Tema 8: Meta Intérpretes

Shell para sistema basado en reglas

Contenido

- Shell Básico
- Shell Interactivo
- Shell interactivo y Reglas
- Shell Pruebas

Intérprete de reglas

- Similar a “Vanilla”: mismo estilo, granularidad
- Diferencias
  - Intérprete de lenguaje de reglas
  - Dos niveles, para permitir explicación de metas que fallan

Sintaxis hechos, reglas

<hecho> ::=hecho(<f_atomica>).
<regla> ::= regla(<cabeza>, <cuerpo>, <nombre>).
<cabeza> ::=<f_atomica>
<cuerpo> ::=<f_atomica> cierto {& <f_atomica> cierto}*

```
hecho( comportamiento(motor, no_arranca)).
hecho( indicador(bateria, 0)).

regla( estado(potencia, off), comportamiento(motor, no_arranca)
   cierto, potencia).
regla( causa(bateria, baja), estado(potencia, off) cierto &
   indicador(bateria, 0) cierto, bateria).
regla( diagnostico(X, Y), causa(X, Y) cierto, diagnostico).
```

Definición Operadores

Para que:

regla(causa(bateria, baja),

estado(potencia, off) cierto &
indicador(bateria, 0) cierto, bateria).

sea un término Prolog válido, es preciso definir los operadores:

```
:-op(40, xfy, &).
:-op(30, xf, cierto).
```

Primer nivel: Monitor

- Necesario para gestionar cómputo con fallo

```
monitor(Meta):- solve(Meta, Resul),
  filtro(Resul), nl,write(Meta).
monitor(Meta):- nl, write(Meta), write('
  no se satisface').

filtro(si).

/* por ahora
filtro(no) :- fail.
*/
```

Segundo nivel: solve

- Similar “vanilla”

```
solve(A, si) :- hecho(A).
solve(A, Resul) :-
  regla(A, B, Nombre),
  solve_cuerpo(B, Resul).
solve(A, no).
```

Procesar cuerpo regla

- Similar solve(A,B) en “vanilla”
  - Algo más complejo por gestión cómputo con fallo

```
solve_cuerpo(A&B, Resul) :-
  solve_cuerpo(A, ResulA),
  solve_and(ResulA, B, Resul).
solve_cuerpo(A cierto, Resul) :- solve(A,
  Resul).
solve_and(no, B, no).
solve_and(si, B, Resul) :-
  solve_cuerpo(B, Resul).
```

Shell básico

```
/*  ESQUELETO INTERPRETE REGLAS DOS NIVELES                          */
:-op(40, xfy, &).
:-op(30, xf, cierto).

monitor(Meta):- solve(Meta, Resul), filtro(Resul), nl,write(Meta).
monitor(Meta):- nl, write(Meta), write(' no se satisface').
filtro(si).
/* filtro(no) :- fail. */

solve(A, si) :- hecho(A).
solve(A, Resul) :- regla(A, B, Nombre), solve_cuerpo(B, Resul).
solve(A, no).

solve_cuerpo(A&B, Resul) :- solve_cuerpo(A, ResulA), solve_and(ResulA, B, Resul).
solve_cuerpo(A cierto, Resul) :- solve(A, Resul).
solve_and(no, B, no).
solve_and(si, B, Resul) :- solve_cuerpo(B, Resul).

/* Base de conocimiento */
hecho( comportamiento(motor, no_arranca)).
hecho( indicador(bateria, 0)).
regla( estado(potencia,off), comportamiento(motor, no_arranca) cierto, potencia).
regla( causa(bateria, baja), estado(potencia, off) cierto & indicador(bateria, 0)
    cierto, bateria).
regla( diagnostico(X, Y), causa(X, Y) cierto, diagnostico).
```

Ejemplo shell básico

```
hecho( comportamiento(motor, no_arranca)).
hecho( indicador(X, Y)).

regla( estado(potencia,off), comportamiento(motor, no_arranca) cierto,
   potencia).
regla( causa(bateria, baja), estado(potencia, off) cierto &
   indicador(bateria, 0) cierto, bateria).
regla( diagnostico(X, Y), causa(X, Y) cierto, diagnostico).


1 ?- monitor(diagnostico(X,Y)).

diagnostico(bateria, baja)
X = bateria,
Y = baja ;

diagnostico(_G344, _G345) no se satisface
true.
```

Shell interactivo (I)

- Permitir al intérprete solicitar información en tiempo de ejecución
- El usuario proporciona valor de verdad: **`si`** o **`no`**.
- Sólo para algunos predicados
  - Los definidos como **`preguntables`**
  - Ejemplo: **`preguntable(`**`indicador(X,Y)).`

