make.tol de ChRules.Iterative

ChRules.Iterative es un programa iterativo de aplicacion de reglas de reescritura que: a) aplica a un area rectangular de caracteres, b) reglas de transformacion de areas rectangulares de caracteres y c) que juntas forman una base de reglas de transformacion del contenido de ese area para alcanzar un cierto objetivo, como por ejemplo, solucionar un problema Las reglas de ChRules.Iterative son del tipo [condicion, accion], esto es: a) si se cumple la condicion, el rectangulo condicion de la regla equipara con alguna subarea del area de trabajo b) entonces se aplica la accion de transformacion cambiando el contenido de la antigua subarea del area de trabajo por el nuevo rectangulo que proporciona la parte de la accion de la regla.

En las reglas de ChRules.Iterative, tanto la parte de la condicion como la de la accion son 2 rectangulos de caracteres. En principio 2 retangulos de identicas dimensiones, por ejemplo, de 3x4 caracteres, de 5x1 caracteres, 4x4 caracteres, etc. // Podrian diseñarse reglas cuyas condicion y accion fueran de diferentes dimensiones, pero en ese caso se cambiaria la forma del area dando lugar a casos dificiles de plantear y resolver, aunque no imposibles. La parte inicial del nombre del programa, ChRules, proviene de la caracteristica de ser reglas, Rules, de caracteres, Ch de Ch(aracters), por lo que se las podria llamar reglas de caracteres.

La idea basica del funcionamiento de ChRules.Iterative es la siguiente: a) si en el estado actual del area de caracteres existe algun subarea rectangular con el mismo contenido que la parte de condicion de una regla, b) entonces dicha regla es aplicable y, de aplicarse, el subarea rectangular del area de caracteres que coincide con la condicion es sobreescrita, conservando la forma, con el area rectangular de caracteres de la accion de la regla. Por tanto, estas reglas de rectangulos de caracteres pueden considerarse como reglas de reescritura, pero, a diferencia de otras reglas de reescritura, en vez de trabajar con secuencias de caracteres trabajan con areas rectangulares de caracteres.

Cada caso de estudio (problema a resolver) para ChRules.Iterative se define mediante: a) sus areas de trabajo iniciales, que pueden ser una o mas, para diferentes variantes del mismo problema y b) sus bases de reglas para su resolucion, generalmente una base, pero puede haber mas para diferentes formas de solucionar el problema y, aun mas, estas bases de reglas se pueden combinar mediante, el algebra de conjuntos de Tol, para crear nuevas bases de reglas para determinados tipos de problemas. // Por ejemplo, para el caso de los robots que buscan como salir de un laberinto se puenden: a) Definir distintos laberintos con diferentes posiciones de entrada y salidada de los robots y con uno o mas robots. b) Se puede definir una base de reglas para que el robot emplee el algoritmo llamado de la mano derecha y, tambien, su simetrico de la mano izquierda. c) Una base de reglas para resolver conflictos de bloqueo mutuo cuando 2 o mas robots intentan salir del laberinto. d) Crear una base de reglas conjunto con la base de reglas de resolucion de conflictos mas la base de reglas del algoritmo de la mano derecha (o de la izquierda).

Esta version del programa ChRules.Iterative tiene 3 particularidades que la definen: a) El ciclo del motor de comprobacion y aplicacion de reglas es iterativo. b) La seleccion de la subarea donde aplicar la regla es determinista. c) Pero en la seleccion de las reglas que hay aplicar se pueden programar 3 metodos distintos. Estos 3 metodos de seleccion de reglas son: c1) Seleccion completamente al azar. c2) Seleccion determinista siguiendo el mismo orden en el que se han definido las reglas de reescritura (cada regla prevalece sobre sus reglas sucesoras). c3) Seleccion segun un parametro rndCtr entre 0 y 1 segun el cual en rndCtr casos la seleccion es al azar y en el otro 1-rndCtr de los casos la seleccion es determinista, segun el orden de declaracion de las reglas.

Los casos de estudio que incluye ChRules.Iterative son, por ejemplo: a) Automatas celulares con sus reglas de la vida, que pueden funcionar tanto de forma determinista como aletatoria. Este caso incluye un area y 2 bases de reglas. b) Robots en laberintos que encuentran la salida con el algoritmo de la mano derecha o el de la izquierda y que solo se pueden resolver aplicando las reglas de forma determinista, ya que al azar los robots suelen perderse y dar vueltas, aunque tarde o temprano terminarian por encontrar la salida, por casualidad. En este caso se incluyen varios laberintos y varias bases de reglas que se pueden combinar para crear otras. c) Relleno de figuras, que se resuelven bien de forma aleatoria y que tambien funcionan de manera determinista, se incluyen 2 areas y una base de reglas sencilla. d) Partidas de pong, el pingpong electronico, en las que de forma completamente al azar es dificil que los jugadores den mas de 2 toques seguidos sin perder, de forma completamente determinista lo habitual es que los jugadores no fallen nunca si las reglas estan bien definidas (pues las reglas del jugador se definen antes que las reglas del movimiento de la pelota), entrando facilmente en una partida que es un ciclo eterno y con una cierta proporcion de azar, por ejemplo, 0.15 o 0.20, en los que las partidas simulan ser mas reales con varios toques consecutivos pero sin hacerse eternas. e) Una simulacion de un sistema de sedimentacion, orden parcial, mediante un metodo similar al metodo de ordenacion llamado de la burbuja.

ChRules.Iterative visualiza sus resultados de 3 formas diferentes: a) Mediante una traza de evolucion del mapa por pantalla. b) Mediante una traza en un fichero en disco, escrita en Javascript, que se traslada tal y como esta a un simulador que permite la posterior simulacion de los resultados. c) Mediante paginas escritas en Html con los casos resueltos paso a paso. El programa se estructura en base a un probrama principal make.tol que se incluye: a) ficheros de funciones comunes de conjuntos y ficheros, b) ficheros de funciones por tipos de objeto que emplea, areas, reglas y motor de resulucion y c) tantos ficheros de aplicacion como tipos de casos se han programado, automatas celulares, robots y laberintos, juego del pong, etc.

La funcion principal de ChRules.Iterative se llama Set EngSol() que retorna la solucion en forma de un conjunto de textos y cuyos parametros son: a) Set linSet, que es el area a resolver, b) Set rulSet, que es la base de reglas que hay que aplicar, c) Real maxSta, que es el numero maximo de iteraciones, nuevos estados por los que puede pasar el area de entrada y que es util frente a ciclos eternos, d) Text preFil, que es la ruta y el nombre de 2 fichero, sin extension, que guardaran las trazas en Html y Javascript. e) Real numCol, que es el numero de columnas de las tablas de estados de la traza de salida en Html y f) Real rndCtr, que es el control del determinismo en la seleccion de las reglas: si es 0 es determinista, si es 1 es aleatorio y si esta entre 0 y 1 pseudoaletorio en una la proporcion que determine el valor de rndCtr.

ChRules.Iterative es un programa que se clasifica como: a) Iterativo y existiendo su version recursiva. b) De reglas, en este caso de ragas de areas rectangulares de caracteres, c) Que implementa, entre otros ejemplos, un automata celular. d) Con un comportamiento que, ademas de determinista, puede ser aleatorio o pseudoaleatorio. e) Que en el caso de los robots dentro de un laberinto, realiza una busqueda con el objetivo de encontrar, desde un punto, la salida. f) En sedimentacion implementa, parcialmente, un proceso de orden por el metodo de la burbuja. Se ha probado su funcionamiento para las versiones de Tol 1.1.1, 1.1.5, 1.1.6 y 2.0.1.

Árbol de ficheros

ChRules.Iterative programa de aplicacion iterativa de reglas de areas de caracteres

- ← make.tol aplica unas bases de reglas de reescritura a varios escenarios
- ← make.bat mandato de ejecucion del programa de aplicacion de reglas
- tol directorios que contienen fichero de codigo fuente Tol
  - cmm funciones comunes
    - ← txt.tol funciones de textos
    - ← set.tol funciones de conjuntos
    - ← dir.tol funciones de directorios
  - app directorio funciones y datos de areas, reglas y motor de reglas
    - ← are.tol funciones de areas sobre las que se aplican las reglas
    - ← rul.tol funciones de reglas que transforman de areas de caracteres
    - ← eng.tol funciones del motor iterativo de aplicacion de reglas
    - ← bub.tol para realizar un orden parcial con metodo de burbuja
    - ← cel.tol automata celular con reglas de reproduccion y muerte
    - ← lab.tol base de reglas para que un robot salga de un laberinto
    - ← fll.tol base de reglas para rellenar, fill, superficies cerradas
    - ← pon.tol base de reglas de un pingpong electronico, pong
  - ← inc.tol para la inclusion de funciones, areas y bases de reglas
- simulator directorio del simulador en Javascript del motor iterativo de reglas
  - css directorio para css, Cascading Style Sheets, del simulador
    - ← simulator.css css para simular areas de aplicacion de las reglas
  - src directorio de codigo fuente Javascript del simulador de reglas
    - ← simulator.js simula el funcionamiento del motor de aplicacion de reglas
    - ← simulatorarray.js array con ejemplos de evolucion para cada base de reglas
- → automata_celular_al_azar.html automata celular de reglas al azar de reproduccion y muerte

→ automata_celular_determinista.html automata celular determinista de reglas de reproduccion y muerte

→ burbuja_y_sedimentos.html orden parcial con metodo de burbuja simulando una sedimentacion

→ laberinto_2_cabezones.html laberinto con 2 robots que se autobloquean y no pueden salir

→ laberinto_2_negociadores.html laberinto con 2 robots y reglas para resolver su mutuo bloqueo

→ laberinto_mas_grande.html robot que sale del laberinto aplicando reglas de la mano derecha

→ laberinto_mano_izquierda_html robot que busca la salida aplicando reglas de la mano izquierda

→ laberinto_mini_1_robot.html pequeño laberinto con un robot y reglas de la mano derecha

→ partida_de_pong_al_15.html partida de pong donde el 15% de las veces las reglas son al azar

→ partida_de_pong_al_20.html partida de pong donde el 20% de las veces las reglas son al azar

→ partida_de_pong_azarosa.html partida de pingpong electronico, pong, con reglas aleatorias

→ partida_de_pong_eterna.html ciclo eterno de una partida de pong, con reglas deterministas

→ rellena_caja_al_azar.html reglas para llenar superficies aplicadas a una caja con lineas

→ rellena_k_al_azar.html rellenado de superficies aplicado al interior de una letra K

→ simulator.html simulador del motor iterativo de reglas de areas de caracteres

→ chrules_iterative.pdf funciones del motor de aplicacion de reglas de caracteres

Declaraciones

Pruebas

- `Text trcDir`
  Directorio de trazas Html y Javascript
- `Text tstCmd`
  Mandatos de que se le quiere dar al motor de reglas los de autómatas celulares, los de robots en laberintos, los ping pong, etc. con su base de hechos (area) y sus caracteristicas de aleatoriedad o no, etc.
- `Real tstExe`
  Aplica la reglas a areas dependiendo de tstCmd.
- `Real tstJsc`

Reune los ficheros Javascript, lo prepara para el simulado y crea la tabla array indice a los diferentes casos.

