4 PBE. MODELOS BÁSICOS

Esta unidad tiene como objetivo ilustrar la metodología descrita y el estudio de problemas mediante la resolución mecánica de los mismos a través de la modelización y simulación, así como introducir unos primeros ejemplos sencillos haciendo uso de la aproximación PBE (P systems By Example).

1 Ejemplo 1. Hola mundo

Este ejemplo trata de presentar un problema muy sencillo a resolver mediante sistemas P, con el objetivo de ir familiarizándonos con el proceso y las herramientas a emplear.

1. 1 Escenario

El problema a estudiar en este primer ejemplo no es un problema del mundo real, sino ilustrativo de ciertos conceptos interesantes en el ámbito de los sistemas P. Concretamente, vamos a estudiar el funcionamiento de un sistema no determinista maximal, en el que se generarán distintos resultados de un modo no determinista, es decir, ante la misma entrada no estará determinada unívocamente la salida ni el camino para llegar a la misma. Para ello, se construirá un sistema P de transición que, partiendo de un estado inicial definido, tenga varias posibilidades para evolucionar, seleccionadas de forma aleatoria.

Este tipo de sistemas P clásicos de transición son interesantes para la resolución de problemas NP-completos (entre otros), son no deterministas e implementan un paralelismo maximal como estrategia de ejecución de las reglas del sistema (existen otras estrategias distintas a la descrita, como la determinista, estocástica, probabilística, paralelismo minimal, etc.)

1. 2 Modelo

El sistema P que resuelve el problema anterior es tremendamente sencillo. Se trata de un sistema incluyendo una única membrana (la membrana piel, que hemos etiquetado como 1), y con un multiconjunto inicial constando de un solo objeto a. En la única membrana existente tendremos dos reglas, que podrán hacer evolucionar el objeto a bien en b o en c respectivamente.

Dada la cercanía de la sintaxis del sistema P en lenguaje formal y la misma en el lenguaje estándar P-Lingua, evitamos la repetición incluyendo directamente el código P-Lingua correspondiente:

```
@model<transition>
def main()
{
  @mu = []'1;
  @ms(1) = a*1;

  [a --> b]'1;
  [a --> c]'1;
}
```

1. 3 Simulación

Este sencillo sistema P de transición parte de una membrana (piel) cuya configuración inicial contiene un único objeto a, con multiplicidad 1. Este compartimento delimitado por la membrana piel contiene además dos reglas, que dado un objeto a pueden hacerlo evolucionar hacia b o hacia c.

Podemos hacer una traza manual de las dos posibles computaciones que se podrían dar en este caso al ejecutar este sistema en la máquina real que implementara este sistema P de transición. No obstante, es más conveniente disponer de simuladores que nos ahorren este trabajo manual. Para ello disponemos de herramientas software introducidas en la unidad 3, P-Lingua y MeCoSim, que nos van a facilitar enormemente la tarea.

Para llevar a cabo la simulación planteada en este primer ejemplo debemos hacer lo siguiente:

1. Si aún no hemos instalado MeCoSim, procedemos a su instalación a través del botón "Launch" de su página Web: `http://www.p-lingua.org/mecosim/`. Esto nos descargará la última versión del programa, creará iconos de acceso directo en el escritorio y lanzará la ventana principal de la aplicación. Si ya lo tiene instalado, lance la aplicación desde el icono o el menú de su sistema operativo.

   MeCoSim dispone de una aplicación por defecto, en la que podemos cargar sistemas P escritos en P-Lingua, no requiriendo datos de entrada ni una interfaz de usuario a medida. Este es el caso que nos ocupa, puesto que tenemos el sistema descrito en el apartado de "Modelo", no requerimos entrada específica y nos bastará con la información disponible en la aplicación "General", que es la que vemos en el listado de MeCoSim.

