Trabajo Práctico N°3: Programación Lógica

Paradigmas de Lenguajes de Programación
1er cuatrimestre 2021

Fecha de entrega: 29 de junio de 2021 (antes de las 17 horas)

Introducción

En este trabajo modelaremos utilizando `Prolog` el lenguaje de programación $\mathcal{S}$ estudiado en la materia Lógica y Computabilidad. A continuación representamos un repaso rápido por las características del lenguaje. Para más detalles, consultar el documento "CartillaS.pdf". **Es importante que, salvo donde se indique lo contrario, los predicados que escriban no generen soluciones repetidas ni se cuelguen**. Está permitido utilizar `cut` y los metapredicados de la sección útil de la página de la materia.

El lenguaje $\mathcal{S}$

El lenguaje $\mathcal{S}$ es imperativo y trabaja con un conjunto mínimo de instrucciones. Tiene infinitas variables de entrada X_i, infinitas variables temporales Z_i y una variable de salida Y. Todas las variables son naturales y comienzan en 0, salvo aquellas de entrada que ya tengan un valor asignado. Un programa en $\mathcal{S}$ es una lista de instrucciones donde cada instrucción puede estar etiquetada. Las instrucciones posibles son:

1. $V \leftarrow V$, que no hace nada.
2. $V \leftarrow V + 1$, que le suma 1 a la variable V.
3. $V \leftarrow V - 1$, que le resta 1 a la variable V (si V valía 0, no hace nada).
4. $\mathsf{IF}\, V \neq 0\, \mathsf{GOTO}\, E$, que compara el valor de la variable V con cero y si son distintos salta a la instrucción etiquetada con E.[1]

Ejecución de un programa

A medida que se ejecuta un programa se puede observar el **estado** de sus variables. Un estado contiene ecuaciones de la forma $V = m$, una por cada variable, que indican el valor de cada variable en ese estado. Llamamos **descripción instantánea** a un par compuesto por un número natural (que indica el número de la próxima instrucción a ejecutar) y un estado. La descripción instantánea inicial tiene número 1 y todas las ecuaciones del estado igualan las variables a 0, salvo aquellas de entrada que ya tengan un valor asignado. Cuando el número de instrucción pasa a ser superior a la longitud del programa (es decir, la cantidad de instrucciones del mismo), este termina, y el resultado de la ejecución es el valor correspondiente a Y en el estado final.

Las instrucciones de tipo 1, 2 y 3 incrementan en 1 el número de instrucción. Las de tipo 2 y 3 además modifican el valor de la variable en cuestión en el estado. Las instrucciones de tipo 4 no modifican el estado, pero modifican el índice de próxima instrucción de la siguiente forma: si la variable en cuestión tiene valor 0 en el estado entonces el índice de próxima instrucción se incrementa en 1. Si no, al índice de próxima instrucción se le asigna el índice de la primera instrucción en el programa que tenga la etiqueta en cuestión (si no hay ninguna, se le asigna 1 más que la longitud del programa).

[1] Si hay más de una salta a la primera. Si no hay ninguna, termina el programa.

Representación en `Prolog`

Representaremos a cada `variable` y a cada `etiqueta` con su codificación:

- Cada variable se codifica con un número natural a partir de 1, según el siguiente orden: Y, X_1, Z_1, X_2, Z_2, ...
- Cada etiqueta se codifica con un número natural a partir de 1, según el siguiente orden: A, B, C, ..., Z, AA, AB, ...

Las `instrucciones` se representarán con los literales **nada**(l,v), **suma**(l,v), **resta**(l,v) y **goto**(l,v,e), siendo v la codificación de la variable asociada a la instrucción, l la codificación de la etiqueta asociada a la instrucción (vale 0 si la instrucción no está etiquetada) y e la codificación de la etiqueta destino (en la instrucción de salto). Los `programas` se representarán como listas de instrucciones.

Los `estados` se representan con listas de pares variable-valor. Toda variable que no esté en el estado se considera que tiene valor 0. Las `descripciones instantáneas` se representan con tuplas cuyas primeras componentes corresponden al índice de la próxima instrucción a ejecutar y cuyas segundas componentes corresponden al estado del programa.

Ejercicios

Ejercicio 1. Definir el predicado `evaluar(+Estado, +Var, -Val)` que dado un `estado` y (el número correspondiente a la codificación de) una `variable`, instancia en el tercer argumento el valor de la variable en el estado. Recordar que el estado se representa como una lista de pares variable-valor y que las variables que no aparecen en la lista tienen valor 0. Por ejemplo, la siguiente consulta pide el valor de Y en el estado $\{X = 0 \ \forall \text{ variable } X\}$.

```
?- evaluar([], 1, V).
  V = 0.
```

Mientras que la siguiente consulta pide el valor de X_1 en el estado que le asigna 3 a X_1 y 0 al resto de las variables.

```
?- evaluar([(4,0), (2,3)], 2, V).
  V = 3.
```

Codificación

Ejercicio 2. Se provee el predicado `codificacionLista(?L, ?Z)` para codificar y decodificar listas según se explica en la cartilla. Si el primer argumento está instanciado, entonces instancia en `Z` su codificación correspondiente. Si el primer argumento no está instanciado, lo instancia de acuerdo a la codificación del segundo parámetro en `Z` (es decir, al menos uno de los parámetros **DEBE** estar instanciado). Utilizando los predicados auxiliares `divide/2` y `esPrimo/1` dados, implementar los siguientes predicados auxiliares para que `codificacionLista` funcione correctamente:

- `maximoExponenteQueDivideA(-X, +P, +Z)` que instancia en `X` el máximo número tal que $\texttt{P}^X$ divide a `Z`.
- `iesimoPrimo(+I, -P)` que instancia en `P` el `I`-ésimo primo. **Pista:** pensar cómo podría calcularse el `I`-ésimo primo a partir del `(I-1)`-ésimo primo.

Algunos ejemplos de consultas:

```
?- iesimoPrimo(3, P).
  P = 5.

?- iesimoPrimo(10, P).
  P = 29.

?- maximoExponenteQueDivideA(X, 7, 600).
  X = 0.

?- maximoExponenteQueDivideA(X, 7, 700).
  X = 1.

?- maximoExponenteQueDivideA(X, 7, 98).
  X = 2.
```

Simulación del lenguaje $\mathcal{S}$

Para esta sección se proveen los siguientes predicados:

- `pi1(+(X,Y), -X)` y `pi2(+(X,Y), -Y)` que dada una tupla `(X,Y)` instancian en el segundo argumento su primera y su segunda componente, respectivamente.
- `iesimo(+L, +I, -X)` que dada una lista `L` y un índice `I` ≥ 1 instancia en el tercer argumento el elemento en la `I`-ésima posición de `L`.
- `etiquetaInstruccion(+I, -E)`, `codigoInstruccion(+I, -C)` y `variableInstruccion(+I, -V)` que dada una instrucción `I` instancian en el segundo argumento su etiqueta, su código[2] y su variable, respectivamente.
- `longitud(+P, -L)` que dado un programa `P` instancia en el segundo argumento su longitud.
- `iEsimaInstruccion(+P, +I, -Instrucción)` que dado un programa `P` y un índice `I` ≥ 1 instancia en el tercer argumento la `I`-ésima instrucción de `P`.

Ejercicio 3. Definir el predicado `snap(+Xs, +P, +T, -Di)` que dada una lista de entradas `Xs`, un programa `P` (es decir, una lista, no un número), y un número de pasos `T`, instancia en `Di` la descripción instantánea resultante de ejecutar el programa `P`, con entradas `Xs` después de `T` pasos. Recordar que en el lenguaje S indexamos las listas **desde 1**.
Sugerencia: definir los siguientes predicados auxiliares.

