Taller de Programación Lógica

Paradigmas de Lenguajes de Programación — 2do cuat. 2018

Fecha de entrega: 15 de noviembre de 2018

1. Presentación y Sintaxis

En este taller trabajaremos con el cálculo λ puro, cuya sintaxis y representación presentamos a continuación.

Los *términos* de este cálculo son los siguientes:

$$M ::= V \mid MM \mid \lambda V.\ M$$

donde V es una *variable*, que se definen de la siguiente manera:

$$V ::= x \mid \hat{}V$$

Notar que cuando en el texto escribamos x_0 nos estaremos refiriendo a x, x_1 a $\hat{}x$, x_2 a $\hat{}\hat{}x$, etc.

Los modelaremos en Prolog de la siguiente manera:

- Variables:
  - A la variable x la representaremos con la constante `x`.
  - A la variable $\hat{}V$ la representaremos como `^(V)`, donde `V` es la representación de la variable V.

  Por ejemplo, x_2 se representa `^(^(x))`
- Términos:
  - Al **término** formado por la variable V lo representaremos como `mvar(V)`, donde `V` respresenta a V.
  - Al término formado por la aplicación MN lo representaremos como `app(M, N)`, donde `M` y `N` representan a M y a N, respectivamente.
  - Al término formado por la abstracción $\lambda V.\ M$ lo representaremos como `lambda(V, M)`, donde `V` representa a V y `M` representa a M.

  Por ejemplo, el término $\lambda x_1.\ \lambda x_2.\ x_2 x_1$ se representa como

```
lambda(^(x), lambda(^(^(x)), app(mvar(^(^(x))), mvar(^(x)) ) )
```

Por suerte, contamos con el predicado `show(T)` que escribe por pantalla el término o la variable representada por `T`.

Ejercicio 1

(a) Definir el predicado **vari(?X)** que es verdadero cuando X pertenece al lenguaje de las **variables.** Observar que si X no está instanciada, **vari/1** debe producir una enumaración de todas las variables sin repetición.

(b) Definir el predicado **term(+M)** que es verdadero cuando M pertenece al lenguaje de los términos. ¿Es posible definir **term(-M)**? ¿Es diferente a **vari(-X)**? ¿Por qué?

2. Variables libres y ligadas

Al igual que en la versión del cálculo que vimos en la materia, en el cálculo λ puro existen las variables libres y ligadas, con exactamente la misma definición.

Ejercicio 2

Definir **fv(+M, -Xs)**, que es verdadero cuando **Xs** es una lista con (exactamente) todas las *variables* libres de M, en algún orden. Podemos suponer que M verifica **term/1**.

Ejercicio 3

Definir **sustFV(+M, +X, +Y, ?MSust)**, que es verdad cuando **MSust** es M con las ocurrencias libres de X reemplazadas por Y. Podemos suponer que X e Y verifican **vari/1** y M y **MSust** verifican **term/1** (si **MSust** no está instanciado, al instanciarse debe verificarlo). Observar que en esta definición, no hace falta preocuparse por la captura de variables.

Al igual que en nuestro cálculo, diremos que dos términos son α-equivalentes si difieren únicamente en el nombre de sus variables ligadas.

Lema: $M \overset{\alpha}{\equiv} N$ sii N se obtiene a partir de M por 0, 1 o más cambios de variable ligada. Un cambio de variable ligada consiste en reemplazar la variable de una abstracción y todas sus apariciones ligadas por otra variable (que no debe aparecer libre en el término).

Ejercicio 4

Definir el predicado **alphaEq(+M, +N)** que decide si M y N son α-equivalentes.

3. Sustituciones

En el cálculo λ puro, el cómputo está definido por una regla de reescritura basada en sustituciones. Para eso, debemos definir una sustición de una variable por un término como :

- $x[P/x] = P$, donde x es una variable y P un término cualquiera.
- $y[P/x] = P$, donde x e y son dos variables *distintas* y P un término cualquiera.
- $(MN)[P/x] = M[P/x]\ N[P/x]$, donde x es una variable y M, N y P son términos cualesquiera.
- $(\lambda x.\ M)[P/x] = (\lambda x.\ M)$, donde x es una variable y P un término cualquiera.
- $(\lambda y.\ M)[P/x] = (\lambda z.\ (M[z/y])[P/x])$, donde x, y y z son variables, M y P términos y z **no aparece libre en M ni en P**.

Ejercicio 5

Definir **sust(+M, +X, +N, ?MSust)** que es verdadero cuando **MSust** $\overset{\alpha}{\equiv}$ M[N / X].