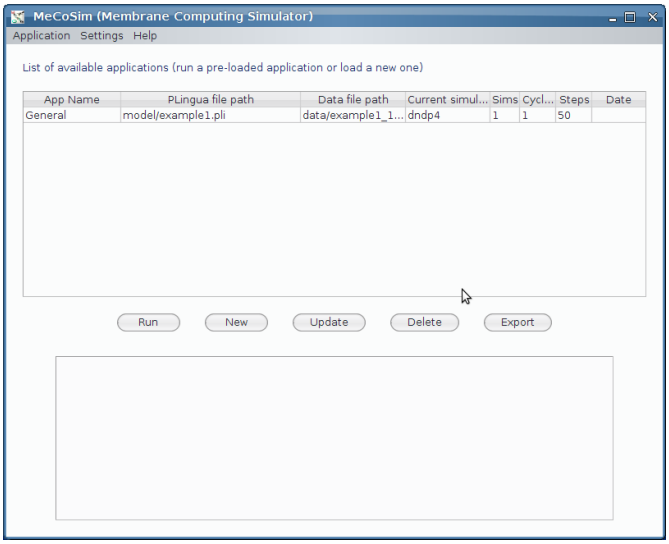

FIGURA 1: MeCoSim - Ventana principal

2. Si no disponemos de los plugins básicos de MeCoSim, procedemos a su instalación.

   Muchas funcionalidades del programa dependen de extensiones del mismo, plugins. Existen repositorios de plugins de donde podemos descargar plugins para MeCoSim. Por ejemplo, vamos a incorporar el plugin básico de MeCoSim mediante la opción de menú *Settings* > *Manage repositories* > *plugins*. Se nos abre una ventana en la que seleccionar el repositorio del que queremos tomar los plugins, y escogemos el que aparece por defecto. Al seleccionarlo, nos sale un listado de plugins. En él podemos hacer doble click sobre el plugin deseado (en nuestro caso el primero), o bien hacer click y pulsar en el botón "Instalar". Con esto quedará instalado el plugin, que descargará el mismo de forma que pueda ser empleado por MeCoSim. Ahora bien, los plugins pueden contener muchas posibles funcionalidades. Para activar una determinada funcionalidad debemos actualizar el archivo de configuración de plugins. Para eso, vamos a la carpeta de *MeCoSim* > *prop* > *plugins-properties* y abrimos el archivo con un editor de textos tipo *notepad*++ o similar. Una vez en el archivo podemos activar distintas funcionalidades según los puntos de entrada contemplados en los plugins. En nuestro caso vamos a activar el editor de archivo P-Lingua y algunos visores de alfabetos, membranas y objetos. Basta con añadir en el fichero lo siguiente:

```
plugin-plinguaeditor = mecosim.plugins.basics.PLinguaEditor
pluginname-plinguaeditor = Edit P-lingua File
pluginmethod-plinguaeditor = openPLinguaFile
pluginorder-plinguaeditor = 1
pluginjar-plinguaeditor-1 = MeCoSimBasicsPlugin.jar
```

```
plugin-alphabetviewer = mecosim.plugins.basics.AlphabetViewer
pluginname-alphabetviewer = Alphabet Viewer
pluginmethod-alphabetviewer = visualizeAlphabet
pluginorder-alphabetviewer = 2
pluginjar-alphabetviewer-1 = MeCoSimBasicsPlugin.jar

plugin-membranestructureviewer = mecosim.plugins.basics.MembraneStructureViewer
pluginname-membranestructureviewer = Membrane Structure Viewer
pluginmethod-membranestructureviewer = visualizeMembraneStructure
pluginorder-membranestructureviewer = 3
pluginjar-membranestructureviewer-1 = MeCoSimBasicsPlugin.jar

plugin-multisetsviewer = mecosim.plugins.basics.MembraneStructureViewer
pluginname-multisetsviewer = Multisets Viewer
pluginmethod-multisetsviewer = visualizeMultiSets
pluginorder-multisetsviewer = 4
pluginjar-multisetsviewer-1 = MeCoSimBasicsPlugin.jar
```

No es necesario conocer el cometido de cada dato a menos que seamos desarrolladores de plugins, ya que vendrán dados por el desarrollador del mismo. Sí es conveniente saber que podemos modificar el pluginorder para alterar el orden en el que aparecerán en la ventana de nuestras aplicaciones basadas en MeCoSim. De momento podemos dejar tal cual estas líneas en el fichero y guardar. A continuación salimos de MeCoSim con *Repositories* > *Exit* y *Application* > *Exit* y volvemos a entrar, de modo que ya nos aparezcan los plugins.

3. Obtener el modelo del repositorio de modelos del curso MATVIDA. Es decir, de: `http://www.p-lingua.org/mecosim/matvida/models/`. Para ello, introduzca el repositorio en MeCoSim mediante *Settings* > *Manage repositories* > *Models*, y una vez ahí pulse el botón "Add" y pegue la url indicada. A continuación selecciónela para que aparezca el listado de modelos en P-Lingua. Haga doble click en el ejemplo 1 o bien click simple y botón "Install", de modo que el modelo quede instalado y disponible para MeCoSim. Ya podemos salir del gestor de repositorios mediante *Repositories* > *Exit*, llegando así a la ventana principal, listos para empezar a trabajar con el modelo.

4. Lanzar la aplicación por defecto haciendo doble click en la aplicación con $AppName$ "General", o bien click simple y botón "Run". Esta aplicación no contiene tablas de datos de entrada, y nos muestra la tabla de simulación por defecto, con la configuración del sistema P en cada paso (Simulation, cycle, step, environment, label, membrane, object, multiplicity). Nos servirá para cargar cualquier sistema P escrito en P-Lingua que no requiera una entrada por parte del usuario, como es el caso de los dos ejemplos básicos de esta unidad.

5. Una vez en la aplicación "General", cargamos el modelo mediante la opción de menú *Model* > *Set model*, y seleccionando el archivo .pli del ejemplo 1 que acabamos de instalar. Ya tenemos el modelo listo para depurar o simular.

6. En lugar de simular, antes vamos a depurar el modelo para comprobar que no contiene errores. Para ello, vamos a la pestaña *Debug console*, y dentro de ella pulsamos en el botón "Init model". Vemos que la pestaña *ParsingInfo* nos muestra los módulos cargados y las reglas del sistema, con el listado

de parámetros vacío (no es un sistema parametrizado, no dependiendo de entrada externa por parte del usuario), y si todo ha ido bien no se muestra ninguna ventana indicándonos la presencia de errores en el archivo. Podemos ver en cualquier caso el contenido de errores y avisos mediante las pestañas $Errors$ y $Warnings$, que en este caso deben estar vacías.

7. Vamos a proceder ahora a seleccionar el simulador a emplear. P-Lingua dispone de una serie de simuladores asociados a cada variante de sistema P contemplada. Como vemos en la ventana abajo, con el rótulo de "Selected simulator", por defecto se nos habrá cargado el simulador "non_deterministic _transition". Podemos cambiar este simulador por uno más adecuado que ilustra nuestro ejemplo: "non_deterministic_transition_without_priorities". Esto se lleva a cabo a través de la opción de menú *Simulation* > *Options* > *Simulation Algorithm* > *non_deterministic_transition_without_priorities*.

