Clase práctica 7: $\mathcal{S}++$, Codificaciones y funciones parcialmente computables

Lenguajes Formales, Autómatas y Computabilidad

Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

Primer cuatrimestre 2025

Lenguaje $\mathcal{S}++$

Variables

- **Variables de entrada:** $X_1, X_2, ...$
- **Variables locales:** $Z_1, Z_2, ...$
- **Variables de salida:** Y

Lenguaje $\mathcal{S}++$

Variables

- **Variables de entrada:** $X_1, X_2, \ldots$
- **Variables locales:** $Z_1, Z_2, \ldots$
- **Variables de salida:** Y

Instrucciones

- **Incremento:** $V++$ (suma uno a la variable V)
- **Decremento:** $V--$ (resta uno a la variable V)
- **Loop:** ***While*** $V \neq 0$ ***do*** P (mientras $V \neq 0$ ejecutar el programa P)
- **instrucción vacía:** *pass* (no hacer nada)

Un **programa** es una secuencia finita de instrucciones. Por default todas lsa variables no inicializadas comienzan con valor 0.

Función computada por un programa

Definición

Dado un programa P, para cada $N \in \mathbb{N}$ se define $\Psi_P^{(n)} : \mathbb{N}^n \to \mathbb{N}$ como la **función computada por P** con n entradas.

Función computada por un programa

Definición

Dado un programa P, para cada $N \in \mathbb{N}$ se define $\Psi_P^{(n)} : \mathbb{N}^n \to \mathbb{N}$ como la **función computada por *P*** con n entradas.

$\Psi_P^{(n)}$ define una *función parcial*, es decir, puede indefinirse (si se cuelga el programa) para algunos valores de entrada. Notamos entonces:

- ⋆ $\Psi_P^{(n)}(x_1, \ldots, x_n) \downarrow$ si está definida con entrada $x_1, \ldots, x_n$.
- ⋆ $\Psi_P^{(n)}(x_1, \ldots, x_n) \uparrow$ si está indefinida con entrada $x_1, \ldots, x_n$.

Función computada por un programa

Definición

Dado un programa P, para cada $N \in \mathbb{N}$ se define $\Psi_P^{(n)} : \mathbb{N}^n \to \mathbb{N}$ como la **función computada por P** con n entradas.

$\Psi_P^{(n)}$ define una *función parcial*, es decir, puede indefinirse (si se cuelga el programa) para algunos valores de entrada. Notamos entonces:

- $\Psi_P^{(n)}(x_1, ..., x_n) \downarrow$ si está definida con entrada $x_1, ..., x_n$.
- $\Psi_P^{(n)}(x_1, ..., x_n) \uparrow$ si está indefinida con entrada $x_1, ..., x_n$.

Llamamos **Dominio** de una función al conjunto de valores para los cuales está definida:

$$Dom(\Psi_P^{(n)}) = \{(x_1, ..., x_n) \mid \Psi_P^{(n)}(x_1, ..., x_n) \downarrow\}$$

Decimos que una función es **total** si su dominio es $\mathbb{N}$.

Cómo obtener Ψ_P

1. Se inicializan las variables $X_1, ..., X_n$ con los valores de la entrada $x_1, ..., x_n$.
2. Se ejecuten en orden y secuencialmente las instrucciones del programa P.
3. Si la ejecución en algún momento termina, entonces $\Psi_P^{(n)}(x_1, ..., x_n) \downarrow$, y el valor de la función es el valor de la variable Y al terminar la ejecución del programa P. Si la ejecución no termina, entonces $\Psi_P^{(n)}(x_1, ..., x_n) \uparrow$.

Funciones (parcialmente) computables

Definición

Decimos que una función f es **parcialmente computable** si existe un programa P en $\mathcal{S}++$ tal que $f = \Psi_P^{(n)}$.

Otra definición

Decimos que una función f es **total computable** o simplemente **computable** si es *parcial computable* y además es *total*.

Macros

$\mathcal{S}++$ es un lenguaje muy simple y definida muy formalmente, lo cual es cómodo para hacer demostraciones, pero incómodo para escribir programas. Para simplificar entonces escribir programas en $\mathcal{S}++$ y no repetir una y otra vez lo mismo vamos a usar **macros**.

Macros

Es una sucesión de instrucciones a las cuales ponemos una *etiqueta* para poder referenciarlas todas las veces que queramos.

Macros

$\mathcal{S}++$ es un lenguaje muy simple y definida muy formalmente, lo cual es cómodo para hacer demostraciones, pero incómodo para escribir programas. Para simplificar entonces escribir programas en $\mathcal{S}++$ y no repetir una y otra vez lo mismo vamos a usar **macros**.