- Podemos suponer que X verifica **vari/1** y que M y N verifican **term/1**.
- Si N está instanciado podemos suponer que verifica **term/1**, mientras que si no lo está, el resultado debe verificar **term/1**.

- En el caso de que `MSust` no esté instanciado, el predicado no debe producir términos α-equivalentes.
  Pista: si `sust(λx1.λx2. x, x, x3, M)` produce `M = λx4.λx5. x3`, no debe producir ni `M = λx5.λx4. x3`, ni `M = λx7.λx8. x3`.
  Sugerencia: escribir un predicado que genere variales apropiadas.

4. Reducción

En este cálculo, los términos admiten más de una reducción. Las reglas que la definen son las siguientes:

- $MN \to M'N$, donde $M \to M'$.
- $MN \to MN'$, donde $N \to N'$.
- $\lambda x.\ M \to \lambda x.\ M'$, donde $M \to M'$.
- $(\lambda x.\ M)N \to M[N/x]$

Este último caso es el que efectivamente hace que el cómputo progrese. A los (sub)términos de la forma $(\lambda x.M)N$ les decimos redexes y a la reducción de este tipo de aplicaciones le decimos β-reducción.

Ejercicio 6

(a) Definr el predicado `betaRedex(+R, ?N)`, que es verdadero si `R` es un redex y `N` es su β-reducción.

(b) Definir `reduce(+M, ?N)`, que es verdadero cuando $M \to N$.

Como siempre, podemos suponer que todos los términos verifican `term/1` y si `N` no está instanciado, no deben producirse términos α-equivalentes.

Ejercicio 7

Definir el predicado `formaNormal(+M)` que decide si `M` es una forma normal.

Definición: un paso de reducción se dice *leftmost* (más a la izquierda) si es producto de reducir el redex que aparece más arriba y más a la izquierda en el árbol sintáctico de formación del término. Observación: Todas las reducciones leftmost son reducciones y un término admite (a lo sumo) una reducción leftmost.

Ejercicio 8

Definir `leftmost(+M, -N)` que es verdadero si `N` es el resultado de realizar una reducción leftmost sobre `M`.

Este predicado deberá definirse en una única regla, utilizando únicamente predicados definidos hasta el momento. Si necesitan modificar alguno de los predicados que ya definieron, aclaren cuáles, de qué manera y por qué.

Teorema: Si M tiene forma normal (i.e., existe N f.n. tal que $M \xrightarrow{*} N$), entonces $M \xrightarrow{l*} N$, donde $\xrightarrow{l*}$ significa 0, 1 o más pasos de reducción leftmost. Además, si M tiene una forma normal, esta es única módulo $\stackrel{\alpha}{\equiv}$.

Ejercicio 9

(a) Definir `formaNormal(+M, -N)` que es verdadero cuando `N` es la forma normal de `M`. Si `M` no tiene forma normal, el predicado puede indefinirse.

(b) ¿Podrá definirse `formaNormal(+M, +N)`? ¿O una versión de `formaNormal(+M, -N)` que sea total? ¿Por qué?

5. Los difíciles

Se define la longitud de los términos de la siguiente manera:

- $long(x) = 1$
- $long(\lambda x.\ M) = 1 + long(M)$
- $long(MN) = long(M) + long(N)$

Ejercicio 10

(a) Definir **long(+M, ?K)** que es verdadero si M tiene longitud K. Podemos suponer que M verifica **term/1**.

(b) Extender la definición anterior para soportar **long(-M, +K)**.
Observar que M puede no estar totalmente instanciado y en caso de que no lo esté, debe dejar variables de Prolog en el lugar de las variables del cálculo. De esta manera, los términos están instanciados ``lo menos posible'', pero su longitud está definida.

Ejercicio 11

(a) Extender la definición de **fv/2** para que soporte un término parcialmente instanciado si la lista de variables está instanciada (**fv(-M, +Xs)**).
Observar que no deben generarse términos α-equivalentes (por ejemplo, si ya se generó $\lambda x_1.\lambda x_2.\ x_1 x_2$, no debe generarse $\lambda x_2.\lambda x_1.\ x_2 x_1$).

(b) Definir **cerrado(?M)** que es verdadero cuando M es un término cerrado (ie, que no contiene variables libres).
Observar que si M esta instanciado, debe decidir si es cerrado, y si no, debe producir una enumeración de todos los términos cerrados del cálculo λ puro, sin repetidos por alfaequivalencia.

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 entregar el código impreso con la implementación de los predicados pedidos. Cada predicado asociado a los ejercicios debe contar con ejemplos que muestren que exhibe la funcionalidad solicitada. Además, 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 e-mail y lo debe hacer en forma de archivo adjunto con nombre `tp2.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 incluída 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 *Cosas útiles* 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.