Modelos de Computación y Complejidad

Grado en Ingeniería Informática. Tecnologías Informáticas

Tema 4: Recursividad Enumerable e Indecidibilidad

David Orellana Martín
Mario de J. Pérez Jiménez

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática
Universidad de Sevilla

marper@us.es

http://www.cs.us.es/~marper/

Curso 2021-2022

Objetivos

Estudiar las limitaciones de los modelos de computación.

- Conjuntos recursivamente enumerables.
- Predicados parcialmente decidibles.
- Problemas de decisión.
  - Decidibilidad.
  - Semidecidibilidad.
  - Indecidibilidad.
- El problema de la parada.

Modelos de computación

¿En qué consiste un modelo de computación?

* Definir **formalmente** el concepto intuitivo de **procedimiento mecánico**.

* **Todo procedimiento informal**, de acuerdo con la idea intuitiva del término, debe ser un caso particular de procedimiento mecánico que ha sido rigurosamente definido en el modelo.

Los modelos de computación tienen la capacidad de **resolver problemas abstractos** a través de sus procedimientos mecánimos.

* Si M es un modelo de computación, notaremos $\mathcal{P}(M)$ el conjunto de problemas abstractos resolubles por los procedimientos mecánicos de M.

La **potencia computacional** de un modelo de computación M viene dada por el conjunto $\mathcal{P}(M)$.

Limitaciones de los modelos de computación

Sean M_1 y M_2 dos modelos de computación.

* ¿Qué significa que M_2 sea **más potente** que M_1?
  * $\mathcal{P}(M_1) \subseteq \mathcal{P}(M_2)$
* ¿Qué significa que M_1 y M_2 tengan la **misma potencia computacional**?
  * $\mathcal{P}(M_1) = \mathcal{P}(M_2)$
* **Modelos** de computación **universales**.
  * Funciones **recursivas** (K. Gödel).
  * Funciones **λ-calculables** (A. Church y S. Kleene).
  * **Máquinas de Turing** (A. Turing).

Limitaciones de un **modelo de computación**:

* **Existen problemas abstractos** que **no** **son resolubles mecánicamente** en ningún modelo de computación universal.

Consideraciones históricas

A. Church: irresolubilidad mecánica de un problema abstracto (abril, 1936).

A. Turing: mismo resultado, utilizando sus famosas máquinas (mayo, 1936).

A. Turing: irresolubilidad mecánica del **problema de la parada** (mayo, 1936).

* "*Dado un procedimiento mecánico y un dato de entrada, determinar si el procedimiento* **para** *sobre ese dato*"

Lo más interesante del resultado obtenido fue el método utilizado para realizar la prueba: **reducibilidad** de un problema a otro.

A. Turing: equivalencia de los modelos de las funciones **recursivas**, las funciones λ-**calculables** y las **máquinas de Turing** (agosto, 1936).

El modelo de computación GOTO

* Modelo de computación **orientado a programas**.
* Los procedimientos mecánicos del modelo son los programas GOTO.
* Modelo de computación **universal**.

Limitaciones del modelo de computación GOTO:

* Existen problemas abstractos que **no son resolubles** a través de programas GOTO.

El modelo de computación GOTO

Todo conjunto de tuplas de números naturales tiene asociado, de manera natural, un problema abstracto, que notaremos X_A:

* Sea $A \subseteq \mathbb{N}^k$, con $k \geq 1$. El problema abstracto asociado al conjunto A es el siguiente:

  Dada una k-tupla $(x_1, \ldots, x_k)$ *de números naturales, determinar si* $(x_1, \ldots, x_k) \in A$.

Resolver mecánicamente el problema abstracto X_A en el modelo de computación GOTO, equivale a demostrar que el conjunto A es GOTO-computable.

Limitaciones del modelo de computación GOTO

Establecer las limitaciones del modelo de computación GOTO:

* Demostrar la existencia de conjuntos que **no son GOTO-computable**.