8. Ya podemos simular, bien mediante el modo de depuración paso a paso o hasta satisfacer el criterio de parada que se establezca. Comencemos ilustrando el modo de simulación mediante depuración paso a paso. En la pestaña de debug, si ya hemos hecho "Init Model", estamos listos para seleccionar la pestaña "Simulation Info" y pulsar en el botón "Step". Esto simulará la ejecución de un paso de computación mediante el simulador seleccionado anteriormente. Si se trata del primero, esta pestaña nos mostrará en modo texto la configuración 0 (inicial), las reglas seleccionadas en el paso 1, y a continuación la configuración 1. Si continuamos ejecutando pasos mediante "Step", nos irá mostrando para cada uno las reglas aplicadas y la configuración resultante. Si a partir de la configuración actual ya no es posible aplicar ninguna regla se mostrará el mensaje "Halting configuration (No rule can be selected to be executed in the next step)". También podríamos lanzar la ejecución de un número de pasos mediante "Run steps", mostrándonos en el texto tanto los pasos como las sucesivas configuraciones. Tanto tras la inicialización del modelo como tras ejecutar cada paso, podemos ver la información mediante los plugins de visualización configurados anteriormente:

   - Visor del alfabeto del sistema P: mediante la opción Plugins > Alphabet Viewer.
   - Visor de la estructura de membranas: mediante la opción Plugins > Membrane Structure Viewer.
   - Visor de los multiconjuntos de objetos presentes en cada membrana: mediante la opción Plugins > Multisets Viewer.

   Este ejemplo es muy sencillo y podemos validar que la información que muestran estos visores es la esperada, pero resultará de mayor utilidad para disponer de información estructurada jerárquicamente en forma de árbol en ejemplos más grandes, donde el análisis de la información textual puede ser demasiado arduo y tedioso.

9. Aparte del modo de depuración, podemos simular y obtener la salida en forma de tabla tanto si somos diseñadores de sistemas P como usuarios finales interesados en el problema a resolver más que en el paradigma empleado. Para ello, podemos volver a la pestaña "Output", inicialmente vacía o conteniendo ceros. Una vez aquí, podemos decidir si queremos simular un número determinado de pasos (estructurados en ciclos y permitiendo lanzar un número determinado de computaciones completas en lugar de una sola, como veremos en ejemplos posteriores) o podemos simular hasta alcanzar una configuración de parada. Para emplear este último modo que es el que nos interesa en este caso, indicamos que vamos a simular 0 ciclos como sinónimo de un número indeterminado. Esto se realiza mediante la opción de menú *Simulate* > *Options* > *Number of cycles* > *0*. Automáticamente desaparecen de la ventana los datos "Simulated cycles", "Simulations by cycle" y "Steps by cycle", y en su lugar aparece resaltado en azul el mensaje "Simulate until a halting condition". Podemos proceder a simular mediante la opción *Simulation* > *Simulate*, lo que lanzará la computación del sistema P correspondiente hasta alcanzar una configuración de parada que no permita seguir simulando. Si todo va bien se mostrará a la finalización el mensaje "The simulation has finished!", y se mostrará en la tabla "Output" el contenido de cada membrana en cada momento, al nivel de granularidad más fino (cada línea representa la multiplicidad de un objeto en una determinada membrana en un paso de computación).

Hasta aquí el detalle de la simulación. En el próximo apartado se detallan los resultados esperados y el análisis de los mismos.

1. 4 Resultados

Este sencillo sistema P de transición parte de una membrana (piel) cuya configuración inicial contiene un único objeto a, con multiplicidad 1. Este compartimento delimitado por la membrana piel contiene además dos reglas, que dado un objeto a pueden hacerlo evolucionar hacia b o hacia c.

Se proponen los siguientes experimentos virtuales:

- Lanzar 10 simulaciones mediante *Simulation* > *Simulate!* y comprobar qué ocurre. ¿Obtenemos siempre la misma salida? ¿A qué se debe?
- Modificar ahora el sistema P (mediante el plugin de edición, en *Plugins* > *Edit P-Lingua File* o bien mediante un editor de texto externo) de modo que la configuración inicial incluya 1.000 objetos a en lugar de 1. ¿Qué sucede ahora? ¿Cuál es la explicación? ¿Cuántos pasos de computación ha dado el simulador?
- Pruebe a introducir ahora una tercera regla que genere un objeto d a partir de un objeto a. ¿Qué cambio observa ahora?

