expresión de formato pueden omitirse).

expresión, ... , *expresión* son cero o más expresiones cuyos valores deben ser formateados e impresos. Cada expresión debe ser de tipo numérico, simbólico o lógico.

archivo es una expresión simbólica cuyo valor especifica el nombre de un archivo de texto al cual se redirigirá la salida. La señal `>` significa la creación de un archivo nuevo y vacío, mientras que la señal `>>` significa anexar la salida a un archivo existente. Si no se especifica un nombre de archivo, la salida se escribe en la salida estándar (terminal).

Ejemplos

```
printf 'Hola, mundo!\n';
printf: "x = %.3f; y = %.3f; z = %.3f\n", x, y, z > "resultado.txt";
printf{i in I, j in J}: "el flujo desde %s hacia %s es %d\n", i, j, x[i,j]
   >> archivo_resultados & ".txt";
printf{i in I} 'el flujo total de %s es %g\n', i, sum{j in J} x[i,j];
printf{k in K} "x[%s] = " & (if x[k] < 0 then "?" else "%g"),
   k, x[k];
```

La sentencia printf es semejante a la sentencia display; sin embargo, la misma permite formatear los datos que se escribirán.

Si el dominio del subíndice no se especifica, la sentencia printf es ejecutada solamente una vez. Especificar el dominio del subíndice produce que la sentencia printf sea ejecutada para cada n-tuplo en el conjunto dominio. En este último caso, el formato y la expresión pueden incluir índices introducidos en la correspondiente expresión indizante.

La cadena de control de formato es el valor de la expresión simbólica *formato* especificado en la sentencia printf. Se compone con cero o más directivas como sigue: caracteres ordinarios (no `%`), que son copiados sin modificación al flujo de salida, y especificaciones de conversión, cada una de las cuales provoca la evaluación de la expresión correspondiente especificada en la sentencia printf, su formateo y la escritura del valor resultante en el flujo de salida.

Las especificaciones de conversión que se pueden usar en la cadena de control de formato son las siguientes: `d`, `i`, `f`, `F`, `e`, `E`, `g`, `G` y `s`. Estas especificaciones tienen la misma semántica y sintaxis que en el lenguaje de programación C.

4.10 Sentencia for

`for` *dominio* `:` *sentencia* `;`

`for` *dominio* `:` `{` *sentencia* ... *sentencia* `}` `;`

dominio es una expresión indizante opcional que especifica el dominio del subíndice de la sentencia for (los dos puntos a continuación de la expresión indizante pueden omitirse);

sentencia es una sentencia que será ejecutada bajo el control de la sentencia for;

sentencia, ..., *sentencia* es una secuencia de sentencias (encerrada entre llaves) que serán ejecutadas bajo el control de la sentencia for.

Solamente se pueden usar las siguientes sentencias dentro de la sentencia for: check, display, printf y otra sentencia for.

Ejemplos

```
for {(i,j) in E: i != j}
{  printf "el flujo desde %s hacia %s es %g\n", i, j, x[i,j];
   check x[i,j] >= 0;
}
for {i in 1..n}
{  for {j in 1..n} printf " %s", if x[i,j] then "Q" else ".";
   printf("\n");
}
for {1..72} printf("*");
```

La sentencia for provoca que la sentencia o secuencia de sentencias especificadas como parte de la sentencia for sea ejecutada para cada n-tuplo en el conjunto dominio. De modo que las sentencias dentro de la sentencia for pueden incluir índices introducidos en la correspondiente expresión indizante.

4.11 Sentencia table

```
table nombre alias IN controlador arg ... arg :
      conjunto <- [ cmp , ... , cmp ] , par ~ cmp , ... , par ~ cmp ;

table nombre alias dominio OUT controlador arg ... arg :
      expr ~ cmp , ... , expr ~ cmp ;
```

nombre es el nombre simbólico de la tabla;

alias es un literal de cadena opcional que especifica un alias para la tabla;

dominio es una expresión indizante que especifica el dominio del subíndice de la tabla (de salida);

`IN` significa leer datos desde la tabla de entrada;

`OUT` significa escribir datos en la tabla de salida;

controlador es una expresión simbólica que especifica el controlador usado para acceder a la tabla (para más detalles, ver Apéndice C, página 64);

arg es una expresión simbólica opcional que es un argumento pasado al controlador de la tabla. Esta expresión simbólica no debe incluir índices especificados en el dominio;

conjunto es el nombre de un conjunto simple opcional llamado *conjunto de control*. Puede ser omitido junto con el delimitador `<-`;

cmp es un nombre de campo. Entre corchetes se debe especificar al menos un campo. El nombre de campo a continuación de un nombre de parámetro o de una expresión es opcional y puede ser omitido junto con el delimitador `~`, en cuyo caso el nombre del objeto del modelo que corresponda se usará como el nombre de campo;

par es el nombre simbólico de un parámetro del modelo;

expr es una expresión numérica o simbólica.

Ejemplos

```
table datos IN "CSV" "datos.csv": M <- [DESDE,HACIA], d~DISTANCIA,
   c~COSTO;
table resultado{(s,h) in M} OUT "CSV" "resultado.csv": s~DESDE, h~HACIA,
   x[s,h]~FLUJO;
```

La sentencia table permite leer datos desde una tabla y asignarlos a objetos del modelo tales como conjuntos y parámetros (no escalares), al igual que escribir datos del modelo en una tabla.

4.11.1 Estructura de tablas

Una *tabla de datos* es un conjunto (desordenado) de *registros*, en el que cada registro consiste del mismo número de *campos*, y cada campo está provisto de un nombre simbólico único denominado *nombre de campo*. Por ejemplo:

		Primer campo ↓	Segundo campo ↓	. . .	Último campo ↓
Encabezado de tabla	→	DESDE	HACIA	DISTANCIA	COSTO
Primer registro	→	Seattle	New-York	2.5	0.12
Segundo registro	→	Seattle	Chicago	1.7	0.08
		Seattle	Topeka	1.8	0.09
. . .		San-Diego	New-York	2.5	0.15
		San-Diego	Chicago	1.8	0.10
Último registro	→	San-Diego	Topeka	1.4	0.07

4.11.2 Lectura de datos desde una tabla de entrada

La sentencia table de entrada produce la lectura de los datos desde la tabla especificada, registro por registro.

Una vez que se ha leído el próximo registro, los valores numéricos o simbólicos de los campos cuyos nombres se han encerrado entre corchetes en la sentencia table se reúnen en un n-tuplo y, si se ha especificado el conjunto de control en la sentencia table, este n-tuplo es agregado al mismo. Además, un valor numérico o simbólico de cada campo asociado con un parámetro del modelo se asigna al miembro del parámetro identificado por subíndices, los cuales son componentes del n-tuplo que se acaba de leer.

Por ejemplo, la siguiente sentencia table de entrada:

```
table datos IN "...": M <- [DESDE,HACIA], d~DISTANCIA, c~COSTO;
```

produce la lectura de valores de cuatro campos llamados `DESDE`, `HACIA`, `DISTANCIA` y `COSTO` de cada registro de la tabla especificada. Los valores de los campos `DESDE` y `HACIA` proveen un par (s, h) que se agrega al conjunto de control `M`. El valor del campo `DISTANCIA` se asigna al miembro del parámetro `d`$[s, h]$ y el valor del campo `COSTO` se asigna al miembro del parámetro `c`$[s, h]$.

Debe notarse que la tabla de entrada puede contener campos adicionales cuyos nombres no sean especificados en la sentencia table, en cuyo caso los valores de estos campos serán ignorados al leer la tabla.

4.11.3 Escritura de datos en una tabla de salida

La sentencia table de salida produce la escritura de datos en la tabla especificada. Debe notarse que algunos controladores (concretamente CSV y xBASE) destruyen la tabla de salida antes de escribir los datos, *i.e.* borran todos sus registros existentes.

Cada n-tuplo en el conjunto dominio especificado genera un registro escrito en la tabla de salida. Los valores de los campos son valores numéricos o simbólicos de las correspondientes expresiones especificadas en la sentencia table. Estas expresiones se evalúan para cada n-tuplo en el conjunto dominio y, de este modo, puede incluir índices que se introdujeron en la correspondiente expresión indizante.

Por ejemplo, la siguiente sentencia table de salida:

```
table resultado{(s,h) in M} OUT "...": s~DESDE, h~HACIA, x[s,h]~FLUJO;
```

produce la escritura de registros en la tabla de salida, a razón de un registro por cada par (s, h) en el conjunto `M`, en el que cada registro consiste de tres campos llamados `DESDE`, `HACIA` y `FLUJO`. Los valores escritos en los campos `DESDE` y `HACIA` son los valores corrientes de los índices `s` y `h`, y el valor escrito en el campo `FLUJO` es un valor del miembro `x`$[s, h]$ del correspondiente parámetro o variable indexada.

Capítulo 5

Datos del modelo

Los *datos del modelo* incluyen conjuntos elementales, los cuales son "valores" de los conjuntos del modelo, y valores simbólicos y numéricos de los parámetros del modelo.

En MathProg hay dos maneras diferentes de proveer valores para los conjuntos y parámetros del modelo. Una manera es simplemente proveyendo los datos necesarios mediante el atributo de asignación. Sin embargo, en muchos casos es más práctico separar el modelo en sí mismo de los datos particulares necesarios para el modelo. Por esta última razón, en MathProg hay otra manera que consiste en separar la descripción del modelo en dos partes: la sección del modelo y la sección de los datos.

La *sección del modelo* es la parte principal de la descripción del modelo que contiene las declaraciones de todos los objetos del modelo y es común a todos los problemas basados en tal modelo.

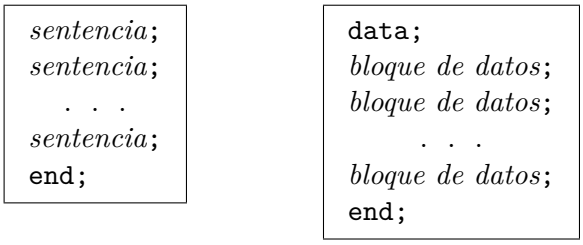
La *sección de los datos* es una parte opcional de la descripción del modelo que contiene datos específicos para un problema particular.

En MathProg las secciones del modelo y de los datos se pueden ubicar tanto en un único archivo de texto, como en dos archivos de texto separados.

1. Si ambas secciones se ubican en un archivo, este debe componerse como sigue:

sentencia;
sentencia;
. . .
sentencia;
`data;`
bloque de datos;
bloque de datos;
. . .
bloque de datos;
`end;`

2. Si ambas secciones se ubican en dos archivos separados, los archivos se componen como sigue:



Archivo del modelo Archivo de datos

Nota: si la sección de datos se ubica en un archivo separado, la palabra clave `data` es opcional y puede omitirse, al igual que el punto y coma que le sigue.

5.1 Codificación de la sección de los datos

La *sección de los datos* es una secuencia de bloques de datos en varios formatos que se discuten en las siguientes secciones. El orden que siguen los bloques de datos en la sección de los datos puede ser arbitrario, no necesariamente el mismo que se siguió en la sección del modelo para sus correspondientes objetos.

Las reglas para codificar la sección de los datos comúnmente son las mismas reglas que para codificar la descripción del modelo (ver Sección 2, página 9), *i.e.* los bloques de datos se componen con unidades léxicas básicas como nombres simbólicos, literales numéricos y de cadena, palabras clave, delimitadores y comentarios. Sin embargo, por conveniencia y para mejorar la legibilidad hay una desviación de la regla común: si un literal de cadena consiste únicamente de caracteres alfanuméricos (incluyendo el carácter de subrayado), los signos `+` y `-` y/o el punto decimal, el mismo puede codificarse sin comillas delimitadoras (simples o dobles).

Todo el material numérico y simbólico provisto en la sección de los datos se codifica en la forma de números y símbolos, *i.e.* a diferencia de la sección del modelo, en la sección de los datos no se permiten expresiones. Sin embargo, los signos `+` y `-` pueden preceder a literales numéricos para permitir la codificación de cantidades numéricas con signo, en cuyo caso no debe haber caracteres de espacio en blanco entre el signo y el literal numérico que le sigue (si hay al menos un espacio en blanco, el signo y el literal numérico que le sigue serán reconocidos como dos unidades léxicas diferentes).

5.2 Bloque de datos de conjunto

`set` *nombre* `,` *registro* `,` ... `,` *registro* `;`

`set` *nombre* `[` *símbolo* `,` ... `,` *símbolo* `]` `,` *registro* `,` ... `,` *registro* `;`

nombre es el nombre simbólico del conjunto;

símbolo, ..., *símbolo* son subíndices que especifican un miembro particular del conjunto (si el conjunto es un arreglo, *i.e.* un conjunto de conjuntos);

registro, ..., *registro* son registros.

Las comas que preceden a los registros pueden omitirse.

Registros

`:=`
: es un elemento de asignación de registro no-significativo que puede ser usado libremente para mejorar la legibilidad;

`(` *porción* `)`
: especifica una porción;

datos-simples
: especifica los datos del conjunto en formato simple;

`:` *datos-matriciales*
: especifica los datos del conjunto en formato matricial;

`(tr) :` *datos-matriciales*
: especifica los datos del conjunto en formato matricial traspuesto (en este caso, los dos puntos que siguen a la palabra clave `(tr)` pueden omitirse).

Ejemplos

```
set mes := Ene Feb Mar Abr May Jun;
set mes "Ene", "Feb", "Mar", "Abr", "May", "Jun";
set A[3,Mar] := (1,2) (2,3) (4,2) (3,1) (2,2) (4,4) (3,4);
set A[3,'Mar'] := 1 2 2 3 4 2 3 1 2 2 4 4 3 4;
set A[3,'Mar'] : 1 2 3 4 :=
               1 - + - -
               2 - + + -
               3 + - - +
               4 - + - + ;
set B := (1,2,3) (1,3,2) (2,3,1) (2,1,3) (1,2,2) (1,1,1) (2,1,1);
set B := (*,*,*) 1 2 3, 1 3 2, 2 3 1, 2 1 3, 1 2 2, 1 1 1, 2 1 1;
set B := (1,*,2) 3 2 (2,*,1) 3 1 (1,2,3) (2,1,3) (1,1,1);
set B := (1,*,*) : 1 2 3 :=
                 1 + - -
```

```
          2 - + +
          3 - + -
(2,*,*) : 1 2 3 :=
          1 + - +
          2 - - -
          3 + - - ;
```

(En estos ejemplos, `mes` es un conjunto simple de singletones, `A` es un arreglo 2-dimensional de duplos y `B` es un conjunto simple de triplos. Los bloques de datos para el mismo conjunto son equivalentes en el sentido de que especifican los mismos datos en formatos distintos.

El *bloque de datos del conjunto* se usa para especificar un conjunto elemental completo, el que se asigna a un conjunto (si es un conjunto simple) o a uno de sus miembros (si el conjunto es un arreglo de conjuntos).[1]

Los bloques de datos sólo pueden ser especificados para conjuntos no-calculables, *i.e.* para conjuntos que no tienen el atributo de asignación (`:=`) en la correspondiente sentencia set.

Si el conjunto es un conjunto simple, sólo se debe especificar su nombre simbólico en el encabezado del bloque de datos. De otro modo, si el conjunto es un arreglo n-dimensional, su nombre simbólico debe proveerse con una lista completa de subíndices separados por coma y encerrados entre corchetes para especificar un miembro particular del arreglo de conjuntos. El número de subíndices debe ser igual a la dimensión del arreglo de conjuntos, en el que cada subíndice deben ser un número o símbolo.

Un conjunto elemental definido en el bloque de datos del conjunto se codifica como una secuencia de registros según se describe luego.[2]

5.2.1 Asignación de registro

La *asignación de registro* (`:=`) es un elemento no-significativo. Se puede usar para mejorar la legibilidad de los bloques de datos.

5.2.2 Registro en porción

El *registro en porción* es un registro de control que especifica una *porción* del conjunto elemental definido en el bloque de datos. Tiene la siguiente forma sintáctica:

$$\texttt{(}\ p_1\ \texttt{,}\ p_2\ \texttt{,}\ \ldots\ \texttt{,}\ p_n\ \texttt{)}$$

donde p_1, p_2, ..., p_n son componentes de la porción.