Se va a establecer "**algo más**".

* Se va a probar la existencia de problemas abstractos que, si bien **no son** GOTO-computables, en realidad, "**casi lo son**".

La diferencia, cómo no, está en el "**casi**".

También se va a establecer la existencia de problemas abstractos que **no son** GOTO-computables, ni siquiera "**con el casi**".

Conjuntos GOTO-computables

Función característica de un conjunto $A \subseteq \mathbb{N}^k$:

- ⋆ Función total $C_A : \mathbb{N}^k \longrightarrow \{0, 1\}$ definida por:

$$C_A(\vec{x}) = \begin{cases} 1 & \text{si } \vec{x} \in A \\ 0 & \text{si } \vec{x} \in \mathbb{N}^k - A \end{cases}$$

- ⋆ Es un **predicado** de aridad k sobre $\mathbb{N}$: $C_A(\vec{x}) \equiv \vec{x} \in A$.

Conjunto GOTO–computable:

- ⋆ Su función característica es GOTO–computable.
- ⋆ El predicado de pertenencia $\theta(\vec{x}) \equiv \vec{x} \in A$ es GOTO-computable.

¿Peculiaridad de los conjuntos GOTO-computables?

Sea $A \subseteq \mathbb{N}^k$ un conjunto GOTO-computable.

- ⋆ Existe un programa GOTO, Q, tal que para cada k-tupla, $\vec{x}$, de números naturales:
  - **Si** $\vec{x} \in A$ entonces **la computación** $Q(\vec{x})$ **para** y **devuelve 1**.
  - **Si** $\vec{x} \notin A$ entonces **la computación** $Q(\vec{x})$ **para** y **devuelve 0**.

  El programa Q es capaz de "**distinguir**/**reconocer**" las tuplas que **son** del conjunto, de aquellas que **no lo son**.

Así pues, en los conjuntos GOTO-computables se puede decidir **mecánicamente** ...

Conjuntos recursivamente enumerables

Supongamos ahora que para un cierto conjunto $A \subseteq \mathbb{N}^k$:

- ⋆ Existe un programa GOTO, Q, tal que para cada k-tupla, $\vec{x}$ de números naturales:
  - **Si** $\vec{x} \in A$ entonces **la computación** $Q(\vec{x})$ **para** y **devuelve 1**.
  - **PERO**, **si** $\vec{x} \notin A$ entonces **la computación** $Q(\vec{x})$ **no para**.

Esta condición es **más débil** que la de GOTO-computabilidad.

En principio, en estos conjuntos, sólo se puede decidir **mecánicamente** ...

- ⋆ Son los conjuntos denominados **recursivamente enumerables** (**r.e.**)

Conjuntos recursivamente enumerables

La **función característica parcial** de un conjunto $A \subseteq \mathbb{N}^k$ es la función parcial C_A^* de $\mathbb{N}^k$ en $\{0,1\}$ definida por:

$$C_A^*(\vec{x}) = \begin{cases} 1 & \text{si } \vec{x} \in A \\ \uparrow & \text{si } \vec{x} \in \mathbb{N}^k - A \end{cases}$$

Definición: Un conjunto $A \subseteq \mathbb{N}^k$ es **recursivamente enumerable** (**r.e.**) si y sólo si la función característica parcial C_A^* es GOTO–computable.

Proposición 1: Sea $A \subseteq \mathbb{N}^k$. Son equivalentes:

(1) A es un conjunto recursivamente enumerable.

(2) Existe una función GOTO–computable f de aridad k tal que $A = dom(f)$.

Demostración:

$\boxed{(1) \Rightarrow (2)}$ Si A es r.e. entonces la función C_A^* es GOTO–computable. Además, se tiene que $dom(C_A^*) = A$.

$\boxed{(2) \Rightarrow (1)}$ Supongamos que existe una función GOTO–computable f de aridad k tal que $A = dom(f)$.

Entonces se tiene que para cada $\vec{x} \in \mathbb{N}^k$: $C_A^*(\vec{x}) = C_1^{(1)}\big(f(\vec{x})\big)$.

Luego, $C_A^* = C_1^{(1)} \circ f$; es decir, la función C_A^* es GOTO–computable.

Por tanto, el conjunto A es **r.e.**

■

Nota 1: $A \subseteq \mathbb{N}^k$ es **r.e.** si y sólo si existe un programa GOTO, P, tal que

$$P(\vec{x})\downarrow \iff \vec{x} \in A\,, \quad \text{para todo } \vec{x} \in \mathbb{N}^k$$

Notación: Para cada $n \in \mathbb{N}$ y cada $k \geq 1$, notaremos $W_n^{(k)} = dom\,(\varphi_n^{(k)})$.

Funciones GOTO–computables

$\varphi_0^{(1)}$	$\varphi_1^{(1)}$	$\varphi_2^{(1)}$	$\ldots\ldots$	$\varphi_n^{(1)}$	$\ldots\ldots$
$\varphi_0^{(2)}$	$\varphi_1^{(2)}$	$\varphi_2^{(2)}$	$\ldots\ldots$	$\varphi_n^{(2)}$	$\ldots\ldots$
$\varphi_0^{(3)}$	$\varphi_1^{(3)}$	$\varphi_2^{(3)}$	$\ldots\ldots$	$\varphi_n^{(3)}$	$\ldots\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\varphi_0^{(k)}$	$\varphi_1^{(k)}$	$\varphi_2^{(k}$	$\ldots\ldots$	$\varphi_n^{(k)}$	$\ldots\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Conjuntos recursivamente enumerables

$W_0^{(1)}$	$W_1^{(1)}$	$W_2^{(1)}$	$\ldots\ldots$	$W_n^{(1)}$	$\ldots\ldots$
$W_0^{(2)}$	$W_1^{(2)}$	$W_2^{(2)}$	$\ldots\ldots$	$W_n^{(2)}$	$\ldots\ldots$
$W_0^{(3)}$	$W_1^{(3)}$	$W_2^{(3)}$	$\ldots\ldots$	$W_n^{(3)}$	$\ldots\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$W_0^{(k)}$	$W_1^{(k)}$	$W_2^{(k}$	$\ldots\ldots$	$W_n^{(k)}$	$\ldots\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Proposición 2: Si $A \subseteq \mathbb{N}^k$ es un conjunto GOTO–computable, entonces A es **r.e.**

Demostración:

Supongamos que $A \subseteq \mathbb{N}^k$ es un conjunto GOTO–computable. Entonces, la función característica C_A es GOTO–computable.

Consideremos el siguiente programa GOTO:

$$P \equiv \boxed{[B_1]\ \ \text{IF } C_A(X_1, X_2, ..., X_k) = 0 \text{ GOTO } B_1}$$

Se verifica:

- $dom([\![P]\!]^{(k)}) = \{\vec{x} \in \mathbb{N}^k : C_A(\vec{x}) = 1\} = A$

Por tanto, el conjunto A es **r.e.**

■

¿Será cierto el recíproco?

¿**Existen conjuntos r.e. que no sean GOTO–computables**?

Más adelante veremos que la respuesta es afirmativa.

Proposición 3: $A, B \subseteq \mathbb{N}^k$ **r.e.** $\Longrightarrow$ $A \cap B$ y $A \cup B$ son **r.e.**

Demostración:

Sean f y g funciones GOTO–computables tales que $dom(f) = A$ y $dom(g) = B$.

Sean e_1 y e_2 los códigos de ciertos programas que calculan f y g.

Consideremos los programas GOTO:

$$P : \begin{cases} Y \longleftarrow f(X_1, ..., X_k) \\ Y \longleftarrow g(X_1, ..., X_k) \end{cases}$$

$$Q : \begin{cases} [A_1] & IF\ STEP^{(k)}(X_1, ..., X_k, e_1, Z)\ GOTO\ E \\ & IF\ STEP^{(k)}(X_1, ..., X_k, e_2, Z)\ GOTO\ E \\ & Z \longleftarrow Z + 1 \\ & GOTO\ A_1 \end{cases}$$

Se verifica: $dom([\![P]\!]^{(k)}) = A \cap B$ y $dom([\![Q]\!]^{(k)}) = A \cup B$.

Por tanto, los conjuntos $A \cap B$ y $A \cup B$ son **r.e.**

■

El **complementario** de un conjunto **r.e.** ¿será **r.e.**?

Teorema del Complemento o de la Negación

Teorema. $A \subseteq \mathbb{N}^k$ es GOTO–computable $\iff$ A y $\overline{A}$ son **r.e.**

Demostración:

$\Longrightarrow$ Si A es GOTO–computable, entonces $\overline{A}$ es GOTO–computable.
Luego A y $\overline{A}$ son **r.e.**

$\Longleftarrow$ Supongamos que A y $\overline{A}$ son **r.e.**

Sean f y g funciones GOTO–computables tales que $dom(f) = A$ y $dom(g) = \overline{A}$.

Sean e_1 y e_2 los códigos de ciertos programas que calculan f y g.

Entonces, el siguiente programa calcula la función característica C_A :

$$Q : \begin{cases} [A_1] & IF\ STEP^{(k)}(X_1, ..., X_k, e_1, Z)\ GOTO\ C \\ & IF\ STEP^{(k)}(X_1, ..., X_k, e_2, Z)\ GOTO\ E \\ & Z \longleftarrow Z + 1 \\ & GOTO\ A_1 \\ [C] & Y \longleftarrow Y + 1 \end{cases}$$

Por tanto, A es un conjunto GOTO–computable. ■

Si un conjunto **r.e.** **no** es GOTO-computable, entonces su complementario **no** es **r.e.**

La gráfica de una función parcial

Dada una función parcial f de $\mathbb{N}^k$ en $\mathbb{N}$, se define la **gráfica** de f como sigue:

$$G(f) = \{(x_1, \ldots, x_k, y) \in \mathbb{N}^{k+1} : \ f(x_1, \ldots, x_k) = y\}$$

Es decir, la gráfica de una función de aridad k es un conjunto de aridad $k+1$.

En el tema 2, se probó que:

- ⋆ Si la función f es **total** y GOTO-computable, entonces la gráfica $G(f)$ es un conjunto GOTO-computable.

¿Qué sucede si la función f de $\mathbb{N}^k$ en $\mathbb{N}$ es parcial?

Proposición 4: Si f es una función **parcial** GOTO-computable de $\mathbb{N}^k$ en $\mathbb{N}$, entonces la **gráfica** de f es un conjunto **r.e.**

Demostración:
Consideremos la función auxiliar $g : \mathbb{N}^{k+1} \dashrightarrow \mathbb{N}$ definida como sigue:

$$g(x_1, \ldots, x_k, y) = \mu z \left[d(f(x_1, \ldots, x_k), y) = 0 \right]$$

Entonces se verifica:

- La función g es GOTO-computable (minimización no acotada de una función GOTO-computable).
- $dom(g)$ es un conjunto **r.e.**
- $dom(g) = \{(x_1, \ldots, x_k, y) \in \mathbb{N}^{k+1} \mid f(x_1, \ldots, x_k) = y\} = G(f)$.

Por tanto, el conjunto $G(f)$ es **r.e.**

■

Nota: La función **distancia** de $\mathbb{N}^2$ en $\mathbb{N}$ se define como sigue: la imagen de (z, t), que notamos $d(z, t)$, es igual a $z - t$, si $z \geq t$, y a $t - z$ si $t > z$. Dicha función es GOTO-computable.

Se puede probar que el **recíproco** del resultado anterior también es cierto.

