Tema 6: El teorema de recursión

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Ciencias de la Computación
(4º curso, Grado en Matemáticas)
Curso 2021–22

Transformaciones de programas

Teorema *s–n–m*

- Aplicaciones

El teorema de recursión

- Aplicaciones del teorema de recursión

Transformaciones de programas

Definición: Una transformación de programas es una aplicación

$$\mathcal{F} : \mathtt{GOTO_P} \times \overset{(m)}{\cdots} \times \mathtt{GOTO_P} \times \mathbb{N}^k \to \mathtt{GOTO_P}$$

Definición: Una transformación $\mathcal{F}$ se dice **recursiva (o automática, o computable**) si es recursiva la función F que hace conmutativo el diagrama

$$\begin{array}{ccccc} & & \mathcal{F} & & \\ \mathtt{GOTO_P} \times \overset{(m)}{\cdots} \times \mathtt{GOTO_P} \times \mathbb{N}^k & \longrightarrow & \mathtt{GOTO_P} & & \\ \#(.)^{(m)} \times Id_{\mathbb{N}^m} \downarrow & & \# \downarrow & & \#(.) \\ \mathbb{N}^{m+k} & \underset{F}{\longrightarrow} & \mathbb{N} & & \end{array}$$

Ejemplo interesante

Sugerido por el teorema de la forma normal: $(\mathtt{p}, n) \mapsto \mathtt{q}_{\mathtt{p},n}$ donde $\mathtt{q}_{\mathtt{p},n}$ es el programa (usando macros):

$$
\begin{array}{ll}
 & \mathtt{Z}_2 \leftarrow \#(\mathtt{p}) \\
[\mathtt{A}] & \mathtt{IF}\ \mathcal{T}_n(\mathtt{X}_1, \ldots, \mathtt{X}_n, \mathtt{Z}_2, \mathtt{Z})\ \mathtt{GOTO\ B} \\
 & \mathtt{Z} \leftarrow \mathtt{Z} + 1 \\
 & \mathtt{GOTO\ A} \\
[\mathtt{B}] & \mathtt{Y} \leftarrow I(\mathtt{Z})
\end{array}
$$

La transformación

$$\mathtt{F} : \mathtt{GOTO}_{\mathtt{P}} \times \mathbb{N} \to \mathtt{GOTO}_{\mathtt{P}}$$

$$(\mathtt{p}, n) \mapsto \mathtt{q}_{\mathtt{p},n}$$

es automática, pues $F(\#(\mathtt{p}), n) = \#(\mathtt{q}_{\mathtt{p},n})$ es p.r.
Se verifica que: $[\![\mathtt{F}(\mathtt{p})]\!]^{(n)} = [\![\mathtt{p}]\!]^{(n)}$

Fijando variables como "parámetros"

- Podemos *especializar* los programas de manera automática
  - Dada $\varphi_e^{(m+n)}$, si prefijamos m valores $a_1, \ldots a_n$, obtenemos una función $h : \mathbb{N}^m \to \mathbb{N}$ definida por

$$h(x_1, \ldots, x_m) = \varphi_e^{(m+n)}(x_1, \ldots, x_m, a_1, \ldots, a_n)$$

    ¿Podemos obtener de manera recursiva el (un) código j de la función *especializada*?
  - Si denotamos por S_m^n a la función $n+1$–aria $(e, \vec{a}) \mapsto j$, entonces la ecuación anterior queda como

$$\varphi^{(m)}_{S_m^n(e,\vec{u})}(\vec{x}) \simeq \varphi_e^{(m+n)}(\vec{x}, \vec{u})$$

- Cuestión: ¿Existe la **función computable** S_m^n?