Text trcDir

```
////////////////////////////////////////////////////////////////////////////
Text trcDir = "trace";
////////////////////////////////////////////////////////////////////////////
PutDescription("Directorio de trazas Html y Javascript",trcDir);
////////////////////////////////////////////////////////////////////////////
```

Text tstCmd

```
////////////////////////////////////////////////////////////////////////////
Text tstCmd = "fllRndupK"; // Caso de estudio
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Mandatos de que se le quiere dar al motor de reglas los de autómatas
celulares, los de robots en laberintos, los ping pong, etc. con su
base de hechos (area) y sus caracteristicas de aleatoriedad o no, etc.",
tstCmd);
////////////////////////////////////////////////////////////////////////////
```

Real tstExe

```
////////////////////////////////////////////////////////////////////////////
Real tstExe = Case(
  tstCmd == "bubSedUp1",
  {
    Text trcPth = trcDir+"/"+"burbuja_y_sedimentos";
    Set  rulApp = EngSol(BubSed, BubUp1, 900, trcPth, 2, TRUE); // Al azar
    Card(rulApp)
  },
  tstCmd == "celRndP31",
  {
    Text trcPth = trcDir+"/"+"automata_celular_al_azar";
    Set  rulApp = EngSol(CelP31, CelRb1, 900, trcPth, 2, TRUE); // Al azar
    Card(rulApp)
  },
  tstCmd == "celDetP31",
  {
    Text trcPth = trcDir+"/"+"automata_celular_determinista";
    Set  rulApp = EngSol(CelP31, CelRb1, 900, trcPth, 2, FALSE); // Determin
    Card(rulApp)
  },
  tstCmd == "ponUpA020",
  {
    Text trcPth = trcDir+"/"+"partida_de_pong_al_20"; // Ruta de las trazas
    Set  rulApp = EngSol(PonUpA, PonRb1, 600, trcPth, 6, 0.20); // 20% azar
    Card(rulApp)
  },
  tstCmd == "ponUpA015",
  {
    Text trcPth = trcDir+"/"+"partida_de_pong_al_15"; // Ruta de las trazas
    Set  rulApp = EngSol(PonUpA, PonRb1, 600, trcPth, 6, 0.15); // 15% azar
    Card(rulApp)
  },
  tstCmd == "ponUpA000",
  {
    Text trcPth = trcDir+"/"+"partida_de_pong_eterna"; // Ruta de las trazas
    Set  rulApp = EngSol(PonDwA, PonRb1, 600, trcPth, 6, FALSE); // Determin
    Card(rulApp)
  },
  tstCmd == "ponUpA100",
  {
    Text trcPth = trcDir+"/"+"partida_de_pong_azarosa"; // Ruta de las trazas
```

```
    Set  rulApp = EngSol(PonUpA, PonRb1, 600, trcPth, 6, TRUE); // Al azar
    Card(rulApp)
  },
  tstCmd == "labMax001",
  {
    Text trcPth = trcDir+"/"+"laberinto_mas_grande"; // Ruta de las trazas
    Set  rulApp = EngSol(LabA11, LabOutRgh, 300, trcPth, 2, FALSE);
    Card(rulApp)
  },
  tstCmd == "labMaxLft",
  {
    Text trcPth = trcDir+"/"+"laberinto_mano_izquierda"; // Ruta de las trazas
    Set  rulApp = EngSol(LabA11, LabOutLft, 300, trcPth, 2, FALSE);
    Card(rulApp)
  },
  tstCmd == "labMinN02", // Cuando se chocan lo arreglan, se aplica la LabRb2
  {
    Text trcPth = trcDir+"/"+"laberinto_2_negociadores"; // Ruta de las trazas
    Set  rulApp = EngSol(LabA02, LabNegOutRgh, 197, trcPth, 4, FALSE);
    Card(rulApp)
  },
  tstCmd == "labMinC02", // Se chocan y se autobloquean
  {
    Text trcPth = trcDir+"/"+"laberinto_2_cabezones"; // Ruta de las trazas
    Set  rulApp = EngSol(LabA02, LabOutRgh, 197, trcPth, 4, FALSE);
    Card(rulApp)
  },
  tstCmd == "labMin001",
  {
    Text trcPth = trcDir+"/"+"laberinto_mini_1_robot"; // Ruta de las trazas
    Set  rulApp = EngSol(LabA01, LabOutRgh, 100, trcPth, 4, FALSE);
    Card(rulApp)
  },
  tstCmd == "labMinRnd",
  {
    Text trcPth = trcDir+"/"+"laberinto_robot_loco"; // Ruta de las trazas
    Set  rulApp = EngSol(LabA11, LabOutRgh, 500, trcPth, 4, TRUE); // Random
    Card(rulApp)
  },
  tstCmd == "fllRndBox",
  {
    Text trcPth = trcDir+"/"+"rellena_caja_al_azar"; // Ruta de las trazas
    Set  rulApp = EngSol(FllBox, FllRb1, 500, trcPth, 2, TRUE); // Random
    Card(rulApp)
  },
  tstCmd == "fllRndupK",
  {
    Text trcPth = trcDir+"/"+"rellena_k_al_azar"; // Ruta de las trazas
    Set  rulApp = EngSol(FllupK, FllRb1, 300, trcPth, 2, TRUE); // Random
    Card(rulApp)
  },
  TRUE,             FALSE); // No hace nada
////////////////////////////////////////////////////////////////////////////
PutDescription("Aplica la reglas a areas dependiendo de tstCmd.", tstExe);
////////////////////////////////////////////////////////////////////////////
```

Real tstJsc

```
////////////////////////////////////////////////////////////////////////////
Real tstJsc = EngJavascript(trcDir, "simulator/src"); // Dirs traza y .js
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Reune los ficheros Javascript, lo prepara para el simulado y crea la tabla
array indice a los diferentes casos.",
tstJsc);
////////////////////////////////////////////////////////////////////////////
```

txt.tol de ChRules.Iterative

Text functions.

Declaraciones

Funciones de nombre corto

- `Text F(Anything anyVal)`

  Retorna numeros, fechas, textos, conjuntos como un texto de formato simple.

Funciones de nombre corto

Text F()

```
////////////////////////////////////////////////////////////////////////////
Text F(Anything anyVal)
////////////////////////////////////////////////////////////////////////////
{
  Text graVal = Grammar(anyVal);
  Case
  (
    graVal=="Text", anyVal, // Ya es texto

    graVal=="Real", If(EQ(anyVal, Round(anyVal)),
                       FormatReal(anyVal, "%.0lf"),  // Entero sin decimales
                       FormatReal(anyVal, "%.2lf")), // 2 decimales

    graVal=="Date", If(Hour(anyVal),
                       FormatDate(anyVal, "%c%Y/%m/%d %h:%i:%s"), // Tiempo
                       FormatDate(anyVal, "%c%Y/%m/%d")),         // Fecha

    graVal=="Set",
    {
      Real crdSet = Card(anyVal);
      Case(
      EQ(crdSet,0), "[]",
      EQ(crdSet,1), "["+F(anyVal[1])+"]",
      TRUE,         "["+F(anyVal[1])+SetSum(For(2,crdSet,Text(Real setPos)
                  { "|"+F(anyVal[setPos]) }))+"]")
    },

    TRUE,           "Not basic type"
  )
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna numeros, fechas, textos, conjuntos como un texto de formato simple.",
F);
////////////////////////////////////////////////////////////////////////////
```

set.tol de ChRules.Iterative

Set functions.

Declaraciones

Funciones

- `Text Set2Txt(Set valSet, Text iniTxt, Text endTxt, Text sepTxt, Text`

sepLst, Text txtDet, Text datFmt, Text dteDet, Text dteFmt)

Returns a text like a list with all the elements of valSet converted in a text format (elements types: Text, Real or Date). Arguments: - iniTxt initial text, for example: '(', '[[', '', etc. - endTxt end text, for example ')', ']]', '', etc. - sepTxt elements separator, for example '; ', ',', '', etc. - sepLst two last elements separator, for example, ' & ', ' and ', etc., you can specify the same as sepTxt - txtDet text delimiters, for exmple, quotes for TOL, single quote for SQL, nothing, etc. - datFmt real numbers format, for explame, '%.0lf' for integers, if none then uses the default TOL real number format. - dteDet date delimiters, for example, simgle quote for SQL. - dteFmt date format, for example, '%c%Y%m%d', if none then uses the default TOL dates format. Only works with TOL types Text, Real or Date, when find a Set type then works in a recursive way with the same arguments.

- Set SetShuffle(Set anySet)

  Retorna el conjunto de lo que sea desordenado, para ello: - Para cada elemento del conjunto, sea lo que sea, Anything, construye una tabla de pares formados por un numero aleatorio y cada uno de los elementos. - Ordena por el elemento aleatorio, por lo que sera un total desorden. - Recorre la tabla desordenada retornando el conjunto de los valores originales.

Text Set2Txt()

```
////////////////////////////////////////////////////////////////////////////
Text Set2Txt(Set  valSet, // Set of elements
             Text iniTxt, // Initial text for list
             Text endTxt, // End text for list
             Text sepTxt, // Element separators
             Text sepLst, // 2 last elements separator
             Text txtDet, // Delimiter for texts
             Text datFmt, // Format for real numbers
             Text dteDet, // Delimiter for dates
             Text dteFmt) // Format for dates
////////////////////////////////////////////////////////////////////////////
{
  Real card = Card(valSet);
  Text body =
    If(EQ(card,0), "",
    If(EQ(card,1), F(valSet[1]),
    If(EQ(card,2), F(valSet[1])+sepLst+F(valSet[2]),
    {
      Set txtVal = For(2,card,Text(Real p)
      {
        If(EQ(p,card),sepLst,sepTxt) + F(valSet[p])
      });
      F(valSet[1]) + BinGroup("+",txtVal)
    })));
  iniTxt+body+endTxt
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Returns a text like a list with all the elements of valSet converted in a
text format (elements types: Text, Real or Date).
Arguments:
 - iniTxt initial text, for example: '(', '[[', '', etc.
 - endTxt end text, for example ')', ']]', '', etc.
 - sepTxt elements separator, for example '; ', ',', '', etc.
 - sepLst two last elements separator, for example, ' & ', ' and ', etc.,
   you can specify the same as sepTxt
 - txtDet text delimiters, for exmple, quotes for TOL, single quote for
   SQL, nothing, etc.
 - datFmt real numbers format, for explame, '%.0lf' for integers, if none
   then uses the default TOL real number format.
 - dteDet date delimiters, for example, simgle quote for SQL.
 - dteFmt date format, for example, '%c%Y%m%d', if none
```

```
   then uses the default TOL dates format.
Only works with TOL types Text, Real or Date, when find a Set type then
works in a recursive way with the same arguments.",
Set2Txt);
////////////////////////////////////////////////////////////////////////////
```

Set SetShuffle()

```
////////////////////////////////////////////////////////////////////////////
Set SetShuffle(Set anySet) // Set of Anything
////////////////////////////////////////////////////////////////////////////
{
  Set tabSet = EvalSet(anySet, Set(Anything anyVal) // Para todo elemento
               { [[ Real Rand(0,1), anyVal ]] });   // Par [random, valor]

  Set tabSrt = Sort(tabSet, Real(Set a, Set b) // Ordenar por el aleatorio
               { Compare(a[1],b[1]) });

  EvalSet(tabSrt, Anything(Set parSet) { parSet[2] }) // Desordenados
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el conjunto de lo que sea desordenado, para ello:
- Para cada elemento del conjunto, sea lo que sea, Anything,
  construye una tabla de pares formados por un numero aleatorio y cada uno
  de los elementos.
- Ordena por el elemento aleatorio, por lo que sera un total desorden.
- Recorre la tabla desordenada retornando el conjunto de los valores
  originales.",
SetShuffle);
////////////////////////////////////////////////////////////////////////////
```

dir.tol de ChRules.Iterative

Directories functions.

