**Prolog - Unit Testing usando PlUnit**

Pre-requisitos

Tener [instalado](http://pdep.com.ar/Home/software/software-swi-prolog) SWI Prolog o similar.

Conceptos básicos

Para testear nuestros programas de Prolog, utilizaremos un framework de testing llamado [PlUnit](http://www.swi-prolog.org/pldoc/package/plunit.html) que ya viene con *SWI Prolog*.

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:

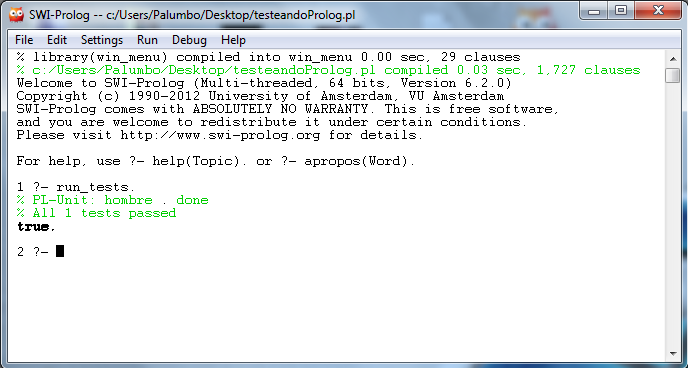
|  |
| --- |
| **test**(socrates\_es\_hombre) :-  hombre(socrates). |

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**(hombre).  **test**(socrates\_es\_hombre) :-  hombre(socrates).    :- **end\_tests**(hombre). |

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

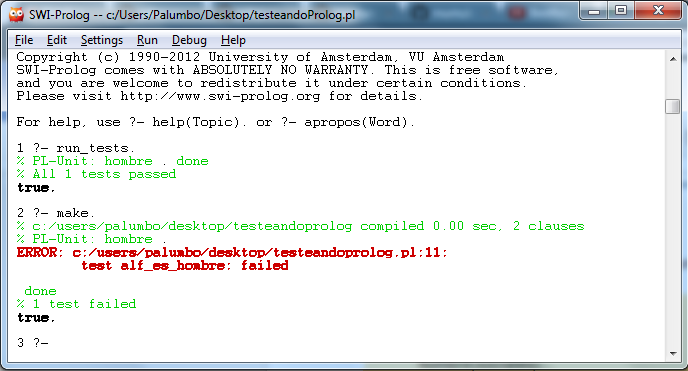
**VERDE**. Podemos ver que dice que pasaron todos nuestros tests (uno solo). ¡Muy bien!



Ahora veamos como rompe cuando agregamos un test que sabemos que es erróneo:

|  |
| --- |
| :- **begin\_tests**(hombre).  **test**(socrates\_es\_hombre) :-  hombre(socrates).  **test**(alf\_es\_hombre) :-  hombre(alf).  :- **end\_tests**(hombre). |

Al buildear nuestro programa con el predicado *make*, ya se correrán todos los tests automáticamente:

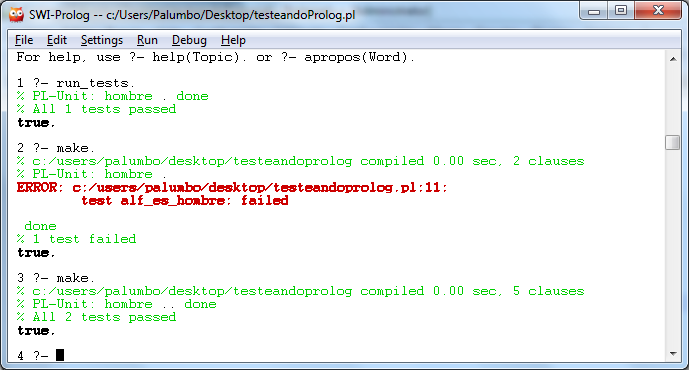


**ROJO**. Vemos como *PlUnit* nos indica que hubo tests que fallaron (uno solo) y cuál de ellos fue.

Una forma de arreglar nuestro test es negándolo, agregando *not/1* a la consulta. Sin embargo en la siguiente sección veremos una manera más ordenada de resolver este conflicto:

|  |
| --- |
| **test**(alf\_no\_es\_hombre) :-  not(hombre(alf)). |

Y así, al hacer *make*, vemos como pasan todos nuestros tests nuevamente:



Buenas prácticas

# Fail

Anteriormente vimos que si queríamos testear algún predicado que tenga que dar falso tuvimos que negar la consulta con *not/1*. Esto es un poco chocante y teóricamente malo, ya que no se está testeando la consulta que se quiere.

Para ello, *PlUnit* nos ofrece el predicado **test/2** que recibe como segundo parámetro uno o una lista de [Options](http://www.swi-prolog.org/pldoc/man?section=unitbox), entre ellos **fail**, que indica que la consulta tiene que dar falsa.