Proposición 5: Sea f una función **parcial** de $\mathbb{N}^k$ en $\mathbb{N}$ tal que **gráfica** de f es un conjunto **r.e.** Entonces, la función f es GOTO-computable.

- ⋆ Este resultado proporciona un **método** interesante para establecer la GOTO-computabilidad de funciones.
- ⋆ Para demostrar que una **función** es **GOTO-computable**, es **suficiente** probar que su **gráfica** es un conjunto **r.e.**

Recuérdese que en el tema 2 se había establecido el **teorema de definición por casos** para **funciones totales GOTO-computables**.

A continuación, se extiende este resultado para **funciones parciales**.

Teorema de definición por casos para funciones parciales GOTO–computables

Teorema. Sean $f_1, ..., f_n$ **funciones parciales** GOTO–computables de aridad $k \geq 1$. Sean $A_1, ..., A_n$ conjuntos GOTO-computables que forman una partición de $\mathbb{N}^k$. Entonces la función:

$$g(\vec{x}) = \begin{cases} f_1(\vec{x}) & \text{si } \vec{x} \in A_1 \\ ... & ... \\ f_n(\vec{x}) & \text{si } \vec{x} \in A_n \end{cases}$$

es GOTO–computable.

Demostración:

Basta observar que:

$$(\vec{x}, y) \in G(g) \Longleftrightarrow [(\vec{x}, y) \in G(f_1) \wedge \vec{x} \in A_1] \vee ... \vee [(\vec{x}, y) \in G(f_n) \wedge \vec{x} \in A_n]$$

y tener presente las proposiciones 4 y 5. ■

Predicados parcialmente decidibles

Los **predicados GOTO–computable** se denominan también **decidibles**. Es decir,

- ⋆ Un **predicado** θ de aridad k es **decidible** si y sólo si el conjunto $A = \{\vec{x} \in \mathbb{N}^k : \theta(\vec{x}) = 1\}$ es GOTO-computable.

Definición. Un **predicado** θ de aridad k es **parcialmente decidible** (p.d.) si y sólo si el conjunto $A = \{\vec{x} \in \mathbb{N}^k : \theta(\vec{x}) = 1\}$ es r.e.

- ⋆ La **conjunción** y la **disyunción** de predicados p.d. son predicados p.d.
  - La **negación** de un predicado p.d. **no tiene porqué ser** un predicado p.d.

- ⋆ Se demuestra que **si un predicado $\theta(\vec{x}, t)$ es p.d. entonces el predicado $\exists t\ \theta(\vec{x}, t)$ también es p.d.**
  - Si $\theta(\vec{x}, t)$ es un predicado p.d. entonces $\forall t\ \theta(\vec{x}, t)$ no tiene porqué ser un predicado p.d.

Problemas de decisión

Informalmente: *problema cuyas únicas posibles respuestas son* **sí** *(afirmativa) o* **no** *(negativa).*

Formalmente: Una terna ordenada $X = (\Sigma_X, E_X, \theta_X)$, en donde:

- $\star$ Σ_X es un alfabeto finito.
- $\star$ E_X es un lenguaje sobre Σ_X (**instancias** del problema).
- $\star$ θ_X es un predicado sobre E_X.

Todo problema de decisión $X = (\Sigma_X, E_X, \theta_X)$ tiene asociado un conjunto:

- $L_X = \{a \in E_X : \ \theta_X(a) = 1\}$.

Obsérvese que la relación de pertenencia al conjunto L_X es el predicado θ_X:

$$a \in L_X \iff \theta_X(a) = 1$$

Normalmente, el problema de decisión $X = (\Sigma_X, E_X, \theta_X)$ se expresa así:

Dado $a \in E_X$, determinar si $\theta_X(a) = 1$

Es decir: **Dada una instancia de X, determinar si la respuesta del problema es afirmativa**

Decidibilidad de problemas de decisión

Un problema de decisión $X = (\Sigma_X, E_X, \theta_X)$ es:

- **Decidible sii** el conjunto L_X es GOTO–computable;
- **Semidecidible sii** el conjunto L_X es r.e.;
- **Indecidible sii** el conjunto L_X **no** es GOTO–computable.