Declaraciones

Funciones

- `Set DirExtAll(Text dirPth, Text chkExt, Real toLowe, Real casSen)`
  Returns a set of relative paths of files with the extension chkExt that are inside the directory dirPth and inside all of its directories. If toLowe then all names are changed to lower case. If casSen then are case sensitive in the extension match. Is a version of DirAll() function that only check extensions and not uses TextMatch() that writes warnings at Tol 2.0.1.
- `Text DirReadFiles(Text dirPth, Text chkExt, Text filSep)`
  Returns as a text the contents of all files in dirPth that theirs file names match with file pattern filPat. In this text, the contenst of each file will be separated with filSep.

Set DirExtAll()

```
////////////////////////////////////////////////////////////////////////////
Set DirExtAll(Text dirPth, // Directory path
              Text chkExt, // File extension
              Real toLowe, // If true all paths are changed to lower case
              Real casSen) // If true the extension match is case sensitive
////////////////////////////////////////////////////////////////////////////
{
  If(Not(DirExist(dirPth)), Empty,
  {
    Set  getDir = GetDir(dirPth);
    Set  filSet = getDir[1];
    Set  dirSet = getDir[2];

    Set  filFnd = EvalSet(filSet, Text(Text filNam)
    {
      If(!FilCheckExtension(filNam, chkExt, casSen), "",
         If(toLowe, ToLower(dirPth+"/"+filNam), dirPth+"/"+filNam))
    });

    Set filSel = Select(filFnd, Real(Text filNam) { filNam != "" });

    Set  dirFnd = EvalSet(dirSet, Set(Text subDir)
      { DirExtAll(dirPth+"/"+subDir, chkExt, toLowe, casSen) });

    Real dirCar = Card(dirFnd);

    If(EQ(dirCar,0), filSel,
    If(EQ(dirCar,1), filSel << dirFnd[1],
                     filSel << BinGroup("+", dirFnd)))
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Returns a set of relative paths of files with the extension chkExt that
are inside the directory dirPth and inside all of its directories.
If toLowe then all names are changed to lower case.
If casSen then are case sensitive in the extension match.
Is a version of DirAll() function that only check extensions and not uses
TextMatch() that writes warnings at Tol 2.0.1.",
DirExtAll);
////////////////////////////////////////////////////////////////////////////
```

Text DirReadFiles()

```
////////////////////////////////////////////////////////////////////////////
Text DirReadFiles(Text dirPth, // Directory path
                  Text chkExt, // File extension
                  Text filSep) // File separator
////////////////////////////////////////////////////////////////////////////
{
  Set pthSet = DirExtAll(dirPth, chkExt, FALSE, TRUE);
  Set txtSet = EvalSet(pthSet, Text(Text filPth) { ReadFile(filPth) });
  Set2Txt(txtSet, "", "", filSep, filSep, "", "", "", "")
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Returns as a text the contents of all files in dirPth that theirs file names
match with file pattern filPat.
In this text, the contenst of each file will be separated with filSep.",
DirReadFiles);
////////////////////////////////////////////////////////////////////////////
```

are.tol de ChRules.Iterative

Funciones para areas de caracteres.

Declaraciones

Funciones

- `Set AreSub(Set linSet, Real x, Real y, Real w, Real h)`
  Dado un conjunto de lineas de texto retorna un conjunto de lineas de texto cuya esquina superior izquierda corresponden a la posicion x e y y su ancho y alto a w y h. Esto es si x=3, y=4, w=4 y h=2 entonces de: 123456789 2ABCDEFGH 3ABCDEFGH 4Apor FGH => <por > 5AaquiFGH <aqui> Se asume que el conjunto de textos es rectangular y no vacio.
- `Real AreCmp(Set linSetA, Set linSetB)`
  Retorna TRUE si dos areas de texto son iguales, esto es, compara el primer conjunto de lineas con el segundo conjunto de lineas. Esta funcion corta y retorna FALSE en cuanto una comparacion falla Para poder cortar se apoya en la funcion While().
- `Set AreWri(Set linSet, Set newSet, Real x, Real y)`
  Dado un conjunto de lineas, setLin, de texto retorna un conjunto de lineas de texto resultado de sobreescribir el conjunto de textos newSet en la posicion determinadas por x e y. Esto es, si x=3, y=4 entonces de: 123456789 123456789 2ABCDEFGH 2ABCDEFGH 3ABCDEFGH 3ABCDEFGH 4ABCDEFGH => <++++> => 4A++++FGH 5ABCDEFGH <++++> 5A++++FGH Esta funcion asume: a) que los conjuntos de textos son rectangulares y no vacios y b) que las coordenadas son razonables, esto es, que x e y no son tales que parte de la escritura de newSet se salga fuera de setLin. Estas suposiciones son razonables si se tiene en cuenta que las coordenadas proceden de un match correcto de la parte de condicion de las reglas de reescritura de areas de caracteres.
- `Set AreFnd(Set linSet, Set fndSet)`
  Retorna el conjunto formado por el par de coordenadas (x,y) de la primera ocurrencia del conjunto de textos fndSet en linSet. Si no lo encuentra retorna el conjunto vacio.
- `Set AreRep(Set linSet, Set oldSet, Set newSet)`
  Si oldSet aparece en linSet entonces retorna el resultado de reemplazar la primera ocurrencia de oldSet por newSet, si no aparece retorna Empty.
- `Set AreCic(Set linSet, Set parSet)`
  Retorna el resultado de cambiar en linSet el primer par de parSet, formado por un cojunto de pares (oldSet, newSet), donde oldSet aparezca en linSet, si no aparece ninguno retorna Empty.
- `Real AreTrc(Set linSet, Real numCel, Text preFil, Real numCol, Real trcCtr)`

Visualiza y guarda la traza de de un estado de 3 formas en lenguaje Html, en Javascript y por pantalla. Dependiendo del parametro trcCtr, si 0 abre los 2 ficheros Html y Javascript, si 1 asume que esta en medio ciclo y si 2 cierra ficheros de forma logica, esto es pone los cierres de las estructuras de programacion de Html y Javascript. En Javascript, cada rectangulo de texto se pone como una sola linea de texto entre comillas, con un separador donde termina realmente cada linea original del rectangulo, luego este separador se por un br, salto de linea Html. El caracter separador se ha de elegir de tal forma que pueda reemplazarse sin probrmas, por ejemplo, el | suele dar problemas pues se usa dentro de las figuras de las areas, el ; da problemas porque es el fin de los caracteres especiales Html como   o > o <. En este caso se ha elegido la virgulilla ~.

Set AreSub()

```
////////////////////////////////////////////////////////////////////////
Set AreSub(Set  linSet, // Conjunto de linea
           Real x,      // Coordenada X
           Real y,      // Coordenada Y
           Real w,      // Ancho, width
           Real h)      // Alto, height
////////////////////////////////////////////////////////////////////////
{
  Real xEnd = Min(x+w-1,TextLength(linSet[1]));
  Real yEnd = Min(y+h-1,Card(linSet));

  For(y, yEnd, Text(Real lin) { Sub(linSet[lin], x, xEnd) })
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Dado un conjunto de lineas de texto retorna un conjunto de lineas de texto
cuya esquina superior izquierda corresponden a la posicion x e y y su ancho y
alto a w y h. Esto es si x=3, y=4, w=4 y h=2 entonces de:
   123456789
   2ABCDEFGH
   3ABCDEFGH
   4Apor FGH => <por >
   5AaquiFGH    <aqui>
Se asume que el conjunto de textos es rectangular y no vacio.",
AreSub);
////////////////////////////////////////////////////////////////////////
```

Real AreCmp()

```
////////////////////////////////////////////////////////////////////////
Real AreCmp(Set linSetA,  // Primer  conjunto de lineas
            Set linSetB)  // Segundo conjunto de lineas
////////////////////////////////////////////////////////////////////////
{
  Real carA = Card(linSetA);
  Real carB = Card(linSetB);
  If(NE(carA, carB), FALSE,
  {
    Real end = Copy(FALSE);
    Real ok  = Copy(TRUE);
    Real lin = 1;
    Real While(Not(end),
    {
      If(linSetA[lin]!=linSetB[lin],
         { Real(end:=TRUE); Real(ok:=FALSE); FALSE },
         {
           Real(lin:=lin+1);
           If(LE(lin,carA), TRUE, { Real(end:=TRUE); FALSE })
         })
    });
    Copy(ok)
  })
```

```
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna TRUE si dos areas de texto son iguales, esto es, compara el primer
conjunto de lineas con el segundo conjunto de lineas.
Esta funcion corta y retorna FALSE en cuanto una comparacion falla
Para poder cortar se apoya en la funcion While().",
AreCmp);
////////////////////////////////////////////////////////////////////////
```

Set AreWri()

```
////////////////////////////////////////////////////////////////////////
Set AreWri(Set  linSet, // Area que hay que transformar
           Set  newSet, // Area mas pequeña que se va escribir sobre linSet
           Real x,      // Coordenada X del punto donde se empieza a escribir
           Real y)      // Coordenada Y del punto donde se empieza a escribir
////////////////////////////////////////////////////////////////////////
{
  Real linLen = TextLength(linSet[1]); // Ancho, se asume rectangulo
  Real newLen = TextLength(newSet[1]); // Ancho, se asume rectangulo

  Set  ini = For(1, y-1, Text(Real pos)
                 { linSet[pos] });

  Set  rep = For(y,   y+Card(newSet)-1, Text(Real pos)
  {
    Sub(linSet[pos],1,x-1)+          // Parte inicial de la linea original
    newSet[pos-y+1]+                 // Nuevo texto
    Sub(linSet[pos],x+newLen,linLen) // Parte final de la linea original
  });

  Set  end = For(y+Card(newSet), Card(linSet), Text(Real pos)
                 { linSet[pos] });

  ini << rep << end
};
////////////////////////////////////////////////////////////////////////
PutDescription(
"Dado un conjunto de lineas, setLin, de texto retorna un conjunto de lineas de
texto resultado de sobreescribir el conjunto de textos newSet en la posicion
determinadas por x e y.
Esto es, si x=3, y=4 entonces de:
   123456789              123456789
   2ABCDEFGH              2ABCDEFGH
   3ABCDEFGH              3ABCDEFGH
   4ABCDEFGH => <++++> => 4A++++FGH
   5ABCDEFGH    <++++>    5A++++FGH
Esta funcion asume:
a) que los conjuntos de textos son rectangulares y no vacios y
b) que las coordenadas son razonables, esto es,
   que x e y no son tales que parte de la escritura de newSet
   se salga fuera de setLin.
Estas suposiciones son razonables si se tiene en cuenta que las coordenadas
proceden de un match correcto de la parte de condicion de las reglas de
reescritura de areas de caracteres.",
AreWri);
////////////////////////////////////////////////////////////////////////
```

