Control de la ejecución y diseño modular

Francisco J. Martín Mateos

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Control de la ejecución

- Ejemplo de fases de un problema:
  - Lectura de datos.
  - Detección de problemas.
- Técnicas de control:
  - Control empotrado en las reglas.
  - Control usando prioridades.
  - Control usando reglas.
  - Control usando módulos.

Control empotrado en las reglas

Control empotrado en las reglas

- Distinguir las fases del problema usando hechos de control:
  - `(fase lectura-de-datos)`
  - `(fase deteccion-de-problemas)`
- Todas las reglas de una misma fase tienen entre sus condiciones el hecho de control correspondiente.
- Las reglas que finalizan una fase tienen que eliminar el hecho de control de dicha fase y añadir el de la siguiente.
- Inconvenientes:
  - Identificación de los hechos de control.
  - Dificultad para precisar la conclusión de cada fase.

Control empotrado en las reglas

- Hechos iniciales:

```
(deffacts inicio
  (fase lectura-de-datos)
  (dispositivos C1 C2 C3))
```

- Fase de lectura de datos:

```
(defrule lectura-de-datos
  (fase lectura-de-datos)
  ?h <- (dispositivos ?id $?res)
  (not (dato ?id ?val))
  =>
  (retract ?h)
  (assert (dispositivos $?res)
          (dato ?id (random 1 100))))

(defrule final-de-lectura-de-datos
  ?h1 <- (fase lectura-de-datos)
  ?h2 <- (dispositivos)
  =>
  (retract ?h1 ?h2)
  (assert (dispositivos C1 C2 C3)
          (fase deteccion-de-problemas)))
```

Control empotrado en las reglas

- Fase de detección de problemas:

```
(defrule deteccion-de-problemas-1
  (fase deteccion-de-problemas)
  ?h <- (dato ?id ?val&:(evenp ?val))
  =>
  (retract ?h)
  (printout t "Problemas en el dispositivo " ?id crlf))

(defrule deteccion-de-problemas-2
  (fase deteccion-de-problemas)
  ?h <- (dato ?id ?val&:(oddp ?val))
  =>
  (retract ?h)
  (printout t "Sin problemas en el dispositivo " ?id crlf))

(defrule final-deteccion-de-problemas
  ?h <- (fase deteccion-de-problemas)
  (not (dato ? ?))
  =>
  (retract ?h)
  (assert (fase lectura-de-datos)))
```

Control empotrado en las reglas: Ejercicio

- Añadir control empotrado en las reglas a la práctica del Sudoku
- Utilizar dos hechos de control:
  - `(fase basica)` en la que se usan las estrategias de valor asignado, par asignado, valores ocultos y pares ocultos.
  - `(fase avanzada)` en la que se usan las estrategias de la intersección y de la cruz.
- Implementar el siguiente comportamiento:
  - Las estrategias asociadas a la `(fase avanzada)` solo se usan cuando no se pueden aplicar las estrategias asociadas a la `(fase basica)`.
  - Una vez aplicada una regla asociada a la `(fase avanzada)`, se tienen que volver a evaluar las estrategias asociadas a la `(fase basica)`.

Control usando prioridades

Prioridades

- Sintaxis:

```
(defrule <nombre>
  (declare (salience <numero>))
  <condicion>*
  =>
  <accion>*)
```

- Valores:
  - Mínimo: -10000
  - Máximo: 10000
  - Defecto: 0
- Ventajas:
  - No es necesario precisar la conclusión de cada fase.
- Inconvenientes:
  - Tendencia a abusar de las prioridades.
  - Contradicción con el objetivo de los sistemas basados en reglas.
  - Las fases se pueden mezclar.
  - Dificultad para comenzar otra vez el ciclo.