Terminología **formal** acerca de problemas de decisión:

- **Decidible** $\equiv$ **resoluble algorítmicamente**.
- **Semidecidible** $\equiv$ **semirresoluble algorítmicamente**.
- **Indecidible** $\equiv$ **irresoluble algorítmicamente**.

Terminología **informal** acerca de los problemas de decisión:

- **Decidible** $\equiv$ **resoluble mecánicamente**.
- **Semidecidible** $\equiv$ **"CASI" resoluble mecánicamente**.
- **Indecidible** $\equiv$ **NO resoluble mecánicamente**.

Equivalentemente, un problema de decisión $X = (\Sigma_X, E_X, \theta_X)$ es:

- **Decidible sii** el predicado θ_X es GOTO–computable.
- **Semidecidible sii** el predicado θ_X es parcialmente decidible.
- **Indecidible sii** el predicado θ_X **no** es GOTO–computable.

El problema de la parada

Enunciado informal: Dado un programa `GOTO`, P, y un dato de entrada $\vec{x} \in \mathbb{N}^k$, determinar si el programa P para sobre $\vec{x}$.

Cuestión informal: ¿Es resoluble mecánicamente el problema de la parada?

Enunciado formal I: Dados dos números naturales x_1, x_2, determinar si el programa de código x_1 con dato de entrada x_2, **para**; es decir, determinar si $\varphi^{(1)}_{x_1}(x_2)\downarrow$

Enunciado formal II: El problema de la parada (que notaremos `HALT`) es el problema de decisión: $(\mathbb{N}^2, \mathbb{N}^2, \texttt{HALT}(x,y))$, siendo $\texttt{HALT}(x,y)) \equiv \varphi^{(1)}_x(y)\downarrow$

Enunciado formal de la resolublidad algorítmica del **problema de la parada**:

Determinar si el conjunto $\mathcal{K}_0 = \{(x,y) \in \mathbb{N}^2 : \ \varphi^{(1)}_x(y)\downarrow\}$ es `GOTO`–computable

Cuestión: ¿Es decidible el problema de la parada?

Proposición 6: El problema de la parada es **indecidible** pero, en cambio, sí es **semidecidible**.

Demostración:

Indecidibilidad: El predicado $\texttt{HALT}(x_1, x_2)$ **no es GOTO–computable**.

- ⋆ Si el predicado $\texttt{HALT}(x_1, x_2) \equiv \varphi^{(1)}_{x_1}(x_2)\downarrow$ **fuese GOTO–computable**, entonces también lo sería la función:

$$f(x) = \begin{cases} \uparrow & \text{si } \texttt{HALT}(x,x) \text{ es verdadero} \\ 0 & e.c.o.c. \end{cases}$$

ya que el conjunto $G(f)$ sería GOTO-computable, pues

$$(x,y) \in G(f) \iff \neg\texttt{HALT}(x,x) \ \wedge \ y = 0$$

Por tanto, existiría un número natural $n \in \mathbb{N}$ tal que $f = \varphi^{(1)}_n$. Entonces:

- El predicado $\texttt{HALT}(n, n)$ **no puede ser verdadero**.
  - ∗ $\texttt{HALT}(n, n)$ verdadero $\Longrightarrow f(n)\uparrow \Longrightarrow \varphi^{(1)}_n(n)\uparrow \Longrightarrow \texttt{HALT}(n, n)$ falso.
- El predicado $\texttt{HALT}(n, n)$ **no puede ser falso**.
  - ∗ $\texttt{HALT}(n, n)$ falso $\Longrightarrow f(n) = 0 \Longrightarrow \varphi^{(1)}_n(n) = 0 \Longrightarrow \texttt{HALT}(n, n)$ verdadero.