Ejemplo

La siguiente función es la suma:

$$f(x,u) = \begin{cases} u & x = 0 \\ \mathcal{S}(f(x-1,u)) & x \neq 0 \end{cases}$$

Si fijamos $u = 2$ obtenemos la función $f_2(x) = x + 2$,

$$f_2(x) = \begin{cases} 2 & x = 0 \\ \mathcal{S}(f_2(x-1)) & x \neq 0 \end{cases}$$

En programas:

```
      Z ← X1
      Y ← X2
[B]   IF Z ≠ 0 GOTO A
      GOTO E
[A]   Z ← Z − 1
      Y ← Y + 1
      GOTO B
```

$\xrightarrow{x_2=2}$

```
      X2 ← X2 + 1
      X2 ← X2 + 1
      Z ← X1
      Y ← X2
[B]   IF Z ≠ 0 GOTO A
      GOTO E
[A]   Z ← Z − 1
      Y ← Y + 1
      GOTO B
```

Es evidente que $[\![P_2]\!]^{(1)}(x) = [\![P]\!]^{(2)}(x,2)$

¿Se puede mecanizar esta transformación para cualquier programa?

El teorema *s-n-m*

Teorema *s-n-m*. Sean $n,\ m \geq 1$. Existe $S_m^n : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ primitiva recursiva e inyectiva tal que para todo $\vec{a} \in \mathbb{N}^n$,

$$\varphi^{(m)}_{S_m^n(e,\vec{a})}(\vec{x}) = \varphi_e^{(m+n)}(\vec{x}, \vec{a})$$

(igualdad entre funciones parciales, claro)

Demostración: por inducción en n:

- Caso $n = 1$: La función S_m^1 debe verificar

$$\varphi^{(m)}_{S_m^1(e,u)}(x_1, \ldots, x_m) = \varphi_e^{(m+1)}(x_1, \ldots, x_m, u)$$

Es decir, dados (e, u), $S_m^1(e, u)$ calcula el programa:

$$(x_1, \ldots, x_m) \mapsto \varphi_e^{(m+1)}(x_1, \ldots, x_m, u)$$

Basta asignarle a $\mathtt{X}_{m+1}$ el valor u, y aplicar seguidamente el programa $\mathtt{p}$ de código $\#(\mathtt{p}) = e$:

$$\left.\begin{array}{l} \mathtt{X}_{m+1} \leftarrow \mathtt{X}_{m+1} + 1 \\ \vdots \\ \mathtt{X}_{m+1} \leftarrow \mathtt{X}_{m+1} + 1 \\ \mathtt{p} \end{array}\right\} (u)$$

El teorema *s-n-m*. Demostración $n = 1$

- ¿Cuál es el código del programa resultante?
  - $\#(\mathtt{X}_{m+1} \leftarrow \mathtt{X}_{m+1} + 1) = \langle 0, \langle 1, 2m+1 \rangle \rangle = 16m + 10$.
  - Hay que desplazar las líneas de

$$\mathtt{P} \equiv \mathtt{I}_1, \ldots, \mathtt{I}_{\mathrm{Lt}(e+1)}$$

en la codificación del programa resultante.

Luego

$$S_m^1(e,u) = [\#(\mathtt{X}_{m+1} \leftarrow \mathtt{X}_{m+1} + 1), \ldots, \#(\mathtt{X}_{m+1} \leftarrow \mathtt{X}_{m+1} + 1), \#(\mathtt{I}_1), \ldots, \#(I_{\mathrm{Lt}(e+1)})] \dot{-} 1$$

que es

$$S_m^1(e,u) = \left[\left(\prod_{i=1}^{u} p_i \right)^{16m+10} \cdot \prod_{j=1}^{\mathrm{Lt}(e+1)} p_{u+j}^{(e+1)_j} \right] \dot{-} 1$$

que es, por tanto, primitiva recursiva.

