Preguntas Frecuentes

Uso del depurador en Ciao Prolog

- Desde dentro de un IDE – Emacs, VSC, o el Playground:

  La manera más fácil de activar y usar el depurador es desde Emacs, VSC o el Playground: abrir el fichero `.pl` que se quiere depurar y, usar el menu CiaoDbg→'(Un)Debug buffer source', botón depuración, o More→Debug. Esto introduce los siguientes comandos en el top level, que son los mismos que se necesitan para activar el depurador a mano en el top level (suponemos que el fichero que queremos depurar es 'code.pl'):

```
?- debug_module_source(code).
{No module is selected for debugging}
{Modules selected for source debugging: [code]}
yes
?- trace.
{The debugger will first creep -- showing everything
   (trace)}
yes
{trace}
?- use_module(code).
yes
{trace}
?-
```

  Explicación de los comandos: `debug_module_source(code)` le dice al sistema que queremos depurar ese fichero. `trace` le dice al depurador que se ponga en modo traza (en el que para en todos los puntos). Luego hay que hacer `use_module(code)` para que se recompile ese módulo en modo debug. A partir de ahi el depurador estará activo para ese módulo en las consultas que hagamos después de eso.

- Fuera de un IDE: Como se ha comentado, los mismos comandos de arriba se le pueden dar manualmente al top level para activar el depurador manualmente fuera del IDE (p.ej., si corremos el top level en un terminal).

Nota: para usar el depurador se recomienda correr el programa en el modo estándar de Prolog, es decir, desactivar por ejemplo el paquete de ejecución en anchura.

El manual del depurador está en:

`https://ciao-lang.org/ciao/build/doc/ciao.html/debugger_doc.html`

o, por ejemplo, desde dentro de Emacs,

`M-x info` → `Ciao` → `PART I - The program development environment` → `The interactive debugger`.

Deliverit

Para preparar vuestro código para los tests de Deliverit, probadlos bien primero en el top level de vuestra instalación del sistema o en el playground:

1. Probad a **llamar en el top level a los predicados de diferentes maneras**, por ejemplo con varios o todos argumentos que sean variables, con varios o todos argumentos cerrados (ground, es decir, términos sin variables), con argumentos que no son valores correctos o que son inconsistentes entre sí para los que el predicado por tanto debe fallar, etc.

2. Acordaos también siempre de **pedir más soluciones** (poniendo `';'<enter>` tras cada solución) y aseguraos que las alternativas, en su caso, son las que tengan que ser, y que al final, después de dar todas las respuestas, el predicado falla diciendo no y no entra en un bucle. Esto último no es siempre sencillo o necesario, pero siempre ayuda al menos intentarlo y comprobar el comportamiento.

3. Si en los tests hay **errores de timeout** o, sobre todo, de **exceso de memoria**, en cuyo caso los tests abortan, es generalmente porque hay bucles en el programa y el sistema llena la memoria disponible.

   Pistas para localizar este tipo de errores: en vuestro ordenador, desde el top-level, haced consultas de diferentes tipos como se indica arriba. Si haciéndolo veis que vuestro programa tarda demasiado o entra en un bucle infinito:

   - Aseguraos de que los casos base (los no recursivos) van antes que las otras cláusulas.
   - Si sigue habiendo bucles o el programa tarda demasiado, pensad en el orden de los literales en los cuerpos de las cláusulas: recordad que la generación de soluciones va de izquierda a derecha, y que hay que poner primero las llamadas que tengan menos soluciones o que estén más instanciadas (tengan más argumentos con datos y no variables cuando se llaman) para que puedan devolver pocas soluciones y así alimentar sólo esas pocas a las siguientes y reducir así su espacio de búsqueda.

LPdoc

1. Generación de ficheros `.pdf` con LPdoc:

   Para generar pdf directamente de LPdoc (`-t pdf`) tenéis que instalar antes algunas dependencias. Buscad en Moodle las instrucciones para generar manuales y ejecutar tests.

   Concretamente:

   - Para generar el pdf directamente desde LPdoc hay que tener instalada una distribución de TeX/LaTeX tal como TeX Live, etc. (dependiendo de la distribución podéis tener que instalar texlive, texinfo, e imagemagick). En Linux o Windows/WSL:
     `$ sudo apt install texlive texinfo imagemagik`
     En Mac OsX:
     `$ brew install texlive texinfo imagemagik`

   Alternativamente, podéis generar el manual en html (la opción por defecto) y luego, en un navegador, abrir el manual en html, y luego "imprimir" la página a fichero pdf (en la mayoría de navegadores se hace seleccionando "pdf file" como impresora, o similar).

2. Visualización del fichero de salida en Windows/WSL:

   Una vez generado el manual con LPdoc en formato html (con `lpdoc code.pl`) la forma más fácil de visualizarlo en Windows/WSL es con el comando `wslview code.html/code.html`

3. Opciones de LPdoc para que se incluyan los tests en la documentación:

   Si tenéis tests en el código está bien que salgan también en la documentación. Para hacer que LPdoc los incluya, tenéis que seleccionar la siguiente opción:

   - `lpdoc --doc_mainopts=tests code.pl`
   - O, si estáis usando un fichero `SETTINGS.pl` para poner las opciones de LPdoc lo podéis poner también ahí.
   - O también, en Emacs, en el buffer `*LPdoc*` podéis incluirlo como opción en el comando para lpdoc:
     `?- doc_cmd('SETTINGS.pl',[name_value(doc_mainopts,tests)],gen(html)).`

   Esto está en el capítulo 'Admissible values for the documentation configuration options' del manual de LPdoc.

4. Inclusión de acentos en la documentación:

   Podéis poner los acentos directamente usando utf8, y normalmente saldrán correctamente en la salida de lpdoc.

   Un método alternativo muy portátil es poner las tildes en el formato nativo de LPdoc (similar al de LaTeX y texinfo):

```
:- doc(title, "Programci@'{o}n L@'{o}gica").
```

Este método puede ser más engorroso (dependiendo del editor) pero tiene la ventaja de que mantiene el fuente en ASCII, generando acentos en la salida.

En todo caso el que no os salgan bien los acentos no importa para la práctica.

5. Los campos de `alumno_prode/4` ponedlos siempre entre comillas simples para aseguraros de que los argumentos son átomos y no cadenas de caracteres (podéis poner acentos si queréis):

   ```
   alumno_prode('Álvaro', 'Fernández', 'Gómez', 'X100100').
   ```

6. Cómo hacer para que aparezca el **nombre y número de matrícula del autor en la portada** del documento si se genera con LPDoc. Lo más fácil es usar la opción `author` de `:- doc`:

   - `:- doc(author, "Fernando Smith, X105792").`