Semidecidibilidad: El predicado $\texttt{HALT}(x_1, x_2)$ **es parcialmente decidible**.

- ⋆ En efecto: para cada par de números naturales x_1, x_2 se verifica lo siguiente

$$\texttt{HALT}(x_1, x_2) = 1 \Longleftrightarrow \varphi^{(1)}_{x_1}(x_2)\downarrow \Longleftrightarrow \exists t\ STEP^{(1)}(x_2, x_1, t)$$

Así pues,

- ∗ $\texttt{HALT}(x_1, x_2)$ es una cuantificación existencial de un predicado $(STEP^{(1)}(x_2, x_1, t))$ que es GOTO-computable.
- ∗ Por tanto, $\texttt{HALT}(x_1, x_2)$ será un predicado parcialmente decidible.

■

Observaciones

- ⋆ El predicado $\texttt{HALT}(x, y) \equiv \varphi_x^{(1)}(y) \downarrow$ es **p.d.** pero **no** es GOTO–computable.
- ⋆ El conjunto $\mathcal{K}_0 = \{(x, y) \in \mathbb{N}^2 : \ \varphi_x^{(1)}(y) \downarrow\}$ es **r.e.** pero **no** es GOTO–computable.
- ⋆ El conjunto $\mathcal{K} = \{x \in \mathbb{N} : \ \varphi_x^{(1)}(x) \downarrow\}$ es **r.e.** pero **no** es GOTO–computable.
- ⋆ El conjunto $\overline{\mathcal{K}} = \mathbb{N} - \mathcal{K}$ **no** es r.e. (por el teorema del complemento).

El conjunto $\mathcal{K}$ se denomina **conjunto del problema de la parada**.

En consecuencia:

- ⋆ Existen conjuntos **r.e.** que no son GOTO-computables (por ejemplo, el conjunto $\mathcal{K}$).
- ⋆ Existen conjuntos que ni siquiera son **r.e.** (por ejemplo, el conjunto $\overline{\mathcal{K}}$).

El teorema de Rice

Notación: $\texttt{GCOMP}^{(k)}$ es el conjunto de funciones GOTO-computables de aridad $k \geq 1$.

Si $\mathcal{F} \subseteq \texttt{GCOMP}^{(k)}$, notaremos: $I_{\mathcal{F}} = \{x \in \mathbb{N} : \ \varphi_x^{(k)} \in \mathcal{F}\}$ (**conjunto de índices** de $\mathcal{F}$).

Enunciado del teorema de Rice: Sea $\mathcal{F} \subseteq \texttt{GCOMP}^{(1)}$ tal que $\mathcal{F} \neq \emptyset$ y $\mathcal{F} \neq \texttt{GCOMP}^{(1)}$. Entonces, el conjunto de índices, $I_{\mathcal{F}}$, **NO es GOTO–computable**.

Observaciones:

- ⋆ El teorema de Rice proporciona un método para **probar** que un cierto conjunto $A \subseteq \mathbb{N}$ **no es GOTO–computable**.
- ⋆ ¿Cómo aplicar el teorema de Rice?

Paso 1. **Búsquese** una familia $\mathcal{F} \subseteq GCOMP^{(1)}$ tal que $I_{\mathcal{F}} = A$.

Paso 2. Pruébese que $\mathcal{F} \neq \emptyset$.

Paso 3. Pruébese que $\mathcal{F} \neq GCOMP^{(1)}$.

- ⋆ Entonces, de los pasos 1, 2 y 3 se concluye, por el teorema de Rice, que el conjunto A **no** es GOTO–computable.