El teorema *s-n-m*. Paso de inducción

- Paso de inducción: Supongamos que hemos obtenido $S_m^n \in \mathcal{PR}$ y veamos cómo se obtiene S_m^{n+1}.
- Basta especializar, en primer lugar, en la última variable (el caso $n = 1$) y aplicar la hipótesis de inducción para especializar en las n restantes:

$$\begin{aligned}\varphi_e^{(m+n+1)}(\vec{x}, u_1, \ldots, u_n, u_{n+1}) &= \varphi^{(m+n)}_{S^1_{m+n}(e,u_{n+1})}(\vec{x}, u_1, \ldots, u_n) \\ &= \varphi^{(m)}_{S^n_m(S^1_{m+n}(e,u_{n+1}),u_1,\ldots,u_n))}(\vec{x})\end{aligned}$$

Por tanto, definimos

$$S_m^{n+1}(e, u_1, \ldots, u_n, u_{n+1}) = S_m^n(S^1_{m+n}(e, u_{n+1}), u_1, \ldots, u_n))$$

es primitiva recursiva e inyectiva (por construcción).

El teorema *s-n-m*: consecuencia en GOTO

Corolario Existe una transformación automática de programas

$$\mathtt{Spec}_m^n : \mathtt{GOTO}_\mathtt{P} \times \mathbb{N}^n \to \mathtt{GOTO}_\mathtt{P}$$

tal que para cualesquiera $\mathtt{p} \in \mathtt{GOTO}_\mathtt{P}$, $\vec{x} \in \mathbb{N}^m$, $\vec{u} \in \mathbb{N}^n$

$$[\![\mathtt{Spec}_\mathtt{m}^\mathtt{n}(\mathtt{p}, \vec{u})]\!]^{(m)}(\vec{x}) = [\![\mathtt{p}]\!]^{(m+n)}(\vec{x}, \vec{u})$$

Demostración: Se obtiene a partir de la función S_m^n. $\mathtt{Spec}_m^n$ es la *única* transformación automática que hace conmutativo el diagrama

$$\begin{array}{ccccc} & & \mathtt{Spec}_m^n & & \\ & \mathtt{GOTO}_\mathtt{P} \times \mathbb{N}^n & \longrightarrow & \mathtt{GOTO}_\mathtt{P} & \\ \#(.) \times Id_{\mathbb{N}^n} & \downarrow & \# & \downarrow & \#(.) \\ & \mathbb{N}^{1+n} & \longrightarrow & \mathbb{N} & \\ & & S_m^n & & \end{array}$$

Aplicaciones (I)

Existe una transformación automática que obtiene el programa que **calcula la composición de funciones computadas por dos programas** **(no la concatenación de éstos)**:

Corolario: Existe $g \in \mathcal{PR}^{(2)}$ tal que $\forall\ e_1,\ e_2 \in \mathbb{N}$

$$\varphi^1_{e_1} \circ \varphi^1_{e_2} \simeq \varphi^1_{g(e_1,e_2)}$$

- **Demostración**: Consideremos la función $h : \mathbb{N}^3 - \to \mathbb{N}$

$$h(z,x,y) = \varphi_x(\varphi_y(z))$$

Se verifica que $h \in \mathcal{P}$.

Esto es cierto pues $h(z,x,y) = \mathcal{U}_1(\mathcal{U}_1(z,y),x)$

Por tanto, existe e tal que $\varphi^3_e = h$. Entonces, aplicando s–2–1, para cualesquiera z, e_1, e_2

$$h(z,e_1,e_2) = \varphi_{e_1} \circ \varphi_{e_2}(z) = \varphi_e(z,e_1,e_2) = \varphi^{(1)}_{S^2_1(e,e_1,e_2)}(z)$$

y por tanto $g(u,v) = S^2_1(e,u,v)$ verifica lo pedido

Aplicaciones (II)

Corolario: Existe una transformación automática de programas, $\mathtt{T}$ tal que para cualesquiera dos programas $\mathtt{p}$ y $\mathtt{q}$,

$$[\![\mathtt{T}(\mathtt{p},\mathtt{q})]\!]^{(1)} = [\![\mathtt{p}]\!]^{(1)} \circ [\![\mathtt{q}]\!]^{(1)}$$

Demostración: $\mathtt{T}$ es la transformación que hace conmutativo el diagrama

$$\begin{array}{ccccc} & & \overset{\mathtt{T}}{\longrightarrow} & & \\ \mathtt{GOTO}_{\mathtt{P}} \times \mathtt{GOTO}_{\mathtt{P}} & & & \mathtt{GOTO}_{\mathtt{P}} & \\ \#(.)^{(2)} \downarrow & & & \downarrow \#(.) & \\ \mathbb{N}^2 & & \overset{g}{\longrightarrow} & \mathbb{N} & \end{array}$$

para g la función del teorema anterior.

