Lógica y Computabilidad

2do cuatrimestre 2020 - **A DISTANCIA**

Departamento de Computación - FCEyN - UBA

Computabilidad - clase 2

Funciones iniciales, composición, recursión, clases PRC, funciones primitivas recursivas

Funciones iniciales

Otra manera de formalizar la idea de función calculable de manera efectiva:

- empezar por funciones muy simples, efectivas intuitivamente
- si mezclamos de alguna manera efectiva dos o más funciones que ya eran efectivas, entonces obtenemos una función calculable de manera efectiva

Definición

Las siguientes funciones se llaman iniciales:

- $s(x) = x + 1$
- $n(x) = 0$
- proyecciones: $u_i^n(x_1, \ldots, x_n) = x_i$ para $i \in \{1, \ldots, n\}$

Composición y recursión primitiva

Definición

Sea $f : \mathbb{N}^k \to \mathbb{N}$ y $g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N}$. $h : \mathbb{N}^n \to \mathbb{N}$ se obtiene a partir de f y $g_1, \ldots, g_k$ por composición si

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$$

Definición

$h : \mathbb{N}^{n+1} \to \mathbb{N}$ se obtiene a partir de $g : \mathbb{N}^{n+2} \to \mathbb{N}$ y $f : \mathbb{N}^n \to \mathbb{N}$ por recursión primitiva si

$$\begin{aligned} h(x_1, \ldots, x_n, 0) &= f(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n, t+1) &= g(h(x_1, \ldots, x_n, t), x_1, \ldots, x_n, t) \end{aligned}$$

(En este contexto, una función 0-aria es una constante k. Si h es 1-aria y $t = 0$, entonces $h(t) = k = s^{(k)}(n(t))$.)

Clases PRC

Una clase $\mathcal{C}$ de funciones totales es PRC (primitive recursive closed) si

1. las funciones iniciales están en $\mathcal{C}$
2. si una función f se obtiene a partir de otras pertenecientes a $\mathcal{C}$ por medio de composición o recursión primitiva, entonces f también está en $\mathcal{C}$

Teorema

La clase de funciones totales Turing computables (o computables en C, o en Java) es una clase PRC.

Funciones primitivas recursivas

Una función es primitiva recursiva (p.r.) si se puede obtener a partir de las funciones iniciales por un número finito de aplicaciones de composición y recursión primitiva.

Teorema

Una función es p.r. sii pertenece a toda clase PRC.

Demostración.

($\Leftarrow$) La clase de funciones p.r. es una clase PRC. Luego, si f pertenece a toda clase PRC, en particular f es p.r.

($\Rightarrow$) Sea f p.r. y sea $\mathcal{C}$ una clase PRC. Como f es p.r., hay una lista

$$f_1, f_2, \ldots, f_n$$

tal que

- $f = f_n$
- f_i es inicial (luego está en $\mathcal{C}$) o se obtiene por composición o recursión primitiva a partir de funciones f_j, $j < i$ (luego también está en $\mathcal{C}$).

Entonces todas las funciones de la lista están en $\mathcal{C}$ (por inducción). En particular, $f_n \in \mathcal{C}$. $\square$

¿Funciones totales Turing computables = funciones primitivas recursivas?

Entonces la clase de funciones p.r. es la clase PRC más chica.

Corolario

Toda función p.r. es total y Turing computable.

Demostración.

Sabemos que la clase de funciones totales Turing computables es PRC. Por el teorema anterior, si f es p.r., entonces f pertenece a la clase de funciones Turing computables. □

No toda función parcial Turing computable es p.r porque toda función p.r. es total. Pero...

¿toda función total Turing computable es p.r.?

Ejemplo de función p.r.

La función

$$suma(x, y) = x + y$$

es p.r., porque

$$\begin{aligned} suma(x, 0) &= u_1^1(x) \\ suma(x, y+1) &= g(suma(x, y), x, y) \end{aligned}$$

donde

$$g(x_1, x_2, x_3) = s(u_1^3(x_1, x_2, x_3))$$

Otras funciones primitivas recursivas

- $x \cdot y$
- $x!$
- x^y
- $x \dot{-} y = \begin{cases} x - y & \text{si } x \geq y \\ 0 & \text{si no} \end{cases}$
- $\alpha(x) = \begin{cases} 1 & \text{si } x = 0 \\ 0 & \text{si no} \end{cases}$
- y muchas más. ¿Todas las funciones totales Turing computables..?

Predicados primitivos recursivos

Los predicados son simplemente funciones que toman valores en $\{0, 1\}$.

- 1 se interpreta como verdadero
- 0 se interpreta como falso

Los predicados p.r. son aquellos representados por funciones p.r. en $\{0, 1\}$.

Por ejemplo, el predicado $x \leq y$ es p.r. porque se puede definir como

$$\alpha(x \dot{-} y)$$

Operadores lógicos

Teorema

Sea $\mathcal{C}$ una clase PRC. Si p y q son predicados en $\mathcal{C}$ entonces $\neg p$, $p \wedge q$ y $p \vee q$ están en $\mathcal{C}$.

Demostración.

- $\neg p$ se define como $\alpha(p)$
- $p \wedge q$ se define como $p \cdot q$
- $p \vee q$ se define como $\neg(\neg p \wedge \neg q)$

$\square$

Corolario

Si p y q son predicados p.r., entonces también lo son los predicados $\neg p$, $p \vee q$ y $p \wedge q$

Corolario

Si p y q son predicados totales Turing computables entonces también lo son los predicados $\neg p$, $p \vee q$ y $p \wedge q$

Definición por casos (2)

Teorema

Sea $\mathcal{C}$ una clase PRC. Sean $h, g : \mathbb{N}^n \to \mathbb{N}$ funciones en $\mathcal{C}$ y sea $p : \mathbb{N}^n \to \{0, 1\}$ un predicado en $\mathcal{C}$. La función

$$f(x_1, \dots, x_n) = \begin{cases} g(x_1, \dots, x_n) & \textit{si } p(x_1, \dots, x_n) \\ h(x_1, \dots, x_n) & \textit{si no} \end{cases}$$

está en $\mathcal{C}$.

Demostración.

$f(x_1, \dots, x_n) =$
$g(x_1, \dots, x_n) \cdot p(x_1, \dots, x_n) + h(x_1, \dots, x_n) \cdot \alpha(p(x_1, \dots, x_n))$ $\square$

Definición por casos $(m+1)$

Teorema

Sea $\mathcal{C}$ una clase PRC. Sean $g_1, \ldots, g_m, h : \mathbb{N}^n \to \mathbb{N}$ funciones en $\mathcal{C}$ y sean $p_1, \ldots, p_m : \mathbb{N}^n \to \{0,1\}$ predicados mutuamente excluyentes en $\mathcal{C}$. La función

$$f(x_1, \ldots, x_n) = \begin{cases} g_1(x_1, \ldots, x_n) & \text{si } p_1(x_1, \ldots, x_n) \\ \quad\vdots & \\ g_m(x_1, \ldots, x_n) & \text{si } p_m(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n) & \text{si no} \end{cases}$$

está en $\mathcal{C}$.

Recursión primitiva

- todavía no respondimos si p.r. $=$ Turing computables totales
- no lo vamos a responder todavía

Observar que el esquema de recursión

$$\begin{aligned} h(x_1, \ldots, x_n, 0) &= f(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n, t+1) &= g(h(x_1, \ldots, x_n, t), x_1, \ldots, x_n, t) \end{aligned}$$

es muy simple:

- la recursión siempre se hace en el último parámetro
- la función variante de $h(x_1, \ldots, x_n, x_{n+1})$ es x_{n+1}