Control usando prioridades

- Hechos iniciales:

```
(deffacts inicio
  (dispositivos C1 C2 C3))
```

- Fase de lectura de datos:

```
(defrule lectura-de-datos
  (declare (salience 30))
  ?h <- (dispositivos $? ?id $?)
  (not (dato ?id ?val))
  =>
  (assert (dato ?id (random 1 100))))
```

Control usando prioridades

- Fase de detección de problemas:

```
(defrule deteccion-de-problemas-1
  (declare (salience 20))
  ?h <- (dato ?id ?val&:(evenp ?val))
  =>
  (retract ?h)
  (printout t "Problemas en el dispositivo " ?id crlf))

(defrule deteccion-de-problemas-2
  (declare (salience 20))
  ?h <- (dato ?id ?val&:(oddp ?val))
  =>
  (retract ?h)
  (printout t "Sin problemas en el dispositivo " ?id crlf))
```

Control usando prioridades: Ejercicio

- Añadir control usando prioridades a la práctica del Sudoku
- Utilizar dos prioridades:
  - Prioridad `20` en la que se usan las estrategias de valor asignado, par asignado, valores ocultos y pares ocultos.
  - Prioridad `10` en la que se usan las estrategias de la intersección y de la cruz.
  - Las reglas que imprimen la solución tienen prioridad `-10`.
- Implementar el siguiente comportamiento:
  - Las estrategias asociadas a la prioridad `10` solo se usan cuando no se pueden aplicar las estrategias asociadas a la prioridad `20`.
  - Una vez aplicada una regla asociada a la prioridad `10`, se tienen que volver a evaluar las estrategias asociadas a la prioridad `20`.

Control usando reglas

Reglas de control

- Reglas específicas para controlar las fases de desarrollo.
  - Se utilizan hechos de control para distinguir las fases.
  - Se definen reglas para cambiar las fases con baja prioridad.
- Ventajas:
  - Separación entre reglas de control y reglas de proceso.
  - Acorde con las metodologías de diseño.
  - No es necesario precisar la conclusión de cada fase.

Control usando reglas (1)

- Una regla para cada cambio de fase:

```
(deffacts inicio
  (fase lectura-de-datos)
  (dispositivos C1 C2 C3))

(defrule lectura-a-deteccion
  (declare (salience -10))
  ?fase <- (fase lectura-de-datos)
  =>
  (retract ?fase)
  (assert (fase deteccion-de-problemas)))

(defrule deteccion-a-lectura
  (declare (salience -10))
  ?fase <- (fase deteccion-de-problemas)
  =>
  (retract ?fase)
  (assert (fase lectura-de-datos)))
```

Control usando reglas (1)

- Fase de lectura de datos:

```
(defrule lectura-de-datos
  (fase lectura-de-datos)
  ?h <- (dispositivos $? ?id $?)
  (not (dato ?id ?val))
  =>
  (assert (dato ?id (random 1 100))))
```

- Fase de detección de problemas:

```
(defrule deteccion-de-problemas-1
  (fase deteccion-de-problemas)
  ?h <- (dato ?id ?val&:(evenp ?val))
  =>
  (retract ?h)
  (printout t "Problemas en el dispositivo " ?id crlf))

(defrule deteccion-de-problemas-2
  (fase deteccion-de-problemas)
  ?h <- (dato ?id ?val&:(oddp ?val))
  =>
  (retract ?h)
  (printout t "Sin problemas en el dispositivo " ?id crlf))
```

Control usando reglas (2)

- Una regla para todos los cambios de fase:

```
(deffacts control
  (fase lectura-de-datos)
  (siguiente-fase lectura-de-datos deteccion-de-problemas)
  (siguiente-fase deteccion-de-problemas lectura-de-datos))

(defrule cambio-de-fase
  (declare (salience -10))
  ?fase <- (fase ?actual)
  (siguiente-fase ?actual ?siguiente)
  =>
  (retract ?fase)
  (assert (fase ?siguiente)))
```

Control usando reglas (3)

- Cambio de fase en secuencia:

```
(deffacts control
  (fase lectura-de-datos)
  (secuencia-de-fases lectura-de-datos
                      deteccion-de-problemas))

(defrule cambio-de-fase
  (declare (salience -10))
  ?fase <- (fase ?actual)
  ?secuencia <- (secuencia-de-fases ?siguiente $?resto)
  =>
  (retract ?fase ?secuencia)
  (assert (fase ?siguiente)
          (secuencia-de-fases $?resto ?siguiente)))
```

Control usando reglas: Ejercicio

- Añadir control usando reglas a la práctica del Sudoku
- Utilizar dos hechos de control:
  - `(fase basica)` en la que se usan las estrategias de valor asignado, par asignado, valores ocultos y pares ocultos.
  - `(fase avanzada)` en la que se usan las estrategias de la intersección y de la cruz.
- Implementar el siguiente comportamiento:
  - Las estrategias asociadas a la `(fase avanzada)` solo se usan cuando no se pueden aplicar las estrategias asociadas a la `(fase basica)`.
  - Una vez aplicada una regla asociada a la `(fase avanzada)`, se tienen que volver a evaluar las estrategias asociadas a la `(fase basica)`.

Control usando módulos

Módulos

- Grupos de constructores (reglas, hechos, funciones, ...) con agenda propia.
- Sólo un módulo está "activo" en cada momento, los restantes están "latentes".
- El módulo por defecto es MAIN y es el primero en construirse.
- Sintaxis:

```
(defmodule <nombre>
  (export <elemento>)*
  (import <modulo> <elemento>)*)

<elemento> := ?ALL |
              ?NONE |
              <constructor> ?ALL |
              <constructor> ?NONE |
              <constructor> <nombre>+

<constructor> := deftemplate | defclass |
                 defglobal | deffunction | defgeneric
```

Comunicación entre módulos

- Lectura secuencial de ficheros:
  - Sólo se puede exportar a módulos sucesivos.
  - Sólo se puede importar de módulos anteriores.
  - El módulo MAIN puede exportar a cualquier otro pero no puede importar de ninguno.
- Solución más general: exportarlo todo e importarlo todo de los módulos anteriores.

```
(defmodule DETECCION
  (export ?ALL))

(defmodule AISLAMIENTO
  (export ?ALL))

(defmodule RECUPERACION
  (import DETECCION ?ALL)
  (import AISLAMIENTO ?ALL))
```

Definición de constructores en módulos

- Indicar delante del nombre del constructor el nombre del módulo.
- Los constructores pueden hacer uso de cualquier elemento definido en el propio módulo o importado de otro módulo.
- Módulo **DETECCION**:

```
(deftemplate DETECCION::fallo
  (slot componente))

(deffacts DETECCION::inicio
  (fallo (componente A)))

(defrule DETECCION::deteccion
  (fallo (componente A | C))
  =>)
```

Definición de constructores en módulos

- Módulo **AISLAMIENTO**:

```
(deftemplate AISLAMIENTO::posible-fallo
  (slot componente))

(deffacts AISLAMIENTO::inicio
  (posible-fallo (componente B)))

(defrule AISLAMIENTO::aislamiento
  (posible-fallo (componente B | D))
  =>)
```

- Módulo **RECUPERACION**:

```
(deffacts RECUPERACION::inicio
  (fallo (componente C))
  (posible-fallo (componente D)))

(defrule RECUPERACION::recuperacion
  (fallo (componente A | C))
  (posible-fallo (componente B | D))
  =>)
```

Memorias de trabajo

- Visualizar la memoria de trabajo de un módulo:

```
CLIPS> (facts DETECCION)
f-1     (fallo (componente A))
f-3     (fallo (componente C))
For a total of 2 facts.
CLIPS> (facts AISLAMIENTO)
f-2     (posible-fallo (componente B))
f-4     (posible-fallo (componente D))
For a total of 2 facts.
CLIPS> (facts RECUPERACION)
f-1     (fallo (componente A))
f-2     (posible-fallo (componente B))
f-3     (fallo (componente C))
f-4     (posible-fallo (componente D))
For a total of 4 facts.
```

Memorias de trabajo

- Visualizar la memoria de trabajo de todos los módulos:

```
CLIPS> (facts *)
f-0     (initial-fact)
f-1     (fallo (componente A))
f-2     (posible-fallo (componente B))
f-3     (fallo (componente C))
f-4     (posible-fallo (componente D))
```

Agendas

- Visualizar la agenda de un módulo:

```
CLIPS> (agenda DETECCION)
0      deteccion: f-3
0      deteccion: f-1
For a total of 2 activations.
CLIPS> (agenda AISLAMIENTO)
0      aislamiento: f-4
0      aislamiento: f-2
For a total of 2 activations.
CLIPS> (agenda RECUPERACION)
0      recuperacion: f-1,f-4
0      recuperacion: f-1,f-2
0      recuperacion: f-3,f-4
0      recuperacion: f-3,f-2
For a total of 4 activations.
```

Agendas

- Visualizar la agenda de todos los módulos:

```
CLIPS> (agenda *)
MAIN:
DETECCION:
   0      deteccion: f-3
   0      deteccion: f-1
AISLAMIENTO:
   0      aislamiento: f-4
   0      aislamiento: f-2
RECUPERACION:
   0      recuperacion: f-1,f-4
   0      recuperacion: f-3,f-4
   0      recuperacion: f-3,f-2
   0      recuperacion: f-1,f-2
For a total of 8 activations.
```

Pila de módulos

- Estructura que establece la secuencia de módulos a ejecutar.
  - Al iniciar el sistema de producción el módulo MAIN se coloca como primer módulo.
  - Se pueden colocar otros módulos con el comando **`focus`**.

```
CLIPS> (reset)
CLIPS> (watch rules)
CLIPS> (run)
CLIPS> (focus DETECCION)
TRUE
CLIPS> (run)
FIRE    1 deteccion: f-3
FIRE    2 deteccion: f-1
```

Pila de módulos

- El comando `(list-focus-stack)` permite ver el estado de la pila de módulos:

```
CLIPS> (reset)
CLIPS> (focus RECUPERACION)
TRUE
CLIPS> (focus AISLAMIENTO)
TRUE
CLIPS> (list-focus-stack)
AISLAMIENTO
RECUPERACION
MAIN
CLIPS> (run)
FIRE    1 aislamiento: f-4
FIRE    2 aislamiento: f-2
FIRE    3 recuperacion: f-1,f-4
FIRE    4 recuperacion: f-3,f-4
FIRE    5 recuperacion: f-3,f-2
FIRE    6 recuperacion: f-1,f-2
```

Pila de módulos

- Se pueden apilar varios módulos en una misma instrucción **focus**:

```
CLIPS> (reset)
CLIPS> (focus AISLAMIENTO RECUPERACION)
TRUE
CLIPS> (list-focus-stack)
AISLAMIENTO
RECUPERACION
MAIN
CLIPS> (run)
FIRE    1 aislamiento: f-4
FIRE    2 aislamiento: f-2
FIRE    3 recuperacion: f-1,f-4
FIRE    4 recuperacion: f-3,f-4
FIRE    5 recuperacion: f-3,f-2
FIRE    6 recuperacion: f-1,f-2
```

Pila de módulos

- Se puede apilar varias veces un mismo módulo, aunque no de forma consecutiva:

```
CLIPS> (reset)
CLIPS> (focus DETECCION DETECCION AISLAMIENTO DETECCION)
TRUE
CLIPS> (list-focus-stack)
DETECCION
AISLAMIENTO
DETECCION
MAIN
CLIPS> (run)
FIRE    1 deteccion: f-3
FIRE    2 deteccion: f-1
FIRE    3 aislamiento: f-4
FIRE    4 aislamiento: f-2
```