Set AreFnd()

```
////////////////////////////////////////////////////////////////////////
Set AreFnd(Set linSet, // Conjunto de lineas de texto donde hay que buscar
           Set fndSet) // Conjunto rectangular de lineas de texto a buscar
////////////////////////////////////////////////////////////////////////
```

```
{
  Real curX   = 1;
  Real curY   = 1;
  Real width  = TextLength(fndSet[1]);
  Real endX   = TextLength(linSet[1]) - width + 1;
  Real height = Card(fndSet);
  Real endY   = Card(linSet)-height+1;
  Real end    = Copy(FALSE);

  Real While(Not(end),
  {
    Real fst = TextFind(linSet[curY], fndSet[1], curX);
    If(LE(fst,0),
    { // No aparece ni la primera linea de fndSet -> Saltar de linea
      Real(curY:=curY+1);
      Real(curX:=1);
      If(LE(curY,endY), TRUE,            // Quedan lineas
                        Real(end:=TRUE)) // Termino
    },
    {
      Set sub = AreSub(linSet, curX, curY, width, height);
      If(AreCmp(sub, fndSet), Real(end:=TRUE), // Lo encontro
      { // Aunque la primera linea coincide el resto no -> Saltar en X
        Real(curX:=curX+1);
        If(LE(curX, endX), TRUE, // Seguir explorando la linea
        {
          // Ya no aparece en esta linea -> Saltar de linea
          Real(curY:=curY+1);
          Real(curX:=1);
          If(LE(curY,endY), TRUE,            // Quedan lineas
                            Real(end:=TRUE)) // Termino con todas las lineas
        })
      })
    })
  });

  If(LE(curY, endY), [[ Copy(curX), Copy(curY) ]], Empty)
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el conjunto formado por el par de coordenadas (x,y) de la primera
ocurrencia del conjunto de textos fndSet en linSet.
Si no lo encuentra retorna el conjunto vacio.",
AreFnd);
////////////////////////////////////////////////////////////////////////////
```

Set AreRep()

```
////////////////////////////////////////////////////////////////////////////
Set AreRep(Set linSet, // Conjunto de lineas de texto con forma rectangular
           Set oldSet, // Viejo texto rectangular que se va a sustituir
           Set newSet) // Nuevo texto rectangular que se va a escribir
////////////////////////////////////////////////////////////////////////////
{
  Set xySet = AreFnd(linSet, oldSet);
  If(EQ(Card(xySet),0), Empty, AreWri(linSet, newSet, xySet[1], xySet[2]))
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Si oldSet aparece en linSet entonces retorna el resultado de reemplazar la
primera ocurrencia de oldSet por newSet, si no aparece retorna Empty.",
AreRep);
////////////////////////////////////////////////////////////////////////////
```

Set AreCic()

```
////////////////////////////////////////////////////////////////////////////
Set AreCic(Set linSet, // Conjunto de lineas de texto rectangular
           Set parSet) // 2 rectangulos (rectangulo viejo, rectangulo nuevo)
```

```
////////////////////////////////////////////////////////////////////////////
{
  Real end   = Copy(FALSE);
  Real par   = 1;
  Real max   = Card(parSet);
  Set  xySet = Copy(Empty);
  Real While(Not(end),
  {
    Set(xySet:=AreFnd(linSet, parSet[par][1]));
    If(GT(Card(xySet),0), Real(end:=TRUE), // Termino con exito
    {
      Real(par:=par+1); // Probar con el siguiente
      If(GT(par,max), Real(end:=TRUE), // Termino sin exito
                      FALSE)           // Cicla
    })
  });

//Text WriteLn("Rule "+FormatReal(par,"%.0lf"));
  If(GT(par,max), Empty, // No lo encontro
     AreWri(linSet, parSet[par][2], xySet[1], xySet[2])) // Lo encontro
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el resultado de cambiar en linSet el primer par de parSet, formado
por un cojunto de pares (oldSet, newSet), donde oldSet aparezca en linSet,
si no aparece ninguno retorna Empty.",
AreCic);
////////////////////////////////////////////////////////////////////////////
```

Real AreTrc()

```
////////////////////////////////////////////////////////////////////////////
Real AreTrc(Set  linSet, // Conjunto de lineas, area rectangular
            Real numCel, // Numero de celda, que es el estado o paso
            Text preFil, // Ruta y nombre del fichero sin extension
            Real numCol, // Numero de columnas de la tabla
            Real trcCtr) // 0 abre fichero, 1 ciclo, 2 cierra fichero
////////////////////////////////////////////////////////////////////////////
{
  Text htmFil = preFil+".htm"; // Fichero Html
  Text srcFil = preFil+".js";  // Fichero Javascript

  Text srcIni = If(trcCtr == 0, // Apertura
  {
    Text htmOpn = "<table border=0 cellspacing=1 "+
                                  "cellpadding=8 bgcolor='#8080f0'>";
    Text WriteFile(htmFil, htmOpn);

    Text srcSep = "\n"+Repeat("/", 78)+"\n\n"; // Separador de arrays
    Text srcVar = GetFilePrefix(srcFil);       // Nombre fichero -> variable
    Text srcOpn = srcSep +
                  "var " + srcVar + " = new Array(\n"; // Array Javascript
    Text WriteFile(srcFil, srcOpn);
     ""       // No hay inicio en la apertura Javascript
  }, ",\n"); // En ciclo javascript cierra la linea anterior del array

  If(trcCtr <= 1, // Para 0 y 1 hay un estado que escribir
  {
    Text areBdy = If(EQ(Card(linSet),0), "EMPTY\n", // Para todas las salidas
                     SetSum(EvalSet(linSet, Text(Text lin) { lin+"\n" })));

                                // Eliminar < y > para Html y Javascript
    Text areRep = ReplaceTable(areBdy, [[ [["<","&lt;"]],[[">","&gt;"]] ]]);

    Text areTit = "State = "+FormatReal(numCel,"%.0lf")+"\n";

    Text srcBdy = "  " + Char(34) + Replace(areRep, "\n", "~") + Char(34);

    Text AppendFile(srcFil, srcIni + srcBdy); // Javascript

    Text htmIni =
      If(EQ(numCel%numCol, 1), "<tr>\n", "") + // Por si es la uno de la fila
```

```
      "<td bgcolor='#ffffff'>\n"+
      "<code><pre>\n";

    Text htmEnd =
      "</code></pre>\n"+
      "</td>\n"+
      If(EQ(numCel%numCol, 0), "</tr>\n", ""); // Por si es la ultima de fila

    Text AppendFile(htmFil, htmIni + areTit + areRep + htmEnd); // Html

    Text panSep = Repeat("_", 78)+"\n";   // Separador para pantalla

    Text Write(panSep + areTit + areBdy); // Por pantalla

    TRUE // Seguimos en ciclo
  },
  {
    // 2 o mayor es el fin, no hay estado que escribir, solo se cierra
    Text AppendFile(htmFil, "\n</tr>\n</table>\n"); // Fin estructura Html
    Text AppendFile(srcFil, "\n);\n");              // Fin del array

    // La pagina Html, htmFil, que se ha construido, solo contiene la tabla
    // de estados, ahora se hace una pagina completa con header, body, etc.
    Text htmTab = ReadFile(htmFil); // Lee fichero terminado, tabla completa
    Text htmSee = ReadFile(GetFilePath(htmFil) + // Ruta del fichero con /
                           "/_seed.htm");        // Semilla Html con body

    Text filPre = GetFilePrefix(htmFil);         // Prefijo del fichero
    Text htmTit = FirstToUpper(Replace(filPre, "_", " ")); // Titulo pagina

    Text filOut = Replace(htmFil, ".htm", "_html.htm");

    Text htmRep = ReplaceTable(htmSee, [[ [["_TIT_", htmTit]],
                                          [["_TAB_", htmTab]] ]]);
    Text WriteFile(filOut, htmRep);
    FALSE
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Visualiza y guarda la traza de de un estado de 3 formas en lenguaje Html, en
Javascript y por pantalla.
Dependiendo del parametro trcCtr, si 0 abre los 2 ficheros Html y Javascript,
si 1 asume que esta en medio ciclo y si 2 cierra ficheros de forma logica,
esto es pone los cierres de las estructuras de programacion de Html y
Javascript.
En Javascript, cada rectangulo de texto se pone como una sola linea de texto
entre comillas, con un separador donde termina realmente cada linea original
del rectangulo, luego este separador se por un br, salto de linea Html.
El caracter separador se ha de elegir de tal forma que pueda reemplazarse
sin probrmas, por ejemplo, el | suele dar problemas pues se usa dentro de
las figuras de las areas, el ; da problemas porque es el fin de los caracteres
especiales Html como   o &gt; o &lt;.
En este caso se ha elegido la virgulilla ~.",
AreTrc);
////////////////////////////////////////////////////////////////////////////
```

rul.tol de ChRules.Iterative

Funciones para reglas de caracteres, rules.

Declaraciones

Funciones

- `Set RulGet(Text rulTxt)`

  Retorna un cojunto de pares (oldSet, newSet) a partir de un texto que describe reglas de la forma: xxx :- yyy , primera linea de la primera regla terminada en , xxx :- yyy , segunda linea de la primera regla terminada en , xxx :- yyy ; ultima linea de la primera regla terminada en ; ... zzz :- yyy , zzz :- yyy , zzz :- yyy ; El simbolo < :- > de separacion de condicion accion, el < ,> de terminacion de linea y el simbolo < ;> de terminacion de regla son simbolos reservados y no pueden aparecer en el interior de los textos.

- `Set RulRnd(Set rulSet, Real rndRat)`

  Dependiendo de rndRat: - Si 0 las reglas en el mismo orden que recibe. - Si 1 las reglas siempre desordenadas. - Si p entre 0 y 1 entonces p veces desordenadas y 1-p veces ordenadas, esto es, - cuanto mayor es p mas al azar se comporta, - cuanto menor es p mas cercano al determinismo se comporta. Esto permite al motor de reglas funcionar: a) de forma totalmente derterminista, b) de forma totalmente aleatoria o c) de forma parcialmente aleatoria. Notese que algunos problemas no pueden resolverse si se barajan las reglas mientras que otros si. Aunque solo con la 3 parte del Case() se puede programar se ha preferido dejar claros los 2 casos extremos de 0 y de 1.

