Representación del conocimento

Representación procedimental (CLIPS)

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

25 de junio de 2021

Conocimiento Procedimental

Temas

Conocimiento Procedimental

Una *representación procedimental,* es aquella en la que la información de control necesaria para utilizar el conocimiento se encuentra embebida en el propio conocimiento.

Aplicación de las reglas

- El conocimento es descrito en términos de *hechos* y *reglas*.
  - Los hechos describen el estado del mundo.
  - Las reglas indican las acciones que tendrán lugar cuando un estado satisfaga sus **precondiciones**.
- En general, las reglas de una máquina de inferencias pueden ser aplicadas:
  - Hacia adelante: de hechos a objetivos.
  - Hacia atrás: desde los objetivos.

Temas

Componentes de CLIPS

C Language Integrated Production System (CLIPS) consta de:

1. Lista de hechos.
2. Base de conocimento: reglas.
3. Máquina de inferencias: controla la ejecución.

http://home.agh.edu.pl/~ligeza/wiki/clips:intro

Características de CLIPS

- CLIPS utiliza únicamente encadenamiento hacia adelante: busca qué reglas aplicar a partir del estado actual.
- Cuando las precondiciones de una regla se satisfacen, la regla es agregada a una agenda, para ser disparada.
- Es inmediato programar *agentes reactivos* en CLIPS.
- Para programar *agentes basados en objetivos*, es necesario diseñar las reglas viendo a los objetivos como precondiciones que deben ser satisfechas.

Descripción del estado: Hechos

Los hechos están integrados por:

- El *nombre de la relación*
- seguido de zero o más *campos con nombre* o *rendijas* y sus valores asociados (sin importar el orden).

```
1  (person (name John Q. Public)
2          (age 23)
3          (eye-color blue)
4          (hair-color black))
```

- Es posible definir plantillas para los hechos añadibles a la memoria.
  `(deftemplate <relation-name> [<optional-comment>] <slot-definition>*)`
  Por ejemplo:

```
1  (deftemplate person ‘‘An example deftemplate’’
2      (multislot name)
3      (slot age)
4      (slot eye-color)
5      (slot hair-color))
```

- Los hechos pueden ser añadidos (`assert`) o removidos (`retract`) de la memoria. `modify` es una abreviatura que remueve y luego añade un hecho, pero con el valor de una rendija modificada.

Operadores: Reglas

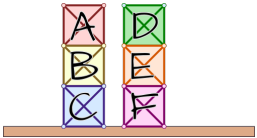
- Si todos los patrones solicitados por la regla cazan con hechos en la memoria, la regla se *activa* y es agregada a una *agenda*, la colección de reglas activadas.

```
(defrule <rule-name> [<comment>]
<patterns>* ; Left-Hand Side
=>
<actions>*) ; Right-Hand Side
```

- El RHS contiene la lista de acciones a ejecutar cuando la regla sea *disparada*.

Temas

Mundo de bloques




```
1   (deftemplate on-top-of
2     (slot upper)
3     (slot lower))
4
5   (deffacts initial-state
6     (block A) (block B)
7     (block C) (block D)
8     (block E) (block F)
9     (on-top-of (upper nothing) (lower A))
10    (on-top-of (upper A) (lower B))
11    (on-top-of (upper B) (lower C))
12    (on-top-of (upper C) (lower floor))
13    (on-top-of (upper nothing) (lower D))
14    (on-top-of (upper D) (lower E))
15    (on-top-of (upper E) (lower F))
16    (on-top-of (upper F) (lower floor))
17    (goal (move C) (on-top-of E))
18  )
```

Regla: move-directly

```
1  (defrule move-directly
2    ?goal <- (goal (move ?block1) (on-top-of ?block2))
3    (block ?block1)
4    (block ?block2)
5    (on-top-of (upper nothing) (lower ?block1))
6    ?stack1 <- (on-top-of (upper ?block1)
7                          (lower ?block3))
8    ?stack2 <- (on-top-of (upper nothing)
9                          (lower ?block2))
10   =>
11   (retract ?goal ?stack1 ?stack2)
12   (assert (on-top-of (upper ?block1)
13                      (lower ?block2))
14           (on-top-of (upper nothing)
15                      (lower ?block3)))
16   (printout t ?block1 " moved on top of " ?block2)
17 )
```

