Respuestas a dudas de la **Segunda práctica** (Versión 1)

Recopilación de respuestas a dudas planteadas en los diferentes grupos sobre la práctica 2:

- Os recordamos que el fichero `codigo.pl` tiene que empezar con la línea:
  `:- module(_,_).`
  para que el corrector pueda importar los hechos con vuestros nombres y vuestros predicados.
- Os recordamos que lo manuales de Ciao están en `http://ciao-lang.org/documentation.html`(hay un puntero en la página principal de Moodle "Documentación de Ciao Prolog"), o, más cómodo aún, si estáis en `Emacs` y habéis instalado Ciao correctamente (con todas las dependencias y generando los manuales), los tenéis dentro de Emacs haciendo `M-x info`, donde os debería aparecer un área de manuales de Ciao.
- Hemos visto que varios estabais teniendo dificultades con el caso de prueba 2.6 en la P2:

  ```
  T2.6: ejecutar_instruccion: el predicado debe ser capaz de deducir
  la instruccion que permite cambiar de un estado a otro.
  ```

  Se refiere al modo en que damos el estado inicial y el final y se generan por backtracking instrucciones que producen esa transición. Siempre está bien que os funcionen las cosas en otros modos además del más directo, pero como este modo no se pide explícitamente en ese apartado, y algunos estaban teniendo dificultades con ello, hemos decidido quitar el test 2.6 de la batería de pruebas, para que os sea más fácil pasar todas las pruebas para los primeros dos predicados. De todas maneras, tened en cuenta que, aunque ese modo de uso no se pida, según cómo programéis el último predicado (hay varias maneras de hacerlo) ese modo en que se generan instrucciones puede quizá ser útil.
- Se pueden crear todos los predicados adicionales que necesitéis, no sólo los que se indican en el enunciado.
- Se pueden usar predicados predefinidos o de librerías, siempre que sean Prolog estándar.
- Cuando se dice que las estructuras regs/n sólo pueden tener constantes válidas en Prolog, se refiere la estructura regs/n que se da como resultado en el predicado generador_de_codigo/3. Para poder operar con estos registros, os pedimos que implementéis ese predicado intermedio, eliminar_comodines/3, que sí que puede contener variables.
- Si os aparece el mensaje: `Warning: clauses of ejecutar_instruccion/3 are discontiguous.` el compilador os está avisando de que tenéis cláusulas así:

  ```
  p(a).
  q(b).
  p(a).
  ```

  en vez de:

  ```
  p(a).
  p(a).
  q(b).
  ```

  es decir, que las cláusulas de p/1 no son contiguas. Es no está mal necesariamente (por eso no es un error) pero generalmente es raro (de ahí el warning) y se debe a un gazapo como:

  ```
  patata(a).
  patat(b).
  patata(a).
  ```

  Si el predicado discontiguo es intencional, con poner:

```
:- discontiguous(patata/1).
```

se quita el warning.

- En el depurador os recomendamos por simplicidad usar spypoints. Una vez activado el depurador haced:

  ```
  ?- spy(predicado/aridad).
  ```

  por ejemplo:

  ```
  ?- spy(r/1).
  {Spypoint placed on 'foo:r'/1}
  yes
  {trace}
  ```

  Y ahí ya podéis usar "l" (leap), que es "saltar hasta la siguiente llamada a un predicado marcado con spy/1". Una vez el depurador se pare en ese predicado, en vez de hacer "l", haced <enter>y el depurador seguirá a partir de allí paso a paso. Si os queréis saltar una llamada, haced "s"(skip). Si queréis entrar <enter>. Si queréis volver a ejecutar una llamada que ha fallado, para ver por qué falla, hacer r"(redo).

  Se pueden usar también los breakpoints, pero es más fácil e intuitivo al principio usar spy/1, porque los breakpoints interactúan con los números de línea.

  Y siempre podéis usar también el método clásico de imprimir términos durante la ejecución con la librería write `http://ciao-lang.org/ciao/build/doc/ciao.html/write.html`

- Algunos habéis preguntado sobre cómo comprobar en los test varias soluciones. Los tests tienen una serie de pseudo-propiedades especiales que se pueden poner en los tests para generar múltiples soluciones, etc. Se pueden ver en el manual de los tests, que viene con el bundle `ciadbg`. Se puede ver con `M-x info` en `Emacs`, o en `http://ciao-lang.org/ciao/build/doc/ciaodbg.html/unittest.html`

  En particular, podéis probar con `try_sols`. Si ponéis:

  ```
  :- test p(X,Y) : (X=a) + (try_sols(2), not_fails).
  p(a,1).
  p(a,2).
  ```

  llamará a p/2 con X=a e intentará obtener *hasta* 2 soluciones (si hay menos no falla, pro prueba hasta 2).

  Si queréis comprobar varias salidas podéis poner por ejemplo una disyunción:

  ```
  :- test p(X,Y) : (X=a) => (Y=1; Y=2) + (try_sols(2), not_fails).
  ```

  o una llamada a `member/2`:

  ```
  :- test p(X,Y) : (X=a) => (member(Y,[1,2]) + (try_sols(2), not_fails).
  ```

Otras preguntas y respuestas:

- ¿Se debe permitir que en regs haya una variable como primer parámetro? Por ejemplo: eliminar_comodines(regs(X,*),Y,Z).

  - No, no se deben aceptar entradas con variables en los argumentos de regs en eliminar_comodines/3 (sí luego como salida).

- No me funciona `?- arg(N, libro(autor, titulo), autor).`

- Ese modo de ejecución para arg/3 no es estándar. Puedes consultar la documentación para ver las formas en las que puedes usar arg/3.