Pila de módulos

- El comando **pop-focus** elimina el primer módulo de la pila.
  - Este comando se puede usar en una regla como acción.

```
CLIPS> (reset)
CLIPS> (focus DETECCION AISLAMIENTO RECUPERACION)
TRUE
CLIPS> (list-focus-stack)
DETECCION
AISLAMIENTO
RECUPERACION
MAIN
CLIPS> (run 2)
FIRE    1 deteccion: f-3
FIRE    2 deteccion: f-1
CLIPS> (pop-focus)
AISLAMIENTO
CLIPS> (list-focus-stack)
RECUPERACION
MAIN
CLIPS> (run)
FIRE    1 recuperacion: f-1,f-4
FIRE    2 recuperacion: f-3,f-4
FIRE    3 recuperacion: f-3,f-2
FIRE    4 recuperacion: f-1,f-2
```

Pila de módulos

- El comando `(watch focus)` permite visualizar como cambian los elementos en la pila de módulos.

```
CLIPS> (reset)
CLIPS> (watch focus)
CLIPS> (focus DETECCION AISLAMIENTO RECUPERACION)
==> Focus RECUPERACION from MAIN
==> Focus AISLAMIENTO from RECUPERACION
==> Focus DETECCION from AISLAMIENTO
TRUE
CLIPS> (run)
FIRE    1 deteccion: f-3
FIRE    2 deteccion: f-1
<== Focus DETECCION to AISLAMIENTO
FIRE    3 aislamiento: f-4
FIRE    4 aislamiento: f-2
<== Focus AISLAMIENTO to RECUPERACION
FIRE    5 recuperacion: f-1,f-4
FIRE    6 recuperacion: f-3,f-4
FIRE    7 recuperacion: f-3,f-2
FIRE    8 recuperacion: f-1,f-2
<== Focus RECUPERACION to MAIN
<== Focus MAIN
```

Acciones sobre la pila de módulos

- El comando `focus` se puede usar como acción en una regla.

```
(defmodule MAIN
  (export ?ALL))

(defrule MAIN::inicio
  =>
  (focus DETECCION AISLAMIENTO RECUPERACION))
```

Acciones sobre la pila de módulos

- Traza de la ejecución:

```
CLIPS> (reset)
CLIPS> (watch focus)
CLIPS> (run)
FIRE    1 inicio: *
==> Focus RECUPERACION from MAIN
==> Focus AISLAMIENTO from RECUPERACION
==> Focus DETECCION from AISLAMIENTO
FIRE    2 deteccion: f-3
FIRE    3 deteccion: f-1
<== Focus DETECCION to AISLAMIENTO
FIRE    4 aislamiento: f-4
FIRE    5 aislamiento: f-2
<== Focus AISLAMIENTO to RECUPERACION
FIRE    6 recuperacion: f-1,f-4
FIRE    7 recuperacion: f-3,f-4
FIRE    8 recuperacion: f-3,f-2
FIRE    9 recuperacion: f-1,f-2
<== Focus RECUPERACION to MAIN
<== Focus MAIN
```

Acciones sobre la pila de módulos

- El comando `pop-focus` se puede usar como acción en una regla:

```
(defrule DETECCION::deteccion
  (fallo (componente A | C))
  =>
  (pop-focus))

(defrule AISLAMIENTO::aislamiento
  (posible-fallo (componente B | D))
  =>
  (pop-focus))

(defrule RECUPERACION::recuperacion
  (fallo (componente A | C))
  (posible-fallo (componente B | D))
  =>
  (pop-focus))
```

Acciones sobre la pila de módulos

- Traza de la ejecución:

```
CLIPS> (reset)
CLIPS> (watch focus)
CLIPS> (run)
FIRE    1 inicio: *
==> Focus RECUPERACION from MAIN
==> Focus AISLAMIENTO from RECUPERACION
==> Focus DETECCION from AISLAMIENTO
FIRE    2 deteccion: f-3
<== Focus DETECCION to AISLAMIENTO
FIRE    3 aislamiento: f-4
<== Focus AISLAMIENTO to RECUPERACION
FIRE    4 recuperacion: f-1,f-4
<== Focus RECUPERACION to MAIN
<== Focus MAIN
```