Cada componente de la porción puede ser un número o un símbolo o el asterisco (`*`). El número de componentes de la porción debe coincidir con la dimensión de los n-tuplos del conjunto elemental que se define. Por ejemplo, si el conjunto elemental contiene 4-tuplos (cuádruplos), la porción debe

[1] Hay otra forma de especificar datos para un conjunto simple junto con los datos para los parámetros. Esta característica se discute en la próxima sección.

[2] Registro es simplemente un término técnico. No significa que los mismos presenten algún formateo especial.

tener cuatro componentes. El número de asteriscos en la porción se denomina la *dimensión de la porción*.

El efecto de usar las porciones es como sigue: si se especifica una porción m-dimensional (*i.e.* una porción que tiene m asteriscos) en el bloque de datos, todos los registros subsecuentes deben especificar tuplos de dimensión m. Cuando se encuentra un m-tuplo, cada asterisco en la porción se reemplaza por los correspondientes componentes del m-tuplo para dar el n-tuplo resultante, el que es incluido en el conjunto elemental que se define. Por ejemplo, si está vigente la porción $(a, *, 1, 2, *)$ y se encuentra el duplo $(3, b)$ en el registro subsecuente, el 5-tuplo resultante que se incluye en el conjunto elemental es $(a, 3, 1, 2, b)$.

Las porciones que no tienen asteriscos en si mismas, definen un n-tuplo completo que se incluye en el conjunto elemental.

Una vez especificada una porción, la misma está vigente hasta que aparezca una nueva porción o bien hasta que se encuentra el final del bloque de datos. Debe notarse que si no se especifica una porción en el bloque de datos, se asume una cuyos componentes son asteriscos en todas las posiciones.

5.2.3 Registro simple

El *registro simple* define un n-tuplo en formato simple y tiene la siguiente forma sintáctica:

$$t_1 \ , \ t_2 \ , \ \ldots \ , \ t_n$$

donde $t_1, t_2, \ldots, t_n$ son componentes del n-tuplo. Cada componente puede ser un número o símbolo. Las comas entre componentes son opcionales y pueden omitirse.

5.2.4 Registro matricial

El *registro matricial* define varios 2-tuplos (duplos) en formato matricial y tiene la siguiente forma sintáctica:

$$\begin{array}{cccccc} \texttt{:} & c_1 & c_2 & \ldots & c_n & \texttt{:=} \\ f_1 & a_{11} & a_{12} & \ldots & a_{1n} & \\ f_2 & a_{21} & a_{22} & \ldots & a_{2n} & \\ & \cdot & \cdot & \cdot & \cdot & \\ f_m & a_{m1} & a_{m2} & \ldots & a_{mn} & \end{array}$$

donde f_1, f_2, ..., f_m son números y/o símbolos que corresponden a filas de la matriz; c_1, c_2, ..., c_n son números y/o símbolos que corresponden a columnas de la matriz, mientras que a_{11}, a_{12}, ..., a_{mn} son los elementos de la matriz, los cuales pueden ser tanto `+` como `-`. (En este registro, el delimitador `:` que precede a la lista de columnas y el delimitador `:=` a continuación de la lista de columnas, no pueden ser omitidos.)

Cada elemento a_{ij} del bloque de datos matricial (donde $1 \leq i \leq m$, $1 \leq j \leq n$) corresponde a 2-tuplos (f_i, c_j). Si en a_{ij} se indica el signo más (`+`), el correspondiente 2-tuplo (o un n-tuplo más largo si se usa una porción) se incluye en el conjunto elemental. De otro modo, si en a_{ij} se indica el signo menos (`-`) el 2-tuplo no se incluye en el conjunto elemental.

Puesto que el registro matricial define 2-tuplos, ya sea el conjunto elemental o bien la porción vigente en uso deben ser 2-dimensionales.

5.2.5 Registro matricial traspuesto

El *registro matricial traspuesto* tiene la siguiente forma sintáctica:

$$\begin{array}{cccccc} \texttt{(tr)} : & c_1 & c_2 & \ldots & c_n & \texttt{:=} \\ f_1 & a_{11} & a_{12} & \ldots & a_{1n} & \\ f_2 & a_{21} & a_{22} & \ldots & a_{2n} & \\ & \cdot & \cdot & \cdot & \cdot & \\ f_m & a_{m1} & a_{m2} & \ldots & a_{mn} & \end{array}$$

(En este caso el delimitador `:` a continuación de la palabra clave `(tr)` es opcional y puede omitirse.)

Este registro es completamente análogo al registro matricial (ver anteriormente) con la única excepción, en este caso, de que cada elemento a_{ij} de la matriz corresponde al 2-tuplo (c_j, f_i) en vez de a (f_i, c_j).

Una vez especificado, el indicador `(tr)` tiene alcance en todos los registros subsecuentes hasta que se encuentra otra porción o bien el fin del bloque de datos.

5.3 Bloque de datos de parámetro

`param` *nombre* `,` *registro* `,` ... `,` *registro* `;`

`param` *nombre* `default` *valor* `,` *registro* `,` ... `,` *registro* `;`

`param :` *datos-tabulación* `;`

`param default` *valor* `:` *datos-tabulación* `;`

nombre es el nombre simbólico del parámetro;

valor es un valor por defecto opcional del parámetro;

registro, ..., *registro* son registros.

datos-tabulación especifica los datos del parámetro en el formato tabulación.

Las comas que preceden a los registros pueden omitirse.

Registros

`:=`
: es un elemento de asignación de registro no-significativo que puede ser usado libremente para mejorar la legibilidad;

`[` *porción* `]`
: especifica una porción;

datos-planos
especifica los datos del parámetro en formato plano;

: *datos-tabulares*
especifica los datos del parámetro en formato tabular;

(tr) : *datos-tabulares*
especifica los datos del parámetro en formato matricial traspuesto (en este caso, los dos puntos que siguen a la palabra clave (tr) pueden omitirse).

Ejemplos

```
param T := 4;
param mes := 1 Ene 2 Feb 3 Mar 4 Abr 5 May;
param mes := [1] 'Ene', [2] 'Feb', [3] 'Mar', [4] 'Abr', [5] 'May';
param stock_inicial := hierro 7.32 niquel 35.8;
param stock_inicial [*] hierro 7.32, niquel 35.8;
param costo [hierro] .025 [niquel] .03;
param valor := hierro -.1, niquel .02;
param       : stock_inicial  costo  valor :=
      hierro      7.32       .025    -.1
      niquel     35.8        .03      .02 ;
param : insumo : stock_inicial  costo  valor :=
        hierro       7.32       .025    -.1
        niquel      35.8        .03      .02 ;
param demanda default 0 (tr)
       :    FRA  DET  LAN  WIN  STL  FRE  LAF :=
   laminas  300   .   100   75   .   225  250
   rollos   500  750  400  250   .   850  500
   cintas   100   .    .    50  200   .   250 ;
param costo_transporte :
   [*,*,laminas]:  FRA  DET  LAN  WIN  STL  FRE  LAF :=
         GARY      30   10    8   10   11   71    6
         CLEV      22    7   10    7   21   82   13
         PITT      19   11   12   10   25   83   15
   [*,*,rollos]:   FRA  DET  LAN  WIN  STL  FRE  LAF :=
         GARY      39   14   11   14   16   82    8
         CLEV      27    9   12    9   26   95   17
         PITT      24   14   17   13   28   99   20
   [*,*,cintas]:   FRA  DET  LAN  WIN  STL  FRE  LAF :=
         GARY      41   15   12   16   17   86    8
         CLEV      29    9   13    9   28   99   18
         PITT      26   14   17   13   31  104   20  ;
```

El *bloque de datos del parámetro* se usa para especificar datos completos a un parámetro (o a varios parámetros si los datos se especifican en el formato tabulaciones)

Los bloques de datos sólo pueden ser especificados para parámetros no-calculables, *i.e.* para parámetros que no tienen el atributo de asignación(:=) en la correspondiente sentencia parameter.

Los datos definidos en el bloque de datos del parámetro se codifican como una secuencia de registros descriptos luego. Adicionalmente, el bloque de datos puede ser provisto con el atributo opcional `default`, el cual especifica un valor numérico o simbólico por defecto para el parámetro (parámetros). Este valor por defecto se asigna al parámetro, o a sus miembros, cuando no se definen valores apropiados en el bloque de datos del parámetro. El atributo `default` no se puede usar si ya se ha especificado en la correspondiente sentencia parameter.

5.3.1 Asignación de registro

La *asignación de registro* (`:=`) es un elemento no-significativo. Se puede usar para mejorar la legibilidad de los bloques de datos.

5.3.2 Registro en porción

El *registro en porción* es un registro de control que especifica una *porción* del arreglo de parámetros. Tiene la siguiente forma sintáctica:

$$\texttt{[}\ p_1\ \texttt{,}\ p_2\ \texttt{,}\ \ldots\ \texttt{,}\ p_n\ \texttt{]}$$

donde p_1, p_2, ..., p_n son componentes de la porción.

Cada componente de la porción puede ser un número o un símbolo o el asterisco (`*`). El número de componentes de la porción debe coincidir con la dimensión del parámetro. Por ejemplo, si el parámetro es un arreglo 4-dimensional, la porción debe tener cuatro componentes. El número de asteriscos en la porción se denomina la *dimensión de la porción*.

El efecto de usar las porciones es como sigue: si se especifica una porción m-dimensional (*i.e.* una porción que tiene m asteriscos) en el bloque de datos, todos los registros subsecuentes deben especificar los subíndices de los miembros del parámetro como si el parámetro fuese m-dimensional y no n-dimensional.

Cuando se encuentran los m subíndices, cada asterisco en la porción se reemplaza por el correspondiente subíndice para dar los n subíndices, los cuales definen al miembro corriente del parámetro. Por ejemplo, si está vigente la porción $(a, *, 1, 2, *)$ y se encuentran los subíndices 3 y b en el registro subsecuente, la lista completa de subíndices que se usa para elegir un miembro del parámetro es $(a, 3, 1, 2, b)$.

Se permite especificar una porción que no tenga asteriscos. Tal porción, en sí misma define una lista completa de subíndices, en cuyo caso el próximo registro debe definir solamente un valor individual del correspondiente miembro del parámetro.

Una vez especificada una porción, la misma está vigente hasta que aparezca una nueva porción o bien hasta que se encuentra el final del bloque de datos. Debe notarse que si no se especifica una porción en el bloque de datos, se asume una cuyos componentes son todos asteriscos.

5.3.3 Registro plano

El *registro plano* define la lista de subíndices y un valor individual en el formato plano. Este registro tiene la siguiente forma sintáctica:

$$t_1 \ , \ t_2 \ , \ \ldots \ , \ t_n \ , \ v$$

donde t_1, t_2, ..., t_n son subíndices y v es un valor. Cada subíndice, al igual que el valor, puede ser un número o un símbolo. Las comas que siguen a los subíndices son opcionales y pueden omitirse.

En el caso de parámetros o porciones 0-dimensionales, el registro plano no tiene subíndice y consiste solamente de un valor individual.

5.3.4 Registro tabular

El *registro tabular* define varios valores, cada uno de los cuales viene provisto con dos subíndices. Este registro tiene la siguiente forma sintáctica:

$$\begin{array}{cccccc} \texttt{:} & c_1 & c_2 & \ldots & c_n & \texttt{:=} \\ f_1 & a_{11} & a_{12} & \ldots & a_{1n} & \\ f_2 & a_{21} & a_{22} & \ldots & a_{2n} & \\ & . & . & . & . & \\ f_m & a_{m1} & a_{m2} & \ldots & a_{mn} & \end{array}$$

donde los f_1, f_2, ..., f_m son números y/o símbolos que corresponden a filas de la tabla; c_1, c_2, ..., c_n son números y/o símbolos que corresponden a columnas de la tabla; mientras que a_{11}, a_{12}, ..., a_{mn} son elementos de la tabla. Cada elemento puede ser un número o símbolo o el punto decimal (`.`) solo (en este registro, el delimitador `:` que precede a la lista de columnas y el delimitador `:=` a continuación de la lista de columnas, no pueden ser omitidos.).

Cada elemento a_{ij} del bloque de datos tabulares ($1 \leq i \leq m$, $1 \leq j \leq n$) define dos subíndices, siendo el primero f_i y el segundo c_j. Estos subíndices se usan junto con la porción vigente para formar la lista completa de subíndices que identifica a un miembro particular del arreglo de parámetros. Si a_{ij} es un número o símbolo, tal valor se asigna al miembro del parámetro. Sin embargo, si a_{ij} es un punto decimal solo, el miembro recibe el valor por defecto especificado ya sea en el bloque de datos del parámetro o en la sentencia parameter, o si no hay un valor por defecto especificado, el miembro permanece indefinido.

Puesto que el registro tabular provee dos subíndices para cada valor, ya sea el parámetro o bien la porción vigente en uso deben ser 2-dimensionales.

5.3.5 Registro tabular traspuesto

El *registro tabular traspuesto* tiene la siguiente forma sintáctica:

$$\begin{array}{cccccc} \texttt{(tr)} \ \texttt{:} & c_1 & c_2 & \ldots & c_n & \texttt{:=} \\ f_1 & a_{11} & a_{12} & \ldots & a_{1n} & \\ f_2 & a_{21} & a_{22} & \ldots & a_{2n} & \\ & . & . & . & . & \\ f_m & a_{m1} & a_{m2} & \ldots & a_{mn} & \end{array}$$

(En este caso el delimitador `:` a continuación de la palabra clave `(tr)` es opcional y puede omitirse.)

Este registro es completamente análogo al registro tabular (ver anteriormente), con la única excepción de que el primer subíndice definido por el elemento a_{ij} es c_j mientras que el segundo es f_i.

Una vez especificado, el indicador `(tr)` tiene alcance en todos los registros subsecuentes hasta que se encuentre otra porción o bien el fin del bloque de datos.

5.3.6 Formato de datos tabulación

El bloque de datos del parámetro en el *formato tabulación* tiene la siguiente forma sintáctica:

$$
\begin{array}{llllllll}
\texttt{param default } \mathit{valor} : c : & p_1 , & p_2 , & \ldots , & p_f \texttt{ :=} \\
f_{11} , & f_{12} , & \ldots , & f_{1n} , & a_{11} , & a_{12} , & \ldots , & a_{1f} , \\
f_{21} , & f_{22} , & \ldots , & f_{2n} , & a_{21} , & a_{22} , & \ldots , & a_{2f} , \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
f_{m1} , & f_{m2} , & \ldots , & f_{mn} , & a_{m1} , & a_{m2} , & \ldots , & a_{mf} \texttt{ ;}
\end{array}
$$

1. La palabra clave `default` puede omitirse junto con el valor que le sigue.
2. El nombre simbólico c puede omitirse junto con los dos puntos que le siguen.
3. Todas las comas son opcionales y pueden omitirse.

El bloque de datos en el formato tabulación mostrado arriba es exactamente equivalente a los siguientes bloques de datos:

`set` c `:=` $(f_{11},f_{12},\ldots,f_{1n})$ $(f_{21},f_{22},\ldots,f_{2n})$ $\ldots$ $(f_{m1},f_{m2},\ldots,f_{mn})$`;`

`param` p_1 `default` *valor* `:=`

$[f_{11},f_{12},\ldots,f_{1n}]$ a_{11} $[f_{21},f_{22},\ldots,f_{2n}]$ a_{21} $\ldots$ $[f_{m1},f_{m2},\ldots,f_{mn}]$ a_{m1}`;`

`param` p_2 `default` *valor* `:=`

$[f_{11},f_{12},\ldots,f_{1n}]$ a_{12} $[f_{21},f_{22},\ldots,f_{2n}]$ a_{22} $\ldots$ $[f_{m1},f_{m2},\ldots,f_{mn}]$ a_{m2}`;`

. . . . . .

`param` p_f `default` *valor* `:=`

$[f_{11},f_{12},\ldots,f_{1n}]$ a_{1f} $[f_{21},f_{22},\ldots,f_{2n}]$ a_{2f} $\ldots$ $[f_{m1},f_{m2},\ldots,f_{mn}]$ a_{mf}`;`

Apéndice A

Uso de sufijos

Se pueden usar sufijos para recuperar valores adicionales relacionados con las variables, restricciones y objetivos del modelo.

Un *sufijo* consiste de un punto (`.`) seguido de una palabra clave no-reservada. Por ejemplo, si `x` es una variable bidimensional, `x[i,j].lb` es un valor numérico igual a la cota inferior de la variable elemental `x[i,j]` que se puede usar como un parámetro numérico dondequiera que sea en expresiones.