- Al hacer los tests de eliminar_comodines(Registros, RegistrosSinComodines, ListaSimbolos), estamos comprobando que RegistrosSinComodines tenga variables anónimas en vez de *. Al probarlo, si metemos:

  ```
  eliminar_comodines(regs(+ ,5 ,*), regs(+ ,5,*), [+, 5]).
  ```

  Da cierto, ya que nuestro programa devuelve RegistrosSinComodines = regs(+ ,5,_), que unifica con regs(+ ,5,*). No sabemos cómo comprobar que efectivamente RegistrosSinComodines devuelve variables anónimas y no cualquier otro elemento que pueda unificar con esas variables anónimas.

  - Las variables anónimas es una "ayuda" o "azúcar sintáctico" que se da al programador, por convención. En realidad, cuando nosotros escribimos en un programa '_', por ejemplo: `p(_ ,_ ,_ ,_).` El intérprete internamente lo representa con: `p(_1,_2,_3,_4).` Dicho de otra forma, al escribir _, estamos diciendo que ese argumento es una variable nueva y no unificada con ninguna otra. Hago esta puntualización, porque el intérprete no distingue si tú escribes "`p(A).`" o "`p(_).`". Por tanto, no se puede comprobar que algo sea una variable anónima. Lo que sí que puedes comprobar es que un término sea una variable, que es lo que se pide para este predicado. Esto lo podéis hacer con el predicado var/1.

- En el enunciado de la práctica pone: "Cada registro puede contener un único símbolo (constante) de un alfabeto finito, definido por el programador" Suponemos que hay que asegurarse de que en cada registro debe haber solo un símbolo, pero no encontramos la manera de verificar esta condición.

  - No hay que comprobar que el término solo tenga un carácter. Solo hay que comprobar que en los registros hay constantes.

- Cuando se llama a ejecutar_instruccion, dandole como argumento un regs inicial y una instrucción, el predicado podría devolver más de un resultado. Por ejemplo en la instrucción que habéis puesto en la práctica del swap, al ejecutar podría devolver regs(* , * , * , * , *) y seguiría siendo válido no?

  - ejecutar_instruccion sólamente realiza la acción especificada por el significado de la instrucción. Por ejemplo, para swap, como se dice en el enunciado: "swap(i,j) para i <j que intercambia el contenido de los registros ri y rj." Por lo tanto sólo hay una acción posible en este caso y una solución posible, que es el resultado de intercambiar las posiciones i con j. Lo que comentas sería cierto si se pidiera que el predicado ejecutar_instruccion produjera un estado de los registros compatible o algo similar.

- No entendemos por que ejemplo en concreto no unifica, dado que parece que al aplicar el SWAP sobre las posiciones 1 y 2 el resultado es el que se encuentra en la variable. ¿Es esto debido a que el * se mantiene?

  ```
  ?- ejecutar_instruccion(regs(1,2,+,5,*),swap(1,2),ES).
  ES = regs(2,1,+,5,*) ? ;
  no
  ```

  - Con este ejemplo queríamos mostrar que sólo debe tener una solución, que es la primera. ejecutar_instrucción devuelve 'no', al pedir la segunda solución (con el ;). Como comentario adicional, cuando hablamos de unificación, nos referimos al proceso de igualar variables y términos, por ejemplo:

    ```
    p(f(X,a)) = p(f(a,Y) va a unificar X con 'a', e Y con 'b'.
    ```

    Aunque el ejemplo que ponemos en la práctica fallara no se puede decir que no unifica, como dices en tu pregunta, si no hay un programa definido. Sólo se puede hablar de que un término no unifique con otro o que un objetivo no unifique con la cabeza de una cláusula. La consecuencia de que dos términos no unifiquen puede ser que un predicado falle, por ejemplo:

```
p(X) :- a = b.
```

Al no unificar a y b, este predicado siempre va a fallar. Sin embargo también puedo escribir lo contrario:

```
p(X) :- \+ a = b.
```

Pero aquí, el hecho de que no unifiquen, hace que el predicado siempre tenga éxito.

Preguntas sobre evaluación:

- ¿El número de pruebas que pase es proporcional a la nota de la practica?
  - No exactamente, pero os puede dar una idea. Luego también cuentan la memoria y otros factores como estilo de programación, uso de tests y documentación, etc.
- Tengo una pregunta con respecto a los criterios de evaluación. Por lo que entiendo del correo que se envió, si hacemos las 2 prácticas y damos el examen de las prácticas, ¿no tenemos que dar el final?. Es decir, ¿la evaluación continua consta, únicamente, de las notas de las 2 prácticas y el examen de las mismas?
  - Sí, con hacer las dos prácticas y el examen de prácticas valdría. Estamos dando la opción de hacer también el examen de evaluación no continua por los problemas que hayan podido surgir, pero si estás aprobado en el modo de continua no hay ninguna necesidad de de hacer el segundo.
- ¿Se puede entregar la primera práctica ahora, si no se entregó antes?
  - Hemos estado permitiendo entregar la primera práctica con retraso (con una penalización) pero entendemos que no tiene sentido entregarla ya en este punto porque se ha corregido en clase y hay un vídeo explicando cómo hacerla. Sin embargo, las prácticas hacen media entre ellas y luego con el examen de prácticas y se podría aprobar con una nota razonable en la segunda práctica y en el examen de prácticas (p.ej., $((0+8)/2+6)/2=5$)). Recomendamos esta solución (y hacer también la práctica 1 aunque no se entregue) como mucho mejor que ir a examen final ya que al ser una asignatura fundamentalmente de programación la mejor manera de lograr los conocimientos necesarios (y para pasar cualquiera de los dos exámenes) es haciendo las prácticas.