Shell interactivo (II)

- La interacción se logra con una nueva cláusula en el nivel solve:

```
solve(A, Resul) :-preguntable(A),
  solve_pregunta(A, Resul).
```

```
solve(A, si) :- hecho(A).
solve(A, Resul) :- regla(A, B, Nombre), solve_cuerpo(B, Resul).
solve(A, Resul) :-preguntable(A), solve_pregunta(A, Resul).
solve(A, no).
```

La información preguntable es **meta-conocimiento explícito** que se puede utilizar como se necesite

Shell interactivo (III)

- `solve_pregunta`
  - Realiza pregunta al usuario
  - SÓLO si no se sabe la respuesta (metaconocimiento)
  - Almacena la respuesta (assert)

```
solve_pregunta(A, Resul):- \+ known(A), pregunta(A, Respuesta),
    responde(Respuesta, A, Resul).

pregunta(A,Respuesta) :- mostrar_pregunta(A), read(Respuesta).
mostrar_pregunta(A) :- nl,write('¿ '), write(A), write(' ?:') .

responde(si, A, si) :- assert(known_cierto(A)).
responde(no, A, no) :- assert(known_falso(A)).

known(A) :- known_cierto(A).
known(A) :- known_falso(A).
```

Modificación de predicados en tiempo de ejecución

```
:- dynamic known_cierto/1 , known_falso/1.
```

Shell interactivo (IV)

```
:- op(40, xfy, &). :- op(30, xf, cierto).
:- dynamic known_cierto/1 , known_falso/1.

/*    ESQUELETO INTERPRETE REGLAS DOS NIVELES                    */
/*                              INTERACTIVO                      */
monitor(Meta) :-  solve(Meta, Resul), filtro(Resul), nl,write(Meta), write(' se satisface'), nl.
monitor(Meta):-  nl, write(Meta), write(' no se satisface'), nl.

filtro(si).
/* filtro(no) :- fail. */

solve(A, si) :- hecho(A).
solve(A, Resul) :- regla(A, B, Nombre), solve_cuerpo(B, Resul).
solve(A, Resul) :-preguntable(A), solve_pregunta(A, Resul).
solve(A, no).

solve_cuerpo(A&B, Resul) :- solve_cuerpo(A, ResulA), solve_and(ResulA, B, Resul).
solve_cuerpo(A cierto, Resul) :- solve(A, Resul).
solve_and(no, A, no).
solve_and(si, B, Resul) :- solve_cuerpo(B, Resul).
solve_pregunta(A, Resul):- \+ known(A), pregunta(A, Respuesta), responde(Respuesta, A, Resul).

pregunta(A,Respuesta) :- mostrar_pregunta(A), read(Respuesta).
mostrar_pregunta(A) :- nl,write('¿ '), write(A), write(' ?:') .

responde(si, A, si) :- assert(known_cierto(A)).
responde(no, A, no) :- assert(known_falso(A)).

known(A) :- known_cierto(A).
known(A) :- known_falso(A).
```

Ejemplo shell interactivo

```
hecho( comportamiento(motor, no_arranca)).
preguntable( indicador(X, Y)).
regla( estado(potencia,off), comportamiento(motor, no_arranca) cierto, potencia).
regla( causa(bateria, baja), estado(potencia, off) cierto & indicador(bateria, 0)
    cierto, bateria).
regla( diagnostico(X, Y), causa(X, Y) cierto, diagnostico).

1 ?- monitor(diagnostico(X,Y)).
¿ indicador(bateria, 0) ?:no.
diagnostico(_G344, _G345) no se satisface
true.

2 ?- monitor(diagnostico(X,Y)).
diagnostico(_G337, _G338) no se satisface
true.

3 ?- retract(known_falso(X)).
X = indicador(bateria, 0).

4 ?- monitor(diagnostico(X,Y)).
¿ indicador(bateria, 0) ?:si.
diagnostico(bateria, baja) se satisface
X = bateria,
Y = baja ;

diagnostico(_G337, _G338) no se satisface
true.
```

Mejora: shell interactivo y reglas

- Permitir al usuario responder “¿Por qué?” a las preguntas del usuario
- Requiere: saber qué reglas ha utilizado el intérprete hasta realizar la pregunta
  - Todos los predicados llevan la lista de reglas como argumento adicional
  - Monitor: inicializa la lista de reglas a []
  - La lista de reglas se actualiza en la segunda cláusula solve, que es la que invoca las reglas