Para las variables del modelo, los sufijos tienen los siguientes significados:

`.lb`	cota inferior
`.ub`	cota superior
`.status`	estatus en la solución: 0 — indefinida 1 — básica 2 — no-básica en la cota inferior 3 — no-básica en la cota superior 4 — variable no-básica libre (no acotada) 5 — variable no-básica fija
`.val`	valor primal en la solución
`.dual`	valor dual (costo reducido) en la solución

Para las restricciones y objetivos del modelo, los sufijos tienen los siguientes significados:

`.lb`	cota inferior de la forma lineal
`.ub`	cota superior de la forma lineal
`.status`	estatus en la solución: 0 — indefinida 1 — no-limitante 2 — limitante en la cota inferior 3 — limitante en la cota superior 4 — fila limitante libre (no-acotada) 5 — restricción de igualdad limitante
`.val`	valor primal de la forma lineal en la solución
`.dual`	valor dual (costo reducido) de la forma lineal en la solución

Debe notarse que los sufijos `.status`, `.val` y `.dual` solamente pueden usarse luego de la sentencia solve.

Apéndice B

Funciones de fecha y hora

por Andrew Makhorin <mao@gnu.org>
y Heinrich Schuchardt <heinrich.schuchardt@gmx.de>

B.1 Obtención del tiempo calendario corriente

Para obtener el tiempo calendario[1] corriente en MathProg existe la función `gmtime`. No tiene argumentos y devuelve el número de segundos transcurridos desde las 00:00:00 del 1 de enero de 1970, Tiempo Universal Coordinado (UTC). Por ejemplo:

```
param utc := gmtime();
```

MathPro no tiene una función para convertir el tiempo UTC devuelto por la función `gmtime` a tiempo calendario *local*. Entonces, si se necesita determinar el tiempo calendario local corriente, se debe agregar al tiempo UTC devuelto la diferencia horaria con respecto al UTC expresada en segundos. Por ejemplo, la hora en Berlín durante el invierno está una hora adelante del UTC, lo que corresponde una diferencia horaria de +1 hora = +3600 segundos, de modo que el tiempo calendario corriente del invierno en Berlín se puede determinar como sigue:

```
param ahora := gmtime() + 3600;
```

Análogamente, el horario de verano en Chicago (Zona Horaria Central) está cinco horas por detrás del UTC, de modo que el correspondiente tiempo calendario local corriente se puede determinar como sigue:

```
param ahora := gmtime() - 5 * 3600;
```

Debe notarse que el valor devuelto por `gmtime` es volátil, *i.e.* si se la invoca varias veces, esta función devolverá valores diferentes.

[1] N. del T.: el tiempo calendario es un punto en el *continuum* del tiempo, por ejemplo 4 de noviembre de 1990 a las 18:02.5 UTC. A veces se lo llama "tiempo absoluto". La definición está tomada de *Sandra Loosemore*, *Richard M. Stallman*, *Roland McGrath*, *Andrew Oram* & *Ulrich Drepper*, "The GNU C Library Reference Manual - for version 2.17", Free Software Foundation, Inc, 2012.

B.2 Conversión de una cadena de caracteres a un tiempo calendario

La función `str2time(`*c*`,` *f*`)` convierte una cadena de caracteres (una *estampa de la fecha y hora*) especificada por su primer argumento *c*, la que debe ser una expresión simbólica, a un tiempo calendario apropiado para cálculos aritméticos. La conversión se controla mediante la especificación de una cadena de formato *f* (el segundo argumento), la que también debe ser una expresión simbólica.

El resultado de la conversión devuelto por `str2time` tiene el mismo significado que los valores devueltos por la función `gmtime` (ver Subsección B.1, página 59). Debe notarse que `str2time` *no corrige* el tiempo calendario devuelto para considerar la zona horaria local, *i.e.* si se aplica a las 00:00:00 del 1 de enero de 1970, siempre devolverá 0.

Por ejemplo, las sentencias del modelo

```
param c, symbolic, := "07/14/98 13:47";
param t := str2time(c, "%m/%d/%y %H:%M");
display t;
```

imprime lo siguiente en la salida estándar:

```
t = 900424020
```

donde el tiempo calendario mostrado corresponde a las 13:47:00 del 14 de julio de 1998.

La cadena de formato que se pasa a la función `str2time` consiste de especificadores de conversión y caracteres ordinarios. Cada especificador de conversión empieza con un carácter de porcentaje (`%`) seguido por una letra.

Los siguientes especificadores de conversión se pueden usar en la cadena de formato[2]:

`%b` El nombre del mes abreviado (insensible a mayúsculas). Al menos las tres primeras letras del nombre del mes deben aparecer en la cadena de entrada.

`%d` El día del mes como un número decimal (rango de 1 a 31). Se permite el cero como primer dígito, aunque no es requerido.

`%h` Lo mismo que `%b`.

`%H` La hora como un número decimal, empleando un reloj de 24 horas (rango de 0 a 23). Se permite el cero como primer dígito, aunque no es requerido.

`%m` El mes como un número decimal (rango de 1 a 12). Se permite el cero como primer dígito, aunque no es requerido.

`%M` El minuto como número decimal (rango de 0 a 59). Se permite el cero como primer dígito, aunque no es requerido.

`%S` El segundo como un número decimal (rango de 0 a 59). Se permite el cero como primer dígito, aunque no es requerido.

[2]N. del T.: en todas las funciones de fecha y hora, nombre del mes y del día de la semana refiere a su denominación en inglés, *e.g.*: `August`, `Aug`, `Wednesday`, `We`.

`%y` El año sin un siglo como un número decimal (rango de 0 a 99). Se permite el cero como primer dígito, aunque no es requerido. Los valores de entrada en el rango de 0 a 68 se consideran como los años 2000 al 2068, mientras que los valores del 69 al 99 como los años 1969 a 1999.

`%z` La diferencia horaria con respecto a GMT en formato ISO 8601.

`%%` Un carácter `%` literal.

Todos los demás caracteres (ordinarios) en la cadena de formato deben tener un carácter de coincidencia con la cadena de entrada a ser convertida. Las excepciones son los espacios en la cadena de entrada, la cual puede coincidir con cero o más caracteres de espacio en la cadena de formato.

Si algún componente de la fecha y/u hora están ausentes en el formato y, consecuentemente, en la cadena de entrada, la función `str2time` usa sus valores por defecto de las 00:00:00 del 1 de enero de 1970, es decir que el valor por defecto para el año es 1970, el valor por defecto para el mes es enero, etc.

La función es aplicable a todos los tiempos calendarios en el rango desde las 00:00:00 del 1 de enero del 0001 hasta las 23:59:59 del 31 de diciembre del 4000 del calendario gregoriano.

B.3 Conversión de un tiempo calendario a una cadena de caracteres

La función `time2str(`*t*`,` *f*`)` convierte el tiempo calendario especificado por su primer argumento *t*, el que debe ser una expresión numérica, a una cadena de caracteres (valor simbólico). La conversión se controla con la cadena de formato *f* especificada (el segundo argumento), la que debe ser una expresión simbólica.

El tiempo calendario que se le pasa a `time2str` tiene el mismo significado que el valor devuelto por la función `gmtime` (ver Subsección B.1, página 59). Debe notarse que `time2str` *no corrige* el tiempo calendario especificado para considerar la zona horaria local, *i.e.* el tiempo calendario 0 siempre corresponde a las 00:00:00 del 1 de enero de 1970.

Por ejemplo, las sentencias del modelo

```
param c, symbolic, := time2str(gmtime(), "%FT%TZ");
display c;
```

puede producir la siguiente impresión:

```
c = '2008-12-04T00:23:45Z'
```

que es una estampa de una fecha y hora en el formato ISO.

La cadena de formato que se pasa a la función `time2str` consiste de especificadores de conversión y caracteres ordinarios. Cada especificador de conversión empieza con un carácter de porcentaje (`%`) seguido por una letra.

Los siguientes especificadores de conversión se pueden usar en la cadena de formato:

`%a` El nombre del día de la semana abreviado (2 caracteres).

`%A` El nombre del día de la semana completo.

`%b` El nombre del mes abreviado (3 caracteres).