Teorema de Recursión:

Sea $g \in \mathcal{P}^{n+1}$ recursiva. Existe $e \in \mathbb{N}$ tal que

$$\forall \vec{x} \in \mathbb{N}^n \left[\varphi_e^{(n)}(x_1, \ldots x_n) = g(e, x_1, \ldots, x_n)\right]$$

- **Demostración** Sea $h \in \mathcal{P}$ definida por

$$h(x_1, \ldots, x_n, v) = g(S_n^1(v, v), x_1, \ldots, x_n)$$

consideremos $e_0 \in \mathbb{N}$ tal que $h = \varphi_{e_0}^{(n+1)}$. Entonces

$$g(S_1^1(v, v), x_1, \ldots, x_n) = \varphi_{e_0}^{n+1}(\vec{x}, v) = \varphi_{S_n^1(e_0, v)}(\vec{x})$$

y tomando $v = e_0$ y $e = S_n^1(e_0, e_0)$ se tiene que

$$\varphi_e^{(n)}(x_1, \ldots, x_n) = g(e, x_1, \ldots, x_n)$$

Teorema de recursión para programas

Corolario Sea $\mathtt{p} \in \mathtt{GOTO}_{\mathtt{P}}$ y $n \geq 1$. Existe q tal que

$$\forall \vec{x} \in \mathbb{N}^n \ \left[[\![\mathtt{p}]\!]^{(n+1)}(\#(\mathtt{q}), \vec{x}) = [\![\mathtt{q}]\!]^{(n)}(\vec{x}) \right]$$

- **Demostración**: Basta aplicar el T. de recursión a $[\![\mathtt{p}]\!]^{(n+1)}$, y q es el programa de código e tal que

$$\varphi_e(\vec{x}) = [\![\mathtt{p}]\!]^{(n+1)}(e, \vec{x})$$

Es evidente (basta examinar la demostración del teorema de recursión) que el programa q se obtiene de **manera efectiva**.

Teorema del punto fijo de Kleene

Corolario (teorema del punto fijo de Kleene).
Sea $f \in \mathcal{R}^{(1)}$. Existe e tal que

$$\varphi_{f(e)} = \varphi_e$$

- **Demostración**. Consideremos la función $g : \mathbb{N}^2 \to \mathbb{N}$

  $$g(z, x) = \varphi_{f(z)}(x)$$

  que es recursiva, pues $g(z, x) = \mathcal{U}_1(x, f(z))$. Por el teorema de recursión existe e tal que

  $$\forall x[\ g(e, x) = \varphi_e(x))]$$

  luego para todo $x \in \mathbb{N}$, $\varphi_e(x) = g(e, x) = \varphi_{f(e)}(x)$

Análogamente se demuestra para cualquier aridad $n \geq 1$: en las condiciones del teorema, existe e tal que $\varphi^{(n)}_{f(e)} = \varphi^{(n)}_e$

Teorema del punto fijo para programas

Corolario (puntos fijos de transformaciones automáticas de programas).
Sea $\mathtt{T} : \mathtt{GOTO}_{\mathtt{P}} \to \mathtt{GOTO}_{\mathtt{P}}$ una transformación automática de programas y $n \geq 1$. Entonces existe $\mathtt{p} \in \mathtt{GOTO}_{\mathtt{P}}$ tal que

$$[\![\mathtt{T}(\mathtt{p})]\!] = [\![\mathtt{p}]\!]$$

- **Demostración**: Aplíquese el teorema del punto fijo a la función recursiva que hace conmutativo el diagrama

$$\begin{array}{ccccc} & & \mathtt{T} & & \\ \mathtt{GOTO}_{\mathtt{P}} & & \longrightarrow & & \mathtt{GOTO}_{\mathtt{P}} \\ \#(.) \quad \downarrow & & & & \downarrow \quad \#(.) \\ \mathbb{N} & & \underset{f}{\longrightarrow} & & \mathbb{N} \end{array}$$

Es decir, **para toda transformación automática de programas existe un programa que calcula la misma función que su transformado**.