Shell interactivo y reglas (I)

```
:-op(40, xfy, &).
:-op(30, xf, cierto).
:- dynamic known_cierto/1 , known_falso/1.

/*  ESQUELETO INTERPRETE REGLAS DOS NIVELES                    */
/*        INTERACTIVO + REGLAS                                 */


monitor(Meta) :-
    solve(Meta, Resul, []), filtro(Resul), nl,write(Meta), write(' se satisface'), nl.
monitor(Meta):- nl, write(Meta), write(' no se satisface'), nl.

filtro(si).
/* filtro(no) :- fail. */


solve(A, si, Reglas) :- hecho(A).
solve(A, Resul, Reglas) :-
    regla(A, B, Nombre), ReglasB=[Nombre |Reglas],
    solve_cuerpo(B, Resul, ReglasB).
solve(A, Resul, Reglas) :-preguntable(A), solve_pregunta(A, Resul, Reglas).
solve(A, no, Reglas).
```

Shell interactivo y reglas (II)

```
solve_cuerpo(A&B, Resul, Reglas) :-
    solve_cuerpo(A, ResulA, Reglas), solve_and(ResulA, B, Resul, Reglas).
solve_cuerpo(A cierto, Resul, Reglas) :- solve(A, Resul, Reglas).
solve_and(no, A, no, Reglas).
solve_and(si, B, Resul,Reglas) :- solve_cuerpo(B, Resul, Reglas).
solve_pregunta(A, Resul, Reglas):-
    \+ known(A), pregunta(A, Respuesta), responde(Respuesta, A, Resul, Reglas).

pregunta(A,Respuesta) :- mostrar_pregunta(A), read(Respuesta).

responde(si, A, si, Reglas) :- assert(known_cierto(A)).
responde(no, A, no, Reglas) :- assert(known_falso(A)).

responde('¿Por qué?', A, Resul, [Regla |Reglas ]):-
    mostrar_regla(Regla), pregunta(A, Respuesta), responde(Respuesta, A, Resul, Reglas).

responde('¿Por qué?', A, Resul, []):- write('No hay más explicación posible.'), nl,
    pregunta(A, Respuesta), responde(Respuesta, A, Resul, []).

known(A) :- known_cierto(A).
known(A) :- known_falso(A).

mostrar_pregunta(A) :- nl,write('¿ '), write(A), write(' ?:') .
mostrar_regla(Regla) :- write('Por regla: '), write(Regla).
```

Ejemplo interactivo y reglas

```
hecho( comportamiento(motor, no_arranca)).
preguntable( indicador(X, Y)).
regla( estado(potencia,off), comportamiento(motor, no_arranca) cierto, potencia).
regla( causa(bateria, baja), estado(potencia, off) cierto & indicador(bateria, 0)
    cierto, bateria).
regla( diagnostico(X, Y), causa(X, Y) cierto, diagnostico).

1 ?- monitor(diagnostico(X, Y)).

¿ indicador(bateria, 0) ?:'¿Por qué?'.
Por regla: bateria
¿ indicador(bateria, 0) ?:'¿Por qué?'.
Por regla: diagnostico
¿ indicador(bateria, 0) ?:'¿Por qué?'.
No hay más explicación posible.

¿ indicador(bateria, 0) ?:si.

diagnostico(bateria, baja) se satisface

X = bateria
Y = baja ;

diagnostico(_G351, _G352) no se satisface

Yes
```

Shell Pruebas

- Ampliamos el shell para generar explicaciones en caso de éxito y de fallo
  - Éxito: la explicación es una prueba o rama éxito
    - Secuencia de reglas encadenadas hasta llegar a los hechos
  - Fallo: la explicación es un árbol de búsqueda en que todas las ramas fallan (árbol de fallo finito)

Modificaciones

- Solve
  - El nivel solve se modifica para que también obtenga la rama de búsqueda (éxito o fallo)
- Monitor
  - El nivel monitor se modifica para gestionar el árbol de búsqueda
  - Elimina ramas de éxito (ya no se necesitan)
  - Almacena ramas fallo (necesarias para explicar el fallo)

Ejemplo Pruebas I

```
hecho( comportamiento(motor, no_arranca)). hecho( indicador(bateria, 0)).
regla( estado(potencia,off), comportamiento(motor, no_arranca) cierto, potencia).
regla( causa(bateria, baja), estado(potencia, off) cierto & indicador(bateria, 0)
    cierto, bateria).
regla( diagnostico(X, Y), causa(X, Y) cierto, diagnostico).

1 ?- monitor(estado(X, Y),P).

estado(potencia, off) se satisface

X = potencia
Y = off
P = estado(potencia, off)por (comportamiento(motor, no_arranca)cierto)con
    hecho(comportamiento(motor, no_arranca)) ;

estado(_G333, _G334) no se satisface

P = fallo(estado(X, Y), [estado(potencia, off)por (comportamiento(motor,
    no_arranca)cierto)con no_confronta(comportamiento(motor, no_arranca)),
    no_confronta(estado(_G441, _G442))]) ;

No
2 ?-
```