- `avanzarIndice(+P, +S, +Ins, +I0, -I)` que dado un programa `P` (es decir, una lista, no un número), un estado `S`, una instrucción `Ins` y un índice `I0` instancia en `I` el índice de la descripción instantánea resultante tras ejecutar la `I0`-ésima instrucción (`Ins`) del programa `P` con estado `S`.
- `avanzarEstado(+Ins, +S0, -S)` que dada una instrucción `Ins` y un estado `S0`, instancia en `S` el estado resultante de ejecutar la instrucción `Ins` con el estado `S0`.

[2] El b, según la codificación de instrucciones en la cartilla

Ejercicio 4. Definir el predicado `stp(+Xs, +P, +T)` que es verdadero si el programa `P`, con entradas `Xs` termina en `T` pasos o menos. Recordar que un programa termina cuando el índice en la descripción instantánea es la **longitud del mismo más 1**.

Pseudo-Halt

Ejercicio 5. El predicado Halt se define de la siguiente manera:

$$\mathsf{Halt}(x, P) = \begin{cases} 1 & \text{si } \Psi_P(x) \downarrow \\ 0 & \text{si no} \end{cases} \tag{1}$$

Notación: $\Psi_P(x) \downarrow$ se lee "el programa P con entrada x termina".

Sabemos que este predicado no es computable así que no lo vamos a pedir, pero el siguiente (al que llamaremos amistosamente Pseudo-Halt) sí lo es:

$$\mathsf{Pseudo\text{-}Halt}(x, P) = \begin{cases} 1 & \text{si } \Psi_P(x) \downarrow \\ \uparrow & \text{si no} \end{cases} \tag{2}$$

Notación: $\uparrow$ significa que la función está indefinida y que el programa correspondiente "se cuelga". Notar que en ambos casos no se les pasa a los programas una lista de entradas sino un **único valor de entrada** que se asignará a X_1.

a) Definir el predicado `pseudoHalt(+X, +P)` que dada una entrada `X` y un programa (es decir, una lista, no un número) `P` es verdadero si $\Psi_P(x)$ termina y se cuelga si no.

b) Definir el predicado `pseudoHalt2(-X, +P)` que dado un programa (es decir, una lista, no un número) `P`, instancia en `X` las entradas para las cuales $\Psi_P(x)$ termina.

c) Definir el predicado `pseudoHalt3(-X, -P)` que instancia pares de programas (es decir, listas, no números) y entradas para las cuales $\Psi_P(x)$ termina. Se puede utilizar el predicado `programa(-P, +N)` que permite instanciar programas en `P` a partir de números `N`.

Pautas de Entrega

El principal objetivo de este trabajo es evaluar el correcto uso del lenguaje Prolog de forma declarativa para resolver el problema planteado.

Se debe enviar un e-mail conteniendo el código fuente en Prolog a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-PL]` seguido inmediatamente del nombre del grupo.
- El código Prolog debe acompañar el email y lo debe hacer en forma de archivo adjunto con nombre `tp3.pl`.

El código debe poder ser ejecutado en SWI-Prolog. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté adecuadamente comentado. Los objetivos a evaluar en la implementación de los predicados son:

- corrección,
- declaratividad,
- reutilización de predicados previamente definidos
- utilización de unificación, backtracking, generate and test y reversibilidad de los predicados.
- **Importante**: salvo donde se indique lo contrario, los predicados no deben instanciar soluciones repetidas ni colgarse luego de devolver la última solución. Vale aclarar que no es necesario filtrar las soluciones repetidas si la repetición proviene de las características de la entrada.

Se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

Referencias y sugerencias

Como referencia se recomienda la bibliografía incluida en el sitio de la materia (ver sección *Bibliografía* → *Programación Lógica*).

Se recomienda que utilicen los predicados ISO y los de SWI-Prolog ya disponibles, siempre que sea posible. Recomendamos especialmente examinar los predicados y metapredicados que figuran en la sección *Útil* de la página de la materia. Pueden hallar la descripción de los mismos en la ayuda de `SWI-Prolog` (a la que acceden con el predicado `help`). También se puede acceder a la documentación online de SWI-Prolog.