[Fases 3, 4] Cómo modular un programa con predicados dinámicos de Prolog

- Para implementar el código que resuelve un mapa de fase 3 o fase 4 podemos distinguir los diferentes comportamientos que puede tener Plman en el mapa y definir, para cada uno, un conjunto de reglas (sub-reglas).

- Cada conjunto de sub-reglas conformarán un "estado" o una situación particular de Plman en el mapa. Por ejemplo, podemos definir un estado cuyas subreglas implementen todas las acciones que tenga que realizar Plman cuando lleva la llave 'a'.

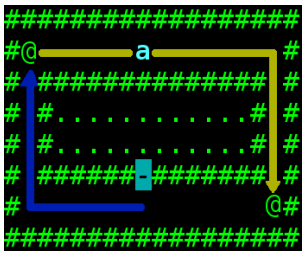
- Siempre que Plman cambie de comportamiento, cambiaremos de estado.

¿Cuántos estados se pueden/deben usar para resolver un mapa?

- Tantos como se estimen oportunos, según los diferentes comportamientos de Plman.
- En cada momento de la resolución se debe saber el estado en que se encuentra Plman.

Cada estado se declara en el programa mediante un hecho dinámico con el fin de que su/s argumento/s pueda variar durante la ejecución.

Ejemplo Solución (incompleta) de maps/fase3/mapa1.pl usando predicados dinámicos




```
:- use_module('pl-man-game/main').
s(D,O) :- see(normal, D, O).
:- dynamic estado/1.
%  hecho inicial cuyo argumento indica la situación por la que quieres comenzar
estado(subir).
% Regla que elimina / inserta el nuevo estado
change(EST) :- retractall(estado(_)),  assert(estado(EST)).
% Regla principal
do(ACT) :-  estado(EST),   do1(EST, ACT).
% Reglas para subir por la izquierda
  do1(subir,     move(left))  :- s( left, '  ').
  do1(subir,      move(up))   :- s( up, '  ').
  do1(subir,  move(right))   :-  change(bajar).
% Reglas para bajar por la derecha
  do1(bajar,  move(right))  :-  s( right, '  ' ).
  do1(bajar, move(down)) :-  s( down, '  ').
  do1(bajar,      get(right))  :-  s( right, 'a ').
```

[Predicados Dinámicos de PROLOG]

Una característica muy útil de Prolog es la posibilidad de **añadir y eliminar** cláusulas (hechos o reglas) de los predicados presentes en el programa, y hacerlo **en tiempo de ejecución**.

Los predicados con esta posibilidad se denominan **predicados dinámicos**.

Para indicar que un predicado es dinámico se le señala con la directiva **dynamic/1** en el código del programa.

Ejemplo.

Abre terminal, crea el fichero, ejemplo1.pl, por ej., con editor gedit

```
$ gedit ejemploj1.pl &
```

y añade este hecho:

```
asignatura(logica).
```

Guarda ejemplo1.pl, por ej., en Escritorio

En /Escritorio, abre SWI-Prolog

```
$ swipl
```

Comprueba que ejemplo1.pl está en este directorio:

```
$ ls
```

Si es así, realiza una consulta para incluir e interpretar el fichero en el sistema Prolog

```
? consult('ejemplo1.pl').
```

Si no hay errores, realiza una pregunta para comprobar que Prolog responde adecuadamente:

```
? asignatura(logica).
true.

? asignatura(X).
X=logica.
```

Ahora queremos insertar más asignaturas, por ej., física, mates, inglés.

Hacemos:

Abrir fichero ejemplo1.pl

```
$ gedit(ejemplo1.pl) &
```

y añadir los hechos:

```
asignatura (física).
asignatura (mates).
asignatura (ingles).
```

La base de conocimiento de ejemplo1.pl quedaría:

```
asignatura (logica).
asignatura (física).
asignatura (mates).
asignatura (ingles).
```

Si queremos eliminar alguno de estos hechos abrimos el fichero y lo quitamos.

A veces es necesario hacer las operaciones de **inclusión/eliminación de cláusulas en tiempo de ejecución**.

Para ello usamos los predicados dinámicos de Prolog.

Vemos cómo se haría con el ejemplo1.pl

Insertar HECHOS con predicados dinámicos

Para insertar cláusulas hechos de un predicado dinámico existe una familia de predicados ISO-standard:

asserta/1 Inserta una nueva cláusula al principio del programa.

assert/1 Inserta una nueva cláusula al final del programa.

Sigue estos pasos para ver cómo funcionan:

Abre un fichero, ejemplo2.pl, por ej., con editor gedit

```
$ gedit ejemplo2.pl &
```

y añade esta directiva:

```
:- dynamic asignatura/1.
```

-dynamic- permite añadir hechos del predicado -asignatura- en tiempo de ejecución.

Guarda ejemplo2.pl, por ej., en Escritorio

En /Escritorio, abre SWI-Prolog

```
$ swipl
```

Comprueba que ejemplo2.pl está en este directorio:

```
$ ls
```

Si es así, realiza una consulta para incluir e interpretar el fichero en el sistema Prolog

```
? consult('ejemplo2.pl').
```

Si no hay errores realiza una pregunta para comprobar que Prolog responde adecuadamente:

```
? asignatura(X).
false.
```

Prolog responde -false- porque no tiene información sobre hechos del predicado -asignatura-.

Incluimos un hecho:

```
? assert(asignatura(logica)).
```

Comprobamos que el hecho -asignatura(logica)- se ha añadido al código del ejemplo2.pl

```
? listing(asignatura(X)).
:- dynamic asignatura/1.
asignatura(logica).
true.
```

Incluimos otro hecho:

? assert(asignatura(mates)).

Comprobamos que el hecho -asignatura(mates)- se ha añadido al código del ejemplo2.pl

? listing(asignatura(X)).

:- dynamic asignatura/1.

asignatura(logica).
asignatura(mates).

true.

Otro hecho más usando assert/1 o asserta/1:

? asserta(asignatura(fisica)).

Y otro más:

? assert(asignatura(mates)).

Comprobamos que el hecho -asignatura(fisica)- se ha añadido al código del ejemplo2.pl

? listing(asignatura(X)).

:- dynamic asignatura/1.

asignatura(fisica).
asignatura(logica).
asignatura(mates).

true.

El **orden** en el que se insertan las cláusulas determina el orden de sucesión de las soluciones, en algunos casos puede ser **importante.**

Eliminar HECHOS con predicados dinámicos

Para eliminar las cláusulas de predicados dinámicos se usan los predicados **retract**:

retract/1	Elimina la primera cláusula que unifique (coincida) con el argumento. Siempre se elimina por el principio del programa.
rectractall/1	Elimina todas las cláusulas que unifican con el hecho.

La cláusula que se quiere eliminar debe estar instanciada (tener algún valor). Se quitará la primera cláusula de la base de conocimientos que empareje con ella. Si se vuelve a preguntar, se irán eliminando, sucesivamente, las cláusulas que coincidan. Con retractall se eliminan todas las cláusulas.

Como retract/1 borra el hecho referenciado en su argumento entonces:

retract(asignatura(logica)). Borraría el hecho asignatura(logica).

retract(asignatura(X)). Preguntaría por el valor de X que quieres borrar.

retractall/1: borraría todos los hechos indicados en su argumento

retractall(asignatura(_)) Borraría todos los hechos de -asignatura/1-

Sigue estos pasos para ver cómo funcionan:

Tenemos en cuenta el código de -ejemplo2.pl-

```
:- dynamic asignatura/1.
asignatura(logica).
asignatura(mates).
```

Eliminamos la asignatura -logica-

```
? retract(asignatura(logica)).
```

Comprobamos que el hecho -asignatura(logica)- se ha eliminado:

```
? listing(asignatura(X)).
:- dynamic asignatura/1.
asignatura(mates).
```

Es útil modificar el código en tiempo de ejecución?

Al modificar el código se provoca ilegibilidad del programa y dificultad para realizar trazas, pero se beneficia **la implementación de estado en los programas.**

Ejemplo Supongamos que necesitamos controlar la temperatura de una sala:

```
:- dynamic temperatura/1.

temperatura(23).

sensor_temperatura :-   leer_tempe(NewT),
                        rectract(temperatura(_)),
                        assert(temperatura(NewT)),
                          !,
                        sensor_temperatura.

leer_tempe(X) :- writeln(´escribe nueva temperatura
```