Acciones sobre la pila de módulos

- La declaración `(declare (auto-focus TRUE))` en una regla coloca el módulo al que pertenece en la pila de módulos al activarse dicha regla.

```
(defmodule RECUPERACION
  (import DETECCION ?ALL)
  (import AISLAMIENTO ?ALL)
  (export ?ALL))

(defrule RECUPERACION::recuperacion
  (fallo (componente A | C))
  (posible-fallo (componente B | D))
  =>
  (assert (alarma)))

(defmodule ALARMA
  (import RECUPERACION ?ALL))

(defrule ALARMA::alarma
  (declare (auto-focus TRUE))
  (alarma)
  =>)
```

Acciones sobre la pila de módulos

- Traza de la ejecución:

```
CLIPS> (reset)
CLIPS> (watch focus)
CLIPS> (run)
FIRE    1 inicio: *
==> Focus RECUPERACION from MAIN
==> Focus AISLAMIENTO from RECUPERACION
==> Focus DETECCION from AISLAMIENTO
FIRE    2 deteccion: f-3
FIRE    3 deteccion: f-1
<== Focus DETECCION to AISLAMIENTO
FIRE    4 aislamiento: f-4
FIRE    5 aislamiento: f-2
<== Focus AISLAMIENTO to RECUPERACION
FIRE    6 recuperacion: f-1,f-4
==> Focus ALARMA from RECUPERACION
FIRE    7 alarma: f-5
<== Focus ALARMA to RECUPERACION
FIRE    8 recuperacion: f-3,f-4
FIRE    9 recuperacion: f-3,f-2
FIRE   10 recuperacion: f-1,f-2
<== Focus RECUPERACION to MAIN
<== Focus MAIN
```

Control usando módulos

- Módulo **MAIN**:

```
(defmodule MAIN
  (export ?ALL))

(deffacts MAIN::inicio
  (ciclo)
  (dispositivos C1 C2 C3))

(defrule MAIN::control
  ?h <- (ciclo)
  =>
  (retract ?h)
  (assert (ciclo))
  (focus LECTURA DETECCION))
```

Control usando módulos

- Módulo **LECTURA**:

```
(defmodule LECTURA
  (import MAIN ?ALL)
  (export ?ALL))

(defrule LECTURA::lectura-de-datos
  ?h <- (dispositivos $? ?id $?)
  (not (dato ?id ?val))
  =>
  (assert (dato ?id (random 1 100))))
```

Control usando módulos

- Módulo **DETECCION**:

```
(defmodule DETECCION
  (import LECTURA ?ALL))

(defrule DETECCION::deteccion-de-problemas-1
  ?h <- (dato ?id ?val&:(evenp ?val))
  =>
  (retract ?h)
  (printout t "Problemas en el dispositivo " ?id crlf))

(defrule DETECCION::deteccion-de-problemas-2
  ?h <- (dato ?id ?val&:(oddp ?val))
  =>
  (retract ?h)
  (printout t "Sin problemas en el dispositivo " ?id crlf))
```

Control usando módulos: Ejercicio

- Añadir control usando módulos a la práctica del Sudoku
- Identificar tres módulos:
  - Módulo **MAIN** para las plantillas, los datos y las reglas para imprimir el resultado.
  - Módulo **BASICO** para las estrategias de valor asignado, par asignado, valores ocultos y pares ocultos.
  - Módulo **AVANZADO** para las estrategias de la intersección y de la cruz.
- Implementar el siguiente comportamiento:
  - Las estrategias del módulo **AVANZADO** solo se usan cuando no hay posibilidad de aplicar las estrategias del módulo **BASICO**.
  - Una vez usada una regla del módulo **AVANZADO**, se tienen que volver a evaluar las estrategias del módulo **BASICO**.

Bibliografía

- Giarratano, J.C. y Riley, G.
  "Expert Systems Principles and Programming (4th ed.)",
  PWS Pub. Co., 2005.
  - Capítulos del 7 al 12
- Giarratano, J.C.
  "CLIPS User's Guide",
  http://clipsrules.sourceforge.net/OnlineDocs.html.