Set RulGet()

```
////////////////////////////////////////////////////////////////////////////
Set RulGet(Text rulTxt) // Regla en texto
////////////////////////////////////////////////////////////////////////////
{
  // Tokenizer() separa por un solo caracter, getTok() lo hace por varios pero
  // asume que el caracter de control Char(1) no aparece en el texto.
  Set getTok(Text t, Text s) { Tokenizer(Replace(t, s, Char(1)), Char(1)) };

  Set  tok001 = getTok(rulTxt," ;"); // Separar las reglas
  // Quedarse con las reglas no vacias
  Set  tok002 = Select(tok001, Real(Text rul) { Compact(rul)!="" });

  // Generar el conjunto de reglas
  Set EvalSet(tok002, Set(Text rul)
  {
    Set linSet = getTok(rul," ,"); // Separar cada linea
    // Separara la parte de la accion de la de la condicion
    Set parSet = EvalSet(linSet, Set(Text lin) { getTok(lin," :- ") });

    // La primera columna son condiciones, la segunda columna acciones, se
    // traspone para facilitar el acceso por filas
    Set traSet = Traspose(parSet);
    Set conSet = traSet[1]; // Los primeros son condiciones
    Set accSet = traSet[2]; // Los segundos son condiciones
    [[ conSet, accSet ]]
  })
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna un cojunto de pares (oldSet, newSet) a partir de un texto que
describe reglas de la forma:
  xxx :- yyy ,      primera linea de la primera regla terminada en ,
```

```
  xxx :- yyy ,      segunda linea de la primera regla terminada en ,
  xxx :- yyy ;      ultima  linea de la primera regla terminada en ;
  ...
  zzz :- yyy ,
  zzz :- yyy ,
  zzz :- yyy ;
El simbolo < :- > de separacion de condicion accion, el < ,> de terminacion
de linea y el simbolo < ;> de terminacion de regla son simbolos reservados
y no pueden aparecer en el interior de los textos.",
RulGet);
////////////////////////////////////////////////////////////////////////////
```

Set RulRnd()

```
////////////////////////////////////////////////////////////////////////////
Set RulRnd(Set  rulSet, // Conjunto de reglas
           Real rndRat) // Desorden total, parcial u orden de las reglas
////////////////////////////////////////////////////////////////////////////
{
  Case(
    rndRat <= 0, rulSet,              // Se conserva el orden
    rndRat >= 1, SetShuffle(rulSet), // Se desordenar siempre
    TRUE,        If(Rand(0,1) <= rndRat, SetShuffle(rulSet), rulSet))
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Dependiendo de rndRat:
- Si 0 las reglas en el mismo orden que recibe.
- Si 1 las reglas siempre desordenadas.
- Si p entre 0 y 1 entonces p veces desordenadas y 1-p veces ordenadas,
  esto es,
  - cuanto mayor es p mas al azar se comporta,
  - cuanto menor es p mas cercano al determinismo se comporta.
Esto permite al motor de reglas funcionar:
a) de forma totalmente derterminista,
b) de forma totalmente aleatoria o
c) de forma parcialmente aleatoria.
Notese que algunos problemas no pueden resolverse si se barajan las reglas
mientras que otros si.
Aunque solo con la 3 parte del Case() se puede programar se ha preferido
dejar claros los 2 casos extremos de 0 y de 1.",
RulRnd);
////////////////////////////////////////////////////////////////////////////
```

eng.tol de ChRules.Iterative

Funciones para el motor de reglas de caracteres, engine.

Declaraciones

Funciones

- `Set EngSol(Set linSet, Set rulSet, Real maxSta, Text preFil, Real numCol, Real rndCtr)`

  Retorna el resultado de aplicar el conjunto de reglas rulSet al area linSet hasta que no haya reglas aplicables o se alcance el numero maximo de estados (ciclos) maxSta. Cada vez que haya un cambio de estado lo visualiza y los va escribiendo en el fichero htmFil, la traza se escribe como una tabla html de numCol columnas.

- `Real EngJavascript(Text dirSol, Text dirSrc)`

  Reune todos los casos realizados y los prepara para que su resolucion pueda ser simulada por un simulador visual Javascript. Genera 2 ficheros en lenguaje Javascript: a) un fichero indice con un array en forma de tabña que contiene todos los casos resueltos, cada fila es (array de pasos, comentario), el nombre del array de pasos es el nombre del fichero sin extension y el comentario es igual cambiando el _ por blanco y b) un fichero mas grande, con tantos arrays como casos resueltos, cada uno de estos arrays contiene todos los pasos de resolucion del caso. El nombre de los arrays esta formado por el nombre de los ficheros. Para la generacion de estos 2 ficheros emplea 2 ficheros semilla que tienen la cabecera de comentarios de los programas Javascript. Recibe como parametros: a) el directorio de casos resueltos de entrada y b) el directorio de salida donde se escribira el codigo Javascript, asume que los ficheros semilla Javascript estan en este mismo directorio y los nombres de los ficheros para el array indice y el banco de datos son fijos. Retorna el numero de casos resueltos.

Set EngSol()

```
////////////////////////////////////////////////////////////////////////
Set EngSol(Set linSet,  // Area a resolver
           Set rulSet,  // Conjunto de reglas
           Real maxSta, // Numero maximo de iteraciones (estados)
           Text preFil, // Ruta y nombre del fichero sin extension
           Real numCol, // Numero de columnas de la tabla de estados
           Real rndCtr) // 0, determinista, 1 aleatorio y >0 <1 pseudoaletorio
////////////////////////////////////////////////////////////////////////
{
  Real AreTrc(linSet, 0, preFil, numCol, 0); // estado 0, abre los ficheros

  Set  staMem = Copy(linSet); // Memoria de estado
  Real staNum = 1;            // Numero de estado (el inical es el 0)
  Real end    = Copy(FALSE);

  Real While(Not(end),
  {
    Set new = AreCic(staMem, RulRnd(rulSet,rndCtr));
    If(EQ(Card(new),0), Real(end:=TRUE), // Termino con exito
    {
      // Se ha alcanzado un nuevo estado
      Real AreTrc(new, staNum, preFil, numCol, 1); // Nuevo estado, en ciclo
      Set (staMem:=Copy(new)); // Memorizar el nuevo estado
      Real(staNum:=staNum+1);  // Probar con el siguiente estado
      If(GT(staNum,maxSta), Real(end:=TRUE), // Termino sin exito
                            FALSE)           // Cicla
```

```
    })
  });
  Real AreTrc(Empty, 0, preFil, numCol, 2); // Cierra, fin del ciclo

  staMem
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Retorna el resultado de aplicar el conjunto de reglas rulSet al area linSet
hasta que no haya reglas aplicables o se alcance el numero maximo de estados
(ciclos) maxSta.
Cada vez que haya un cambio de estado lo visualiza y los va escribiendo en el
fichero htmFil, la traza se escribe como una tabla html de numCol columnas.",
EngSol);
////////////////////////////////////////////////////////////////////////////
```

Real EngJavascript()

```
////////////////////////////////////////////////////////////////////////////
Real EngJavascript(Text dirSol, // Directorio de casos resueltos
                   Text dirSrc) // Directorio de Javascript
////////////////////////////////////////////////////////////////////////////
{
  Set  filSet = DirExtAll(dirSol, "js", FALSE, TRUE); // Todos los ficheros
  Text filCod = DirReadFiles(dirSol, "js", "\n");     // Todo los scripts

  // Contruir el array de casos, indice
  Set  arrSet = For(1, Card(filSet), Text(Real filPos) // Para los casos
  {
    Text arrNam = GetFilePrefix(filSet[filPos]);
    Text arrLbl = Replace(arrNam, "_", " ");
    Text arrNew = "new Array(" + arrNam + ", '" + arrLbl + "')";
    Text arrEnd = If(Card(filSet)==filPos, "", ",\n"); // El ultimo diferente
    "  "+arrNew+arrEnd
  });
  Text arrTxt = SetSum(arrSet); // Une todos los textos
  Text arrSed = dirSrc+"/simulatorarray.sed"; // Fichero semilla
  Text arrPth = Replace(arrSed,".sed",".js"); // Fichero javascript
  Text WriteFile(arrPth, Replace(ReadFile(arrSed), "_ARR_", arrTxt));

  // Construir el banco de datos de pasos, steps, de los casos
  Text sdbSed = dirSrc+"/simulatordb.sed";
  Text sdbPth = Replace(sdbSed,".sed",".js");
  Text WriteFile(sdbPth, Replace(ReadFile(sdbSed), "_SDB_", filCod));

  Card(filSet) // Numero de casos
};
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Reune todos los casos realizados y los prepara para que su resolucion pueda
ser simulada por un simulador visual Javascript.
Genera 2 ficheros en lenguaje Javascript:
a) un fichero indice con un array en forma de tabña que contiene todos los
   casos resueltos, cada fila es (array de pasos, comentario),
   el nombre del array de pasos es el nombre del fichero sin extension y
   el comentario es igual cambiando el _ por blanco y
b) un fichero mas grande, con tantos arrays como casos resueltos,
   cada uno de estos arrays contiene todos los pasos de resolucion del caso.
El nombre de los arrays esta formado por el nombre de los ficheros.
Para la generacion de estos 2 ficheros emplea 2 ficheros semilla que tienen
la cabecera de comentarios de los programas Javascript.
Recibe como parametros:
a) el directorio de casos resueltos de entrada y
b) el directorio de salida donde se escribira el codigo Javascript,
   asume que los ficheros semilla Javascript estan en este mismo directorio y
   los nombres de los ficheros para el array indice y
   el banco de datos son fijos.
Retorna el numero de casos resueltos.",
EngJavascript);
////////////////////////////////////////////////////////////////////////////
```

bub.tol de ChRules.Iterative

Base de reglas para ordenar parcialmente con el metodo de la burbuja, son lentas y parcialmente eficaces.

Declaraciones

Constantes

- `Set BubSed`
  Mezcla de elementos para sedimentar por peso.
- `Set BubUp1`
  Reglas que realizan una semiordenacion por peso un conjunto de 4 elementos, que simula un proceso de sedimentacion.
- `Set BubLt1`
  Reglas de asentamiento lateral, no del todo eficaces.

Constantes

Set BubSed

```
////////////////////////////////////////////////////////////////////////////
Set BubSed =
[["Oo·OoºoOº·Ooº·º·",
  "oOoºº·O·o·Ooºº·O",
  "ºOo·Oº·Ooºº·º·Oo",
  "·ºOoº·OoOoº·ºOo·",
  "·º·oOoºOoºOoO·º·",
  "Ooº·º·Ooº·oOº·Oo",
  "ºoOo·ºOoºO·oº··O",
  "·º·oOoºOoºOo·Oº·",
  "oOoºº·O·o·Ooºº·O",
  "ºOo·Oº·Ooºº·º·Oo",
  "·Oo·Ooº·ºOo·ºOoº",
  "·º·oOoºOoºOoO·º·",
  "Ooº·º·Ooº·oOº·Oo",
  "ºoOo·ºOoºO·oº··O",
  "·º·oOoºOoºOo·Oº·",
  "º·O·o·Ooºoooºº·O",
  "oOoº·º·Ooº·º·OoO"]];
////////////////////////////////////////////////////////////////////////////
PutDescription("Mezcla de elementos para sedimentar por peso.",BubSed);
////////////////////////////////////////////////////////////////////////////
```

Set BubUp1

```
////////////////////////////////////////////////////////////////////////////
Set BubUp1 = RulGet(
   "º :- · ,"   + // Agua · sube
   ". :- º ;"   +
   "o :- · ,"   +
   "· :- o ;"   +
   "O :- · ,"   +
   "· :- O ;"   +

   "o :- º ,"   + // Arena º sube, pero bajo el agua
```

```
  "º :- o ;"   +
  "O :- º ;"   +
  "º :- O ;"   +

  "O :- o ,"   + // Piedrilla o solo supera al pedrusco O
  "o :- O ;");

////////////////////////////////////////////////////////////////////////////
PutDescription(
"Reglas que realizan una semiordenacion por peso un conjunto de 4 elementos,
que simula un proceso de sedimentacion.",
BubUp1);
////////////////////////////////////////////////////////////////////////////
```

Set BubLt1

```
////////////////////////////////////////////////////////////////////////////
Set BubLt1 = RulGet(
  "º. :- .º ;"  + // Asentamiento lateral al final para no hacer diagonales
  "o. :- .o ;"  +
  "O. :- .O ;"  +
  "oº :- ºo ;"  +
  "Oº :- ºO ;"  +
  "Oo :- oO ;");

////////////////////////////////////////////////////////////////////////////
PutDescription("Reglas de asentamiento lateral, no del todo eficaces.",
BubLt1);
////////////////////////////////////////////////////////////////////////////
```

cel.tol de ChRules.Iterative

Racipiente (pantano o area, base de hechos) y reglas de la vida (base de reglas con reglas de escritura) para un automata celular.

Declaraciones

Constantes

- `Set CelP31`
  Recipiente para el automatar celular.
- `Set CelRb1`
  Reglas de la vida para un automata celular con seres adultos, huevos, formas de reproduccion y de muerte.
- `Set CelRb2`
  Reglas de la vida para un automata celular con seres adultos (O), con huevos (o), formas de reproduccion y de muerte en 2 fases pasando por una fase de cadaver (+).

Constantes

Set CelP31

```
////////////////////////////////////////////////////////////////////////////
Set CelP31 =
[["----********************----",
  "---*....................*---",
  "--*......................*--",
  "-*..........O............*-",
  "*..........O···O..........*",
  "*...........O............*",
  "*......O···O···O···O.......*",
  "*....O···O···O···O···O.....*",
  "*......O···O···O···O.......*",
  "*...........O............*",
  "*..........O···O..........*",
  "-*..........O............*-",
  "--*......................*--",
  "---*....................*---",
  "----********************----"]];
////////////////////////////////////////////////////////////////////////////
PutDescription("Recipiente para el automatar celular.", CelP31);
////////////////////////////////////////////////////////////////////////////
```

Set CelRb1

```
////////////////////////////////////////////////////////////////////////////
Set CelRb1 = RulGet(
    "O·O :- ·O· ,"   +
    "··· :- O·O ;"   + // Huevo por reproduccion hacia abajo con movimiento

    "··· :- O·O ,"   +
    "O·O :- ·O· ;"   + // Huevo por reproduccion hacia arriba con movimiento

     "O· :- ·O ,"    +
     "·· :- O· ,"    +
     "O· :- ·O ;"    + // Huevo por reproduccion lateral con movimiento
```

```
      "·O :- O· ,"    +
      "·· :- ·O ,"    +
      "·O :- O· ;"    + // Huevo por reproduccion lateral con movimiento

      "OO :- ·O ,"    +
      "O· :- O· ;"    + // Muerte por exceso de poblacion, muera la apretada

      "OO :- O· ,"    +
      "·O :- ·O ;"    + // Muerte por exceso de poblacion, muera la apretada

      "·O :- ·O ,"    +
      "OO :- O· ;"    + // Muerte por exceso de poblacion, muera la apretada

      "O· :- O· ,"    +
      "OO :- ·O ;"    + // Muerte por exceso de poblacion, muera la apretada

       "O :- O ,"     +
       "O :- · ,"     +
       "O :- O ;"     + // Muerte por exceso de poblacion, muera la apretada

     "OOO :- O·O ;"   + // Muerte por exceso de poblacion, muera la apretada

      "O* :- ·* ;"    + // Muerte al irse tocar la membrana exterior

      "*O :- *· ;"    + // Muerte al irse tocar la membrana exterior

       "O :- · ,"     +
       "* :- * ;"     + // Muerte al irse tocar la membrana exterior

       "* :- * ,"     +
       "O :- · ;"     + // Muerte al irse tocar la membrana exterior

     "··· :- ··· ,"   +
     "·O· :- ··· ,"   +
     "··· :- ··· ;");   // Muerte por aislamiento
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Reglas de la vida para un automata celular con seres adultos, huevos, formas
de reproduccion y de muerte.",
CelRb1);
////////////////////////////////////////////////////////////////////////////
```

Set CelRb2

```
////////////////////////////////////////////////////////////////////////////
Set CelRb2 = RulGet(
     "O·O :- ·O· ,"   +
     "··· :- O·O ;"   + // Huevo por reproduccion hacia abajo con movimiento

     "··· :- O·O ,"   +
     "O·O :- ·O· ;"   + // Huevo por reproduccion hacia arriba con movimiento

      "O· :- ·O ,"    +
      "·· :- O· ,"    +
      "O· :- ·O ;"    + // Huevo por reproduccion lateral con movimiento

      "·O :- O· ,"    +
      "·· :- ·O ,"    +
      "·O :- O· ;"    + // Huevo por reproduccion lateral con movimiento

      "OO :- +O ,"    +
      "O· :- O· ;"    + // Muerte por exceso de poblacion, muera la apretada

      "OO :- O+ ,"    +
      "·O :- ·O ;"    + // Muerte por exceso de poblacion, muera la apretada

      "·O :- ·O ,"    +
      "OO :- O+ ;"    + // Muerte por exceso de poblacion, muera la apretada

      "O· :- O· ,"    +
      "OO :- +O ;"    + // Muerte por exceso de poblacion, muera la apretada
```

```
      "O :- O ,"     +
      "O :- + ,"     +
      "O :- O ;"     + // Muerte por exceso de poblacion, muera la apretada

    "OOO :- O+O ;"   + // Muerte por exceso de poblacion, muera la apretada

     "O* :- +* ;"    + // Muerte al irse tocar la membrana exterior

     "*O :- *+ ;"    + // Muerte al irse tocar la membrana exterior

      "O :- + ,"     +
      "* :- * ;"     + // Muerte al irse tocar la membrana exterior

      "* :- * ,"     +
      "O :- + ;"     + // Muerte al irse tocar la membrana exterior

      "o :- O ;"     + // Eclosion del huevo del que sale un adulto

      "+ :- · ;"     + // Putrefaccion del cadaver que desaparece

    "... :- ... ,"   +
    "·O· :- ·+· ,"   +
    "... :- ... ;");   // Muerte por aislamiento
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Reglas de la vida para un automata celular con seres adultos (O),
con huevos (o), formas de reproduccion y de muerte en 2 fases pasando por una
fase de cadaver (+).",
CelRb2);
////////////////////////////////////////////////////////////////////////////
```

lab.tol de ChRules.Iterative

Base de reglas con reglas para que un robot salga de un laberinto y areas que son laberintos, no pueden funcionar de manera aleatoria.

Declaraciones

Constantes

- `Set LabA01`
  Laberinto pequeño con un solo robot.
- `Set LabA02`
  Laberinto pequeño con un solo robot.
- `Set LabA11`
  Laberinto grande con un solo robot.
- `Set LabOut`
  Reglas para encontrar la salida, son independientes del algoritmo que se emplee, si ve la salida sale. En el caso de la salida, los algoritmos de la mano derecha o izquierda son un poco torpes, pues: - El de la derecha si pasa por la salida, pero no esta a su derecha la ignora y puede dar una larga vuelta hasta que consigue pasar por la salida quedando esta a su derecha. - El de la izquierda si pasa por la salida, pero no esta a su izquierda la ignora y puede dar una larga vuelta hasta que consigue pasar por la salida quedando esta a su izquierda.
- `Set LabRgh`
  Algoritmo de la mano de derecha, para salir de laberintos, codificado como reglas de reescritura de areas rectangulares de caracteres. Es como si el robot llevara su mano derecha pegada siempre a un pared y se prohibiera a si mismo separarla de la pared. Notese que hay casos en los que no es capaz de sacar al robot de un recinto cerrado aun cuando el robot este fuera, pero su mano derecha pegada al exterior del recinto.
- `Set LabOutRgh`
  Reglas para tomar la salida y algoritmo de la mano derecha para encontrar dicha salida salida.
- `Set LabLft`
  Algoritmo de la mano de izquierda, para salir de laberintos, codificado como reglas de reescritura de areas rectangulares de caracteres. Es como si el robot llevara su mano izquierda pegada siempre a un pared y se prohibiera a si mismo separarla de la pared. Notese que hay casos en los que no es capaz de sacar al robot de un recinto cerrado aun cuando el robot este fuera, pero su mano izquierda pegada al exterior del recinto. Es como el algoritmo de la mano derecha pero con las reglas impares intercambiadas de 2 en 2, la 1 y la 3, la 5 y la 7, etc. Las reglas estan numeradas con la misma numeracion que las de la derecha para que se observe este efecto.
- `Set LabOutLft`
  Reglas para tomar la salida y algoritmo de la mano izquierda para encontrar dicha salida salida.

- `Set LabNeg`

  Ampliacion de la base de reglas para que cuando 2 robots intentan salir del mismo laberinto y se enfrentan el uno al otro, se dejen pasar mutuamente, negocial su solucion al conflicto. Otra forma de verlo es que cambian de direccion, pues el cuerpo de los robots al ser identico no permite distinguirlos, en cualquier caso, no se quedan bloqueados. Son solo 2 reglas a las que se añaden todas las anteriores mediante contactenacion de conjuntos. Estas 2 reglas solventan los 2 conflictos mas basicos, pero no todos.

- `Set LabNegOutRgh`

  Reglas para tomar la salida y algoritmo de la mano derecha para encontrar dicha salida salida.

- `Set LabNegOutLft`

  Reglas para tomar la salida y algoritmo de la mano izquierda para encontrar dicha salida salida.

Constantes

Set LabA01

```
////////////////////////////////////////////////////////////////////////////
Set LabA01 =
[["+V--+---------+-+",
  "|...|...|.....|.|",
  "|.|.|.|.|.+-+...|",
  "|.|.|.|.|.|.|.|.|",
  "|.|.|.|.|.|.|.|.|",
  "|.|.|...|...|.|.|",
  "|.+-+.|.+--.+-+.|",
  "|.....|.......|.|",
  "+-----+-------+S+"]];
////////////////////////////////////////////////////////////////////////////
PutDescription("Laberinto pequeño con un solo robot.",LabA01);
////////////////////////////////////////////////////////////////////////////
```

Set LabA02

```
////////////////////////////////////////////////////////////////////////////
Set LabA02 =
[["+V--+V--+-----+-+",
  "|...|...|.....|.|",
  "|.|.|.|.|.+-+...|",
  "|.|.|.|.|.|.|.|.|",
  "|.|.|.|.|.|.|.|.|",
  "|.|.|...|...|.|.|",
  "|.+-+.|.+--.+-+.|",
  "|.....|.......|.|",
  "+-----+-------+S+"]];
////////////////////////////////////////////////////////////////////////////
PutDescription("Laberinto pequeño con un solo robot.",LabA02);
////////////////////////////////////////////////////////////////////////////
```

Set LabA11

```
////////////////////////////////////////////////////////////////////////////
Set LabA11 =
[["+V--+---+-----+-+---+---+-----+",
  "|...|...|.....................|",
```

```
   "|·|·|·|·|·+-+·+-+·|·+-+·|·+-+·|",
   "|·|·|·|·|·|·|·|·|·|·|·|·|·|·|·|",
   "|·|·|·|·|·|·|·|·|·|·|·|·|·|·|·|",
   "|·|·|·|·|···|·|·|·|···|·|···|·|",
   "|·+-+·|·+--·+-+·|·+--|·|·+---+·|",
   "|···········|···|·····|·|·····|·|",
   "|·|·+-+-+---+-+·+-----+·|·+-+·|·|",
   "|·|·|···|·····|·|·····|·|·|·|·|·|",
   "|·|·|·|·|·+-+···|·|·+-+-+-+·|·|·|",
   "|·|·|·|·|·|·|·|·|·|·|·|···|·|·|·|",
   "|·|·|···|·|·|·|···|·|·|·|·|·|·|·|",
   "|···|·|·|···|·|·|·|·|···|···|·|·|",
   "+---+·|·+--·+-+·|·+-+·|·|·+---+·|",
   "|·····|·····|·|·····|·|·····|·|·|",
   "+S----+-----+-+-----+-----+"]];
////////////////////////////////////////////////////////////////////////////
PutDescription("Laberinto grande con un solo robot.", LabA11);
////////////////////////////////////////////////////////////////////////////
```

Set LabOut

```
////////////////////////////////////////////////////////////////////////////
Set LabOut = RulGet(
   "Sv :- S· ;" + // 01) Encontro la salida a la derecha

    "v :- · ,"   + // 02) Encontro la salida al frente
    "S :- S ;"   +

  "vS :- ·S ;"  + // 03) Encontro la salida a la izquierda

    "S :- S ,"   + // 04) Encontro la salida a su espalda
    "v :- · ;"   +
                   // ----------

  "^S :- ·S ;"  + // 05) Encontro la salida a la derecha

    "S :- S ,"   + // 06) Encontro la salida al frente
    "^ :- · ;"   +

  "S^ :- S· ;"  + // 07) Encontro la salida a la izquierda

    "^ :- · ,"   + // 08) Encontro la salida a su espalda
    "S :- S ;"   +
                   // ----------

    "> :- · ,"   + // 09) Encontro la salida a la derecha
    "S :- S ;"   +

  ">S :- ·S ;"  + // 10) Encontro la salida al frente

    "S :- S ,"   + // 11) Encontro la salida a la izquierda
    "> :- · ;"   +

  "S> :- S· ;"  + // 12) Encontro la salida a su espalda

                   // ----------

    "S :- S ,"   + // 13) Encontro la salida a su derecha
    "< :- · ;"   +

  "S< :- S· ;"  + // 14) Encontro la salida al frente

    "< :- · ,"   + // 15) Encontro la salida a su izquierda
    "S :- S ;"   +

  "<S :- ·S ;"); // 16) Encontro la salida a su espalda
////////////////////////////////////////////////////////////////////////////
```

```
PutDescription(
"Reglas para encontrar la salida, son independientes del algoritmo que se
emplee, si ve la salida sale.
En el caso de la salida, los algoritmos de la mano derecha o izquierda son
un poco torpes, pues:
- El de la derecha si pasa por la salida, pero no esta a su derecha la
  ignora y puede dar una larga vuelta hasta que consigue pasar por la salida
  quedando esta a su derecha.
- El de la izquierda si pasa por la salida, pero no esta a su izquierda la
  ignora y puede dar una larga vuelta hasta que consigue pasar por la salida
  quedando esta a su izquierda.",
LabOut);
////////////////////////////////////////////////////////////////////////////
```

Set LabRgh

```
////////////////////////////////////////////////////////////////////////////
Set LabRgh = RulGet(
  "·v :- <· ;"  + // 01) Encontro un paso a la derecha

   "v :- · ,"   + // 02) Encontro un paso al frente
   "· :- v ;"   +

  "v· :- ·> ;"  + // 03) Encontro un paso a la izquierda

   "· :- ^ ,"   + // 04) Encontro un paso a su espalda
   "v :- · ;"   +
                  // ----------
  "^· :- ·> ;"  + // 05) Encontro un paso a la derecha

   "· :- ^ ,"   + // 06) Encontro un paso al frente
   "^ :- · ;"   +

  "·^ :- <· ;"  + // 07) Encontro un paso a la izquierda

   "^ :- · ,"   + // 08) Encontro un paso a su espalda
   "· :- v ;"   +
                  // ----------
   "> :- · ,"   + // 09) Encontro un paso a la derecha
   "· :- v ;"   +

  ">· :- ·> ;"  + // 10) Encontro un paso al frente

   "· :- ^ ,"   + // 11) Encontro un paso a su izquierda
   "> :- · ;"   +

  "·> :- <· ;"  + // 12) Encontro un paso a su espalda

                  // ----------

   "· :- ^ ,"   + // 13) Encontro un paso a la derecha
   "< :- · ;"   +

  "·< :- <· ;"  + // 14) Encontro un paso al frente

   "< :- · ,"   + // 15) Encontro un paso a su izquierda
   "· :- v ;"   +

  "<· :- ·> ;");  // 16) Encontro un paso a su espalda
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Algoritmo de la mano de derecha, para salir de laberintos, codificado como
reglas de reescritura de areas rectangulares de caracteres.
Es como si el robot llevara su mano derecha pegada siempre a un pared y se
prohibiera a si mismo separarla de la pared.
Notese que hay casos en los que no es capaz de sacar al robot de un recinto
cerrado aun cuando el robot este fuera, pero su mano derecha
pegada al exterior del recinto.",
LabRgh);
////////////////////////////////////////////////////////////////////////////
```

Set LabOutRgh

```
////////////////////////////////////////////////////////////////////////////
Set LabOutRgh = LabOut << LabRgh;
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Reglas para tomar la salida y algoritmo de la mano derecha para encontrar
dicha salida salida.",
LabOutRgh);
////////////////////////////////////////////////////////////////////////////
```

Set LabLft

```
////////////////////////////////////////////////////////////////////////////
Set LabLft = RulGet(
  "v· :- ·> ;"  + // 03) Encontro un paso a la izquierda

   "v :- · ,"   + // 02) Encontro un paso al frente
   "· :- v ;"   +

  "·v :- <· ;"  + // 01) Encontro un paso a la derecha

   "· :- ^ ,"   + // 04) Encontro un paso a su espalda
   "v :- · ;"   +
                  // ----------
  "·^ :- <· ;"  + // 07) Encontro un paso a la izquierda

   "· :- ^ ,"   + // 06) Encontro un paso al frente
   "^ :- · ;"   +

  "^· :- ·> ;"  + // 05) Encontro un paso a la derecha

   "^ :- · ,"   + // 08) Encontro un paso a su espalda
   "· :- v ;"   +
                  // ----------
   "· :- ^ ,"   + // 11) Encontro un paso a su izquierda
   "> :- · ;"   +

  ">· :- ·> ;"  + // 10) Encontro un paso al frente

   "> :- · ,"   + // 09) Encontro un paso a la derecha
   "· :- v ;"   +

  "·> :- <· ;"  + // 12) Encontro un paso a su espalda

                  // ----------

   "< :- · ,"   + // 15) Encontro un paso a su izquierda

   "· :- v ;"   +
  "·< :- <· ;"  + // 14) Encontro un paso al frente

   "· :- ^ ,"   + // 13) Encontro un paso a la derecha
   "< :- · ;"   +

  "<· :- ·> ;"); // 16) Encontro un paso a su espalda
////////////////////////////////////////////////////////////////////////////
```

```
PutDescription(
"Algoritmo de la mano de izquierda, para salir de laberintos, codificado como
reglas de reescritura de areas rectangulares de caracteres.
Es como si el robot llevara su mano izquierda pegada siempre a un pared y se
prohibiera a si mismo separarla de la pared.
Notese que hay casos en los que no es capaz de sacar al robot de un recinto
cerrado aun cuando el robot este fuera, pero su mano izquierda
pegada al exterior del recinto.
Es como el algoritmo de la mano derecha pero con las reglas impares
intercambiadas de 2 en 2, la 1 y la 3, la 5 y la 7, etc.
Las reglas estan numeradas con la misma numeracion que las de la derecha
para que se observe este efecto.",
LabLft);
////////////////////////////////////////////////////////////////////////////
```

Set LabOutLft

```
////////////////////////////////////////////////////////////////////////////
Set LabOutLft = LabOut << LabLft;
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Reglas para tomar la salida y algoritmo de la mano izquierda para encontrar
dicha salida salida.",
LabOutLft);
////////////////////////////////////////////////////////////////////////////
```

Set LabNeg

```
////////////////////////////////////////////////////////////////////////////
Set LabNeg = RulGet("><  :-  <>  ;"  + // 01) Enfrentados -> cambio de direccion

                    "v  :-  ^  ,"   + // 02) Enfrentados -> cambio de direccion
                    "^  :-  v  ;");
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Ampliacion de la base de reglas para que cuando 2 robots intentan salir del
mismo laberinto y se enfrentan el uno al otro, se dejen pasar mutuamente,
negocial su solucion al conflicto.
Otra forma de verlo es que cambian de direccion, pues el cuerpo de los
robots al ser identico no permite distinguirlos, en cualquier caso,
no se quedan bloqueados.
Son solo 2 reglas a las que se añaden todas las anteriores mediante
contactenacion de conjuntos.
Estas 2 reglas solventan los 2 conflictos mas basicos, pero no todos.",
LabNeg);
////////////////////////////////////////////////////////////////////////////
```

Set LabNegOutRgh

```
////////////////////////////////////////////////////////////////////////////
Set LabNegOutRgh = LabNeg << LabOutRgh;
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Reglas para tomar la salida y algoritmo de la mano derecha para encontrar
dicha salida salida.",
LabNegOutRgh);
////////////////////////////////////////////////////////////////////////////
```

Set LabNegOutLft

```
////////////////////////////////////////////////////////////////////////////
Set LabNegOutLft = LabNeg << LabOutLft;
```

```
////////////////////////////////////////////////////////////////////////
PutDescription(
"Reglas para tomar la salida y algoritmo de la mano izquierda para encontrar
dicha salida salida.",
LabNegOutLft);
////////////////////////////////////////////////////////////////////////
```

fll.tol de ChRules.Iterative

Base de reglas con reglas para rellenar, fill, superficies cerradas, pueden funcionar de manera aleatoria.

Declaraciones

Constantes

- `Set FllBox`
  Superficie cerrada con una K mayuscula.
- `Set FllupK`
  Superficie cerrada con una K mayuscula.
- `Set Fllrb1`
  Base de reglas para rellenar superficies cerradas mediante reglas de reescritura de areas rectangulares de caracteres.

Constantes

Set FllBox

```
////////////////////////////////////////////////////////////////////////////
Set FllBox =
[["·····o·····o·····o·····",
  "·····o·····o·····o·····",
  "·····o·····o·····o·····",
  "·····o·····o·····o·····",
  "·····o·····o·····o·····",
  "·····o·····o·····o·····",
  "·····o·····o·····o·····",
  "·····o·····o·····o·····",
  "·····o·····o·····o·····",
  "·····o·····o·····o·····",
  "·····o·····o·····o·····",
  "·····o·····o·····o·····",
  "····························",
  "····························",
  "····························",
  "··············+·············" ]];
////////////////////////////////////////////////////////////////////////////
PutDescription("Superficie cerrada con una K mayuscula.",FllBox);
////////////////////////////////////////////////////////////////////////////
```

Set FllupK

```
////////////////////////////////////////////////////////////////////////////
Set FllupK =
[["··ooooooo·····ooooooooo·····",
  "··o·····o·····o·····o·····",
  "··o·····o·····o·····o·····",
  "··o·····o····o·····o·····",
  "··o·····o···o·····o·····",
  "··o·····o··o·····o·····",
  "··o·····ooo·····o·····",
  "··o·····o·····",
  "··o·····o·····",
  "··o·····o·····",
```

```
  "··o··········ooo·······o·······",
  "··o·········oo···o·······o·····",
  "··o·····oo·····o·····o·····o···",
  "··o·····o·····o·····o·····o····",
  "··o·····o·····o···+···o···",
  "··o·····o·····o·····o··",
  "··oooooooo·····oooooooo·" ]];
////////////////////////////////////////////////////////////////////////
PutDescription("Superficie cerrada con una K mayuscula.",FllupK);
////////////////////////////////////////////////////////////////////////
```

Set FllRb1

```
////////////////////////////////////////////////////////////////////////
Set FllRb1 = RulGet(
  "+· :- ++ ;" + // 01) Relleno derecha

  "·+ :- ++ ;" + // 02) Relleno izquierda

   "+ :- + ,"  + // 03) Relleno abajo
   "· :- + ;"  +

   "· :- + ,"  + // 04) Relleno arriba
   "+ :- + ;");
////////////////////////////////////////////////////////////////////////
PutDescription(
"Base de reglas para rellenar superficies cerradas mediante reglas de
reescritura de areas rectangulares de caracteres.",
FllRb1);
////////////////////////////////////////////////////////////////////////
```

pon.tol de ChRules.Iterative

Base de reglas con reglas de un pingpong electronico, pong, incluye el area de juego, pueden funcionar de manera aleatoria, creo.