Sabiendo esto, podemos cambiar nuestro test feo:

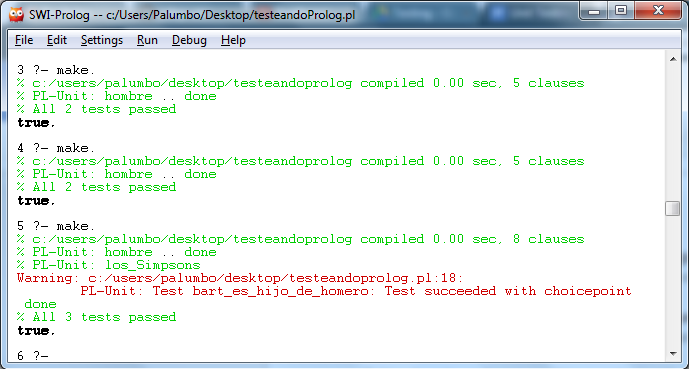
|  |
| --- |
| **test**(alf\_no\_es\_hombre, **fail**) :-  hombre(alf). |

# Nondet

Seguramente muchas veces queramos testear algún predicado NO [determinístico](http://es.wikipedia.org/wiki/Modelo_determinista), como por ej:

|  |
| --- |
| :- **begin\_tests**(los\_Simpsons).  **test**(bart\_es\_hijo\_de\_homero) :-  hijoDe(homero, bart).    :- **end\_tests**(los\_Simpsons). |

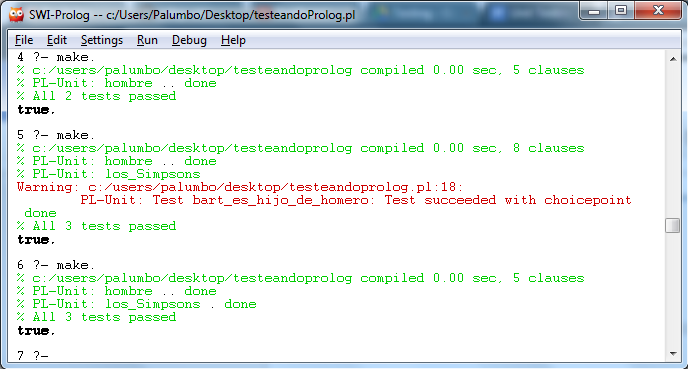
Y al correr nuestros tests, si bien todos pasan, nos tira un **Warning**.



Para evitar eso, basta con agregar el *Option* **nondet** al predicado:

|  |
| --- |
| **test**(bart\_es\_hijo\_de\_homero, **nondet**) :-  hijoDe(homero, bart). |

Y de esa manera le decimos a *PlUnit* que ya sabemos que nuestro predicado no es determinístico y nos deja tranquilos:



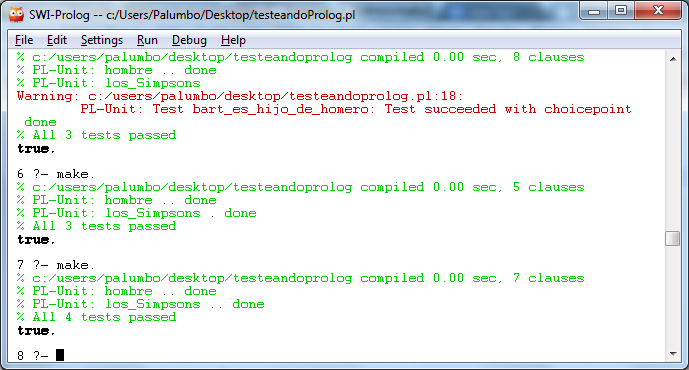
Conceptos avanzados

# Inversibilidad

Es probable que a veces nos pidan que algún predicado sea inversible, y queramos tener eso testeado para asegurarnos.

Para lograr esto existe el *Option* **true/1**, que evalúa que al finalizar el test se cumpla la expresión pasada por parámetro (como en este caso homero tiene más hijos además de bart no es determinístico, por lo cual también hay que indicar eso para que no nos tire el Warning; en forma de lista):

|  |
| --- |
| **test**(un\_posible\_hijo\_para\_homero\_es\_bart,  [**true**(Hijo == bart), **nondet**]) :-  hijoDe(homero, Hijo). |



Ahora, si queremos dar un paso más para saber que todos los valores posibles para la consulta sean correctos, podemos utilizar **set/1**. Este *Option* junta todos los valores que den cierta la consulta en una lista y la unifica a nuestra variable (sin importar el orden ni repetidos).

Así podemos hacer:

|  |
| --- |
| **test**(se\_puede\_saber\_todos\_los\_hijos\_de\_homero,  **set**(Hijos == [lisa, bart, maggie])) :-  hijoDe(homero, Hijos). |

Y vemos como pasan todos los tests:

