Práctica 2: Programación en ISO Prolog

CPU Anular

Considérese una CPU que cuenta con N registros $r_1,\ldots, r_n$ con $n \geq 2$ organizados de forma anular. Cada registro puede contener un único símbolo (constante) de un alfabeto finito, definido por el programador. Esta máquina consta únicamente de dos instrucciones (representadas por las estructuras enfatizadas en negrita), a saber:

- ***move(i)*** que copia el contenido del registro r_i en el registro r_{i+1} para $1 \leq i < n$, y de r_n a r_1 para $i=n$.
- ***swap(i,j)*** para $i < j$ que intercambia el contenido de los registros r_i y r_j.

El problema planteado en esta práctica consiste en construir un programa que permita generar automáticamente una lista de instrucciones que induzca a la máquina a transitar desde un estado inicial conocido (es decir, a partir de un conjunto conocido de valores para los registros) hacia un estado final determinado (de nuevo representado por un conjunto de valores conocidos para los registros). Es posible que tanto en el estado inicial como en el estado final alguno de los registros pueda tener un contenido indeterminado. Esto se representará mediante el símbolo * (comodín), que en el estado inicial significa que *"no sé cual es el valor que contiene el registro",* y en el estado final significa *"me da igual el valor contenido en el registro"*.

El anillo de registros se representará mediante la estructura ***regs/n*****,** donde el registro asociado al argumento i-ésimo módulo n de esta estructura está conectado a los registros correspondientes a los argumentos (i-1)-iésimo módulo n e (i+1)-ésimo módulo n. Por ejemplo, una CPU anular con 5 registros, donde r1, …, r5 contienen los términos 1, 2, +, 5, * respectivamente se representaría mediante la estructura siguiente:

*regs(1 ,2, +, 5, *)*

Debe tenerse en cuenta que un comodín en cualquier registro del estado inicial implica necesariamente que su contenido tiene que ser modificado para alcanzar el estado final cuando ese mismo registro contiene un símbolo diferente del comodín en el estado final especificado. Para que una solución sea correcta será necesario que todos los símbolos (no comodines) especificados en el estado final coincidan. Si no es así, la solución no sería correcta. Por ejemplo, para un supuesto donde el estado inicial fuese *EI = regs(a,*,c)* y el estado final fuese *EF = regs(c,a,*),* el programa *P = [move(3)]* no sería correcto, dado que al ejecutar la instrucción *move(3)* desde el estado inicial EI = *regs(a,*,c)* se obtiene el estado ET = *regs(c,*,c)*, que al compararlo con el estado final esperado EF = *egs(c,a,*)* se podría pensar que son estados equivalentes, pero esto no es así. Si se compara cuidadosamente ET con EF puede comprobarse que i) el contenido de *r1* es el mismo en ambos estados, y por tanto hay coincidencia, ii) en EF el contenido de *r3* nos es indiferente, por lo

que siempre habrá coincidencia independientemente del valor de *r3* en ET y iii) el valor de *r2* en ET es indeterminado, pero se requiere que *r2* contenga el símbolo *a* en EF para que la solución sea válida. Dado que el valor de *r2* es indeterminado en ET, este podría efectivamente ser *a* pero tampoco no serlo, por lo que se consideraría que NO hay coincidencia y esta solución sería incorrecta.

Una manera sencilla de tratar los comodines consiste en sustituirlos por variables. Para ello, se pide al alumno que escriba el predicado ***eliminar_comodines/3***, con cabecera ***eliminar_comodines(Registros, RegistrosSinComodines, ListaSimbolos)***, que es cierto si *RegistrosSinComodines/n* es una estructura de tipo *reg/n*, que resulta de sustituir los comodines que aparecen en el argumento *Registros/n* (también una estructura de tipo *reg/n*) por variables. *ListaSimbolos* es una lista que contiene todos los símbolos utilizados en el término *Registros/n* en el mismo orden en los que estos aparecen en los registros, permitiéndose que haya símbolos repetidos. Por ejemplo:

?- eliminar_comodines(regs(1,1,+,5,), R, L).*

R = regs(1,1,+,5,_)
L = [1, 1, +, 5];
no

Nótese que como no puede haber ningún símbolo en el estado final (ni tampoco en ningún estado intermedio) que no esté en el estado inicial, *ListaSimbolos* contiene efectivamente el alfabeto utilizable por el programa, que puede ser utilizado en el resto de predicados para hacer las comprobaciones pertinentes si se considera oportuno.

Dado que el movimiento entre estados se produce mediante la ejecución de instrucciones, se pide al alumno que escriba el predicado ***ejecutar_instrucción/3,*** con cabecera ***ejecutar_instruccion(EstadoActual, Instruccion, EstadoSiguiente)*** que materializa la transición entre los estados actual y siguiente mediante la ejecución de una instrucción. Por ejemplo:

?- ejecutar_instruccion(regs(1,2,+,5,),swap(1,2),ES).*

ES = regs(2,1,+,5,) ? ;*

no

Para finalizar, se pide al alumno programar el predicado ***generador_de_codigo/3*** con cabecera ***generador_de_codigo(EstadoInicial, EstadoFinal, ListaDeInstrucciones)*** que sera cierto si *ListaDeInstrucciones* unifica con una lista de instrucciones de la CPU que aplicadas secuencialmente desde el estado inicial de los registros representado por *EstadoInicial* permite transitar hacia el estado final de los registros representado por *EstadoFinal*.

Como se ha mencionado anteriormente, los diferentes estados del problema se representarán mediante la estructura N-aria ***regs/n***, cuyos argumentos representan el contenido de los registros $r_1,\ldots, r_n$ respectivamente. El contenido de los registros puede ser cualquier constante válida en Prolog, incluyendo el símbolo * utilizado para los comodines.

Dado que en general existen multiples soluciones para el problema, bastará únicamente con que el programa devuelva una única solución válida siempre y cuando esta sea mínima (es decir, no debe existir ninguna solución alternativa que conste de menos instrucciones que la lista de instrucciones entregada por el programa como solución al problema), y que el programa no entre en un bucle infinito tras la obtención de la primera solución. No obstante, solamente será posible obtener la maxima calificación si el programa entregado devuelve todas las soluciones mínimas existentes (podría haber varias) por vuelta hacia atrás sin quedarse atrapado en bucles infinitos.

A continuación se proporcionan varios ejemplos de las salidas esperadas del programa:

?- generador_de_codigo(regs(a,b,c,d),regs(a,d,a,b),L).

L = [swap(1,2),swap(1,4),move(2),swap(1,2)]

?- generador_de_codigo(regs(a,*,c),regs(c,a,*),L).

L = [swap(1,2),swap(1,3)]

?- generador_de_codigo(regs(a,b,c),regs(a,a,*),L).

L = [move(1)]

Normas

Esta práctica puede hacerse en grupos de 3 personas, siendo posible también hacerla de forma individual. Todos los miembros del grupo deben pertenecer al mismo grupo de teoría. Uno de los alumnos del grupo asumirá el rol de portavoz del grupo, y será el encargado de subir la práctica a tanto a Moodle como Deliverit (http://vps142.cesvima.upm.es). Es obligatorio subir la práctica a ambos sistemas. Es importante que UNICAMENTE EL PORTAVOZ del grupo suba la práctica a Moodle y Deliverit. Una vez asumida la portavocía de grupo, esta se mantiene para las prácticas subsiguientes. El código debe entregarse en un fichero denominado exactamente codigo.pl donde deben estar implementados todos los predicados requeridos en el enunciado. Dado que esta práctica será corregida utilizando un corrector automático, es

OBLIGATORIO que el functor, aridad y orden de los argumentos DE TODOS LOS PREDICADOS mencionados en el enunciado sea IDÉNTICO al indicado en el enunciado. Si esto no se hace así, el corrector automático rechazará la práctica y esta se calificará como suspenso. El fichero codigo.pl debe estar DEBIDAMENTE COMENTADO, explicando la semántica de cada uno de los predicados desarrollados en la práctica. Dentro del fichero codigo.pl se indicará quienes son los componentes del grupo de prácticas mediante la inclusión de tantos hechos de tipo alumno_prode/4 como alumnos haya en el grupo. Estas estructuras, con cabecera:

alumno_prode(Apellido1,Apellido2,Nombre,NumMatricula)

indican al corrector automático los datos de cada componente del grupo. Por ejemplo, si uno de los componentes del grupo se llama Ignacio Javier García Lombardía, con número de matrícula D16025, entonces será necesario incluir el hecho:

alumno_prode('Garcia', 'Lombardia', 'Ignacio Javier', 'D160125')

en la base de hechos del programa. Es SUMAMENTE IMPORTANTE seguir este paso, ya que de no hacerse correctamente, el corrector automático no podrá discernir quien es el autor de la práctica, y por tanto, la rechazará y como consecuencia la calificará como suspensa. Es responsabilidad de cada uno de los componentes del grupo el asegurarse de que estos hechos están correctamente declarados antes de entregar la práctica.

Debe también adjuntarse un fichero con la memoria de la práctica, que debe llamarse memoria.pdf o memoria.txt. La memoria debe indicar también claramente los nombre de los alumnos que forman el grupo, así como sus números de matrícula. La memoria debe incluir:

a) el código, junto con la explicación y justicación de las decisiones;

b) las consultas realizadas con el programa y las respuestas obtenidas con dichas consultas; y

c) opcionalmente, cualquier comentario aclaratorio que se estime oportuno. La memoria también se puede generar a partir del código con la herramienta lpdoc y las consultas del apartado b) se pueden expresar como un conjunto de aserciones test. Si se ha generado la memoria con Lpdoc, la memoria se acepta en formato pdf, html, o un txt que explique como se ha generado la memoria a partir de codigo.pl