Paradigma Lógico

Unit Testing usando PlUnit

por Nahuel Palumbo
Fernando Dodino
Juan Contardo
Valentina Rau

revisado por Matías Freyre

Versión 1.1
Julio 2025

Indice

Pre-requisitos

Tener instalado SWI Prolog o similar.

Test individual

Para testear nuestros programas de Prolog, utilizaremos un framework de testing llamado PlUnit que ya viene con *SWI Prolog*.

Supongamos la siguiente base de conocimientos:

```
cursa(diego, pdp, dodain).
cursa(mary, pdp, gaston).
cursa(elias, pdp, juan).
cursa(celeste, pdp, juan).
cursa(mora, pdp, alf).
cursa(Alguien, pdp, dodain):-cursa(Alguien, pdp, juan).
cursa(Alguien, operativos, adriano):-cursa(Alguien, pdp, gaston).

cursadaBuena(alf, pdp).
cursadaBuena(adriano, operativos).

tuvoCursadaBuena(Alguien):-cursa(Alguien, Materia, Profesor),
  cursadaBuena(Profesor, Materia).
```

Para crear un test solo hace falta crear una regla para el predicado **test/1** que recibe como parámetro el nombre del test en forma de átomo:

```
test(persona_que_curso_con_profe_por_transitividad):-
  cursa(mary, operativos, adriano).
```

Para poder correr nuestros tests de una manera fácil y organizarlos de una manera cómoda, se recomienda encerrar nuestros tests entre las directivas **begin_tests/1** y **end_tests/1**, que esperan como parámetro algún átomo que identifique a todos los tests implicados:

```
:- begin_tests(utneanos).
 test(persona_que_curso_con_profe_por_transitividad):-
   cursa(mary, operativos, adriano).
:- end_tests(utneanos).
```

De esta forma, una vez cargado nuestro archivo de testing, corremos todos los tests con el predicado **run_tests/0**.

¿Qué estamos testeando? Que una consulta se verifique para un determinado individuo (es una consulta existencial). Idealmente el predicado que estamos testeando debe tener cierta complejidad (no vamos a testear hechos, porque por definición son ciertos).

VERDE. Podemos ver que dice que pasaron todos nuestros tests (uno solo). ¡Muy bien!

```
?- run_tests.
% PL-Unit: utneanos
Warning: /home/dodain/workspace/prolog-2021/kata1-utneanos/solucion.pl:24:
      PL-Unit: Test persona_que_curso_con_profe_por_transitividad: Test
succeeded with choicepoint
 done
% test passed
true.
```

Predicados no determinísticos

En la ejecución anterior aparece un mensaje de advertencia: "Test succeeded with choicepoint". Esto ocurre porque estamos en presencia de un predicado no determinístico: en un determinado punto hay muchas formas de continuar[1] con el siguiente paso.

```
cursa(mary, operativos, adriano).
```

falla para el árbol de soluciones posibles de este predicado:

```
cursa(Alguien, pdp, dodain):-cursa(Alguien, pdp, juan).
```

pero se verifica para este otro:

```
cursa(Alguien, operativos, adriano):-cursa(Alguien, pdp, gaston).
```

Entonces PLUnit emite un mensaje de advertencia: "en alguno de los caminos posibles este predicado se cumple, pero en otros no". Si nosotros sabemos de antemano que un predicado es no-determinístico, podemos definir el test de aridad 2:

```
 test(persona_que_curso_con_profe_por_transitividad, nondet):-
   cursa(mary, operativos, adriano).
```

[1] Al igual que ocurre con los autómatas finitos no determinísticos

Con make volvemos a ejecutar los tests, y ahora sí pasan sin ninguna advertencia:

```
?- make.
% /home/dodain/workspace/prolog-2021/kata1-utneanos/solucion compiled 0.00
sec, -1 clause
% PL-Unit: utneanos . done
% test passed
true.
```

Test individual: negación

Sabemos que Diego no tuvo una buena cursada, aun así definimos este test:

```
test(persona_que_no_tuvo_cursada_buena):-tuvoCursadaBuena(diego).
```

Al hacer make, vemos que el segundo test falla:

```
?- make.
% /home/dodain/workspace/prolog-2021/kata1-utneanos/solucion compiled 0.00
sec, 0 clauses
% PL-Unit: utneanos .
ERROR: /home/dodain/workspace/prolog-2021/kata1-utneanos/solucion.pl:29:
      test persona_que_no_tuvo_cursada_buena: failed

 done
% 1 test failed
% 1 tests passed
true.
```

Claro, lo que queremos es testear que no se cumple el predicado que escribimos a continuación. La forma recomendada es utilizar test/2 con fail como segundo parámetro:

```
test(persona_que_no_tuvo_cursada_buena, fail):-
  tuvoCursadaBuena(diego).
```

Esta opción es recomendable cuando tenés un solo predicado (tené en cuenta que si estás probando p ∧ q ∧ r y esperás que solo falle r, quizás te convenga ir por el not/1 para estar seguro de no tener un falso positivo si falla p ó q).

Inversibilidad

Es probable que a veces nos pidan que algún predicado sea inversible, y queramos tener eso testeado para asegurarnos. Por ejemplo, el predicado buenaCursada/1. La forma recomendada es indicar el set de individuos que verifica que dicha cláusula se cumple:

```
test(personas_con_buena_cursada, set(Persona == [mora, mary])):-
  tuvoCursadaBuena(Persona).
```

- Dentro de las opciones, indicamos cuál es el conjunto de individuos que satisfacen la relación para la variable Persona (solo se puede trabajar con una variable)
- La ventaja de que sea un set es que no tiene en cuenta duplicados (si aparece más de una vez mora o mary solo cuentan una sola) y tampoco importa el orden. El test pasa igual si lo definimos así:

```
test(personas_con_buena_cursada, set(Persona == [mary, mora])):-
  tuvoCursadaBuena(Persona).
```

Opción true desaconsejada

Otra variante es utilizar la opción **true/1**, que permite unificar un individuo como posible parte de la solución. No obstante esta forma de testear tiene varios problemas:

1. El predicado a testear debe ser determinístico (debe ser exitoso exactamente una vez y solo una vez, no debe dejar choicepoints como ocurrió en el primer ejemplo)
2. Debe respetarse el orden en el que aparece el individuo en la solución
3. Solo se puede testear un valor dentro del universo de individuos.

Consejos para definir los casos de prueba

- Mientras sea posible, extraer el caso de prueba y no hablar de un valor concreto: “personas_con_buena_cursada” es mejor que “mora_tuvo_buena_cursada_con_adriano”
- No tiene sentido hacer tests sobre hechos
- Cuando el predicado es no-determinístico, es confuso pensar en los casos borde porque la consulta por un camino se satisface y por otro camino falla. En esos casos es preferible concentrarse en testear la inversibilidad mediante la opción set, aun sabiendo que son tests frágiles (cualquier agregado va a romper los tests, pero es algo entendible en nuestra cursada porque nosotros modelamos **todo el universo**)