Declaraciones

Constantes

- `Set PonUpA`
  Mesa de pingpong electronico, saque desde arriba.
- `Set PonDwA`
  Mesa de pingpong electronico, saque desde abajo.
- `Set PonRb1`
  Base de reglas de un ping pong electronico, pong, mediante reglas de reescritura de areas rectangulares de caracteres. Los 4 estados de la pelota se codifican mediante 4 caracteres.

Constantes

Set PonUpA

```
////////////////////////////////////////////////////////////////////////////
Set PonUpA =
[["+---------+",
  "|··=·····|",
  "|···b····|",
  "|·········|",
  "|·········|",
  "|·········|",
  "|·········|",
  "|·········|",
  "|·········|",
  "|·········|",
  "|·····=·|",
  "+---------+"]];
////////////////////////////////////////////////////////////////////////////
PutDescription("Mesa de pingpong electronico, saque desde arriba.",PonUpA);
////////////////////////////////////////////////////////////////////////////
```

Set PonDwA

```
////////////////////////////////////////////////////////////////////////////
Set PonDwA =
[["+---------+",
  "|····=····|",
  "|·········|",
  "|·········|",
  "|·········|",
  "|·········|",
  "|·········|",
  "|·········|",
  "|·········|",
  "|·····p···|",
  "|····=····|",
  "+---------+"]];
////////////////////////////////////////////////////////////////////////////
```

```
PutDescription("Mesa de pingpong electronico, saque desde abajo.",PonDwA);
////////////////////////////////////////////////////////////////////////////
```

Set PonRb1

```
////////////////////////////////////////////////////////////////////////////
Set PonRb1 = RulGet(
    "b·· :- ··p ,"  + // Rebota en la pala inferior
    "·=· :- ·=· ;"  +

    "b·| :- ·p| ,"  + // Rebota en la pala inferior pegado a la pared
    "·=| :- ·=| ;"  +

    "·=· :- ·=· ,"  + // Rebota en la pala superior
    "p·· :- ··b ;"  +

    "·=| :- ·=| ,"  + // Rebota en la pala superior pegado a la pared
    "p·| :- ·b| ;"  +

    "··d :- q·· ,"  + // Rebota en la pala inferior
    "·=· :- ·=· ;"  +

    "|·d :- |q· ,"  + // Rebota en la pala inferior pegado a la pared
    "|=· :- |=· ;"  +

    "·=· :- ·=· ,"  + // Rebota en la pala superior
    "··q :- d·· ;"  +

    "|=· :- |=· ,"  + // Rebota en la pala superior pegado a la pared
    "|·q :- |d· ;"  +

    "b·· :- b·· ,"  + // Intenta cojerlo
    "··= :- ·=· ;"  +

     "b· :- b· ,"   + // Intenta cojerlo
     "=· :- ·= ;"   +

     "b· :- b· ,"   + // Intenta cojerlo
     "·· :- ·· ,"   +
     "=· :- ·= ;"   +

     "b· :- b· ,"   + // Intenta cojerlo
     "·· :- ·· ,"   +
     "·· :- ·· ,"   +
     "=· :- ·= ;"   +

     "b· :- b· ,"   + // Intenta cojerlo
     "·· :- ·· ,"   +
     "·· :- ·· ,"   +
     "·· :- ·· ,"   +
     "=· :- ·= ;"   +

     "b· :- b· ,"   + // Intenta cojerlo
     "·· :- ·· ,"   +
     "·· :- ·· ,"   +
     "·· :- ·· ,"   +
     "·· :- ·· ,"   +
     "=· :- ·= ;"   +

    "··= :- ·=· ,"  + // Intenta cojerlo
    "p·· :- p·· ;"  +

     "=· :- ·= ,"   + // Intenta cojerlo
     "p· :- p· ;"   +

     "=· :- ·= ,"   + // Intenta cojerlo
     "·· :- ·· ,"   +
     "p· :- p· ;"   +

     "=· :- ·= ,"   + // Intenta cojerlo
     "·· :- ·· ,"   +
     "·· :- ·· ,"   +
```

```
  "p· :- p· ;"   +

  "=· :- ·= ,"   + // Intenta cojerlo
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "p· :- p· ;"   +

  "=· :- ·= ,"   + // Intenta cojerlo
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "p· :- p· ;"   +

 "··d :- ··d ,"  + // Intenta cojerlo
 "=·· :- ·=· ;"  +

  "·d :- ·d ,"   + // Intenta cojerlo
  "·= :- =· ;"   +

  "·d :- ·d ,"   + // Intenta cojerlo
  "·· :- ·· ,"   +
  "·= :- =· ;"   +

  "·d :- ·d ,"   + // Intenta cojerlo
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·= :- =· ;"   +

  "·d :- ·d ,"   + // Intenta cojerlo
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·= :- =· ;"   +

  "·d :- ·d ,"   + // Intenta cojerlo
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·= :- =· ;"   +

 "=·· :- ·=· ,"  + // Intenta cojerlo
 "··q :- ··q ;"  +

  "·= :- =· ,"   + // Intenta cojerlo
  "·q :- ·q ;"   +

  "·= :- =· ,"   + // Intenta cojerlo
  "·· :- ·· ,"   +
  "·q :- ·q ;"   +

  "·= :- =· ,"   + // Intenta cojerlo
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·q :- ·q ;"   +

  "·= :- =· ,"   + // Intenta cojerlo
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·q :- ·q ;"   +

  "·= :- =· ,"   + // Intenta cojerlo
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·· :- ·· ,"   +
  "·q :- ·q ;"   +

"b··· :- b··· ," + // Intenta cojerlo de mas lejos
"···· :- ···· ," +
"···= :- ··=· ;" +
```

```
  "···= :- ··=· ," + // Intenta cojerlo de mas lejos
  "···· :- ···· ," +
  "p··· :- p··· ;" +

  "···d :- ···d ," + // Intenta cojerlo de mas lejos
  "···· :- ···· ," +
  "=··· :- ·=·· ;" +

  "=··· :- ·=·· ," + // Intenta cojerlo de mas lejos
  "···· :- ···· ," +
  "···q :- ···q ;" +

    "b· :- ·· ,"   + // Avanza hacia la derecha bajando
    "·· :- ·b ;"   +

    "·d :- ·· ,"   + // Avanza hacia la izquierda bajando
    "·· :- d· ;"   +

    "·· :- q· ,"   + // Avanza hacia la izquierda subiendo
    "·q :- ·· ;"   +

    "·· :- ·p ,"   + // Avanza hacia la derecha subiendo
    "p· :- ·· ;"   +

    "b| :- d| ;"   + // Rebota en la pared derecha bajando

    "|d :- |b ;"   + // Rebota en la pared izquierda bajando

    "|q :- |p ;"   + // Rebota en la pared izquierda subiendo

    "p| :- q| ;"   + // Rebota en la pared derecha subiendo

     "- :- - ,"    + // Gol
     "p :- o ;"    +

     "- :- - ,"    + // Gol
     "q :- o ;"    +

     "b :- o ,"    + // Gol
     "- :- - ;"    +

     "d :- o ,"    + // Gol
     "- :- - ;");
////////////////////////////////////////////////////////////////////////////
PutDescription(
"Base de reglas de un ping pong electronico, pong, mediante reglas de
reescritura de areas rectangulares de caracteres.
Los 4 estados de la pelota se codifican mediante 4 caracteres.",
PonRb1);
////////////////////////////////////////////////////////////////////////////
```

inc.tol de ChRules.Iterative

Inclusion de ficheros de funciones especificas de aplicacion

Declaraciones

Inclusiones comunes

- `Set txtInc`
  Text functions.
- `Set setInc`
  Set functions.
- `Set filInc`
  File functions.
- `Set dirInc`
  Directory functions.

Inclusiones de aplicación

- `Set areInc`
  Funciones para areas de caracteres.
- `Set rulInc`
  Funciones para reglas de caracteres, rules.
- `Set engInc`
  Funciones para el motor de reglas, engine.
- `Set bubInc`
  Reglas que ordenan como el metodo de la burbuja.
- `Set celInc`
  Reglas para automatas celulares.
- `Set labInc`
  Reglas para robots en laberintos.
- `Set fllInc`
  Reglas rellenar, fill, superficies cerradas.
- `Set ponInc`
  Reglas de un ping pong electronico, pong.

Set txtInc

```
////////////////////////////////////////////////////////////////////////////
Set  txtInc = Include("cmm/txt.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Text functions.", txtInc);
////////////////////////////////////////////////////////////////////////////
```

Set setInc

```
////////////////////////////////////////////////////////////////////////////
Set  setInc = Include("cmm/set.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Set functions.", setInc);
////////////////////////////////////////////////////////////////////////////
```

Set filInc

```
////////////////////////////////////////////////////////////////////////////
Set  filInc = Include("cmm/fil.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("File functions.", filInc);
////////////////////////////////////////////////////////////////////////////
```

Set dirInc

```
////////////////////////////////////////////////////////////////////////////
Set  dirInc = Include("cmm/dir.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Directory functions.", dirInc);
////////////////////////////////////////////////////////////////////////////
```

Set areInc

```
////////////////////////////////////////////////////////////////////////////
Set  areInc = Include("app/are.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Funciones para areas de caracteres.", areInc);
////////////////////////////////////////////////////////////////////////////
```

Set rulInc

```
////////////////////////////////////////////////////////////////////////////
Set  rulInc = Include("app/rul.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Funciones para reglas de caracteres, rules.", rulInc);
////////////////////////////////////////////////////////////////////////////
```

Set engInc

```
////////////////////////////////////////////////////////////////////////////
Set  engInc = Include("app/eng.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Funciones para el motor de reglas, engine.", engInc);
////////////////////////////////////////////////////////////////////////////
```

Set bubInc

```
////////////////////////////////////////////////////////////////////////////
Set  bubInc = Include("app/bub.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Reglas que ordenan como el metodo de la burbuja.", bubInc);
```

```
////////////////////////////////////////////////////////////////////////////
```

Set celInc

```
////////////////////////////////////////////////////////////////////////////
Set  celInc = Include("app/cel.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Reglas para automatas celulares.", celInc);
////////////////////////////////////////////////////////////////////////////
```

Set labInc

```
////////////////////////////////////////////////////////////////////////////
Set  labInc = Include("app/lab.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Reglas para robots en laberintos.", labInc);
////////////////////////////////////////////////////////////////////////////
```

Set fllInc

```
////////////////////////////////////////////////////////////////////////////
Set  fllInc = Include("app/fll.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Reglas rellenar, fill, superficies cerradas.", fllInc);
////////////////////////////////////////////////////////////////////////////
```

Set ponInc

```
////////////////////////////////////////////////////////////////////////////
Set  ponInc = Include("app/pon.tol");
////////////////////////////////////////////////////////////////////////////
PutDescription("Reglas de un ping pong electronico, pong.", ponInc);
////////////////////////////////////////////////////////////////////////////
```