Ejemplo Pruebas II

```
hecho( comportamiento(motor, no_arranca)). hecho( indicador(bateria, 0)).
regla( estado(potencia,off), comportamiento(motor, no_arranca) cierto, potencia).
regla( causa(bateria, baja), estado(potencia, off) cierto & indicador(bateria, 0)
    cierto, bateria).
regla( diagnostico(X, Y), causa(X, Y) cierto, diagnostico).

2 ?- monitor(causa(X,Y), P).

causa(bateria, baja) se satisface

X = bateria
Y = baja
P = causa(bateria, baja)por (estado(potencia, off)cierto&indicador(bateria,
    0)cierto)con ((estado(potencia, off)por (comportamiento(motor,
    no_arranca)cierto)con hecho(comportamiento(motor,
    no_arranca)))&hecho(indicador(bateria, 0))) ;

causa(_G320, _G321) no se satisface

P = fallo(causa(X, Y), [causa(bateria, baja)por (estado(potencia,
    off)cierto&indicador(bateria, 0)cierto)con ((estado(potencia, off)por
    (comportamiento(motor, no_arranca)cierto)con hecho(comportamiento(motor,
    no_arranca)))&no_confronta(indicador(bateria, 0))),
    causa(bateria, baja)por (estado(potencia, off)cierto&indicador(bateria,
    0)cierto)con ((estado(potencia, off)por (comportamiento(..., ...)cierto)con
    no_confronta(comportamiento(..., ...)))&no_buscado),
    causa(bateria, baja)por (estado(potencia, off)cierto&indicador(bateria,
    0)cierto)con (no_confronta(estado(potencia, off))&no_buscado),
    no_confronta(causa(_G428, _G429))]) ;
No
```

Elementos de las pruebas

- hecho(A)
- A por B con Prueba
- no_confonta(A)
- no_buscado

```
P = estado(potencia, off)por
  (comportamiento(motor,no_arranca)cierto)con
  hecho(comportamiento(motor, no_arranca))
```

Modificación solve

```
:-op(40, xfy, por).
:-op(30, xfy, con).

solve(A, si, Arbol) :- hecho(A),Arbol=hecho(A).
solve(A, Resul, Arbol) :-
            regla(A, B, Nombre),
            solve_cuerpo(B, Resul, Prueba),
            Arbol = A por B con Prueba.
solve(A, no, Arbol):- Arbol=no_confronta(A)
```

Modificación solve_cuerpo

```
solve_cuerpo(A&B, Resul, Prueba) :-
  solve_cuerpo(A, ResulA, PruebaA),
  solve_and(ResulA, B, Resul, PruebaB),
  Prueba=PruebaA & PruebaB.
solve_cuerpo(A cierto, Resul, Prueba) :-
  solve(A, Resul, Prueba).

solve_and(no, _, no, no_buscado).
solve_and(si, B, Resul, Prueba) :-
  solve_cuerpo(B, Resul, Prueba).
```

Modificación monitor

```
monitor(Meta, Prueba) :-
   inicializa_arbol, solve(Meta, Resul, Prueba),
   filtro(Resul, Prueba), nl, write(Meta),
     write(' se satisface'), nl.
monitor(Meta, Prueba):-
   eliminar_prueba(P), reverse(P, P1),
   Prueba=fallo(Meta, P1), nl, write(Meta),
     write(' no se satisface'), nl.

filtro(si, Prueba):- reinicializa_arbol.
filtro(no, Prueba) :- almacenar_prueba(Prueba),
   fail.
```

Operaciones sobre la prueba

```
eliminar_prueba(Prueba) :-
  retract(arbol(Prueba)).
almacenar_prueba(Prueba):-
  retract(arbol(Arbol)),
  assert(arbol([Prueba | Arbol])).

inicializa_arbol :- assert(arbol([])).
reinicializa_arbol :-
  retract(arbol(Prueba)),
  assert(arbol([]))
```

Una última mejora I

- Mejorar la presentación de las explicaciones:

```
1 ?- explicacion(estado(potencia, off)).

estado(potencia, off) se obtiene usando la regla:
REGLA: potencia
SI comportamiento(motor, no_arranca)
ENTONCES estado(potencia, off)

comportamiento(motor, no_arranca) es un hecho de la
   base de conocimiento.

Yes
```