Teorema de Rice: Ejemplo de ilustración

Ejemplo: Aplicando el teorema de Rice demostrar que **no** es GOTO-computable, el conjunto: $A = \{x \in \mathbb{N} : \text{ el dominio de la función } \varphi_x^{(1)} \text{es un conjunto infinito}\}$

Solución:

Paso 1. Hemos de buscar una clase de funciones $\mathcal{F} \subseteq \mathtt{GCOMP}^{(1)}$ cuyo conjunto de índices, $I_{\mathcal{F}} = \{x \in \mathbb{N} : \ \varphi_x^{(1)} \in \mathcal{F}\}$, sea, precisamente, A. Pues bien, fijándonos en la definición de A consideramos el conjunto de funciones $\mathcal{F} = \{f \in \mathtt{GCOMP}^{(1)} \mid dom(f) \text{ es un conjunto infinito}\}$. De acuerdo con esta definición, es inmediato que $I_{\mathcal{F}} = A$.

Paso 2. Veamos que $\mathcal{F}$ es no vacío; es decir que $\mathcal{F}$ contiene a alguna función GOTO-computable 1-aria. En efecto: la función identidad $(f(x) = x \ , \ \forall x \in \mathbb{N})$ **pertenece** a $\mathcal{F}$ pues su dominio (que coincide con $\mathbb{N}$) es un conjunto infinito (obviamente, toda función total de $\mathbb{N}$ en $\mathbb{N}$ satisface esa condición).

Paso 3. Veamos que $\mathcal{F}$ es distinto de $\mathtt{GCOMP}^{(1)}$; es decir que existen funciones GOTO-computables 1-aria que no pertenecen a $\mathcal{F}$.
En efecto: la función vacía es GOTO-computable (se puede considerar 1-aria) que **no pertenece** a $\mathcal{F}$, pues su dominio es un conjunto finito (concretamente, es el conjunto vacío).

Entonces, de los pasos 1, 2 y 3 se concluye, por el teorema de Rice, que el conjunto A **no** es GOTO–computable.

MÉTODOS

1. **MÉTODOS para determinar si una FUNCIÓN es GOTO-computable**

   (a) Hallar un programa GOTO que calcule la función.

   (b) Describir la función como una **composición**/**recursión primitiva**/ **minimización** de funciones GOTO-computables.

   (c) Describir la función como **definida por casos**, a partir de funciones GOTO-computables, y de casos descritos por predicados GOTO-computables.

   (d) Probar que la **gráfica** de la función es un conjunto R.E.

2. **MÉTODOS para determinar si una FUNCIÓN NO es GOTO-computable**

   NO hay un método **directo**: utilizar el **método por reducción al absurdo**.

MÉTODOS

3. MÉTODOS para determinar si un CONJUNTO es GOTO-computable

(a) Probar que la correspondiente **función característica** es GOTO-computable.

(b) Probar que la **relación de pertenencia** a dicho conjunto es un **predicado** GOTO-computable.

4. MÉTODOS para determinar si un CONJUNTO NO es GOTO-computable

(a) Utilizar el **teoremas de RICE**.

(b) Método por **reducción al absurdo**.

MÉTODOS

5. MÉTODOS para determinar si un CONJUNTO es R.E.

(a) Probar que la correspondiente **función característica PARCIAL** es GOTO-computable.

(b) Probar que la **relación de pertenencia** a dicho conjunto es un **predicado parcialmente decidible**.

6. MÉTODOS para determinar si un CONJUNTO NO es R.E.

(a) Utilizar el **teorema del complemento**.

(b) Método por **reducción al absurdo**.

MÉTODOS

7. MÉTODOS para determinar la RESOLUBILIDAD ALGORÍTMICA de PROBLEMAS DE DECISIÓN

Probar que el **conjunto asociado al problema de decisión**:

(a) **ES** `GOTO`-**computable** (en el caso de la **DECIDIBILIDAD**).

(b) **NO ES** `GOTO`-**computable** (en el caso de la **INDECIDIBILIDAD**).

(c) **ES R.E.** (en el caso de la **SEMIDECIDIBILIDAD**).

(d) **NO ES R.E.** (en el caso de la **NO SEMIDECIDIBILIDAD**).