`%B` El nombre del mes completo.

`%C` El siglo del año, es decir el mayor entero no mayor que el año dividido por 100.

`%d` El día del mes como un número decimal (rango de 01 a 31).

`%D` La fecha, usando el formato `%m/%d/%y`.

`%e` El día del mes, como con `%d`, pero rellenado con un espacio en blanco en vez de cero.

`%F` La fecha, usando el formato `%Y-%m-%d`.

`%g` El año correspondiente al número de semana ISO, pero sin el siglo (rango de 00 a 99). Tiene el mismo formato y valor que `%y`, excepto que si el número de semana ISO (ver `%V`) pertenece al año previo o siguiente, se usa aquel año en su lugar.

`%G` El año correspondiente al número de semana ISO. Tiene el mismo formato y valor que `%Y`, excepto que si el número de semana ISO (ver `%V`) pertenece al año previo o siguiente, se usa aquel año en su lugar.

`%h` Lo mismo que `%b`.

`%H` La hora como un número decimal, empleando un reloj de 24 horas (rango de 00 a 23).

`%I` La hora como un número decimal, empleando un reloj de 12 horas (rango de 01 a 12).

`%j` El día del año como un número decimal (rango de 001 a 366).

`%k` La hora como un número decimal, empleando un reloj de 24 horas como con `%H`, pero rellenado con un espacio en blanco en vez de cero.

`%l` La hora como un número decimal, empleando un reloj de 12 horas como con `%I`, pero rellenado con un espacio en blanco en vez de cero.

`%m` El mes como un número decimal (rango de 01 a 12).

`%M` El minuto como un número decimal (rango de 00 a 59).

`%p` Tanto `AM` como `PM`, de acuerdo con el valor horario dado. La medianoche es tratada como `AM` y el mediodía como `PM`.

`%P` Tanto `am` como `pm`, de acuerdo con el valor horario dado. La medianoche es tratada como `am` y el mediodía como `pm`.

`%R` La hora y los minutos en números decimales, usando el formato `%H:%M`.

`%S` Los segundos como un número decimal (rango de 00 a 59).

`%T` La hora del día en números decimales, usando el formato `%H:%M:%S`.

`%u` El día de la semana como un número decimal (rango de 1 a 7), siendo 1 el lunes.

`%U` El número de semana del año corriente como un número decimal (rango de 00 a 53), empezando con el primer domingo como el primer día de la primer semana. Se considera que los días del año anteriores al primer domingo son parte de la semana 00.

`%V` El número de semana ISO como un número decimal (rango 01 a 53). Las semanas de ISO empiezan los lunes y terminan los domingos. La semana 01 de un año es la primer semana que tiene la mayoría de sus días en ese año, lo cual es equivalente a la semana que contiene al 4 de enero. La semana 01 de un año puede contener días del año previo. La semana anterior a la semana 01 de un año es la última semana (52 o 53) del año previo, aún si esta contiene días del nuevo año. En otras palabras, si el 1 de enero cae en lunes, martes, miércoles o jueves, está en la semana 01; si el 1 de enero cae en viernes, sábado o domingo, está en la semana 52 o 53 del año previo.

`%w` El día de la semana como un número decimal (rango de 0 a 6), siendo 0 el domingo.

`%W` El número de semana del año corriente como un número decimal (rango de 00 a 53), empezando con el primer lunes como el primer día de la primer semana. Se considera que los días del año anteriores al primer lunes son parte de la semana 00.

`%y` El año sin el siglo como un número decimal (rango de 00 a 99), es decir año `mod` 100.

`%Y` El año como un número decimal, usando el calendario gregoriano.

`%%` Un carácter `%` literal.

Todos los demás caracteres (ordinarios) en la cadena de formato simplemente se copian a la cadena resultante.

El primer argumento (tiempo calendario) que se le pasa a la función `time2str` debe están en el rango entre -62135596800 y $+64092211199$, lo que corresponde al período desde las 00:00:00 del 1 de enero del 0001 hasta las 23:59:59 del 31 de diciembre del 4000 del calendario gregoriano.

Apéndice C

Controladores de tablas

por Andrew Makhorin <mao@gnu.org>
y Heinrich Schuchardt <heinrich.schuchardt@gmx.de>

El *controlador de tablas* es un módulo del programa que permite la trasmisión de datos entre objetos de un modelo MathProg y tablas de datos.

Actualmente, el paquete GLPK tiene cuatro controladores de tablas:

— controlador interno de tablas CSV;

— controlador interno de tablas xBASE;

— controlador de tablas ODBC;

— controlador de tablas MySQL.

C.1 Controlador de tablas CSV

El controlador de tablas CSV asume que la tabla de datos está representada en la forma de un archivo de texto plano, en el formato de archivo CSV (valores separados por coma) como se describe más adelante.

Para elegir el controlador de tablas CSV, su nombre en la sentencia table debe especificarse como `"CSV"` y el único argumento debe especificar el nombre de un archivo de texto plano conteniendo la tabla. Por ejemplo:

```
table datos IN "CSV" "datos.csv": ... ;
```

El sufijo del nombre de archivo puede ser arbitrario; sin embargo, se recomienda usar el sufijo '`.csv`'.

Al leer tablas de entrada, el controlador de tablas CSV provee un campo implícito llamado `RECNO`, el cual contiene el número del registro corriente. Este campo puede especificarse en la sentencia table de entrada, como si existiera un verdadero campo llamado `RECNO` en el archivo CSV. Por ejemplo:

```
table lista IN "CSV" "lista.csv": num <- [RECNO], ... ;
```

Formato CSV[1]

El formato CSV (valores separados por coma) es un formato de archivo de texto plano definido como sigue:

1. Cada registro se ubica en una línea separada, delimitada por un salto de línea. Por ejemplo:

```
aaa,bbb,ccc\n
xxx,yyy,zzz\n
```

donde `\n` significa el carácter de control `LF` (`0x0A`).

2. El último registro en el archivo puede tener un salto de línea final o no. Por ejemplo:

```
aaa,bbb,ccc\n
xxx,yyy,zzz
```

3. Debería haber una línea de encabezado que aparezca en la primera línea del archivo, en el mismo formato que las líneas de registro normales. Este encabezado debería contener nombres que correspondan a los campos en el archivo. El número de nombres de campo en la línea de encabezado debe ser igual al número de campos de los registros en el archivo. Por ejemplo:

```
nombre1,nombre2,nombre3\n
aaa,bbb,ccc\n
xxx,yyy,zzz\n
```

4. Dentro del encabezado y de cada registro puede haber uno o más campos separados por comas. Cada línea debe contener el mismo número de campos a través de todo el archivo. Los espacios se consideran parte del campo y, consecuentemente, no son ignorados. El último campo en el registro no debe estar seguido de una coma. Por ejemplo:

```
aaa,bbb,ccc\n
```

5. Los campos pueden estar encerrados entre comillas dobles o no. Por ejemplo:

```
"aaa","bbb","ccc"\n
zzz,yyy,xxx\n
```

6. Si el campo se encierra entre comillas dobles, cada comilla doble que sea parte del campo debe codificarse dos veces. Por ejemplo:

```
"aaa","b""bb","ccc"\n
```

[1] Este material está basado en el documento RFC 4180.

Ejemplo

```
DESDE,HACIA,DISTANCIA,COSTO
Seattle,New-York,2.5,0.12
Seattle,Chicago,1.7,0.08
Seattle,Topeka,1.8,0.09
San-Diego,New-York,2.5,0.15
San-Diego,Chicago,1.8,0.10
San-Diego,Topeka,1.4,0.07
```

C.2 Controlador de tablas xBASE

El controlador de tablas xBASE asume que la tabla de datos se almacenó en formato de archivo .dbf.

Para elegir el controlador de tablas xBASE, su nombre en la sentencia table debe especificarse como `"xBASE"` y el primer argumento debe especificar el nombre de un archivo .dbf que contenga la tabla. Para la tabla de salida, debe haber un segundo argumento definiendo el formato de la tabla en la forma `"FF...F"`, donde `F` es tanto `C(`n`)`, el cual especifica un campo de caracteres de longitud n, como `N(`n`[,`p`])`, el cual especifica un campo numérico de longitud n y precisión p (por defecto, p es 0).