Aplicaciones (I): Existencia de virus

Programa **Virus**: se reproduce completo o alguna de sus características.
Un programa p es un **virus autorreplicante** (v.a.) si al ejecutarse reproduce su código, es decir

$$\forall x \in \mathbb{N}, \qquad [\![\mathtt{p}]\!]^{(1)}(x) = \#(\mathtt{p})$$

Encontrar v.a. en GOTO es difícil.
Proposición: Existen virus autorreplicantes.

- **Demostración** Consideremos la función recursiva

$$g(u, v) = u$$

Por el teorema de recursión, existe $e \in \mathbb{N}$ tal que

$$\forall x \qquad \varphi_e(v) = g(e, v) = e$$

Si p es tal que $\#(\mathtt{p}) = e$, entonces p es un tal virus.

Aplicaciones (II): Inexistencia de antivirus

Teorema: El problema de reconocer los virus autorreplicantes

$$\texttt{AUTORREPLICANTE}(\texttt{p}) \Longleftrightarrow \texttt{p es un virus autorreplicante}$$

No es recursivo

Demostración

- **Demostración** Supongamos que el predicado

$$P(u) \iff u \text{ es el código de un virus autorreplicante}$$
$$\iff \forall x\, \varphi_u(x) = u$$

es recursivo. Fijemos un índice e_0 que no sea el código de un v.a., y consideremos la función (*diagonalizamos*).

	0	1	2	...
φ_0	$\boxed{\varphi_0(0)}$	$\varphi_0(1)$	$\varphi_0(2)$	...
φ_1	$\varphi_1(0)$	$\boxed{\varphi_1(1)}$	$\varphi_1(2)$	...
φ_2	$\varphi_2(0)$	$\varphi_2(1)$	$\boxed{\varphi_2(2)}$	...
$\vdots$				
f	δ_0	δ_1	δ_2	...

$$f(u,x) = \begin{cases} e_o & \text{si } P(u) \\ u & \text{si } \neg P(u) \end{cases}$$

Demostración (II)

- f es recursiva. Aplicando el teor. de recursión existe $e \in \mathbb{N}$ tal que

$$\varphi_e(x) = f(e, x)$$

Pero considerando e:

  - Si e es un v.a. entonces $\varphi_e(e) = e$, pero $f(e) = e_0$. Contradicción
  - Si e no es un v.a., entonces $\varphi_e(x) \neq \mathcal{C}_e$ pero $f = \mathcal{C}_e$. Contradicción

Aplicaciones (III): Resolución de ecuaciones

Uso del t. de recursión para encontrar funciones computables verificando ciertas condiciones.

Ejemplo 1: Demostrar la existencia de una única f computable tal que

$$f(x) = \begin{cases} 1 & \text{si } x = 0 \\ f(\lfloor \frac{x}{2} \rfloor) + x & \text{si } x > 0 \end{cases}$$

(se puede establecer utilizando la función historia, y se prueba incluso que es p.r.)

Utilizando el t. de recursión: Debe ocurrir que $f = \varphi_e$ para cierto e. Por tanto, buscamos e tal que:

$$\varphi_e(x) = \begin{cases} 1 & \text{si } x = 0 \\ \varphi_e(\lfloor \frac{x}{2} \rfloor) + x & \text{si } x > 0 \end{cases}$$

aplicando el t. de recursión...