Deberá considerar la descripción de los sistemas P de transición que hemos comentado, atendiendo a las características fundamentales que explican su dinámica.

2 Ejemplo 2. Cuadrados de números

Este ejemplo trata de mostrar un ejemplo de sistema P de transición que genera el conjunto de los cuadrados de los números naturales comenzando en 1.

2. 1 Escenario

El problema a resolver consiste en la generación no determinista de elementos del conjunto $\{n^2 : n \geq 1\}$, es decir, los cuadrados de los números naturales (excluyendo el caso particular trivial del 0).

2. 2 Modelo

El sistema P que resuelve el problema es el siguiente:

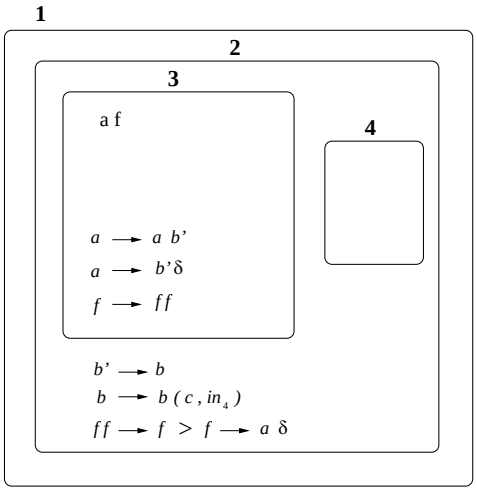

FIGURA 2: Sistema P de transición generador de los elementos del conjunto $\{n^2 : n \geq 1\}$

Podemos verlo descrito en P-Lingua:

```
@model<transition>
def main()
```

```
{
  call n_cuadrados();
}
def n_cuadrados()
{
  @mu = [[[]'3 []'4]'2]'1;
  @ms(3) = a,f;
  [a --> a,bp]'3;
  [a --> bp,@d]'3;
  [f --> f*2]'3;
  [bp --> b]'2;
  [b []'4 --> b [c]'4]'2;
  (1) [f*2 --> f ]'2;
  (2) [f --> a,@d]'2;
}
```

2. 3 Simulación

Como hemos comentado, este sistema genera elementos del conjunto de los cuadrados de los números naturales. Al tratarse de un sistema no determinista, no podemos saber a priori qué rama del árbol de posibles computaciones va a recorrerse, pero podemos estar seguros de que se obtendrá un cuadrado de un número natural. Experimente con el sistema lanzando simulaciones, y observe cómo se van obteniendo distintos resultados.

El diseño de sistemas P no es un proceso trivial, de modo que la imagen anterior no es fácil de interpretar, especialmente para aquellos no familiarizados con este modelo de computación. Por ello pasamos a continuación a explicar el funcionamiento de este sistema, que podrán analizar realizando una depuración paso a paso y observando las reglas que se van aplicando y las configuraciones que se van sucediendo.

Este sistema P de transición con prioridades parte, como vemos en la figura, de una membrana (3) cuya configuración inicial contiene un objeto a y otro f. No hay ningún objeto en la membrana 2, luego no se puede aplicar ninguna regla en esa región. La única posibilidad es, por tanto, empezar la computación en la membrana 3, usando los objetos libres a y f, de los que disponemos de una copia de cada uno. Usando las reglas $a \to ab'$ y $f \to ff$ en paralelo para todas las ocurrencias de a y f disponibles, tras n pasos, con $n \geq 0$, obtenemos n ocurrencias de b' y 2^n ocurrencias de f. En cualquier momento, en lugar de la regla $a \to ab'$ se puede disparar la regla $a \to ab'\delta$ (obsérvese que en todo momento tenemos una única copia de a). En el momento que se ejecute esa regl tendremos $n+1$ ocurrencias de b' y 2^{n+1} ocurrencias de f, y se disolverá la membrana 3, quedando la configuración:

$$[_1[_2 b'^{n+1} f^{2^{n+1}}, b' \to b, b \to b(c, in_4), r_1 : ff \to af, r_2 : f \to a\delta, r_1\ r_2, [_4]_4]_2]_1.$$

Como es natural, las reglas de la anterior membrana 3 se habrán perdido y los objetos pasarán a la membrana 2, cuyas reglas serán ahora las únicas activas (hasta el momento no podían serlo ya que no había objetos en dicha membrana).