Regla: move-to-floor

```
1   (defrule move-to-floor
2     ?goal <- (goal (move ?block1) (on-top-of floor))
3     (block ?block1)
4     (on-top-of (upper nothing) (lower ?block1))
5     ?stack <- (on-top-of (upper ?block1)
6                          (lower ?block2))
7     =>
8     (retract ?goal ?stack)
9     (assert (on-top-of (upper ?block1)
10                       (lower floor))
11            (on-top-of (upper nothing)
12                       (lower ?block2)))
13    (printout t ?block1 " moved on top of floor")
14  )
```

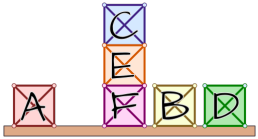
Regla: clear-upper-block

```
1  (defrule clear-upper-block
2    (goal (move ?block1))
3    (block ?block1)
4    (on-top-of (upper ?block2) (lower ?block1))
5    (block ?block2)
6    =>
7    (assert (goal (move ?block2) (on-top-of floor)))
8  )
```

Regla: clear-lower-block

```
1  (defrule clear-lower-block
2    (goal (on-top-of ?block1))
3    (block ?block1)
4    (on-top-of (upper ?block2) (lower ?block1))
5    (block ?block2)
6    =>
7    (assert (goal (move ?block2) (on-top-of floor)))
8  )
```

Ejecución




```
CLIPS> (run)
A moved on top of floor.
B moved on top of floor.
D moved on top of floor.
C moved on top of E.
```

Sintaxis

Sintaxis

- CLIPS es sensitivo al uso de mayúsculas y minúsculas.
- Las llamadas a funciones van entre paréntesis.
  Ej:

```
1 CLIPS> (+ 3 4) ↩
2 7
3 CLIPS> (watch all) ↩
```

Comandos

La secuencia básica para ejecutar un archivo[1] y observar las acciones de la máquina es:

```
1 CLIPS> (load blockworld1.cls) ↩
2 CLIPS> (watch all) ↩
3 CLIPS> (reset) ↩
4 CLIPS> (run) ↩
```

[1] NOTA: Este archivo tiene una regla sin precondiciones, cuyo efecto es insertar todos los hechos correspondientes al estado inicial.

Comandos II

CLIPS cuenta con varios comandos para seguir la evolución de las inferencias:

`(facts)` Lista los hechos en la base de conocimientos.

`(watch <watch-item>)` Donde los artículos a observar pueden ser *facts*, *rules*, *activations*, *statistics*, *compilations*, *focus* y *all*.

`(run [<limit>])` Ejecuta el programa disparando las reglas en la agenda. Es posible fijar el número máximo de reglas a disparar.

Campos

CLIPS tiene siete tipos primitivos:

- float
- integer
- symbol
- string
- external address
- instance name
- instance address

Los últimos tres se utilizan cuando el usuario define funciones en C para extender el sistema.

Refracción

- Después de que una neurona transmite sus impulsos (dispara), ningún tipo de estímulo provocará que dispare de nuevo durante un tiempo.
- En los sistemas expertos se define una regla análoga para evitar caer en ciclos trivialmente.
- CLIPS recuerda los identificadores de los hechos que dispararon una regla y no volverá a activarla con la misma combinación de identificadores.
- Para que una regla vuelva a ser disparada con los mismos hechos, se puede utilizar el comando `(refresh <rule-name>)`, que volverá a colocar las activaciones en la agenda.

Estructuras de datos

Pilas

```
1 (defrule push-value
2   ?push-value <- (push-value ?name ?value)
3   ?stack <- (stack ?name ?rest)
4   =>
5   (retract ?push-value ?stack)
6   (assert (stack ?value $?rest))
7   (printout t "Pushing value " ?value crlf))
```

Pilas (pop)

```
1  (defrule pop-value-valid
2    ?pop-value <- (pop-value ?name)
3    ?stack <- (stack ?name ?value $?rest)
4    =>
5    (retract ?pop-value ?name ?stack)
6    (assert (stack ?name ?rest))
7    (printout t "Popping value " ?value crlf))
8
9  (defrule pop-value-invalid
10   ?pop-value <- (pop-value ?name)
11   (stack ?name)
12   =>
13   (retract ?pop-value ?name)
14   (printout t "Popping from empty stack " ?name crlf))
```

Referencias I


Riley, Gary (24 de jun. de 2021). *CLIPS, A Tool for Building Expert Systems*. URL: http://www.clipsrules.net/.

Licencia