El siguiente es un ejemplo simple que ilustra la creación y lectura de un archivo .dbf:

```
table tab1{i in 1..10} OUT "xBASE" "foo.dbf"
   "N(5)N(10,4)C(1)C(10)": 2*i+1 ~ B, Uniform(-20,+20) ~ A,
   "?" ~ FOO, "[" & i & "]" ~ C;
set M, dimen 4;
table tab2 IN "xBASE" "foo.dbf": M <- [B, C, RECNO, A];
display M;
end;
```

C.3 Controlador de tablas ODBC

El controlador de tablas ODBC permite conexiones con bases de datos SQL usando una implementación de la interfaz ODBC basada en la Call Level Interface (CLI).[2]

Debian GNU/Linux. Bajo Debian GNU/Linux, el controlador de tablas ODBC usa el paquete iODBC,[3] el cual debe estar instalado antes de compilar el paquete GLPK. La instalación se puede efectuar con el siguiente comando:

```
sudo apt-get install libiodbc2-dev
```

Debe notarse que para la configuración del paquete GLPK para activar el uso de la librería iODBC, se debe pasar la opción '`--enable-odbc`' al script de configuración.

[2] La norma de software correspondiente se define en ISO/IEC 9075-3:2003.

[3] Ver `<http://www.iodbc.org/>`.

Para su uso en todo el sistema, las bases de datos individuales deben ingresarse en `/etc/odbc.ini` y `/etc/odbcinst.ini`. Las conexiones de las bases de datos usadas por un usuario individual se especifican mediante archivos en el directorio home (`.odbc.ini` y `.odbcinst.ini`).

Microsoft Windows. Bajo Microsoft Windows, el controlador de tablas ODBC usa la librería ODBC de Microsoft. Para activar esta característica, el símbolo:

```
#define ODBC_DLNAME "odbc32.dll"
```

debe definirse en el archivo de configuración de GLPK '`config.h`'.

Las fuentes de datos pueden crearse por intermedio de las Herramientas Administrativas del Panel de Control.

Para elegir el controlador de tablas ODBC, su nombre en la sentencia table debe especificarse como `'ODBC'` o `'iODBC'`.

La lista de argumentos se especifica como sigue.

El primer argumento es la cadena de conexión que se pasa a la librería ODBC, por ejemplo:

`'DSN=glpk;UID=user;PWD=password'`, o

`'DRIVER=MySQL;DATABASE=glpkdb;UID=user;PWD=password'`.

Las diferentes partes de la cadena se separan con punto y coma. Cada parte consiste de un par *nombre de campo* y *valor* separados por el signo igual. Los nombres de campo permitidos dependen de la librería ODBC. Típicamente, se permiten los siguientes nombres de campo:

`DATABASE` base de datos;

`DRIVER` controlador ODBC;

`DSN` nombre de una fuente de datos;

`FILEDSN` nombre de un archivo de fuente de datos;

`PWD` clave de usuario;

`SERVER` base de datos;

`UID` nombre de usuario.

El segundo argumento, y todos los siguientes, son considerados como sentencias SQL.

Las sentencias SQL se pueden extender sobre múltiples argumentos. Si el último carácter de un argumento es un punto y coma, este indica el final de una sentencia SQL.

Los argumentos de una sentencia SQL se concatenan separados por espacios. El eventual punto y coma final será removido.

Todas las sentencias SQL, excepto la última, se ejecutarán directamente.

Para table-IN, la última sentencia SQL puede ser un comando SELECT que empieza con `'SELECT '` en letras mayúsculas. Si la cadena no se inicia con `'SELECT '`, se considera que es un nombre de tabla y automáticamente se genera una sentencia SELECT.

Para table-OUT, la última sentencia SQL puede contener uno o múltiples signos de interrogación. Si contiene un signo de interrogación, se considera como una plantilla para la rutina de

escritura. De otro modo, la cadena es considerada un nombre de tabla y se genera automáticamente una plantilla INSERT.

La rutina de escritura usa la plantilla con el signo de interrogación y reemplaza al primer signo de interrogación con el primer parámetro de salida, el segundo signo de interrogación con el segundo parámetro de salida, y así sucesivamente. Luego se emite el comando SQL.

El siguiente es un ejemplo de la sentencia table de salida:

```
table ta { l in LOCALIDADES } OUT
   'ODBC'
   'DSN=glpkdb;UID=glpkuser;PWD=glpkpassword'
   'DROP TABLE IF EXISTS resultados;'
   'CREATE TABLE resultados ( ID INT, LOC VARCHAR(255), CANT DOUBLE );'
   'INSERT INTO resultados 'VALUES ( 4, ?, ? )' :
   l ~ LOC, cantidad[l] ~ CANT;
```

Alternativamente, se puede escribir como sigue:

```
table ta { l in LOCALIDADES } OUT
   'ODBC'
   'DSN=glpkdb;UID=glpkuser;PWD=glpkpassword'
   'DROP TABLE IF EXISTS resultados;'
   'CREATE TABLE resultados ( ID INT, LOC VARCHAR(255), CANT DOUBLE );'
   'resultados' :
   l ~ LOC, cantidad[l] ~ CANT, 4 ~ ID;
```

El uso de plantillas con '?' no sólo permite INSERT, sino también UPDATE, DELETE, etc. Por ejemplo:

```
table ta { l in LOCALIDADES } OUT
   'ODBC'
   'DSN=glpkdb;UID=glpkuser;PWD=glpkpassword'
   'UPDATE resultados SET FECHA = ' & fecha & ' WHERE ID = 4;'
   'UPDATE resultados SET CANT = ? WHERE LOC = ? AND ID = 4' :
   cantidad[l], l;
```

C.4 Controlador de tablas MySQL

El controlador de tablas permite conexiones con bases de datos MySQL.

Debian GNU/Linux. Bajo Debian GNU/Linux, el controlador de tablas MySQL usa el paquete MySQL,[4] el cual debe estar instalado antes de compilar el paquete GLPK. La instalación se puede efectuar con el siguiente comando:

```
sudo apt-get install libmysqlclient15-dev
```

Debe notarse que para la configuración del paquete GLPK para activar el uso de la librería MySQL, se debe pasar la opción '`--enable-mysql`' al script de configuración.

[4] Para descargar los archivos de desarrollo, ver `<http://dev.mysql.com/downloads/mysql/>`.

Microsoft Windows. Bajo Microsoft Windows, el controlador de tablas MySQL también usa la librería MySQL. Para activar esta característica, el símbolo:

```
#define MYSQL_DLNAME "libmysql.dll"
```

debe definirse en el archivo de configuración de GLPK '`config.h`'.

Para elegir el controlador de tablas MySQL, su nombre en la sentencia table debe especificarse como `'MySQL'`.

La lista de argumentos se especifica como sigue.

El primer argumento especifica como conectar la base de datos en el estilo DSN, por ejemplo:

`'Database=glpk;UID=glpk;PWD=gnu'`.

Las diferentes partes de la cadena se separan con punto y coma. Cada parte consiste de un par *nombre de campo* y *valor* separados por el signo igual. Se permiten los siguientes nombres de campo:

`Server` servidor corriendo la base de datos (localhost por defecto);

`Database` nombre de la base de datos;

`UID` nombre de usuario;

`PWD` clave de usuario;

`Port` puerto usado por el servidor (3306 por defecto).

El segundo argumento, y todos los siguientes, son considerados como sentencias SQL.

Las sentencias SQL se pueden extender sobre múltiples argumentos. Si el último carácter de un argumento es un punto y coma, este indica el final de una sentencia SQL.

Los argumentos de una sentencia SQL se concatenan separados por espacios. El eventual punto y coma final será removido.

Todas las sentencias SQL, excepto la última, se ejecutarán directamente.

Para table-IN, la última sentencia SQL puede ser un comando SELECT que empieza con `'SELECT '` en letras mayúsculas. Si la cadena no se inicia con `'SELECT '`, se considera que es un nombre de tabla y automáticamente se genera una sentencia SELECT.

Para table-OUT, la última sentencia SQL puede contener uno o múltiples signos de interrogación. Si contiene un signo de interrogación, se considera como una plantilla para la rutina de escritura. De otro modo, la cadena es considerada un nombre de tabla y se genera automáticamente una plantilla INSERT.