Debido a la relación de prioridad, se tiene que emplear la regla $ff \rightarrow af$ tanto como sea posible. En un paso, pasamos de b'^{n+1} a b^{n+1}, mientras que el número de ocurrencias del objeto f se divide entre 2. En el siguiente paso, a partir de b^{n+1}, se introducen $n+1$ ocurrencias de c en la membrana 4 (cada ocurrencia del símbolo b introduce una ocurrencia del objeto c). Al mismo tiempo, el número de ocurrencias de f es de nuevo dividido entre 2. Podemos continuar. En cada paso, $n+1$ ocuurencias adicionales de c son introducidas en la membrana de salida (4). Esto se puede hacer durante $n+1$ pasos: n veces cuando se dispara la regla $ff \rightarrow af$ (disminuyendo así el número de ocurrencias de f a una), y una cuando se usa la regla $f \rightarrow a\delta$ (al tener menor prioridad, es únicamente en ese momento cuando se puede aplicar, una vea tenemos una única f y no un número par de ellas. En este momento, se disuelve la membrana 2, con lo cuál todas sus reglas son eliminadas y no hay más pasos posibles. La configuración que se obtiene es la siguiente:

$$[_1 a^{2^{n+1}} b^{n+1}, [_4 c^{(n+1)^2}]_4]_1.$$

Aquí se muestra una tabla con la traza resumida:

Paso	Membrana 1	Membrana 2	Membrana 3	Membrana 4
0			af	
1			$ab'f^2$	
2			$ab'^2f^{2^2}$	
3			$ab'^3f^{2^3}$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
m			$ab'^m f^{2^m}$	
$m+1$		$b'^{(m+1)}f^{2^{m+1}}$	$disuelta$	
$m+2$		$b^{m+1}f^{2^m}$	$disuelta$	
$(m+2)+1$		$b^{m+1}f^{2^{m-1}}$	$disuelta$	c^{m+1}
$(m+2)+2$		$b^{m+1}f^{2^{m-2}}$	$disuelta$	$c^{2(m+1)}$
$(m+2)+3$		$b^{m+1}f^{2^{m-3}}$	$disuelta$	$c^{3(m+1)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(m+2)+m$		$b^{m+1}f^{2^{m-m}}$	$disuelta$	$c^{m(m+1)}$
$2m+3$	ab^{m+1}	$disuelta$	$disuelta$	$c^{(m+1)(m+1)}$

2. 4 Resultados

Se ha explicado en el apartado anterior el proceso llevado a cabo para la simulación del sistema, y los detalles acerca de la dinámica del sistema. La cuestión es experimentar con el sistema P dado, realizando sucesivas simulaciones y observando los resultados obtenidos. ¿Qué observa?

Se proponen los siguientes experimentos virtuales:

- Analice una traza completa a través de la pestaña de depuración de MeCoSim ("Debug console") paso a paso, y compruebe que su comportamiento es el descrito en el apartado anterior.

- Lanzar un buen conjunto de simulaciones (al menos 10) mediante *Simulation > Simulate!* y comprobar qué ocurre. ¿Cuántos objetos c hay en el último paso de computación? Observe la relación entre el número de objetos c y el número de objetos b. ¿Qué tipo de relación es? Compruebe que el número de objetos es el esperado según la traza explicada en el apartado anterior. ¿Cuál es el mayor valor de multiplicidad obtenido para c? Reflexione acerca de ello.