Consideremos $g(u,x) = \begin{cases} 1 & \text{si } x = 0 \\ \varphi_e(\lfloor \frac{x}{2} \rfloor) + x & \text{si } x > 0 \end{cases}$
se establece su computabilidad:

$$g(u,x) = \begin{cases} 1 & \text{si } x = 0 \\ \mathcal{U}_1(\lfloor \frac{x}{2} \rfloor, u) + x & \text{si } x > 0 \end{cases}$$

y aplicando el teorema de recursión, existe e tal que

$$\varphi_e(x) = g(e,x)$$

Entonces $f = \varphi_e$ satisface las ecuaciones.
La unicidad se prueba demostrando que la solución debe ser total (si existe) aplicando inducción.

Ejemplo de no unicidad

Ejemplo: Demostrar que existe $f \in \mathcal{P}$ tal que

$$\forall x \in \mathbb{N}, \quad f(x) = f(x+11)$$

Sea g la función computable
$g(u,x) = \varphi_u(x+11) = \mathcal{U}_1(x+11,u)$
Por el teorema de recursión, existe e tal que

$$\varphi_e(x) = g(e,x) = \varphi_e(x+11)$$

Nótese que:

- Es una igualdad entre funciones parciales. Por ejemplo, la **función vacía** es solución.
- No hay solución única, toda función constante lo es
- El t. de recursión **no** asegura la unicidad ni de e (por ejem., todo e tal que $\varphi_e = \emptyset$ sirve), ni de la función φ_e.

Ejemplo: La función 91 de McCarthy

Probar la existencia de $f \in \mathcal{P}$ tal que:

$$f(x) = \begin{cases} x - 10 & \text{si } x > 100 \\ f(f(x+11)) & \text{e.o.c.} \end{cases}$$

Consideremos $g(u,x) = \begin{cases} x - 10 & \text{si } x > 100 \\ \varphi_u(\varphi_u(x+11)) & \text{e.o.c.} \end{cases}$

es recursiva, pues

$$g(u,x) = \begin{cases} x - 10 & \text{si } x > 100 \\ \mathcal{U}_1(\mathcal{U}_1(x+11,u),u) & \text{e.o.c.} \end{cases}$$

por el t. de recursión, existe e tal que $\forall x\ \varphi_e(x) = g(e,x)$ y por tanto e es el código de un programa que calcula f. En este caso, la única función[1] f solución es

$$f(x) = \begin{cases} x - 10 & \text{si } x > 100 \\ 91 & \text{e.o.c.} \end{cases}$$

[1] La unicidad queda como ejercicio

Ejemplo: La función de Ackermann

Dada la sucesión

$$\begin{cases} A_0(x) = x + 1 \\ \\ A_{n+1}(x) = (A_n \circ \overset{(x+1)}{\cdots} \circ A_n)(1) \end{cases}$$

la función de Ackermann es $A(n, x) \overset{def.}{=} A_n(x)$. es decir, la función

$$(*) \begin{cases} A(0, x) = x + 1 \\ \\ A(n + 1, 0) = n + 2 \\ \\ A(n + 1, x) = A(n, A(n + 1, x - 1)) \text{ si } x > 0 \end{cases}$$

Aplicando el t. de recursión...

Veamos que es recursiva (esta función no es p.r.).
Consideremos $g : \mathbb{N}^3 - \to \mathbb{N}$ definida por

$$g(u,n,x) = \begin{cases} x+1 & \text{si } n = 0 \\ n+1 & \text{si } x = 0, n > 0 \\ \varphi_u(n-1, \varphi_u(n, x-1)) & \text{si } x, n > 0 \end{cases}$$

que es computable, pues

$$g(u,n,x) = \begin{cases} x+1 & \text{si } n = 0 \\ n+1 & \text{si } x = 0, n > 0 \\ \mathcal{U}_2, (n-1, \mathcal{U}_2(n, x-1, u), u) & x, n > 0 \end{cases}$$

aplicando el teorema de recursión, existe e tal que

$$\varphi_e(n,x) = g(e,n,x)$$

y como $A = \varphi_e^{(2)}$, la función de Ackermann es recursiva

Detalles...

La demostración no está completa:

- Se necesita demostrar la unicidad de la función que verifica las ecuaciones $(*)$, que se hace probando por inducción en n la unicidad de A_n.
- Previamente se demuestra que las funciones A_n son totales.