Ejemplo de implementación Razonamiento por defecto en CLISP

Juan Luis Castro

Objetivo

Implementar en CLISP un sistema que pregunte por un animal y responda si ese animal vuela o no, basado en el siguiente conocimiento:

Las aves casi todas vuelan

La mayor parte de los animales no vuelan

Las aves y los mamíferos son animales

Los gorriones, las palomas, las águilas y los pingüinos son aves

La vaca, los perros y los caballos son mamíferos

Los pingüinos no vuelan

Ideas para modelar la incertidumbre con lógica por defecto

Casi todos las aves vuelan → por defecto un ave vuela

La mayor parte de los animales no vuelan → a falta de información podríamos asumir que si no se sabe nada, un pájaro no vuele

Solución: dos valores de certeza (seguro y por_defecto)

;;;;;;;;;;;;;;;;;;;Representación ;;

; (ave ?x) representa “?x es un ave ”

; (animal ?x) representa “?x es un animal”

; (vuela ?x si|no seguro|por_defecto) representa

;“?x vuela si|no con esa certeza”

Observaciones

la única afirmación donde hay casos por defecto es en si vuela o no, por eso es ahí dónde incluimos la certeza

Hechos

;Las aves y los mamíferos son animales
;Los gorriones, las palomas, las águilas y los pingüinos son aves
;La vaca, los perros y los caballos son mamíferos
;Los pingüinos no vuelan

(deffacts datos
(ave gorrion) (ave paloma) (ave aguila) (ave pinguino)
(mamifero vaca) (mamifero perro) (mamifero caballo)
(vuela pinguino no seguro))

Reglas seguras

```
; Las aves son animales
(defrule aves_son_animales
(ave ?x)
=>
(assert (animal ?x))
(bind ?expl (str-cat "sabemos que un " ?x " es un animal porque las aves son un tipo de animal"))
(assert (explicacion animal ?x ?expl))    )

; añadimos un hecho que contiene la explicación de la deducción
```

Reglas seguras

```
; Los mamiferos son animales (A3)
(defrule mamiferos_son_animales
(mamifero ?x)
=>
(assert (animal ?x))
(bind ?expl (str-cat "sabemos que un " ?x " es un animal porque los mamiferos son un tipo de animal"))
(assert (explicacion animal ?x ?expl))     )

; añadimos un hecho que contiene la explicación de la deducción
```

Regla por defecto: añade

```
;;; Casi todos las aves vuela --> puedo asumir por defecto que las aves vuelan

; Asumimos por defecto
(defrule ave_vuela_por_defecto
(declare (salience -1))   ; para disminuir probabilidad de añadir erróneamente
(ave ?x)
=>
(assert (vuela ?x si por_defecto))
(bind ?expl (str-cat "asumo que un " ?x " vuela, porque casi todas las aves vuelan"))
(assert (explicacion vuela ?x ?expl))            )
)
```

Regla por defecto: retracta

```
; Retractamos cuando hay algo en contra
(defrule retracta_vuela_por_defecto
(declare (salience 1))        ; para retractar antes de inferir cosas erroneamente
?f<- (vuela ?x ?r por_defecto)
(vuela ?x ?s seguro)
=>
(retract ?f)
(bind ?expl (str-cat "retractamos que un " ?x ?r " vuela por defecto, porque sabemos seguro que " ?x ?s " vuela"))
(assert (explicacion retracta_vuela ?x ?expl))     )

;;; COMETARIO: esta regla también elimina los por defecto cuando ya esta seguro
```

Regla por defecto para razonar con información incompleta

```
;;; La mayor parte de los animales no vuelan --> puede interesarme asumir por defecto
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;que un animal no va a volar

(defrule mayor_parte_animales_no_vuelan
(declare (salience -2))  ;;;; es mas arriesgado, mejor después de otros razonamientos
(animal ?x)
(not (vuela ?x ? ?))
=>
(assert (vuela ?x no por_defecto))
(bind ?expl (str-cat "asumo que " ?x " no vuela, porque la mayor parte de los animales no vuelan"))
(assert (explicacion  vuela ?x ?expl))          )
```

Ejercicio

Completar esta base de conocimiento para que el sistema pregunte que de qué animal esta interesado en obtener información sobre si vuela y:

- si es uno de los recogidos en el conocimiento indique si vuela o no
- si no es uno de los recogidos pregunte si es un ave o un mamífero y según la respuesta indique si vuela o no.
- Si no se sabe si es un mamífero o un ave también responda según el razonamiento por defecto indicado