Una última mejora II

```
1 ?- explicacion(estado(X,Y)).

estado(potencia, off) se obtiene usando la regla:
REGLA: potencia
SI comportamiento(motor, no_arranca)
ENTONCES estado(potencia, off)

comportamiento(motor, no_arranca) es un hecho de la base de conocimiento.

X = potencia
Y = off ;

estado(_G339, _G340), ha fallado con las siguientes ramas de fallo:

estado(potencia, off) se obtiene usando la regla:
REGLA: potencia
SI comportamiento(motor, no_arranca)
ENTONCES estado(potencia, off)

comportamiento(motor, no_arranca) no confronta con más hechos o reglas de la base
    de conocimiento.

NUEVA RAMA

estado(_G430, _G431) no confronta con más hechos o reglas de la base de
    conocimiento.

NUEVA RAMA
Yes
```

Una última mejora III a

```
1 ?- explicacion(diagnostico(X,Y)).

diagnostico(bateria, baja) se obtiene usando la regla:
REGLA: diagnostico
SI causa(bateria, baja)
ENTONCES diagnostico(bateria, baja)

causa(bateria, baja) se obtiene usando la regla:
REGLA: bateria
SI estado(potencia, off) Y indicador(bateria, 0)
ENTONCES causa(bateria, baja)

estado(potencia, off) se obtiene usando la regla:
REGLA: potencia
SI comportamiento(motor, no_arranca)
ENTONCES estado(potencia, off)

comportamiento(motor, no_arranca) es un hecho de la base de
   conocimiento.

indicador(bateria, 0) es un hecho de la base de conocimiento.

X = bateria
Y = baja ;
```

Una última mejora IIIb

```
diagnostico(_G369, _G370), ha fallado con las siguientes ramas de
   fallo:

diagnostico(bateria, baja) se obtiene usando la regla:
REGLA: diagnostico
SI causa(bateria, baja)
ENTONCES diagnostico(bateria, baja)

causa(bateria, baja) se obtiene usando la regla:
REGLA: bateria
SI estado(potencia, off) Y indicador(bateria, 0)
ENTONCES causa(bateria, baja)

estado(potencia, off) se obtiene usando la regla:
REGLA: potencia
SI comportamiento(motor, no_arranca)
ENTONCES estado(potencia, off)

comportamiento(motor, no_arranca) es un hecho de la base de
   conocimiento.

indicador(bateria, 0) no confronta con más hechos o reglas de la base
   de conocimiento.
```

Una última mejora IIIc

```
NUEVA RAMA

diagnostico(bateria, baja) se obtiene usando la regla:
REGLA: diagnostico
SI causa(bateria, baja)
ENTONCES diagnostico(bateria, baja)

causa(bateria, baja) se obtiene usando la regla:
REGLA: bateria
SI estado(potencia, off) Y indicador(bateria, 0)
ENTONCES causa(bateria, baja)

estado(potencia, off) se obtiene usando la regla:
REGLA: potencia
SI comportamiento(motor, no_arranca)
ENTONCES estado(potencia, off)

comportamiento(motor, no_arranca) no confronta con más hechos o reglas
   de la base de conocimiento.
El resto de la conjunción no se ha buscado.
```

Una última mejora IIId

NUEVA RAMA

diagnostico(bateria, baja) se obtiene usando la regla:
REGLA: diagnostico
SI causa(bateria, baja)
ENTONCES diagnostico(bateria, baja)

causa(bateria, baja) se obtiene usando la regla:
REGLA: bateria
SI estado(potencia, off) Y indicador(bateria, 0)
ENTONCES causa(bateria, baja)

estado(potencia, off) no confronta con más hechos o reglas de la base de conocimiento.
El resto de la conjunción no se ha buscado.

Una última mejora III

```
NUEVA RAMA

diagnostico(_G469, _G470) se obtiene usando la regla:
REGLA: diagnostico
SI causa(_G469, _G470)
ENTONCES diagnostico(_G469, _G470)

causa(_G469, _G470) no confronta con más hechos o reglas de la base de
   conocimiento.

NUEVA RAMA

diagnostico(_G460, _G461) no confronta con más hechos o reglas de la
   base de conocimiento.

NUEVA RAMA

Yes
```

Shell explicación

```
explicacion(Meta) :- monitor(Meta,
  Prueba), interpretar(Prueba).

monitor(Meta, Prueba) :-
  inicializa_arbol,
  solve(Meta, Resul, Prueba),
  filtro(Resul, Prueba).
monitor(Meta, Prueba):-
  eliminar_prueba(P),
  reverse(P, P1),
  Prueba=fallo(Meta, P1).
```