Macros

Es una sucesión de instrucciones a las cuales ponemos una *etiqueta* para poder referenciarlas todas las veces que queramos.

Un programa escrito usando macros se denomina **pseudoprograma**. Todo pseudoprograma tiene un programa equivalente, que se obtiene expandiendo todas las macros utilizadas.

Ejercicio 1

Enunciado

Dar una macro para la pseudoinstrucción $V := 0$.

Ejercicio 1

Enunciado

Dar una macro para la pseudoinstrucción $V := 0$.

Reolución

$$\textbf{\textit{While}}\ V \neq 0\ \textbf{\textit{do}}:$$
$$V - -$$

Ejercicio 2

Enunciado

Demostrar que las siguientes funciones son computables, donde P es cualquier predicado computable. Se puede utilizar macros.

1. Minimización acotada: $f(z) = min_{x \leqslant z} P(x)$
2. Existencial acotado: $f(z) = (\exists x)_{\leqslant z} P(x)$
3. Para todo acotado: $f(z) = (\forall x)_{\leqslant z} P(x)$

Ejercicio 2

Enunciado

Demostrar que las siguientes funciones son computables, donde P es cualquier predicado computable. Se puede utilizar macros.

1. Minimización acotada: $f(z) = min_{x \leqslant z} P(x)$
2. Existencial acotado: $f(z) = (\exists x)_{\leqslant z} P(x)$
3. Para todo acotado: $f(z) = (\forall x)_{\leqslant z} P(x)$

Resolución I

$$Y = 0$$
$$\textbf{\textit{While}}\ P(Y) = 0 \wedge Y < X_1\ \textbf{\textit{do}}\ \{Y++\}$$

Pensar cómo serían los otros dos programas.

Codificación de tuplas y listas

Codificación y observadores de tuplas

Definimos la siguiente forma de codificar tuplas y de acceder a sus partes, donde $z = \langle x, y \rangle$:

$$\langle x, y \rangle = 2^x(2y+1) - 1$$
$$l(z) = min_{x \leqslant z}((\exists y)_{\leqslant z} z = \langle x, y \rangle)$$
$$r(z) = min_{y \leqslant z}((\exists x)_{\leqslant z} z = \langle x, y \rangle)$$

Codificación de tuplas y listas

Codificación y observadores de tuplas

Definimos la siguiente forma de codificar tuplas y de acceder a sus partes, donde $z = \langle x, y \rangle$:

$$\langle x, y \rangle = 2^x(2y+1) - 1$$
$$l(z) = min_{x \leqslant z}((\exists y)_{\leqslant z} z = \langle x, y \rangle)$$
$$r(z) = min_{y \leqslant z}((\exists x)_{\leqslant z} z = \langle x, y \rangle)$$

Codificación de listas

Usaremos la codificación de Gödel, donde p_i representa al i-ésimo primo:

$$[a_1, ..., a_n] = \prod_{i=1}^{n} p_1^{a_i}$$

Observadores de listas

Observadores

Se pueden definir además funciones para observar la longitud, el *i*-ésimo elemento, la concatenacion y otras funciones sobre listas.

Propiedad

Todos los codificadores y observadores de tuplas y listas son computables.

Vamos a usar estas codificaciones para codificar *programas*.

Codificación de instrucciones

Codificación

$$\#(I) = \begin{cases} \langle a, b\rangle + 1 & I \neq \textbf{\textit{pass}} \\ 0 & \text{si no} \end{cases}$$

donde $a = \#(V)$ siendo V la variable involucrada en la instrucción, utilizando la siguiente numeración:

$$\#(Y) = 0$$
$$\#(X_i) = 2i - 1$$
$$\#(Z_i) = 2i$$

y donde el valor de b es:

- 0 si I es un incremento.
- 1 si I es un decremento.
- $\#(P) + 2$ si I es un ciclo cuyo cuerpo es P.

Codificación de programas

Codificación

Si un programa P es una secuencia de instrucciones $I_1, ..., I_n$, definimos su codificación como:

$$\#(P) = [\#(I_1), ..., \#(I_n)]$$

Codificación de programas

Codificación

Si un programa P es una secuencia de instrucciones $I_1, ..., I_n$, definimos su codificación como:

$$\#(P) = [\#(I_1), ..., \#(I_n)]$$

Notación

Definimos entonces $\Phi_e^{(n)} : \mathbb{N}^n \to \mathbb{N}$ como la función computada por el programa de número e. No confundir con Ψ. Si P es un programa de número e, entonces:

$$\Psi_P^{(n)} = \Phi_e^{(n)}$$

Configuración instantánea

Dado un programa P y una entrada $x_1, \ldots, x_n$, podemos pensar en su ejecución como una serie de pasos de cómputo. Entonces, dado un momento del tiempo, vamos a representar el estado del cómputo mediante la configuración instantánea, que puede pensarse como una tupla de dos números que representan:

Configuración instantánea

1. La **instrucción** que toca ejecutar en el sigueinte paso. La representamos con una lista $[i_1, \ldots, i_j]$ donde i_1 es la instrucción que toca ejecutar a continuación y, si i_1 es un loop, i_2 es el índice de la instrucción *dentro del loop* que toca ejecutar a continuación.
   Si la ejecución del programa ya terminó, esta lista tendrá un único elemento que será la cantidad de instrucciones del programa más uno.
2. Los **valores de las variables**, representados en una lista con el siguiente orden: $Y, X_1, Z_1, X_2, Z_2, \ldots$ hasta la última variable con valor distinto de 0.

STEP y SNAP

STEP

El predicado $STEP^{(n)}(e, t, x_1, ..., x_n) : \mathbb{N}^{n+2} \to \mathbb{N}$ devuelve 1 si el programa de número e con entrada $x_1, ..., x_n$ termina su ejecución tras t o menos pasos, y 0 sino.

SNAP

Definimos la función $SNAP^{(n)} : \mathbb{N}^{n+2} \to \mathbb{N}$, tal que $SNAP(e, t, x_1, ..., x_n)$ codifica la configuración instantánea del programa de número e con entrada $x_1, ..., x_n$ en tiempo t.

STEP y SNAP

STEP

El predicado $STEP^{(n)}(e, t, x_1, ..., x_n) : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ devuelve 1 si el programa de número e con entrada $x_1, ..., x_n$ termina su ejecución tras t o menos pasos, y 0 sino.

SNAP

Definimos la función $SNAP^{(n)} : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$, tal que $SNAP(e, t, x_1, ..., x_n)$ codifica la configuración instantánea del programa de número e con entrada $x_1, ..., x_n$ en tiempo t.

Ambas funciones son computables.

Propiedad importante

Teorema

Sea P un predicado computable, entonces la función definida como:

$$(\exists x)(P(x))$$

que devuelve 1 si existe tal x y se indefine sino es parcial computable.

Ejercicio 3

Enunciado

Demostrar que la siguiente función es parcial computable:

$$f(x, y) = \begin{cases} 1 & \text{si } y \in \text{Dom } \Phi_x^{(1)} \\ \uparrow & \text{en otro caso} \end{cases}$$

Ejercicio 3

Enunciado

Demostrar que la siguiente función es parcial computable:

$$f(x, y) = \begin{cases} 1 & \text{si } y \in \text{Dom } \Phi_x^{(1)} \\ \uparrow & \text{en otro caso} \end{cases}$$

Resolución

Hay dos posibilidades. Podemos dar un programa que compute la función , o podemos usar la propiedad anterior. Usando eso, podemos escribir la función como:

$$f(x, y) = (\exists t)(STEP^{(1)}(x, t, y) = 1)$$

Y como *STEP* es un predicado computable, la función resulta parcial computable.

Ejercicio 4

Enunciado

Demostrar que la siguiente función es parcial computable:

$$g(x, y) = \begin{cases} 1 & \text{si } y \in \text{Im } \Phi_x^{(1)} \\ \uparrow & \text{en otro caso} \end{cases}$$

Ejercicio 4

Enunciado

Demostrar que la siguiente función es parcial computable:

$$g(x, y) = \begin{cases} 1 & \text{si } y \in \text{Im } \Phi_x^{(1)} \\ \uparrow & \text{en otro caso} \end{cases}$$

Resolución

De vuelta, podemos dar un programa que compute la función, o podemos usar la propiedad anterior. En este caso, para escribir la función con un existencial necesitaríamos dos, que existe una entrada z y un tiempo t tal que el programa de número x termina con entrada z y da y. Para eso, podemos usar la codificación de tuplas que ya definimos y sabemos que es computable:

$$g(x, y) = (\exists u)(STEP^{(1)}(x, l(u), r(u)) \wedge r(SNAP^{(1)}(x, l(u), r(u)))[0] = y)$$

Ejercicio 4 - continuación

Continuación resolución

Para que quede más clara la resolución podemos escribir la función como:

$$g(x, y) = (\exists \langle t, z \rangle)(STEP^{(1)}(x, t, z) \wedge r(SNAP^{(1)}(x, z, t))[0] = y)$$

Y como tanto *STEP* como *SNAP* son computables, la función resulta computable.

FIN

$\star \star \star$