La rutina de escritura usa la plantilla con el signo de interrogación y reemplaza al primer signo de interrogación con el primer parámetro de salida, el segundo signo de interrogación con el segundo parámetro de salida y así sucesivamente. Luego se emite el comando SQL.

El siguiente es un ejemplo de la sentencia table de salida:

```
table ta { l in LOCALIDADES } OUT
   'MySQL'
   'Database=glpkdb;UID=glpkuser;PWD=glpkpassword'
   'DROP TABLE IF EXISTS resultados;'
   'CREATE TABLE resultados ( ID INT, LOC VARCHAR(255), CANT DOUBLE );'
   'INSERT INTO resultados VALUES ( 4, ?, ? )' :
   l ~ LOC, cantidad[l] ~ CANT;
```

Alternativamente, se puede escribir como sigue:

```
table ta { l in LOCALIDADES } OUT
   'MySQL'
   'Database=glpkdb;UID=glpkuser;PWD=glpkpassword'
   'DROP TABLE IF EXISTS resultados;'
   'CREATE TABLE resultados ( ID INT, LOC VARCHAR(255), CANT DOUBLE );'
   'resultados' :
   l ~ LOC, cantidad[l] ~ CANT, 4 ~ ID;
```

El uso de plantillas con '?' no sólo permite INSERT, sino también UPDATE, DELETE, etc. Por ejemplo:

```
table ta { l in LOCALIDADES } OUT
   'MySQL'
   'Database=glpkdb;UID=glpkuser;PWD=glpkpassword'
   'UPDATE resultados SET FECHA = ' & fecha & ' WHERE ID = 4;'
   'UPDATE resultados SET CANT = ? WHERE LOC = ? AND ID = 4' :
   cantidad[l], l;
```

Apéndice D

Solución de modelos con glpsol

El paquete GLPK[1] incluye el programa `glpsol`, un *solver* autónomo de PL/PEM. Este programa puede ser invocado desde la línea de comando o desde el *shell* para resolver modelos escritos en el lenguaje de modelado GNU MathProg.

Para comunicarle al solver que el archivo de entrada contiene una descripción del modelo, es necesario especificar la opción `--model` en la línea de comando. Por ejemplo:

```
glpsol --model foo.mod
```

A veces es necesario usar la sección de datos colocada en un archivo separado, en cuyo caso se debe usar el siguiente comando:

```
glpsol --model foo.mod --data foo.dat
```

Debe notarse que, si el archivo del modelo también contiene una sección de datos, esta sección será ignorada.

También se permite especificar más de un archivo conteniendo la sección de datos, por ejemplo:

```
glpsol --model foo.mod --data foo1.dat --data foo2.dat
```

Si la descripción del modelo contiene algunas sentencias display y/o printf, por defecto la salida es enviada a la terminal. Si se necesita redirigir la salida a un archivo, se puede usar el siguiente comando:

```
glpsol --model foo.mod --display foo.out
```

Si se necesita inspeccionar el problema, el cual ha sido generado por el traductor del modelo, se puede usar la opción `--wlp` como sigue:

```
glpsol --model foo.mod --wlp foo.lp
```

En este caso el problema se escribe en el archivo `foo.lp`, en formato CPLEX LP apropiado para el análisis visual.

[1] `http://www.gnu.org/software/glpk/`

A veces sólo se necesita chequear la descripción del modelo, sin resolver la instancia generada del problema. En este caso, se debe especificar la opción `--check`, por ejemplo:

```
glpsol --check --model foo.mod --wlp foo.lp
```

Si se necesita escribir una solución numérica obtenida por el solver en un archivo, se puede usar el siguiente comando:

```
glpsol --model foo.mod --output foo.sol
```

en cuyo caso la solución se escribe en el archivo `foo.sol` en formato de texto plano apropiado para el análisis visual.

La lista completa de opciones de `glpsol` se puede encontrar en el manual de referencia de GLPK incluido en la distribución de GLPK.

Apéndice E

Ejemplo de descripción del modelo

E.1 Descripción del modelo escrita en MathProg

Abajo hay un ejemplo completo de la descripción de un modelo escrito en el lenguaje de modelado GNU MathProg.

```
# UN PROBLEMA DE TRANSPORTE
#
# Este problema determina la logística de costo mínimo de flete
# que cumple los requerimientos en los mercados y en las fábricas
# de suministro.
#
# Referencia:
#              Dantzig G B. 1963. Linear Programming and Extensions.
#              Princeton University Press, Princeton, New Jersey.
#              Sección 3-3.

set I;
/* plantas de enlatado */

set J;
/* mercados */

param a{i in I};
/* producción de la planta i, en cajas */

param b{j in J};
/* demanda en el mercado j, en cajas */

param d{i in I, j in J};
/* distancia, en miles de millas */
```

```
param f;
/* flete, en dólares por caja cada mil millas */

param c{i in I, j in J} := f * d[i,j] / 1000;
/* costo de transporte, en miles de dólares por caja */

var x{i in I, j in J} >= 0;
/* cantidades despachadas, en cajas */

minimize costo: sum{i in I, j in J} c[i,j] * x[i,j];
/* costo total de transporte, en miles de dólares */

s.t. suministro{i in I}: sum{j in J} x[i,j] <= a[i];
/* observar el límite de suministro de la planta i */

s.t. demanda{j in J}: sum{i in I} x[i,j] >= b[j];
/* satisfacer la demanda del mercado j */

data;

set I := Seattle San-Diego;

set J := New-York Chicago Topeka;

param a := Seattle     350
           San-Diego   600;

param b := New-York    325
           Chicago     300
           Topeka      275;

param d :              New-York   Chicago   Topeka :=
           Seattle     2.5        1.7       1.8
           San-Diego   2.5        1.8       1.4  ;

param f := 90;

end;
```

E.2 Instancia generada del problema de PL

Abajo está el resultado de la traducción del modelo de ejemplo producido por el solver `glpsol`, con la opción `--wlp` y escrita en el formato CPLEX LP.

```
\* Problem: transporte *\

Minimize
 costo: + 0.225 x(Seattle,New~York) + 0.153 x(Seattle,Chicago)
 + 0.162 x(Seattle,Topeka) + 0.225 x(San~Diego,New~York)
 + 0.162 x(San~Diego,Chicago) + 0.126 x(San~Diego,Topeka)

Subject To
 suministro(Seattle): + x(Seattle,New~York) + x(Seattle,Chicago)
 + x(Seattle,Topeka) <= 350
 suministro(San~Diego): + x(San~Diego,New~York) + x(San~Diego,Chicago)
 + x(San~Diego,Topeka) <= 600
 demanda(New~York): + x(Seattle,New~York) + x(San~Diego,New~York) >= 325
 demanda(Chicago): + x(Seattle,Chicago) + x(San~Diego,Chicago) >= 300
 demanda(Topeka): + x(Seattle,Topeka) + x(San~Diego,Topeka) >= 275

End
```

E.3 Solución óptima del problema de PL

Abajo está la solución óptima de la instancia generada del problema de PL encontrada por el solver `glpsol`, con la opción `--output` y escrita en formato de texto plano.

```
Problem:    transporte
Rows:       6
Columns:    6
Non-zeros:  18
Status:     OPTIMAL
Objective:  costo = 153.675 (MINimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 costo        B        153.675
     2 suministro[Seattle]
                    NU           350                         350         < eps
     3 suministro[San-Diego]
                    B            550                         600
     4 demanda[New-York]
                    NL           325           325                       0.225
     5 demanda[Chicago]
                    NL           300           300                       0.153
     6 demanda[Topeka]
                    NL           275           275                       0.126
```

```
   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 x[Seattle,New-York]
                    B             50             0
     2 x[Seattle,Chicago]
                    B            300             0
     3 x[Seattle,Topeka]
                    NL             0             0                       0.036
     4 x[San-Diego,New-York]
                    B            275             0
     5 x[San-Diego,Chicago]
                    NL             0             0                       0.009
     6 x[San-Diego,Topeka]
                    B            275             0

End of output
```

Reconocimientos

Los autores desean agradecer a las siguientes personas, quienes amablemente leyeron, comentaron y corrigieron el borrador de este documento:

Juan Carlos Borrás `<borras@cs.helsinki.fi>`

Harley Mackenzie `<hjm@bigpond.com>`

Robbie Morrison `<robbie@actrix.co.nz>`